**Python Pandas:-**

Pandas is a **Python** library used for working with data sets. It has functions for

1. analysing,
2. cleaning,
3. exploring, and
4. manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by **Wes McKinney** in 2008.

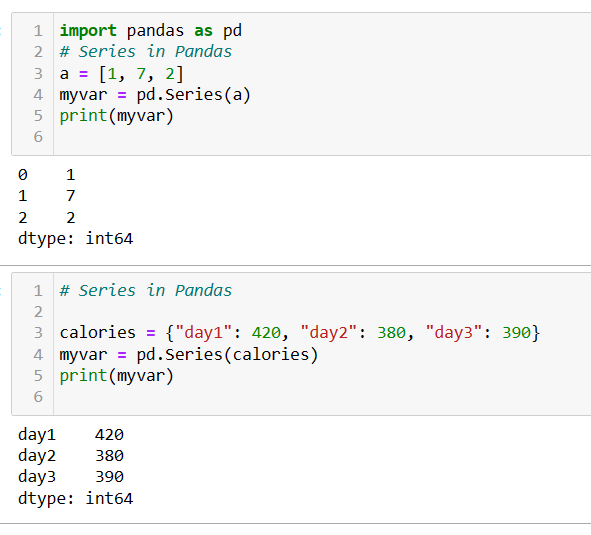
Pandas allow us to analyse big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

Usually we work on two type of datasets

1. **Series:**
2. **Dataframe:**

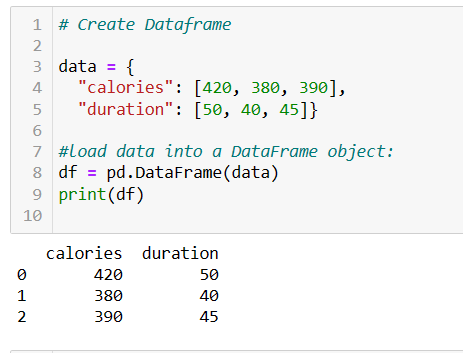
**Series:**

A pandas Series is a **one-dimensional** labelled array capable of holding any data type (e.g., integers, floats, strings, etc.). It is similar to a Python list or NumPy array but provides additional functionalities and features. Each element in a Series is associated with an index, which labels the elements and allows for efficient data retrieval and manipulation.



**DataFrame:**

In pandas, a DataFrame is a **two-dimensional** labeled data structure, similar to a table or **spreadsheet**. It consists of **rows and columns**, where each column can contain data of different types (e.g., integers, floats, strings, etc.). DataFrames are incredibly versatile and are widely used for data manipulation, analysis, and exploration in Python.



Different Operation of DataFrame: -

1. **Reading Dataframe**
2. **Creating Dataframe**
3. Filtering Dataframes:
4. Handling the missing Values

* fillna
* dropna

1. Dropping Row or Columns
2. Working on Index
3. *iterate over data frame*
4. Group by
5. Joining Dataframe

* Concatenation
* Merge
* Join

1. Some stats Function (measure of central tendency)

* Mean
* Median
* Mode

1. Reshaping
2. Basic pandas Functions/Attributes

**Reading different file**

**Data Input**:

1. read\_csv() : Read a CSV file into a DataFrame.
2. read\_excel() : Read an Excel file into a DataFrame.
3. read\_json() : Read a JSON file into a DataFrame.
4. read\_sql() : Read from a SQL database into a DataFrame.

**Save file**

1. to\_csv() : Write a DataFrame to a CSV file.
2. to\_excel() : Write a DataFrame to an Excel file.
3. to\_json() : Write a DataFrame to a JSON file.
4. to\_sql() : Write records stored in a DataFrame to a SQL database.

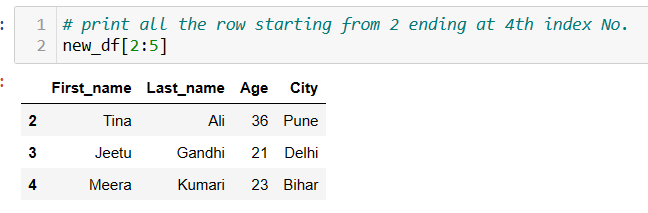
**Filtering Dataframes:**

Below is the list of ways you can filter the datasets

1. Indexing and slicing
2. .Ioc
3. .iloc
4. Conditional filtering

**indexing or slicing:**

Bleow will print all the rows starting from 2 and ending at 4th index No.



**loc:**

It is primarily label-based and is used for selecting rows and columns by labels or a boolean array.

df.loc[**rowNumber**,[**ColumnName**]]

**df.loc[2:4,[“Firstname”,”LastName”]]**

**at:**

It is label-based and is used for accessing a **single value** in a DataFrame by row and column labels.

dff.at[**rowNumber**,**columnName**]

**df.at[2,”FirstName”]**

**iloc:**

It is primarily integer position-based and is used for selecting rows and columns by integer position or a boolean array.

df.iloc[**rowNumber**,[**columnNumber**]]

**df.iloc[2:4,[1,3]**

**iat:**

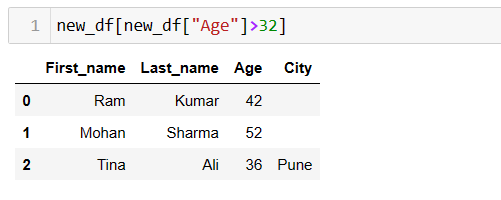
It is integer position-based and is used for accessing a single value in a DataFrame by row and column integer positions.

df.iat[**rowNumber**,**columnNumber**]

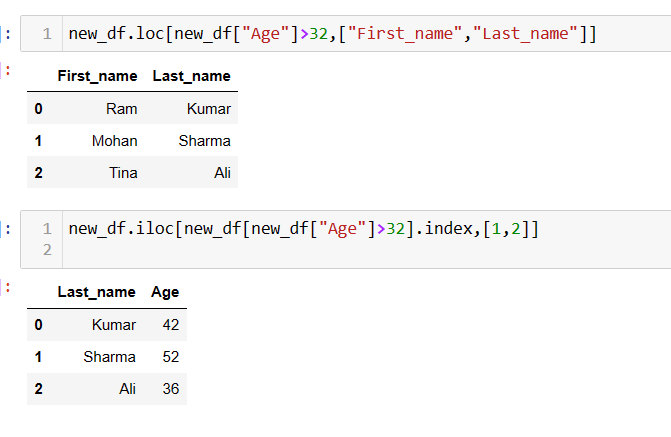
**df.iat[2,3]**

**Filtering using condition**

1. Below will print temp which is greater than 32



1. This will print only the **“Age”** and **“Last\_name”** columns where the temperature is greater than 32.

****

**Use of fillna and dropna**

**df.fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None)**

**df.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)**

|  |
| --- |
| This will replace NA values with 0. |
| new\_df = df.fillna(0) |
| Copy paste earlier cell value to the empty cell |
| new\_df = df.fillna(method=“ffill”) |
| Copy paste below cell value to the empty cell |
| new\_df = df.fillna(method=“bfill”) |
| Copy paste next column cell value to the empty cell |
| new\_df = df.fillna(method=“bfill”,axis= “column”) |
| Copy and paste the earlier cell value to the next empty cell, however, this will only happen for one empty cell. |
| new\_df = df.fillna(method=“ffill”,limit=1) |
| replace empty cells with a specific value: |
| new\_df = df.fillna(130, inplace = True) |
| Replace NULL values in the "Calories" columns with the number 130: |
| new\_df = df["Calories"].fillna(130,inplace = True) |
| Remove all rows containing NA |
| new\_df = df.dropna() |
| This will keep those rows that have some or the other data and delete only those data that are competently empty. |
| new\_df = df.dropna(how=“all”) |
| This will delete those rows which have two empty cells |
| new\_df = df.dropna(thresh=2) |
| remove all rows containing NULL |
| df.dropna(inplace = True) |
| This will return True or False if the data frame contains duplicate |
| print(df.duplicated()) |
| To remove the duplicate line |
| df.drop\_duplicates(inplace = True) |

**Setting Index**

This will set day column as index of the data-Frame

df.set\_index(‘day’,inpace=True)

This will reset original index of the Data-Frame

df.reset\_index(inplace=True)

*Reseting index*

new\_df.reset\_index(drop=True,inplace=True)

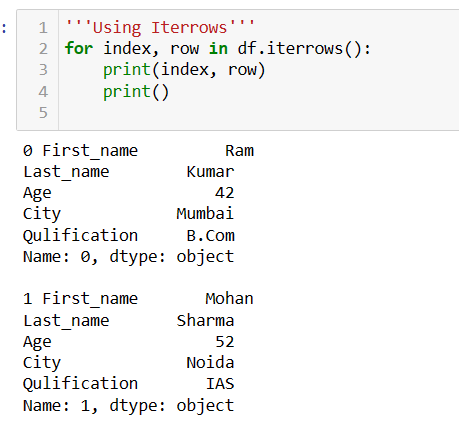
**Basic of pandas**

|  |  |
| --- | --- |
| *Print First 5 Lines* | df.head() |
| *Print Last 5 Lines* | df.tail() |
| *Info about data* | df.info() |
| *How to print all the name of the column.* | df.columns |
| *How to print data type of column* | Type(df[‘Name’]) |
| *Print max value of the column* | df.[‘temperature’].max() |
| *Print min value of the column* | df.[‘temperature’].min() |
| *Print min value of the standard deviation* | df.[‘temperature’].std() |
| *This will print statistics* | df.describe |

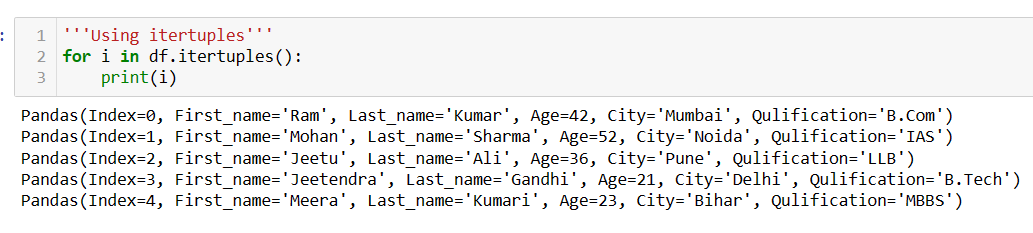
***Different ways to iterate over data frame:***

* ***.Iterows()***
* ***.Itertupples()***
* ***.Items()***
* ***Using index***

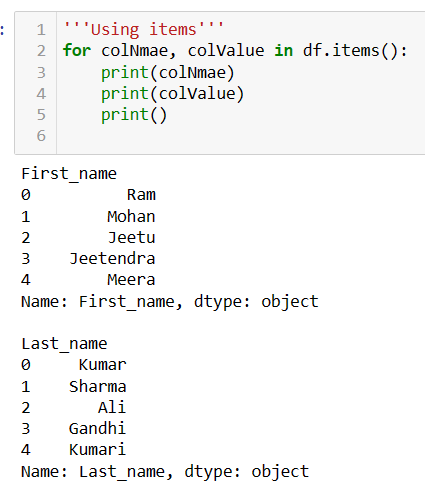
***.Iterows()***

******

***.Itertupples()***

******

***.Items()***

****

**Group by**

**df.groupby(by=None, axis=0, level=None, as\_index=True, sort=True, group\_keys=True, squeeze=<object object>, observed=False, dropna=True)**

**new\_df = df.groupby(‘city’)**

This will group the data as per city.

**for city in city\_df in new\_df:**

**print(city)**

**print(city\_df)**

This will print the data with group.

**new\_df.get\_group(‘mumbai’)**

**Concatenate: -**

**pd.concat(objs, axis=0, join='outer', ignore\_index=False, keys=None, levels=None, names=None, verify\_integrity=False, sort=False, copy=True)**

- Paste data one below the other(appending the data)

**new\_df = pd.concat(df1,df2,ignore\_index=True)**

Paste data next to each other

**new\_df = pd.concat(df1,df2,ignore\_index=True,axis=1)**

**Merge**

**pd.merge(right, how='inner', on=None, left\_on=None, right\_on=None, left\_index=False, right\_index=False, sort=False, suffixes=('\_x', '\_y'), copy=True, indicator=False, validate=None)**

Create Data-frame with inner join **df3 = pd.merge(pd1,df2,on=‘City’)**

Create Data-frame with outer join

**df3 = pd.merge(pd1,df2,on=‘City’,how=“outer”)**

Create Data-frame with left join

**df3 = pd.merge(pd1,df2,on=‘City’,how=“left”)**

**Pivot**

**pd.pivot(data, index=None, columns=None, values=None)**

**pd.pivot\_table(data, values=None, index=None, columns=None, aggfunc='mean', fill\_value=None, margins=False, dropna=True, margins\_name='All', observed=False)**

|  |  |
| --- | --- |
| This will create pivit table with city as row and date in column | **new\_df =pd.pivot\_table(index= “city”, column= “date”, aggfunc= “sum”)** |

**Reshaping the data:  
df.melt(id\_vars=None, value\_vars=None, var\_name=None, value\_name='value', col\_level=None, ignore\_index=True)**

|  |  |
| --- | --- |
| This wll update the column data one bleow the other | **new\_df = pd.melt(df,id\_vars= [“day”])** |

**df.stack(level=-1, dropna=True)**

|  |  |
| --- | --- |
| This will create stacked data | **new\_df = df.stack(level=0)** |

Pandas uses the **mean() median()** and **mode()** methods to calculate the respective values for a specified column:

|  |  |
| --- | --- |
|  |  |
| **mean()** | x = df["Calories"].mean() df["Calories"].fillna(x, inplace = True) |
| **median()** | x = df["Calories"].median() df["Calories"].fillna(x, inplace = True) |
| **mode()** | x = df["Calories"].mode()[0] df["Calories"].fillna(x, inplace = True) |

**Convert Data Frame into a Correct Format:**

|  |  |
| --- | --- |
| Convert all cells in the 'Date' column into dates. | import pandas as pd df = pd.read\_csv('data.csv') df['Date'] = pd.to\_datetime(df['Date']) print(df.to\_string()) |

**Data Exploration**:

1. head(), tail(): View the first/last n rows of a DataFrame.
2. info(): Print a concise summary of a DataFrame.
3. describe(): Generate descriptive statistics of a DataFrame.
4. value\_counts(): Count unique values in a Series.
5. unique(), nunique(): Get unique values/count of unique values in a Series.
6. isnull(), notnull(): Check for missing values.
7. corr(), cov(): Compute pairwise correlation/covariance of columns.

**Data Manipulation**:

1. groupby(): Group DataFrame using a mapper or by a Series of columns.
2. pivot\_table(): Create a spreadsheet-style pivot table.
3. stack(), unstack(): Pivot a level of the (possibly hierarchical) index labels.
4. melt(): Unpivot DataFrame from wide format to long format.
5. drop(): Drop specified labels from rows or columns.
6. fillna(): Fill missing values.
7. dropna(): Drop missing values.
8. rename(): Alter axes labels.
9. map(), apply(), applymap(): Apply functions to elements of DataFrame or Series.
10. replace(): Replace values in a DataFrame or Series.

astype(): Cast a pandas object to a specified dtype.

**Sorting:**

1. sort\_values(), sort\_index(): Sort DataFrame by values or index.

**Time Series**:

1. date\_range(), period\_range(): Generate date and time ranges.
2. to\_datetime(): Convert arguments to datetime.
3. resample(): Convenience method for frequency conversion and resampling.

These are just some of the many functions provided by pandas for data manipulation and analysis.

**String Handling**:

**Accessors**:

str: Accessor for vectorized string operations. Used to access string methods on Pandas Series.

**Counting and Length**:

str.count(): Count occurrences of a pattern/substring in each string.

str.len(): Compute the length of each string.

**Substitution and Replacement**:

str.replace(): Replace occurrences of a pattern/substring with another string.

**String Searching and Matching**:

str.contains(): Check if pattern or regex is contained within a string.

str.startswith(), str.endswith(): Check if string starts/ends with a specified substring.

**Splitting and Joining**:

str.split(), str.rsplit(): Split strings into substrings by a delimiter.

str.join(): Join lists contained as elements in the Series with a separator.

str.cat(): Concatenate strings.

**String Slicing and Indexing**:

str.slice(), str.slice\_replace(): Slice substrings from each element in the Series/Index.

str.get(): Get the ith element of each string.

str.slice\_replace(): Replace the ith element of each string with another string.

**String Formatting**:

str.format(): Format strings.

**Case Conversion**:

str.lower(), str.upper(), str.title(): Convert case of strings.

**Stripping and Padding**:

str.strip(), str.lstrip(), str.rstrip(): Remove leading/trailing whitespace.

str.pad(): Pad strings.

**Alliging Text**:

str.center(), str.ljust(), str.rjust(): Center, left-align, or right-align strings.

**Use of Regex**:

str.extract(), str.extractall(): Extract groups from the first match of a regular expression pattern or from all matches.

str.match(): Determine if each string matches a regular expression.

str.findall(): Find all occurrences of pattern or regular expression in the Series/Index.