DATA WRANGLING IN DATA SCIENCE.

Data Wrangling in data science.

Data Wrangling - generally refers to transforming raw data into a useable form for your analyses of interest, including loading, aggregating, merging, grouping, concatenating and formatting.

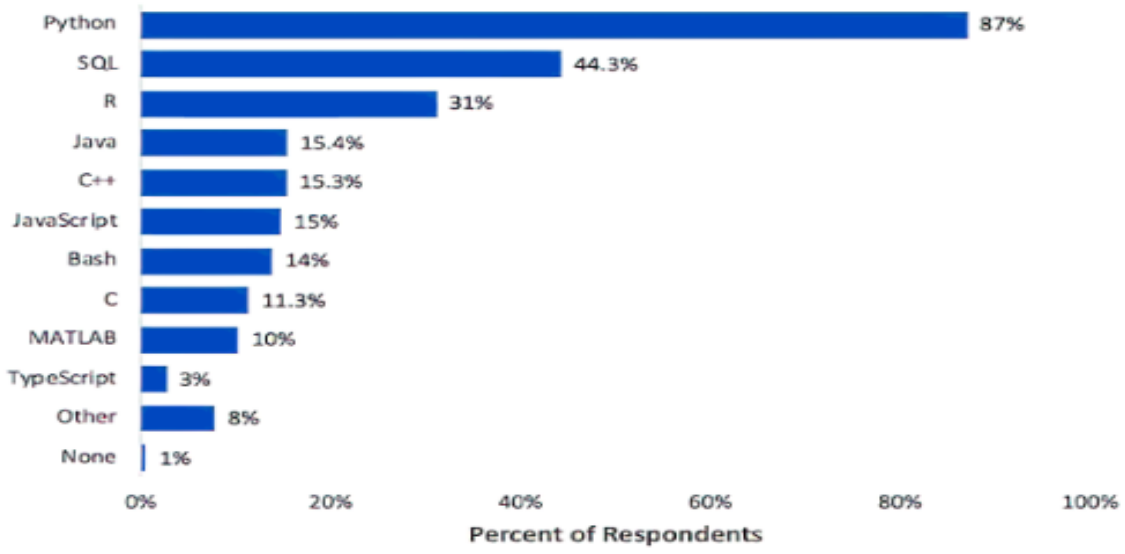
Also known as data munging or data preprocessing, is the process of cleaning, structuring, and transforming raw data into a format suitable for analysis or machine learning. It is a fundamental step in the data science and data analysis pipeline and is crucial for ensuring that the data you work with is accurate, complete, and ready for further exploration and modeling.

- **Data is not useful until** it can be analyzed and presented as insight that drives better decision making.
- **data cannot be effectively analyzed** until it is *well structured, clean, and converted into a suitable format*. Simply put, **that is why** good data wrangling is important.

Data wrangling with Python

Python is generally considered to be a data scientist's best friend. According to a 2019 survey, 87% of data scientists said they regularly used Python, far more than the next most used languages, SQL (44%) and R (31%). Data wrangling with Python is a common and powerful approach due to the availability of various libraries and tools designed for data manipulation and preprocessing. This process ensures that **data is prepared for automation and additional analysis.**

what programming language do you use on a regular basis?



Note: Data are from the 2019 Kaggle ML and Data Science Survey. You can learn more about the study here: https://www.kaggle.com/c/kaggle-survey-2019.

Here are the goals of data wrangling:

The important needs of data wrangling include,

Data scientists spend 75% - 80% of their time wrangling the data, which is not a surprise at all.

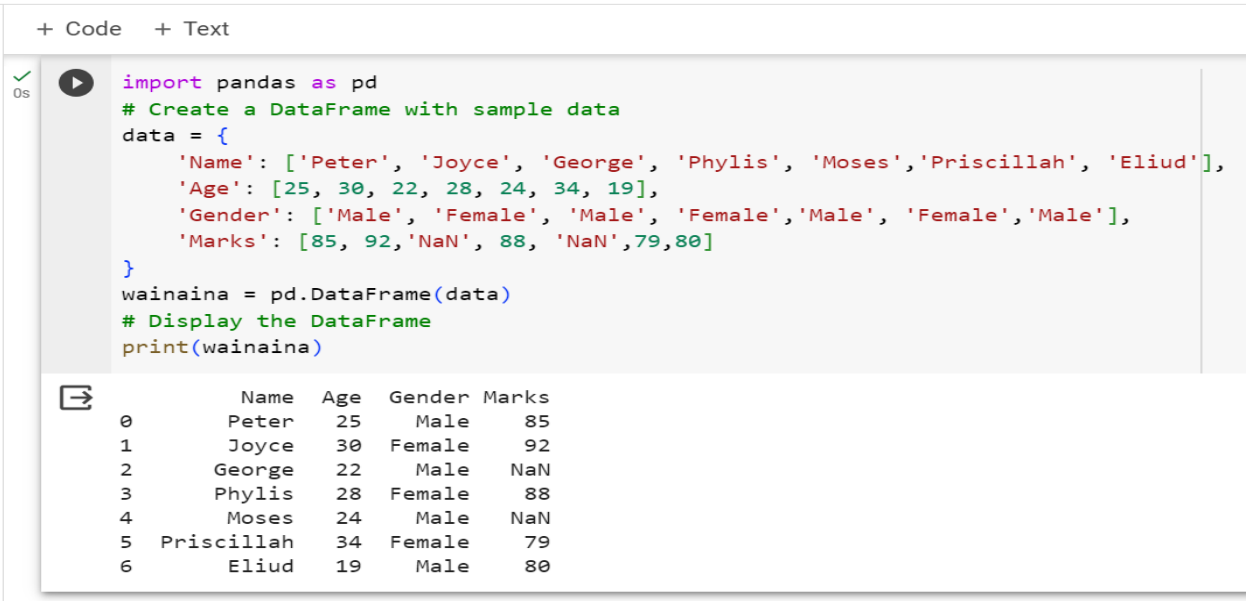
Need	Description
Data Quality Assurance	Ensuring the quality and reliability of data by identifying and addressing errors, inconsistencies, and outliers.
Timely Decision-Making	Supporting faster decision-making by making data readily available in a clean, structured format, enabling quick insights and analysis.
Handling Noisy and Flawed Data	Cleaning and preprocessing noisy, flawed, and incomplete data to ensure it is suitable for analysis and modeling.
Data Preparation for Mining	Structuring and organizing data in a way that makes it suitable for data mining processes, ensuring it's ready for pattern discovery and modeling.
Informed Decision-Making	Enabling data-driven decision-making by cleaning and structuring raw data into a usable format, providing a basis for informed choices.
Data Integration and Formatting	Combining and formatting raw data from various sources into a consistent and coherent structure, facilitating analysis and reporting.
Centralized Data Management	Establishing a centralized data repository for efficient data management, improving compliance, and ensuring data availability for decision-makers.
Prompt and Effective Decision-Making	Allowing decision-makers to access well-prepared data quickly, which enhances the efficiency of the decision-making process.

Data wrangling Key Competencies:

No.	Competency	Description	Example Usage
1	Outlier/Anomaly Detection	Apply Outlier Detection techniques to identify and handle outliers in the data.	Detecting and removing unusually high or low values in a dataset using methods like Z-score or IQR. Z-score and IQR (Interquartile Range) are both statistical methods used in outlier detection, specifically for identifying and handling outliers in a dataset.
2	Missing values in data	Clean data by finding and replacing missing values using data science libraries.	Filling missing age values in a dataset with the mean age or using interpolation techniques.
3	Duplicate values in data	Clean data by finding and removing duplicate values using data science libraries.	Identifying and eliminating rows with identical data in a database of customer information.
4	Categorical data to numeric data	Transform categorical data into numerical data using data science libraries.	Encoding categorical variables like "Gender" (e.g., Male -> 0, Female -> 1) for machine learning models.
5	Group data based on values	Group data within a single dataset based on specific values or criteria using data science libraries.	Grouping sales data by region to calculate regional sales totals or averages.

6	Concatenate data along an axis	Concatenate data along a specified axis (rows or columns) using Python data science libraries.	Combining multiple CSV files with the same structure into one larger dataset.
7	Merge multiple sets of data into a dataset	Join multiple sets of data into a single dataset using data science libraries, typically through merging or joining operations.	Combining customer data from two different databases using a common identifier like customer ID.

1. **Data exploration** - *here we assign the data, and then we visualize the data in a tabular format.*



```
+ Code  + Text

0s
import pandas as pd
# Create a DataFrame with sample data
data = {
    'Name': ['Peter', 'Joyce', 'George', 'Phylis', 'Moses','Priscillah', 'Eliud'],
    'Age': [25, 30, 22, 28, 24, 34, 19],
    'Gender': ['Male', 'Female', 'Male', 'Female','Male', 'Female','Male'],
    'Marks': [85, 92,'NaN', 88, 'NaN',79,80]
}
wainaina = pd.DataFrame(data)
# Display the DataFrame
print(wainaina)

          Name  Age  Gender Marks
0       Peter   25    Male    85
1       Joyce   30  Female    92
2      George   22    Male   NaN
3      Phylis   28  Female    88
4       Moses   24    Male   NaN
5  Priscillah   34  Female    79
6       Eliud   19    Male    80
```

Line(s)	Explanation
import pandas as pd	Import the Pandas library and alias it as **pd** to use its functions and classes in the code.
data = { ... }	Create a Python dictionary called **data** that contains sample data for four columns: 'Name', 'Age', 'Gender', and 'Marks'. Each column is associated with a list of values.
wainaina = pd.DataFrame(data)	Create a Pandas DataFrame named **wainaina** by passing the **data** dictionary to the **pd.DataFrame()** constructor. This DataFrame organizes the data into a tabular format.
print(wainaina)	Display the **wainaina** DataFrame to the console. The DataFrame shows the tabular representation of the data with columns for 'Name', 'Age', 'Gender', and 'Marks'.

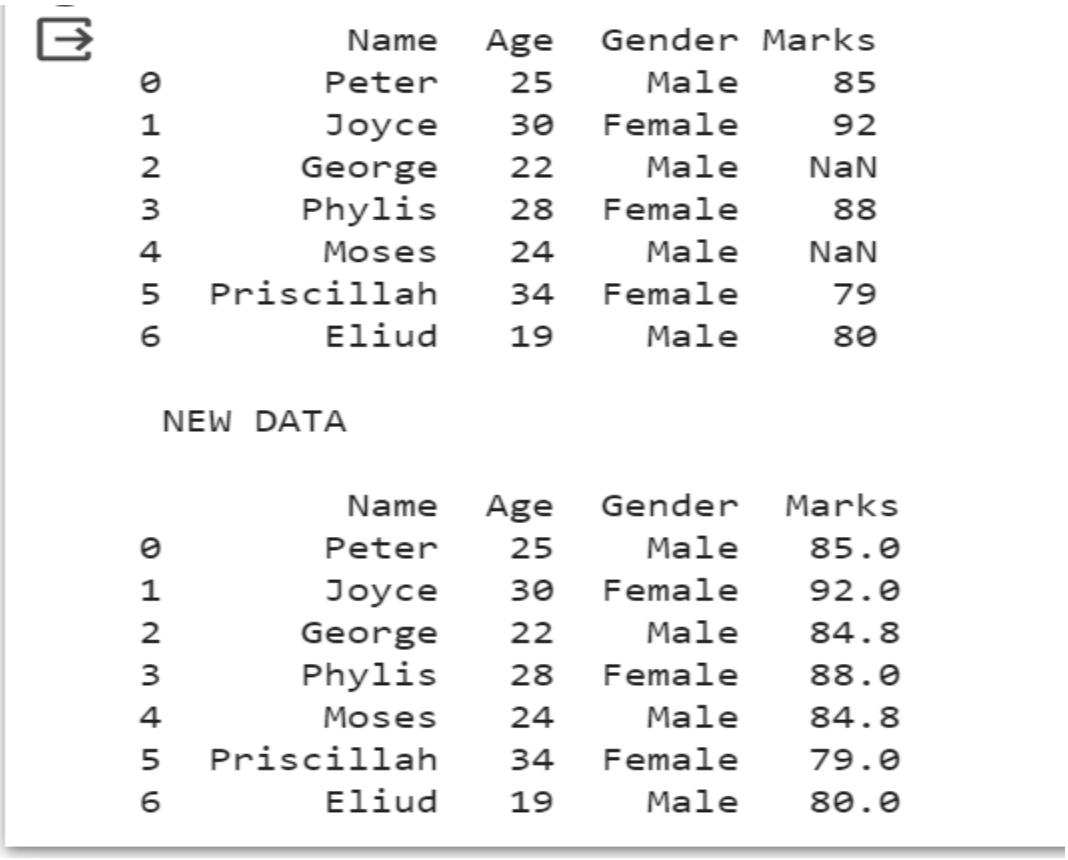
2. **Dealing with missing values**, *as we can see from the previous output, there are NaN values present in the MARKS column which are going to be taken care of by replacing them with the column mean.*

+ Code + Text

```
import pandas as pd
import numpy as np  # Import numpy for numeric operations
# Create a DataFrame with sample data
data = {
    'Name': ['Peter', 'Joyce', 'George', 'Phylis', 'Moses','Priscillah', 'Eliud'],
    'Age': [25, 30, 22, 28, 24, 34, 19],
    'Gender': ['Male', 'Female', 'Male', 'Female','Male', 'Female','Male'],
    'Marks': [85, 92,'NaN', 88, 'NaN',79,80]
}
wainaina = pd.DataFrame(data)
# Display the DataFrame
print(wainaina)
c=avg=0
for me in wainaina['Marks']:
  if str(me).isnumeric():
    c+=1
    avg+=me
avg/=c
wainaina=wainaina.replace(to_replace="NaN",value=avg)
print("\n NEW DATA \n")
print(wainaina)
```

Line(s)	Explanation
import pandas as pd	Import the Pandas library and alias it as **pd** to use its functions and classes in the code.
import numpy as np	Import the NumPy library to perform numeric operations.
data = { ... }	Create a Python dictionary named **data** containing sample data for four columns: 'Name', 'Age', 'Gender', and 'Marks'.
wainaina = pd.DataFrame(data)	Create a Pandas DataFrame named **wainaina** by passing the **data** dictionary to the **pd.DataFrame()** constructor. This DataFrame organizes the sample data into a tabular format.
print(wainaina)	Display the initial DataFrame **wainaina** to the console, showing the tabular representation of the sample data.
c=avg=0	Initialize two variables, **c** and **avg**, to zero. These variables will be used to count numeric values in the 'Marks' column and calculate their average.
for me in wainaina['Marks']:	Start a loop to iterate through each value in the 'Marks' column of the DataFrame **wainaina**.
if str(me).isnumeric():	Check if the current value **me** is numeric by converting it to a string and using the **.isnumeric()** method.
c+=1	If the value is numeric, increment the count **c** by 1.
avg+=me	If the value is numeric, add it to the **avg** variable.
avg/=c	Calculate the average by dividing the sum of numeric values (**avg**) by the count of numeric values (**c**).

wainaina=wainaina.replace(to_re ="NaN",value=avg)	Use the **.replace()** method to replace all occurrences of the string "NaN" in the DataFrame **wainaina** with the calculated average value **avg**.
print("\n NEW DATA \n")	Print a separator to distinguish the original data from the modified data in the console output.
print(wainaina)	Display the modified DataFrame **wainaina** after replacing the missing values with the calculated average. This shows the updated tabular data.



```
          Name  Age  Gender Marks
0        Peter   25    Male    85
1        Joyce   30  Female    92
2       George   22    Male   NaN
3       Phylis   28  Female    88
4        Moses   24    Male   NaN
5   Priscillah   34  Female    79
6        Eliud   19    Male    80

 NEW DATA

          Name  Age  Gender  Marks
0        Peter   25    Male   85.0
1        Joyce   30  Female   92.0
2       George   22    Male   84.8
3       Phylis   28  Female   88.0
4        Moses   24    Male   84.8
5   Priscillah   34  Female   79.0
6        Eliud   19    Male   80.0
```

3. Reshaping data*, in the GENDER column, we can reshape the data by categorizing them into different numbers.*

The essence of reshaping data in data science is to transform and reorganize data from one structure or format into another to make it more suitable for analysis, visualization, modeling, or other specific tasks. Data reshaping is a critical step in the data preprocessing and preparation process, and it serves several important purposes:

- *Data Compatibility:* Reshaping data can make it compatible with the requirements of a specific analysis or modeling technique. Different analytical methods often require data in different formats, so reshaping ensures that the data is in the right shape for the chosen approach.

- *Simplification:* Data reshaping can simplify complex data structures, making it easier to work with and understand. For example, pivoting or melting data can simplify multi-dimensional data into a more manageable format.
- *Aggregation and Summarization:* Reshaping can involve aggregating or summarizing data to create more meaningful insights. This is particularly useful for creating summary reports, dashboards, or generating key performance indicators (KPIs).
- *Data Transformation:* Reshaping may involve transforming data to meet specific requirements. This can include encoding categorical variables, normalizing or standardizing numerical data, and other data preprocessing steps.
- *Visualization:* Data reshaping can prepare data for visualization. Certain visualization tools or libraries may expect data in specific formats, so reshaping helps in creating informative and visually appealing charts and plots.
- *Efficiency:* Reshaping can improve data access and retrieval efficiency. By structuring data appropriately, you can reduce the time and resources needed to retrieve and analyze the data.
- *Data Integration:* Reshaping can facilitate the integration of data from multiple sources into a unified dataset, enabling cross-referencing and analysis across different data sets.
- *Modeling:* Data reshaping can help prepare data for machine learning and statistical modeling tasks. Models often have specific input requirements, and reshaping ensures that the data aligns with these requirements.
- *Data Exploration:* Reshaping can assist in data exploration by organizing data in a way that makes patterns and relationships more apparent, simplifying the initial exploration phase.
- *Data Cleaning:* Data reshaping often goes hand-in-hand with data cleaning, as it may involve handling missing values, outliers, and other data quality issues.

Panda. map () function from series is used to substitute each value in series with another value.

```
# Check the data type of the 'Gender' column before mapping
print(wainaina['Gender'].dtype)
# Map 'Male' to 0 and 'Female' to 1 and cast to float
wainaina['Gender'] = wainaina['Gender'].map({'Male': 0, 'Female': 1}).astype(float)
# Display the updated DataFrame
print(wainaina)
```

```
object
         Name  Age  Gender  Marks
0       Peter   25     0.0   85.0
1       Joyce   30     1.0   92.0
2      George   22     0.0   84.8
3      Phylis   28     1.0   88.0
4       Moses   24     0.0   84.8
5  Priscillah   34     1.0   79.0
6       Eliud   19     0.0   80.0
```

Line(s)	Explanation
print(wainaina['Gender'].dtype)	Check and print the data type of the 'Gender' column in the DataFrame **wainaina**.
wainaina['Gender'] = wainaina['Gender'].map({'Male': 0, 'Female': 1}).astype(float)	Map 'Male' to 0 and 'Female' to 1 in the 'Gender' column, and then cast the column to a float data type.
print(wainaina)	Display the updated DataFrame **wainaina** after applying the mapping and casting operations.

ALTERNATIVELY Use replace function ().

```
# Replace 'Gender' values: 'Female' -> 1.0, 'Male' -> 0.0
wainaina['Gender'] = wainaina['Gender'].replace({'Female': 1.0, 'Male': 0.0})

# Display the updated DataFrame
print(wainaina)
```

```
         Name  Age  Gender  Marks
0       Peter   25     0.0   85.0
1       Joyce   30     1.0   92.0
2      George   22     0.0   84.8
3      Phylis   28     1.0   88.0
4       Moses   24     0.0   84.8
5  Priscillah   34     1.0   79.0
6       Eliud   19     0.0   80.0
```

Line(s)	Explanation
wainaina['Gender'] = wainaina['Gender'].replace({'Female': 1.0, 'Male': 0.0})	Replace 'Gender' values in the DataFrame **wainaina**: 'Female' is replaced with 1.0, and 'Male' is replaced with 0.0.
print(wainaina)	Display the updated DataFrame **wainaina** after replacing the 'Gender' values.

4. **Filtering data**, *suppose there is a requirement for the details regarding name, gender, marks of the top-scoring students. Here we need to remove some unwanted data.*

```
wainaina=wainaina[wainaina['Marks']>=80]
wainaina=wainaina.drop(['Age'],axis=1)
print(wainaina)
```

```
     Name  Gender  Marks
0   Peter    Male   85.0
1   Joyce  Female   92.0
2  George    Male   84.8
3  Phylis  Female   88.0
4   Moses    Male   84.8
6   Eliud    Male   80.0
```

Line(s)	Explanation
wainaina = wainaina[wainaina['Marks'] >= 80]	Filter the DataFrame **wainaina** to keep only rows where the 'Marks' column has a value greater than or equal to 80. Replace the original DataFrame with this filtered subset.
wainaina = wainaina.drop(['Age'], axis=1)	Remove the 'Age' column from the DataFrame **wainaina**. The **axis=1** argument indicates that we are dropping a column. The modified DataFrame no longer contains the 'Age' column.
print(wainaina)	Display the modified DataFrame **wainaina** after applying the filtering and column removal operations.

What does Axis 1 in pandas mean?

A data frame object has two axes. "axis 0" and "axis 1"

- "axis 0" represents rows and
- "axis 1"represents columns.

Hence, we have finally obtained an efficient dataset which can be further used for various purposes. Hence, we have finally obtained an efficient dataset which can be further used for various purposes.

Assuming you have a DataFrame named wainaina with columns 'Name', 'Gender', and 'Marks', and you want to retrieve details about the 3 top-scoring students:

```
# Sort the DataFrame by 'Marks' column in descending order
wainaina.sort_values(by='Marks', ascending=False, inplace=True)
# Select the top-scoring students (e.g., top 5)
top_scorers = wainaina[['Name', 'Gender', 'Marks']].head(3)
# Display the details of the top-scoring students
print(top_scorers)
```

```
     Name  Gender  Marks
1   Joyce  Female   92.0
3  Phylis  Female   88.0
0   Peter    Male   85.0
```

Line(s)	Explanation
wainaina.sort_values(by='Marks', ascending=False, inplace=True)	Sort the DataFrame **wainaina** by the 'Marks' column in descending order, so the top-scoring students appear at the top.
top_scorers = wainaina[['Name', 'Gender', 'Marks']].head(5)	Select the top 5 rows from the sorted DataFrame, including columns 'Name', 'Gender', and 'Marks', and store the result in a new DataFrame called **top_scorers**.
print(top_scorers)	Display the details of the top-scoring students, including their names, genders, and marks, to the console.

- Now that we know the basics of data wrangling. Below we will **discuss various operations using which we can perform data wrangling:**
  a) Merge operation
  b) Grouping Method

(a) Wrangling Data Using Merge Operation

Merge operation is used to merge raw data and into the desired format.

Syntax:
pd.merge(data_frame1,data_frame2, on="field ")

For example: Suppose that a Teacher has two types of Data, first type of Data consists of Details of Students and Second type of Data Consist of Pending Fees Status which is taken from Account Office. So The Teacher will use merge operation here in order to merge the data and provide it meaning. So that teacher will analyze it easily and it also reduces time and effort of Teacher from Manual Merging.

- ***First type of Data consists of Details of Students:***

```
import pandas as pd
STUDENTDETAILS = {
    'IDNO': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
    'NAME': ['Peter', 'Joyce', 'George', 'Phylis', 'Moses','Priscillah', 'Eliud','Veronicah','John', 'Juliet'],
    'Campus': ['Main', 'Ruiru', 'Nairobi', 'Main', 'Ruiru', 'Nairobi','Main', 'Ruiru', 'Nairobi','Main']
}
wainaina1 = pd.DataFrame(STUDENTDETAILS)
print(wainaina1)
```

```
   IDNO        NAME   Campus
0   101       Peter     Main
1   102       Joyce    Ruiru
2   103      George  Nairobi
3   104      Phylis     Main
4   105       Moses    Ruiru
5   106  Priscillah  Nairobi
6   107       Eliud     Main
7   108   Veronicah    Ruiru
8   109        John  Nairobi
9   110      Juliet     Main
```

Line(s)	Explanation
import pandas as pd	Import the Pandas library as **pd** to work with DataFrames.
STUDENTDETAILS = { ... }	Create a Python dictionary named **STUDENTDETAILS** containing student information such as 'IDNO', 'NAME', and 'Campus'.
wainaina1 = pd.DataFrame(STUDENTDETAILS)	Create a Pandas DataFrame named **wainaina1** by passing the **STUDENTDETAILS** dictionary to the **pd.DataFrame()** constructor. This DataFrame organizes the student information into a tabular format.
print(wainaina1)	Display the DataFrame **wainaina1** to the console, showing the tabular representation of the student details.

❖ *Second type of Data Consist of Pending Fees Status:*

```
import pandas as pd
FEESDETAILS = {
    'IDNO': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
    'PENDING': ['6000', '375', 'NIL', '7640', '3800','NIL', '1250','900','5200', 'NIL']
}
wainaina2 = pd.DataFrame(FEESDETAILS)
print(wainaina2)
```

```
   IDNO PENDING
0   101    6000
1   102     375
2   103     NIL
3   104    7640
4   105    3800
5   106     NIL
6   107    1250
7   108     900
8   109    5200
9   110     NIL
```

Line(s)	Explanation
import pandas as pd	Import the Pandas library as **pd** to work with DataFrames.
FEESDETAILS = { ... }	Create a Python dictionary named **FEESDETAILS** containing student fee information, including 'IDNO' and 'PENDING'.
wainaina2 = pd.DataFrame(FEESDETAILS)	Create a Pandas DataFrame named **wainaina2** by passing the **FEESDETAILS** dictionary to the **pd.DataFrame()** constructor. This DataFrame organizes the fee details into a tabular format.
print(wainaina2)	Display the DataFrame **wainaina2** to the console, showing the tabular representation of the fee details.

❖ *Wrangling Data given Using Merge Operation*

```
import pandas as pd
STUDENTDETAILS = {
    'IDNO': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
    'NAME': ['Peter', 'Joyce', 'George', 'Phylis', 'Moses','Priscillah', 'Eliud','Veronicah','John', 'Juliet'],
    'Campus': ['Main', 'Ruiru', 'Nairobi', 'Main', 'Ruiru', 'Nairobi','Main', 'Ruiru', 'Nairobi','Main']
}
wainaina1 = pd.DataFrame(STUDENTDETAILS)

FEESDETAILS = {
    'IDNO': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
    'PENDING': ['6000', '375', 'NIL', '7640', '3800','NIL', '1250','900','5200', 'NIL']
}
wainaina2 = pd.DataFrame(FEESDETAILS)

#merging Dataframe
print(pd.merge(wainaina1,wainaina2,on='IDNO'))
```

```
   IDNO       NAME   Campus PENDING
0   101      Peter     Main    6000
1   102      Joyce    Ruiru     375
2   103     George  Nairobi     NIL
3   104     Phylis     Main    7640
4   105      Moses    Ruiru    3800
5   106  Priscillah  Nairobi     NIL
6   107      Eliud     Main    1250
7   108   Veronicah    Ruiru     900
8   109       John  Nairobi    5200
9   110     Juliet     Main     NIL
```

Line(s)	Explanation
import pandas as pd	Import the Pandas library as **pd** to work with DataFrames.
STUDENTDETAILS = { ... }	Create a Python dictionary named **STUDENTDETAILS** containing student information such as 'IDNO', 'NAME', and 'Campus'.
wainaina1 = pd.DataFrame(STUDENTDETAILS)	Create a Pandas DataFrame named **wainaina1** by passing the **STUDENTDETAILS** dictionary to the **pd.DataFrame()** constructor. This DataFrame organizes the student details into a tabular format.
FEESDETAILS = { ... }	Create another Python dictionary named **FEESDETAILS** containing student fee information, including 'IDNO' and 'PENDING'.
wainaina2 = pd.DataFrame(FEESDETAILS)	Create a Pandas DataFrame named **wainaina2** by passing the **FEESDETAILS** dictionary to the **pd.DataFrame()** constructor. This DataFrame organizes the fee details into a tabular format.
pd.merge(wainaina1, wainaina2, on='IDNO')	Merge the two DataFrames **wainaina1** and **wainaina2** on the 'IDNO' column using the Pandas **pd.merge()** function. This combines the datasets based on the common 'IDNO'.
print(...)	Print the merged DataFrame to the console, displaying the tabular representation of the combined student and fee details.

(b) Wrangling Data using Grouping Method

The grouping method in Data analysis is used to provide results in terms of various groups taken out from Large Data. This method of pandas is used to group the outset of data from the large data set.

To demonstrate data wrangling using the grouping method, you can modify the code to group data by 'IDNO' and then perform some aggregation on the grouped data. The code below not only merges the student and fee details but also performs grouping and aggregation to calculate the total pending fees for each campus.

```
+ Code  + Text

[ ]  import pandas as pd
     # Create a DataFrame for student details
     STUDENTDETAILS = {
         'IDNO': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
         'NAME': ['Peter', 'Joyce', 'George', 'Phylis', 'Moses', 'Priscillah', 'Eliud', 'Veronicah', 'John', 'Juliet'],
         'Campus': ['Main', 'Ruiru', 'Nairobi', 'Main', 'Ruiru', 'Nairobi', 'Main', 'Ruiru', 'Nairobi', 'Main']
     }
     wainaina1 = pd.DataFrame(STUDENTDETAILS)
     # Create a DataFrame for student fees
     FEESDETAILS = {
         'IDNO': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
         'PENDING': [6000, 375, 0, 7640, 3800, 0, 1250, 900, 5200, 0]
     }
     wainaina2 = pd.DataFrame(FEESDETAILS)
     # Merge the two DataFrames on 'IDNO' to combine them
     merged_df = pd.merge(wainaina1, wainaina2, on='IDNO')
     # Group the merged DataFrame by 'Campus' and calculate the total pending fees for each campus
     campus_fee_totals = merged_df.groupby('Campus')['PENDING'].sum().reset_index()
     # Display the total pending fees for each campus
     print(campus_fee_totals)

       Campus  PENDING
0        Main    14890
1     Nairobi     5200
2       Ruiru     5075
```

Line(s)	Explanation
import pandas as pd	Import the Pandas library as **pd** to work with DataFrames.
STUDENTDETAILS = { ... }	Create a Python dictionary named **STUDENTDETAILS** containing student information such as 'IDNO', 'NAME', and 'Campus'.
wainaina1 = pd.DataFrame(STUDENTDETAILS)	Create a Pandas DataFrame named **wainaina1** by passing the **STUDENTDETAILS** dictionary to the **pd.DataFrame()** constructor. This DataFrame organizes the student details into a tabular format.
FEESDETAILS = { ... }	Create another Python dictionary named **FEESDETAILS** containing student fee information, including 'IDNO' and 'PENDING'.
wainaina2 = pd.DataFrame(FEESDETAILS)	Create a Pandas DataFrame named **wainaina2** by passing the **FEESDETAILS** dictionary to the **pd.DataFrame()** constructor. This DataFrame organizes the fee details into a tabular format.
pd.merge(wainaina1, wainaina2, on='IDNO')	Merge the two DataFrames **wainaina1** and **wainaina2** on the 'IDNO' column using the Pandas **pd.merge()** function. This combines the datasets based on the common 'IDNO'.
campus_fee_totals = merged_df.groupby ('Campus')['PENDING'].sum(). reset_index()	Group the merged DataFrame **merged_df** by the 'Campus' column and calculate the total pending fees for each campus using **.groupby()** and **.sum()**. The result is stored in the **campus_fee_totals** DataFrame.
print(campus_fee_totals)	Display the total pending fees for each campus using **print()**.

Example 2: There is a Car Selling company and this company have different Brands of various Car Manufacturing Company like Maruti, Toyota, Mahindra, Ford, etc. and have data where different cars are sold in different years. So the Company wants to wrangle only that data where cars are sold during the year 2010. For this problem, we use another Wrangling technique that is groupby() method.

Line(s)	Explanation
import pandas as pd	Import the Pandas library as **pd** to work with DataFrames.
cardata = { ... }	Create a Python dictionary named **cardata** containing car sales data with columns 'Brand', 'Year', 'Model', and 'Sales'.
wainaina= pd.DataFrame(cardata)	Create a Pandas DataFrame **wainaina** from the **cardata** dictionary to organize the car sales data into a tabular format.
print(wainaina)	Display the **wainaina** DataFrame, which contains the original car sales data.
grouped=wainaina.groupby('Year')	Group the **wainaina** DataFrame by the 'Year' column using the **groupby()** method. This creates a groupby object **grouped**.
print("\n GROUPED DATA \n")	Print a separator line to indicate the start of the grouped data section.
print(grouped.get_group(2010))	Retrieve and display the group of data corresponding to the year 2010 from the **grouped** object using **.get_group()**.

```
      Brand  Year    Model  Sales
0    Maruti  2009    Swift    120
1    Toyota  2010  Corolla    150
2  Mahindra  2011  Scorpio     95
3      Ford  2010   Fiesta     50
4    Maruti  2010     Alto    105
5    Toyota  2009    Camry    130
6    Toyota  2010   Innova     90
7      Ford  2012     Figo     75
8  Mahindra  2010   Bolero     85
9    Maruti  2009   WagonR    115

 GROUPED DATA

      Brand  Year    Model  Sales
1    Toyota  2010  Corolla    150
3      Ford  2010   Fiesta     50
4    Maruti  2010     Alto    105
6    Toyota  2010   Innova     90
8  Mahindra  2010   Bolero     85
```

Using Several years in our Condition:

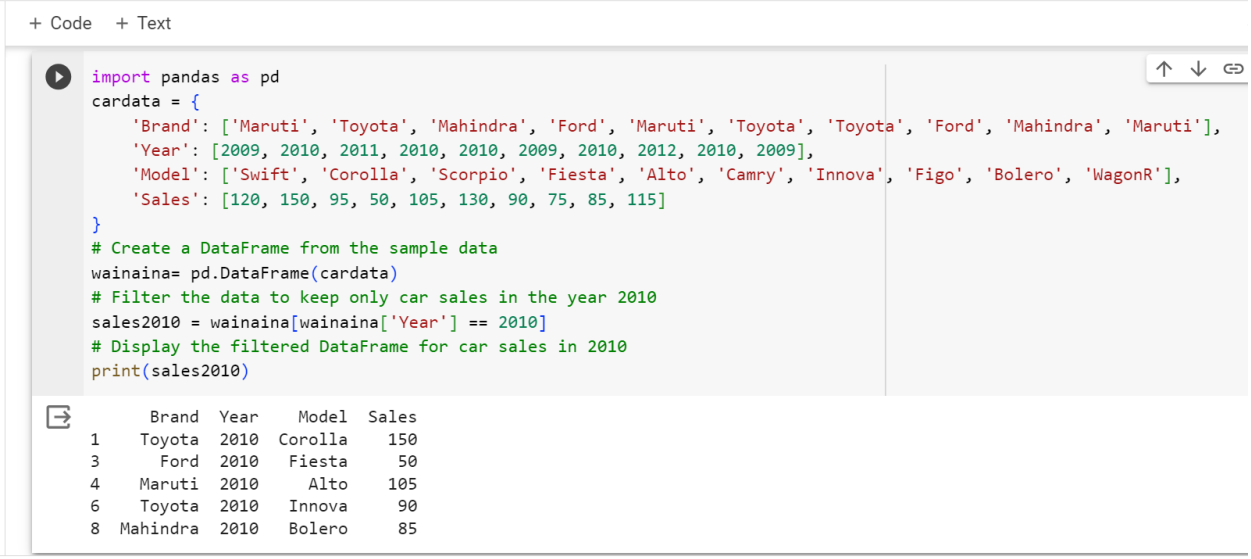
```
import pandas as pd
cardata = {
    'Brand': ['Maruti', 'Toyota', 'Mahindra', 'Ford', 'Maruti', 'Toyota', 'Toyota', 'Ford', 'Mahindra', 'Maruti'],
    'Year': [2009, 2010, 2011, 2010, 2010, 2009, 2010, 2012, 2010, 2009],
    'Model': ['Swift', 'Corolla', 'Scorpio', 'Fiesta', 'Alto', 'Camry', 'Innova', 'Figo', 'Bolero', 'WagonR'],
    'Sales': [120, 150, 95, 50, 105, 130, 90, 75, 85, 115]
}
# Create a DataFrame from the sample data
wainaina= pd.DataFrame(cardata)
print(wainaina)
print("\n GROUPED DATA \n")
grouped=wainaina.groupby('Year')
mychoice = [2011, 2012]
# Display the filtered DataFrame for car sales in a given year
for year in mychoice:
    print(f"\nData for Year {year}:\n")
    print(grouped.get_group(year))
```

Line(s)	Explanation
import pandas as pd	Import the Pandas library as **pd** to work with DataFrames.
cardata = { ... }	Create a Python dictionary named **cardata** containing car sales data with columns 'Brand', 'Year', 'Model', and 'Sales'.
wainaina= pd.DataFrame(cardata)	Create a Pandas DataFrame **wainaina** from the **cardata** dictionary to organize the car sales data into a tabular format.
print(wainaina)	Display the **wainaina** DataFrame, which contains the original car sales data.
print("\n GROUPED DATA \n")	Print a separator line to indicate the start of the grouped data section.
grouped=wainaina.groupby('Year')	Group the **wainaina** DataFrame by the 'Year' column using the **groupby()** method. This creates a groupby object **grouped**.
mychoice = [2011, 2012]	Define a list called **mychoice** that contains the years you want to retrieve data for (2011 and 2012).
for year in mychoice:	Start a **for** loop to iterate over the years in the **mychoice** list.
print(f"\nData for Year {year}:\n")	Print a message indicating the year for which the data is being displayed.
print(grouped.get_group(year))	Retrieve and display the group of data corresponding to the current year from the **grouped** object using **.get_group()**. This shows car sales data for the specified years.

```
      Brand  Year     Model  Sales
0    Maruti  2009     Swift    120
1    Toyota  2010   Corolla    150
2  Mahindra  2011   Scorpio     95
3      Ford  2010    Fiesta     50
4    Maruti  2010      Alto    105
5    Toyota  2009     Camry    130
6    Toyota  2010    Innova     90
7      Ford  2012      Figo     75
8  Mahindra  2010    Bolero     85
9    Maruti  2009    WagonR    115
```

```
 GROUPED DATA


Data for Year 2011:

        Brand   Year     Model   Sales
2   Mahindra   2011   Scorpio      95

Data for Year 2012:

   Brand   Year Model   Sales
7   Ford   2012   Figo      75
```

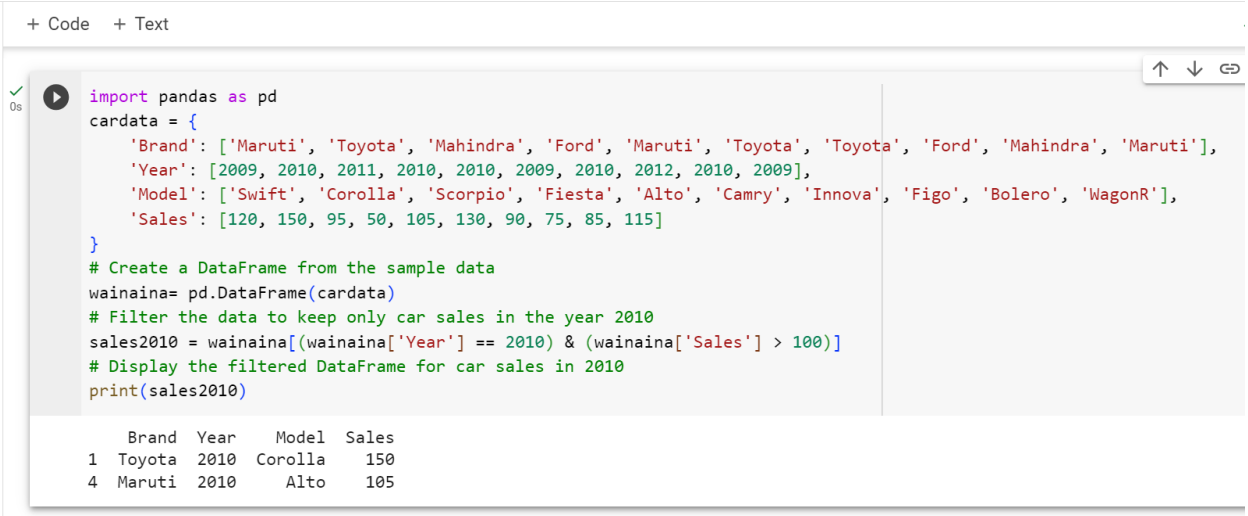
Alternatively, you can use relational operator:



```
+ Code  + Text

import pandas as pd
cardata = {
    'Brand': ['Maruti', 'Toyota', 'Mahindra', 'Ford', 'Maruti', 'Toyota', 'Toyota', 'Ford', 'Mahindra', 'Maruti'],
    'Year': [2009, 2010, 2011, 2010, 2010, 2009, 2010, 2012, 2010, 2009],
    'Model': ['Swift', 'Corolla', 'Scorpio', 'Fiesta', 'Alto', 'Camry', 'Innova', 'Figo', 'Bolero', 'WagonR'],
    'Sales': [120, 150, 95, 50, 105, 130, 90, 75, 85, 115]
}
# Create a DataFrame from the sample data
wainaina= pd.DataFrame(cardata)
# Filter the data to keep only car sales in the year 2010
sales2010 = wainaina[wainaina['Year'] == 2010]
# Display the filtered DataFrame for car sales in 2010
print(sales2010)

      Brand  Year    Model  Sales
1    Toyota  2010  Corolla    150
3      Ford  2010   Fiesta     50
4    Maruti  2010     Alto    105
6    Toyota  2010   Innova     90
8  Mahindra  2010   Bolero     85
```

Line(s)	Explanation
import pandas as pd	Import the Pandas library as **pd** to work with DataFrames.
car_sales_data = { ... }	Create a Python dictionary named **car_sales_data** containing car sales data with columns 'Brand', 'Year', 'Model', and 'Sales'.
car_sales_df = pd.DataFrame(car_sales_data)	Create a Pandas DataFrame **car_sales_df** from the **car_sales_data** dictionary to organize the car sales data into a tabular format.
car_sales_2010 = car_sales_df[car_sales_df['Year'] == 2010]	Filter the **car_sales_df** DataFrame to keep only rows where the 'Year' column is equal to 2010. The result is stored in the **car_sales_2010** DataFrame.
print(car_sales_2010)	Display the **car_sales_2010** DataFrame, which contains car sales data for the year 2010.

Using logical Operator & (AND):



```
import pandas as pd
cardata = {
    'Brand': ['Maruti', 'Toyota', 'Mahindra', 'Ford', 'Maruti', 'Toyota', 'Toyota', 'Ford', 'Mahindra', 'Maruti'],
    'Year': [2009, 2010, 2011, 2010, 2010, 2009, 2010, 2012, 2010, 2009],
    'Model': ['Swift', 'Corolla', 'Scorpio', 'Fiesta', 'Alto', 'Camry', 'Innova', 'Figo', 'Bolero', 'WagonR'],
    'Sales': [120, 150, 95, 50, 105, 130, 90, 75, 85, 115]
}
# Create a DataFrame from the sample data
wainaina= pd.DataFrame(cardata)
# Filter the data to keep only car sales in the year 2010
sales2010 = wainaina[(wainaina['Year'] == 2010) & (wainaina['Sales'] > 100)]
# Display the filtered DataFrame for car sales in 2010
print(sales2010)
```

```
    Brand  Year    Model  Sales
1  Toyota  2010  Corolla    150
4  Maruti  2010     Alto    105
```

Line(s)	Explanation
import pandas as pd	Import the Pandas library as **pd** to work with DataFrames.
cardata = { ... }	Create a Python dictionary named **cardata** containing car sales data with columns 'Brand', 'Year', 'Model', and 'Sales'.
wainaina= pd.DataFrame(cardata)	Create a Pandas DataFrame **wainaina** from the **cardata** dictionary to organize the car sales data into a tabular format.
sales2010 = wainaina[(wainaina['Year'] == 2010) & (wainaina['Sales'] > 100)]	Filter the **wainaina** DataFrame using logical operators. In this case, it selects rows where 'Year' is equal to 2010 **and** 'Sales' is greater than 100. The result is stored in the **sales2010** DataFrame.
print(sales2010)	Display the filtered DataFrame **sales2010**, which contains car sales data for the year 2010 with Sales greater than 100.

Using logical Operator | (OR):

```
import pandas as pd
# Sample data for car sales
cardata = {
    'Brand': ['Maruti', 'Toyota', 'Mahindra', 'Ford', 'Maruti', 'Toyota', 'Toyota', 'Ford', 'Mahindra', 'Maruti'],
    'Year': [2009, 2010, 2011, 2010, 2010, 2009, 2010, 2012, 2010, 2009],
    'Model': ['Swift', 'Corolla', 'Scorpio', 'Fiesta', 'Alto', 'Camry', 'Innova', 'Figo', 'Bolero', 'WagonR'],
    'Sales': [120, 150, 95, 50, 105, 130, 90, 75, 85, 115]
}
# Create a DataFrame from the sample data
wainaina = pd.DataFrame(cardata)
# Filter data for the year 2010 or 2011 using logical OR operator |
filtered_data = wainaina[(wainaina['Year'] == 2010) | (wainaina['Year'] == 2011)]
# Display the filtered DataFrame
print(filtered_data)
```

```
      Brand  Year    Model  Sales
1    Toyota  2010  Corolla    150
2  Mahindra  2011  Scorpio     95
3      Ford  2010   Fiesta     50
4    Maruti  2010     Alto    105
6    Toyota  2010   Innova     90
8  Mahindra  2010   Bolero     85
```

The line filtered_data = wainaina[(wainaina['Year'] == 2010) | (wainaina['Year'] == 2011)] filters the wainaina DataFrame to include only rows where the 'Year' is either 2010 or 2011 using a logical OR operation.