# Pandas in Python

# Pandas

- **Pandas** is Python package for **data analysis**.
- **Pa**nal **Da**ta **S**ystem and **P**ython **an**d **da**ta analy**s**is
- Pandas is a high-level data manipulation tool developed by Wes McKinney.
- It Provides built-in data structures which simplify the manipulation and analysis of data sets.
- Pandas is easy to use and powerful, but “with great power comes great responsibility”
- it is built on top of Numpy.
- Pandas is a software library written for the Python programming language.
- http://pandas.pydata.org/pandas-docs/stable/

# Pandas

- It provides special data structures and operations for the manipulation of numerical tables and time series.
- Pandas deals with the following three data structures
  - Series
  - DataFrame
  - Panel
- Series
  - 1D labeled homogeneous array, size immutable.
- DataFrame
  - General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
- Panel
  - General 3D labeled, size-mutable array.

# Pandas

- Pandas Series
  - Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.).
  - The axis labels are collectively called index.
  - It can be seen as a data structure with two arrays: one functioning as the index, i.e. the labels, and the other one contains the actual data.

# Pandas

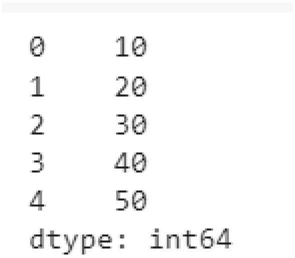
- Pandas Series
  - Creation of series using a constructor:
    - **pandas.Series( data, index, dtype, copy)**
  - data → data takes various forms like ndarray, list, constants
  - Index → Index values must be unique and hashable, same length as data. Default np.arrange(n) if no index is passed.
  - dtype → dtype is for data type. If None, data type will be inferred
  - copy →Copy data. Default False

# Pandas

- A series can be created using various inputs like:
  - Array
  - Dict
  - Scalar value or constant

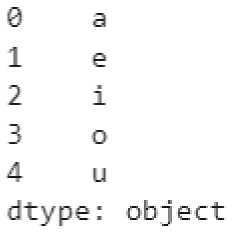
# Pandas - Series

```
import numpy as np
import pandas as pd
"""### Creating a series from an array"""

data=np.array([10,20,30,40,50])
ser=pd.Series(data)
print(ser)
```



```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

```
""" ### Creating a series from a list"""

l=['a','e','i','o','u']
ser=pd.Series(l)
print(ser)
```



```
0    a
1    e
2    i
3    o
4    u
dtype: object
```

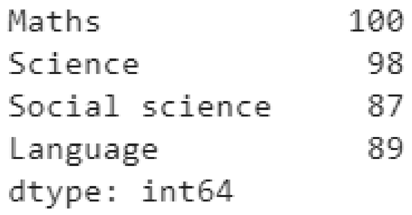
# Pandas - Series

**#To set index for the series**

subj=["Maths","Science","Social science'

marks=[100,98,87,89]

pd.Series(marks, index=subj)

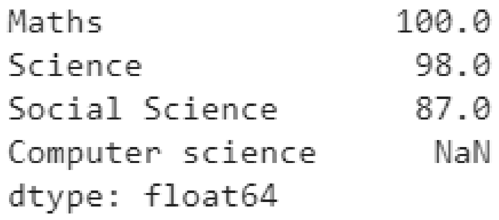


```
Maths             100
Science            98
Social science     87
Language           89
dtype: int64
```

**#Create a series from dictionary**

sub_mark={"Maths":100,"Science":98,"Social Science":87,"Language":89}

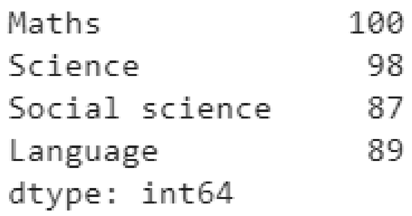
pd.Series(sub_mark)



```
Maths               100.0
Science              98.0
Social Science       87.0
Computer science      NaN
dtype: float64
```

# Pandas - Series

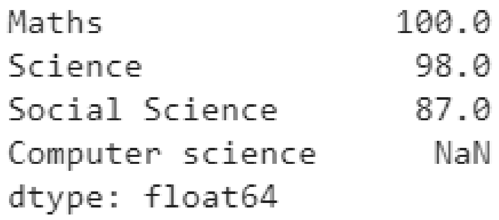
```
#To set index for the series
subj=["Maths","Science","Social science","Language"]
marks=[100,98,87,89]
pd.Series(marks, index=subj)
```



```
Maths             100
Science            98
Social science     87
Language           89
dtype: int64
```

```
#Create a series from dictionary
sub_mark={"Maths":100,"Science":98,"Social Science":87,"Language":89}
pd.Series(sub_mark)
```

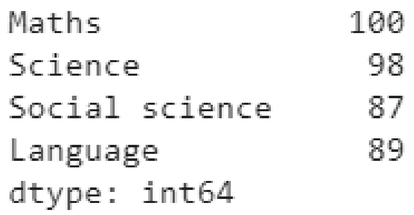
```
#Series with missing values
subj=["Maths","Science","Social Science","Computer science"]
mark_series=pd.Series(sub_mark,index=subj)
mark_series
```



```
Maths               100.0
Science              98.0
Social Science       87.0
Computer science      NaN
dtype: float64
```

```
#Manipulating Series
# to check null values
mark_series.isnull()

mark_series.notnull()
```

# Pandas - Series



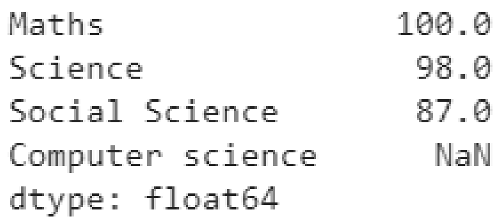
```
Maths             100
Science            98
Social science     87
Language           89
dtype: int64
```

**#Create a series from dictionary**

```
sub_mark={"Maths":100,"Science":98,"Social Science":87,"Language":89}
pd.Series(sub_mark)
```

**#Series with missing values**

```
subj=["Maths","Science","Social Science","Computer science"]
mark_series=pd.Series(sub_mark,index=subj)
mark_series
```



```
Maths               100.0
Science              98.0
Social Science       87.0
Computer science      NaN
dtype: float64
```

# Pandas - Series

```
mark_series>90

#extracting subjects above 90
mark_series[mark_series>90]

# to sort values
#mark_series.sort_values()
mark_series.sort_values(ascending=False)

#ranking values
#mark_series.rank()
mark_series.rank(ascending=False)
```

# Pandas - Series

```
#basic statistics
mark_series.sum()
mark_series.mean()
mark_series.median()
mark_series.std()
mark_series.max()
mark_series.idxmax()
mark_series.min()
mark_series.idxmin()
mark_series.count()

#summary statistics
mark_series.describe()
```

# Pandas - Series

**#to get unique values in a series**

```
mark_series.unique()
```

**#to get count of unique values**

```
mark_series.nunique()
```

**#to get the count of each unique value**

```
mark_series.value_counts()
```

```
mark_series.dropna()
#mark_series.dropna(inplace=True)
```

```
mark_series
```

# Pandas - Series

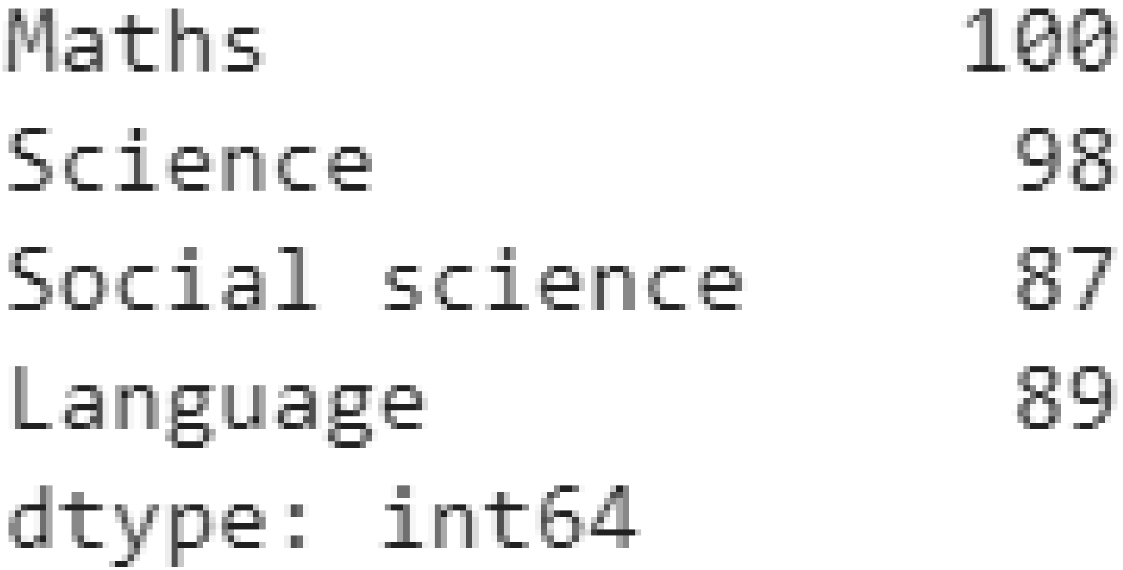
**Accessing elements of a series**

**There are two ways through which we can access element of series, they are :**

**- Accessing Element from Series with Position**
**- Accessing Element Using Label (index)**

```
#accessing element from series with position
mark_series[:2]
```

```
#accessing element using Label
data=np.array(['a','e','i','o','u'])
ser=pd.Series(data,index=[5,10,15,20,25])
ser
```



```
Maths             100
Science            98
Social science     87
Language           89
dtype: int64
```

# Pandas - Series

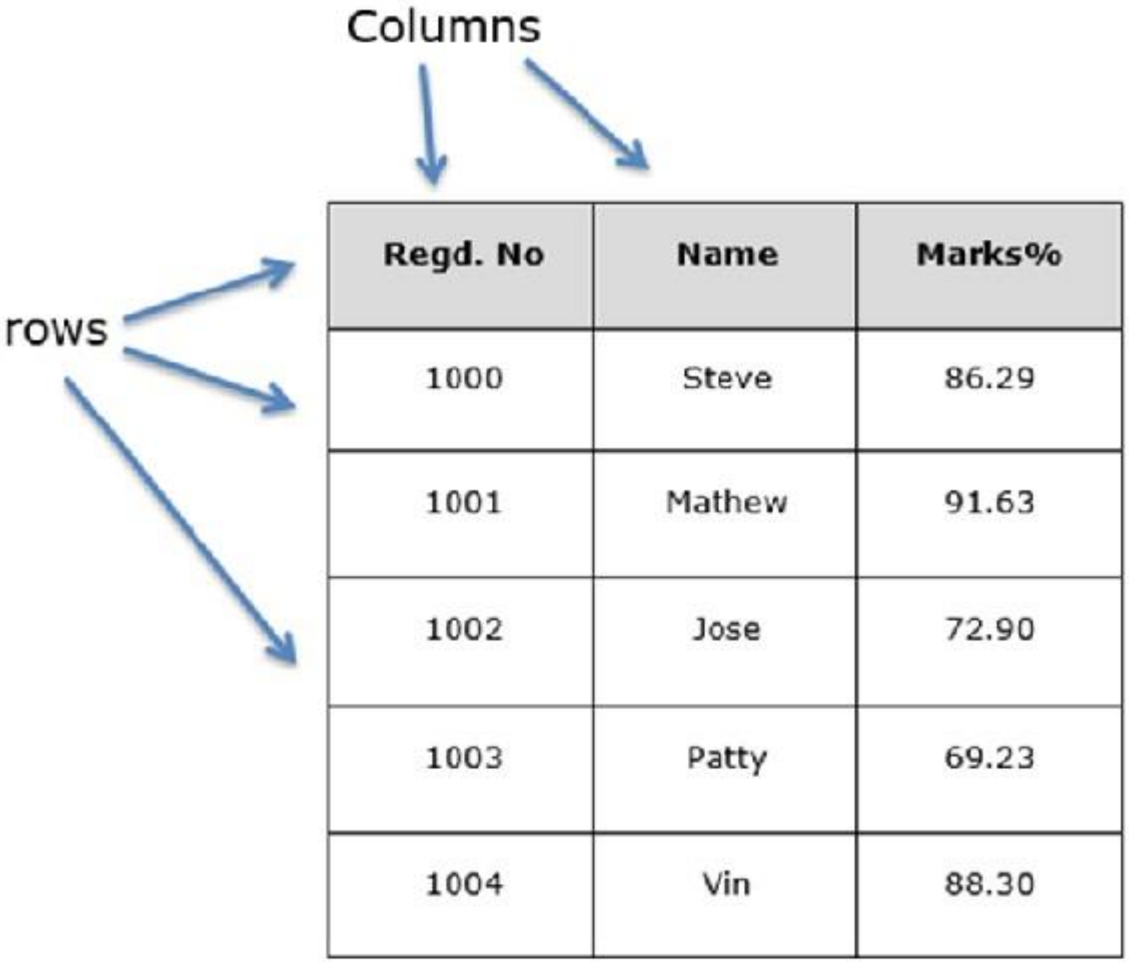
```
#ser[2]
ser[15]

#using iloc
ser.iloc[2]

#using loc
ser.loc[15]
mark_series

mark_series.iloc[1]

mark_series.loc["Science"]

mark_series

mark_series.iloc[:2]

mark_series.loc[:"Social Science"]
```

# Pandas Data frame

- A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.
  - Potentially columns are of different types
  - Size – Mutable
  - Labeled axes (rows and columns)
  - Can Perform Arithmetic operations on rows and columns

# Pandas Data frame

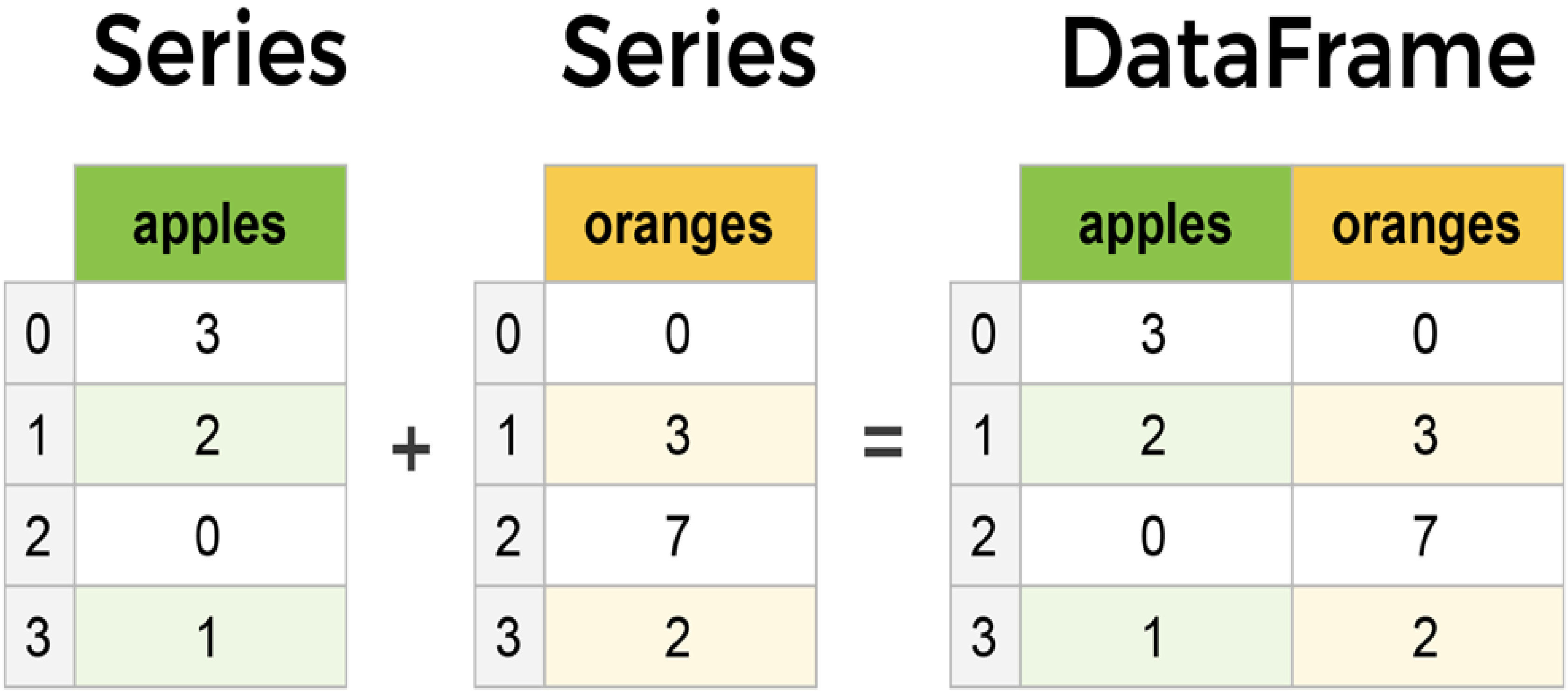
- Structure


Columns
rows

| Regd. No | Name | Marks% |
|---|---|---|
| 1000 | Steve | 86.29 |
| 1001 | Mathew | 91.63 |
| 1002 | Jose | 72.90 |
| 1003 | Patty | 69.23 |
| 1004 | Vin | 88.30 |

# Pandas Data frame

- Structure



Series

|   | apples |
|---|---|
| 0 | 3 |
| 1 | 2 |
| 2 | 0 |
| 3 | 1 |

+

Series

|   | oranges |
|---|---|
| 0 | 0 |
| 1 | 3 |
| 2 | 7 |
| 3 | 2 |

=

DataFrame

|   | apples | oranges |
|---|---|---|
| 0 | 3 | 0 |
| 1 | 2 | 3 |
| 2 | 0 | 7 |
| 3 | 1 | 2 |

# Pandas Data frame

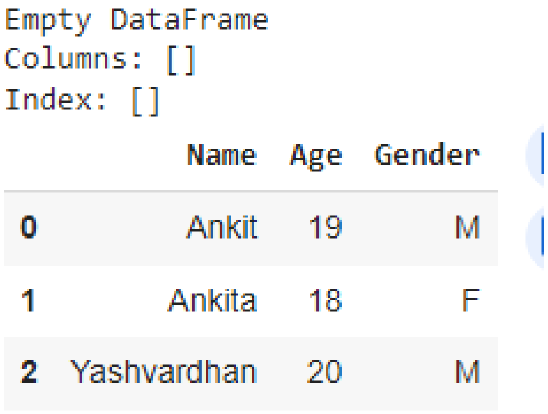
- A pandas DataFrame can be created using a constructor
  - **pandas.DataFrame( data, index, columns, dtype, copy)**
  - Data → data takes various forms like ndarray, series, lists, dict, constants and also another DataFrame.
  - Index → For the row labels, the Index to be used for the resulting frame is Optional Default np.arange(n) if no index is passed.
  - Columns → For column labels, the optional default syntax is - np.arange(n). This is only true if no index is passed.
  - Dtype → Data type of each column.
  - Copy → This command (or whatever it is) is used for copying of data, if the default is False.

# Pandas Data frame

- Creation of DataFrame:
  - Lists
  - Dict
  - Series
  - Numpy ndarrays
  - Another dataframe

# Pandas Data frame

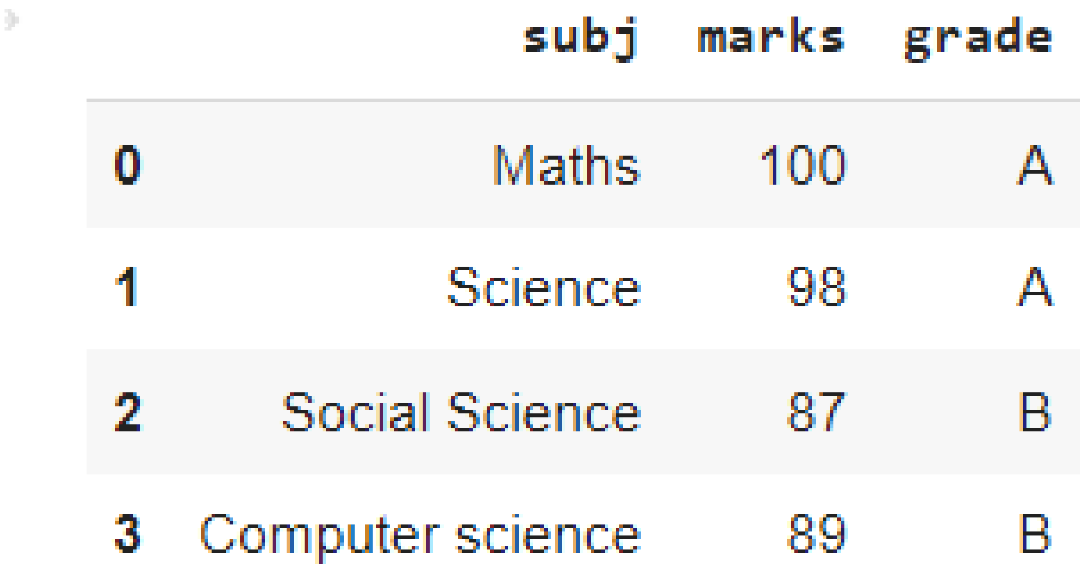
```
import pandas as pd
# create an Empty DataFrame object
df = pd.DataFrame()
print(df)
# append columns to an empty DataFrame
df['Name'] = ['Ankit', 'Ankita', 'Yashvardhan']
df['Age'] = [19,18,20]
df['Gender'] = ['M','F','M']
df
del(df['Age'])
df
```



```
Empty DataFrame
Columns: []
Index: []
```

|   | Name | Age | Gender |
|---|---|---|---|
| **0** | Ankit | 19 | M |
| **1** | Ankita | 18 | F |
| **2** | Yashvardhan | 20 | M |

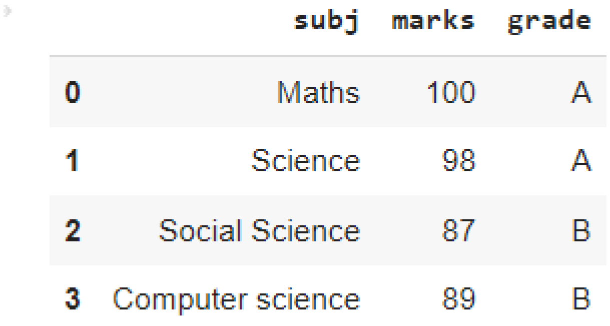
# Pandas Data frame



| | subj | marks | grade |
|---|---|---|---|
| **0** | Maths | 100 | A |
| **1** | Science | 98 | A |
| **2** | Social Science | 87 | B |
| **3** | Computer science | 89 | B |

# Pandas - Dataframe

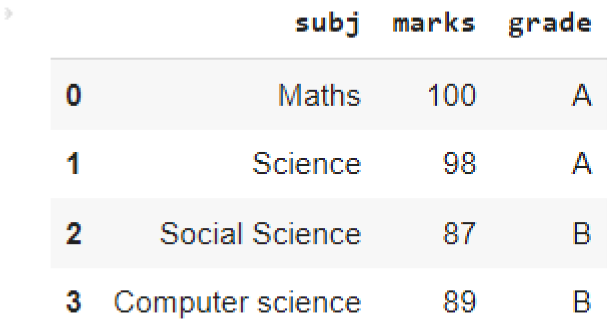
```
#pandas dataframe
#creating from dictionary
data={'subj':["Maths","Science","Social Science","Computer science"],'marks':[100,98,87,89],'grade':('A','A','B','B')}
data
df=pd.DataFrame(data)
df
```



| | subj | marks | grade |
|---|---|---|---|
| 0 | Maths | 100 | A |
| 1 | Science | 98 | A |
| 2 | Social Science | 87 | B |
| 3 | Computer science | 89 | B |

# Pandas - Dataframe

```
#creating from series
subj=pd.Series(["Maths","Science","Social Science","Computer science"])
marks=pd.Series([100,98,87,89])
grade=pd.Series(('A','A','B','B'))

df=pd.DataFrame([subj,marks,grade],index=['subj','marks','grade']).T
print(df)
```



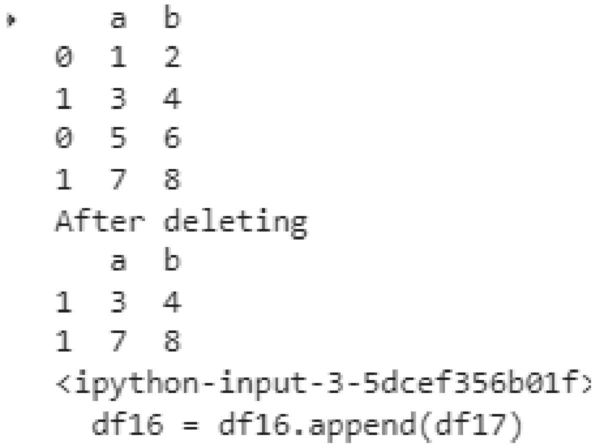
| | subj | marks | grade |
|---|---|---|---|
| 0 | Maths | 100 | A |
| 1 | Science | 98 | A |
| 2 | Social Science | 87 | B |
| 3 | Computer science | 89 | B |

# Pandas - Dataframe

```
#creating from list

df1 = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
print(df1)
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
Print(df2)
```

# Append and Drop

```
df1 = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])

df1 = df1.append(df2)
print(df1)
print("After deleting")
df1 = df1.drop(0)
print(df1)
```



```
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
After deleting
   a  b
1  3  4
1  7  8
<ipython-input-3-5dcef356b01f>
  df16 = df16.append(df17)
```

Note: To append a row we can use

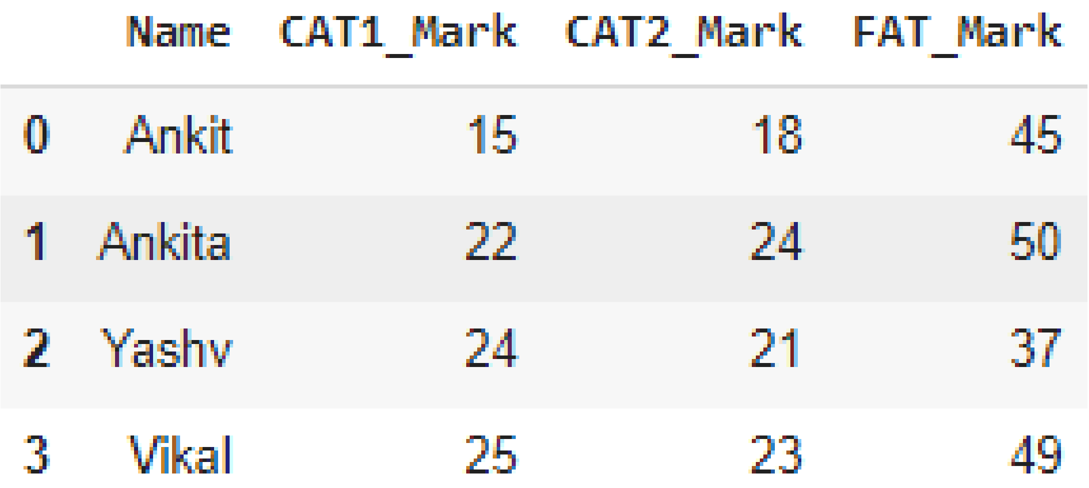
`df1.loc[len(df1.index)] = [7,7]`

# Pandas Data frame – Practice Exercise

- Row Selection, Addition, and Deletion:
  - **Selection by Label:**
    - Rows can be selected by passing row label to a **loc** function.
  - **Selection by integer location**
    - Rows can be selected by passing integer location to an **iloc** function.
  - Slice Rows
    - Multiple rows can be selected using ‘ : ’ operator.

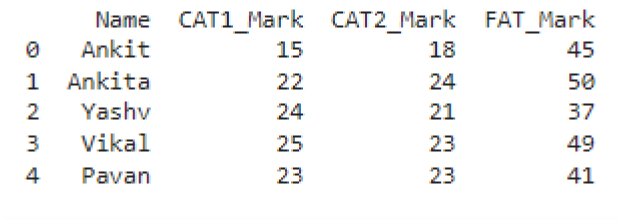
# DataFrame

Create a dataframe to store the following details:

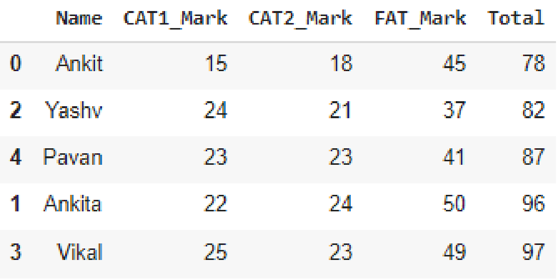


| | Name | CAT1_Mark | CAT2_Mark | FAT_Mark |
|---|---|---|---|---|
| 0 | Ankit | 15 | 18 | 45 |
| 1 | Ankita | 22 | 24 | 50 |
| 2 | Yashv | 24 | 21 | 37 |
| 3 | Vikal | 25 | 23 | 49 |

Then add the details of a new student "Pavan, 23,23,41" to the existing dataframe. Include a column 'Total' to store the marks total marks scored by each student and sort the records based on their total scores.

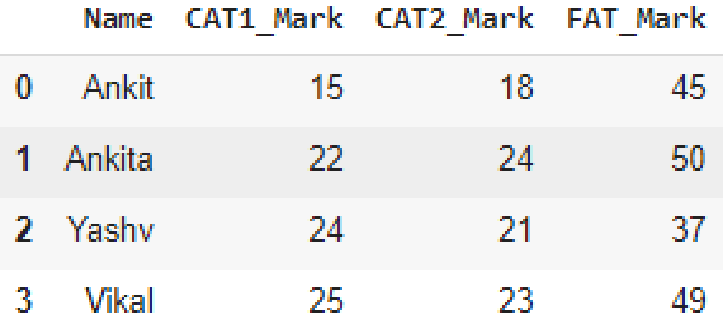


```
     Name  CAT1_Mark  CAT2_Mark  FAT_Mark
0   Ankit         15         18        45
1  Ankita         22         24        50
2   Yashv         24         21        37
3   Vikal         25         23        49
4   Pavan         23         23        41
```



| | Name | CAT1_Mark | CAT2_Mark | FAT_Mark | Total |
|---|---|---|---|---|---|
| 0 | Ankit | 15 | 18 | 45 | 78 |
| 2 | Yashv | 24 | 21 | 37 | 82 |
| 4 | Pavan | 23 | 23 | 41 | 87 |
| 1 | Ankita | 22 | 24 | 50 | 96 |
| 3 | Vikal | 25 | 23 | 49 | 97 |

```
#create an Empty DataFrame object
df = pd.DataFrame()
# append columns to an empty DataFrame
df['Name'] = ['Ankit', 'Ankita', 'Yashv','Vikal']
df['CAT1_Mark'] = [15,22,24,25]
df['CAT2_Mark'] = [18,24,21,23]
df['FAT_Mark']=[45,50,37,49]
print(df)
```



| | Name | CAT1_Mark | CAT2_Mark | FAT_Mark |
|---|---|---|---|---|
| 0 | Ankit | 15 | 18 | 45 |
| 1 | Ankita | 22 | 24 | 50 |
| 2 | Yashv | 24 | 21 | 37 |
| 3 | Vikal | 25 | 23 | 49 |

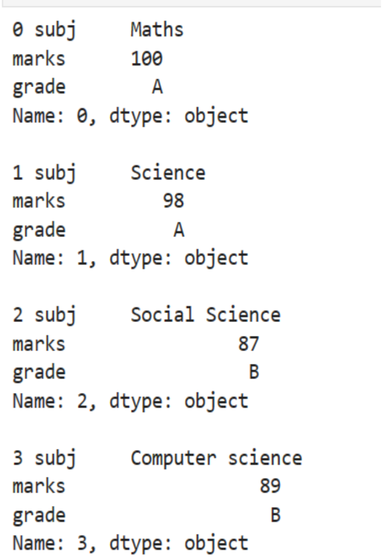
```
#create an Empty DataFrame object
df = pd.DataFrame()
# append columns to an empty DataFrame
df['Name'] = ['Ankit', 'Ankita', 'Yashv','Vikal']
df['CAT1_Mark'] = [15,22,24,25]
df['CAT2_Mark'] = [18,24,21,23]
df['FAT_Mark']=[45,50,37,49]
print(df)
df.loc[len(df.index)] = ['Pavan',23,23,41]
print(df)
df['Total']=df['CAT1_Mark']+df['CAT2_Mark']+df['FAT_Mark']
df
df.sort_values('Total')
```

# Iterating over Rows and Columns

## Iterating over rows :

Apply iterrows() function to get each element of rows.

```
#creating from dictionary
data={'subj':["Maths","Science","Social Science","Computer science"], 'marks’: [100,98,87,89],’grade':('A','A','B','B’)}
df=pd.DataFrame(data)
# iterating over rows using iterrows() function
for i, j in df.iterrows():
        print(i, j)
```

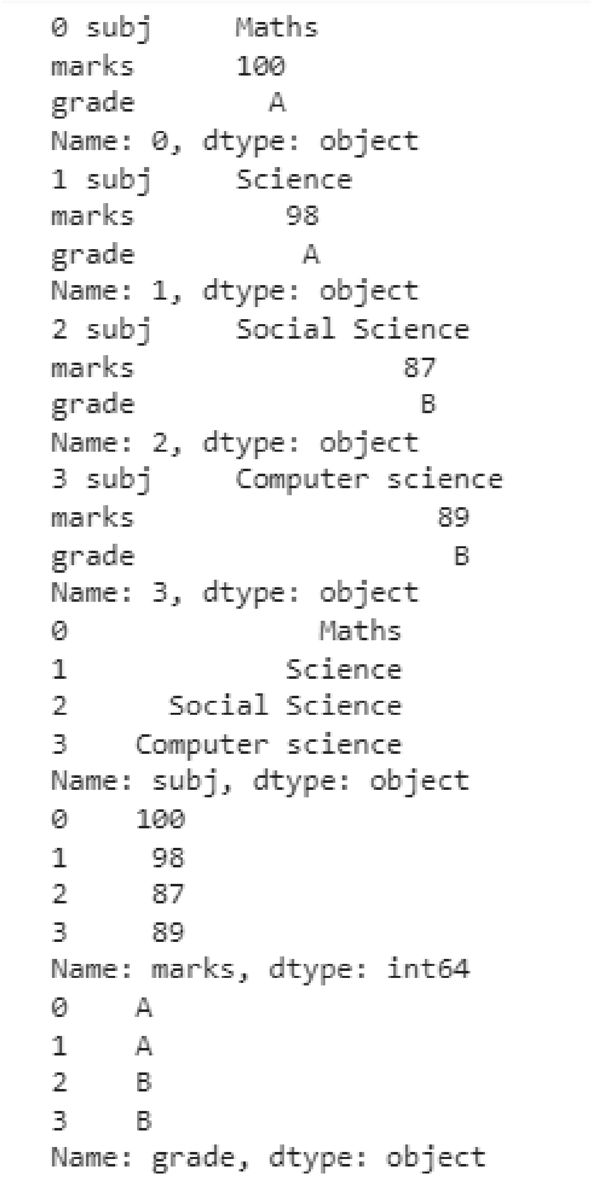


```
0 subj      Maths
marks       100
grade         A
Name: 0, dtype: object

1 subj      Science
marks           98
grade            A
Name: 1, dtype: object

2 subj      Social Science
marks                   87
grade                    B
Name: 2, dtype: object

3 subj      Computer science
marks                     89
grade                      B
Name: 3, dtype: object
```

# Iterating over Columns

**Iterating over columns :**

```
#creating from dictionary
data={'subj':["Maths","Science","Social Science","Computer science"],
'marks’: [100,98,87,89],’grade':('A','A','B','B’)}
df=pd.DataFrame(data)
# iterating over columns
for i in df.columns():
            print(df[i])
```



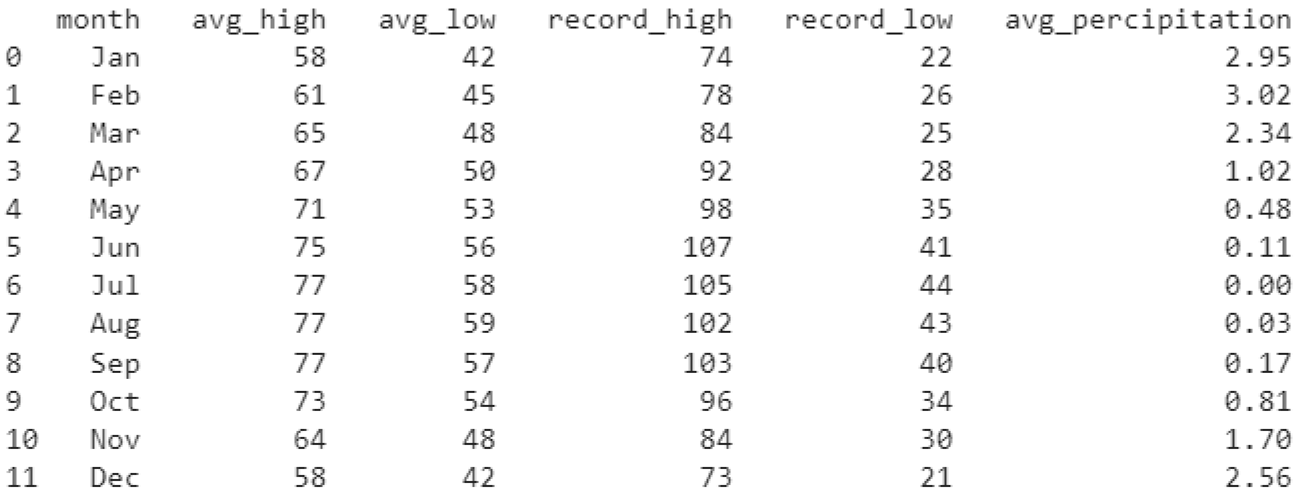
```
0 subj     Maths
marks      100
grade        A
Name: 0, dtype: object
1 subj     Science
marks         98
grade          A
Name: 1, dtype: object
2 subj     Social Science
marks                87
grade                 B
Name: 2, dtype: object
3 subj     Computer science
marks                  89
grade                   B
Name: 3, dtype: object
0                Maths
1              Science
2       Social Science
3     Computer science
Name: subj, dtype: object
0    100
1     98
2     87
3     89
Name: marks, dtype: int64
0    A
1    A
2    B
3    B
Name: grade, dtype: object
```

# Pandas - Dataframe

```
#loading hard coded data
df_h=pd.DataFrame([['Jan', 58, 42, 74, 22, 2.95],
                ['Feb', 61, 45, 78, 26, 3.02],
               ['Mar', 65, 48, 84, 25, 2.34],
               ['Apr', 67, 50, 92, 28, 1.02],
               ['May', 71, 53, 98, 35, 0.48],
               ['Jun', 75, 56, 107, 41, 0.11],
               ['Jul', 77, 58, 105, 44, 0.0],
               ['Aug', 77, 59, 102, 43, 0.03],
               ['Sep', 77, 57, 103, 40, 0.17],
               ['Oct', 73, 54, 96, 34, 0.81],
               ['Nov', 64, 48, 84, 30, 1.7],
               ['Dec', 58, 42, 73, 21, 2.56]],
                index=[0,1,2,3,4,5,6,7,8,9,10,11],
               columns=['month','avg_low','avg_high','record_high','record_low','avg_preci'])
print(df_h)
```

# Pandas - Dataframe

```
#reading data from csv files
filename='E:\data\weather.csv'
df=pd.read_csv(filename)
print(df)
```



```
   month  avg_high  avg_low  record_high  record_low  avg_percipitation
0    Jan        58       42           74          22               2.95
1    Feb        61       45           78          26               3.02
2    Mar        65       48           84          25               2.34
3    Apr        67       50           92          28               1.02
4    May        71       53           98          35               0.48
5    Jun        75       56          107          41               0.11
6    Jul        77       58          105          44               0.00
7    Aug        77       59          102          43               0.03
8    Sep        77       57          103          40               0.17
9    Oct        73       54           96          34               0.81
10   Nov        64       48           84          30               1.70
11   Dec        58       42           73          21               2.56
```

# Pandas - Dataframe

Examples in the demo session

# Pandas Data frame – Practice Exercise

- Creation of Empty Data Frame:
- Create a DataFrame from Lists
- Create a DataFrame from Dict
- Create a DataFrame from series
- Column Selection
  - Selection of a column from the DataFrame.
- Column Addition
- Column Deletion:

# Pandas Data frame – Practice Exercise

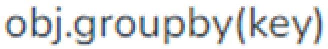
- Row Selection, Addition, and Deletion:
  - **Selection by Label:**
    - Rows can be selected by passing row label to a **loc** function.
  - **Selection by integer location**
    - Rows can be selected by passing integer location to an **iloc** function.
  - Slice Rows
    - Multiple rows can be selected using ‘ : ’ operator.

# Pandas Data frame – Practice Exercise

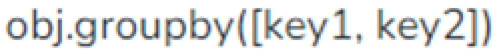
- Row Selection, Addition, and Deletion:
  - **Adding rows:**
    - Add new rows to a DataFrame
  - Deletion of Rows:
    - Use index label to delete or drop rows from a DataFrame, If label is duplicated, then multiple rows will be dropped.

# Pandas Data frame – groupby

- groupby operation allows you to partition an array into groups based on the value in one or more columns and then perform operations on each group separately



```
obj.groupby(key)
```



```
obj.groupby([key1, key2])
```

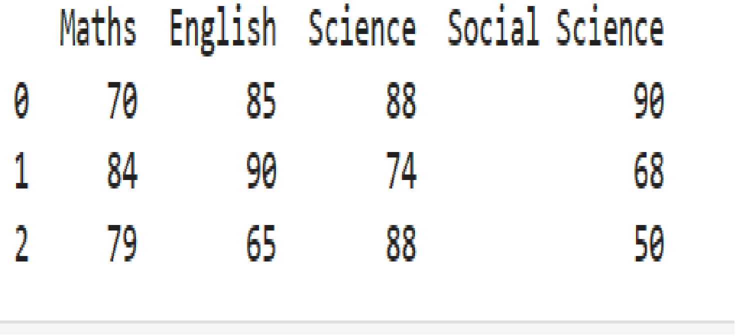
Aggregation :

- Aggregation is a process in which we compute a summary statistic about each group. Aggregated function returns a single aggregated value for each group. After splitting a data into groups using groupby function, several aggregation operations can be performed on the grouped data.

# Aggregation in Pandas

Grouping and Aggregation helps to group and summarize the data and make complex analysis comparatively easy.

```
import pandas as pd
# Creating dataset
df = pd.DataFrame([[70, 85, 88, 90], [84, 90, 74, 68], [79, 65, 88, 50]], columns=['Maths', 'English', 'Science', 'Social Science'])
# display dataset
print(df)
```



```
   Maths  English  Science  Social Science
0     70       85       88              90
1     84       90       74              68
2     79       65       88              50
```

# Aggregation in Pandas

- Aggregation in pandas provides various functions that perform a mathematical or logical operation on the dataset and returns a summary of that function.
- Aggregation can be used to get a summary of columns in the dataset like getting sum, min, max, mean, size, describe, first, last, count, std, var, sem from a particular column of the dataset.
- The function used for aggregation is agg().

# Aggregation in Pandas

df.describe()

The agg() function can be used to calculate the sum, min, and max of each column in the dataset.

df.agg(['sum', 'min', 'max’])

df.describe() - *Generates descriptive sta*

df.mean()

df.std()-*Standard deviation of column*

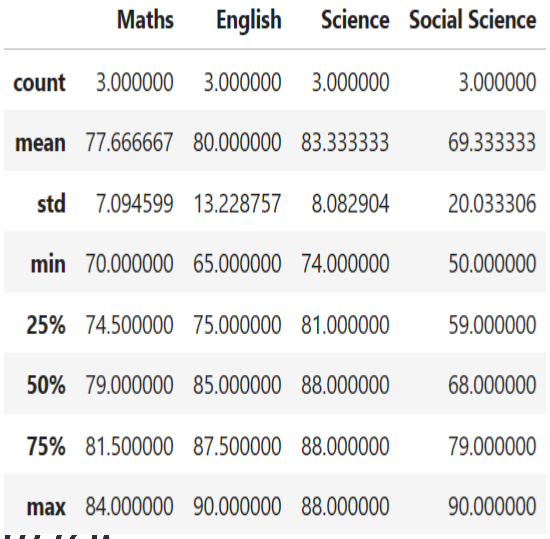
df.var()-*Compute variance of column*

df.sem()-*Standard error of the mean of*

df.count()-*Compute count of column values*

df.first()- *Compute first of group values*

df.last() - *Compute last of group values*



| | Maths | English | Science | Social Science |
|---|---|---|---|---|
| **count** | 3.000000 | 3.000000 | 3.000000 | 3.000000 |
| **mean** | 77.666667 | 80.000000 | 83.333333 | 69.333333 |
| **std** | 7.094599 | 13.228757 | 8.082904 | 20.033306 |
| **min** | 70.000000 | 65.000000 | 74.000000 | 50.000000 |
| **25%** | 74.500000 | 75.000000 | 81.000000 | 59.000000 |
| **50%** | 79.000000 | 85.000000 | 88.000000 | 68.000000 |
| **75%** | 81.500000 | 87.500000 | 88.000000 | 79.000000 |
| **max** | 84.000000 | 90.000000 | 88.000000 | 90.000000 |

# Grouping in Pandas

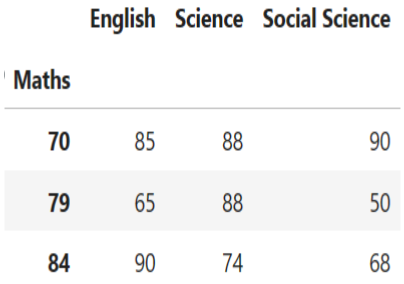
Grouping is used to group data using some criteria from the dataset.

It is used as split-apply-combine strategy.

- ❖ Splitting the data into groups based on some criteria.
- ❖ Applying a function to each group independently.
- ❖ Combining the results into a data structure.

We use groupby() function to group the data on “Maths” value.
It returns the object as result.
To view result of formed groups us n.

```
a = df.groupby('Maths’)
a.first()
```

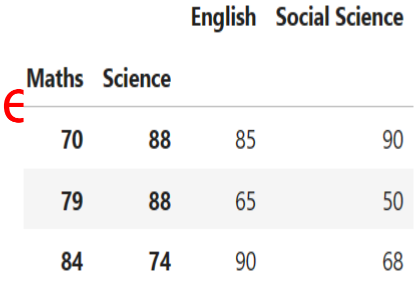


| | English | Science | Social Science |
|---|---|---|---|
| **Maths** | | | |
| **70** | 85 | 88 | 90 |
| **79** | 65 | 88 | 50 |
| **84** | 90 | 74 | 68 |

# Grouping in Pandas

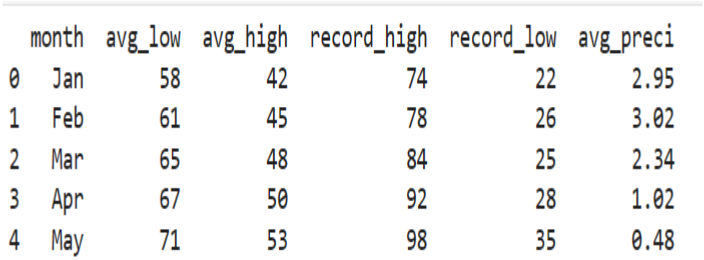
First grouping based on “Maths” within each team we are grouping based on “Science”

```
b = df.groupby(['Maths', 'Science
b.first()
```



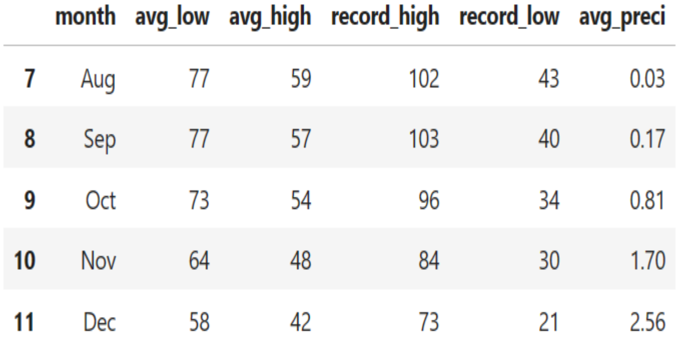
| | | English | Social Science |
|---|---|---|---|
| **Maths** | **Science** | | |
| **70** | **88** | 85 | 90 |
| **79** | **88** | 65 | 50 |
| **84** | **74** | 90 | 68 |

```
import numpy as np
import pandas as pd
# reading csv file
dataset = pd.read_csv("E:/weather.csv )
# printing first 5 rows
print(dataset.head(5))
```



```
  month  avg_low  avg_high  record_high  record_low  avg_preci
0   Jan       58        42           74          22       2.95
1   Feb       61        45           78          26       3.02
2   Mar       65        48           84          25       2.34
3   Apr       67        50           92          28       1.02
4   May       71        53           98          35       0.48
```

```
# printing last 5 rows
dataset.tail(5)
```



| | month | avg_low | avg_high | record_high | record_low | avg_preci |
|---|---|---|---|---|---|---|
| **7** | Aug | 77 | 59 | 102 | 43 | 0.03 |
| **8** | Sep | 77 | 57 | 103 | 40 | 0.17 |
| **9** | Oct | 73 | 54 | 96 | 34 | 0.81 |
| **10** | Nov | 64 | 48 | 84 | 30 | 1.70 |
| **11** | Dec | 58 | 42 | 73 | 21 | 2.56 |

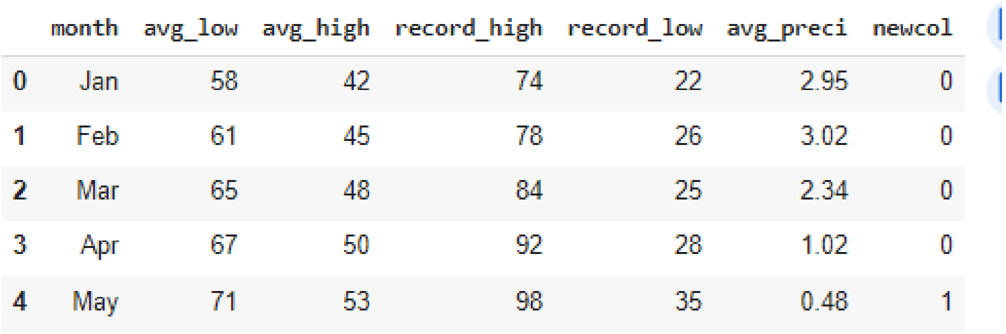
# Grouping in Pandas

```
df['avg_high'].aggregate([min,np.median,max])
df.groupby('newcol').aggregate({'avg_low':[min,max],
'avg_high':[np.mean,np.median]})
df.groupby('newcol')['avg_low'].std()
df['Catcol']=['A','A','A','B','B','B','C','C','C','D','D','D']
df.head()
df.groupby(['newcol','catcol']).sum()
```

# Pandas Data frame – groupby

In weather dataset add a new column:

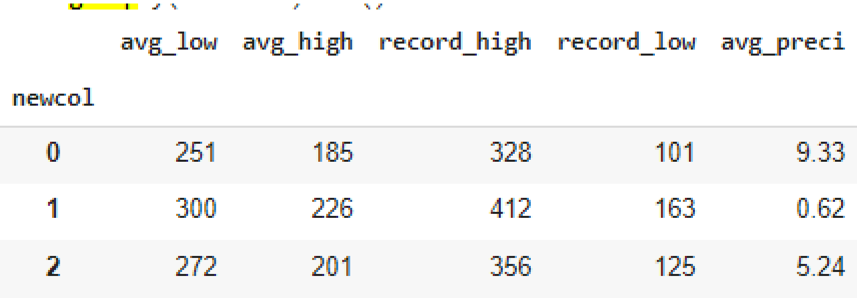
```
df['newcol']=[0,0,0,0,1,1,1,1,2,2,2,2]
df.head()
```



|   | month | avg_low | avg_high | record_high | record_low | avg_preci | newcol |
|---|---|---|---|---|---|---|---|
| 0 | Jan | 58 | 42 | 74 | 22 | 2.95 | 0 |
| 1 | Feb | 61 | 45 | 78 | 26 | 3.02 | 0 |
| 2 | Mar | 65 | 48 | 84 | 25 | 2.34 | 0 |
| 3 | Apr | 67 | 50 | 92 | 28 | 1.02 | 0 |
| 4 | May | 71 | 53 | 98 | 35 | 0.48 | 1 |

# Pandas Data frame – groupby

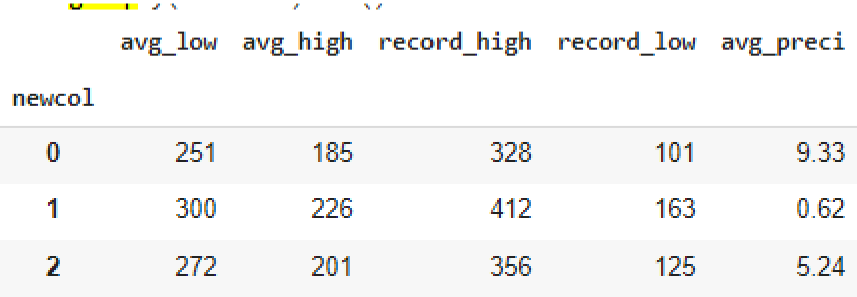
```
df['newcol']=[0,0,0,0,1,1,1,1,2,2,2,2]
df.head()
df.groupby('newcol').sum()
```



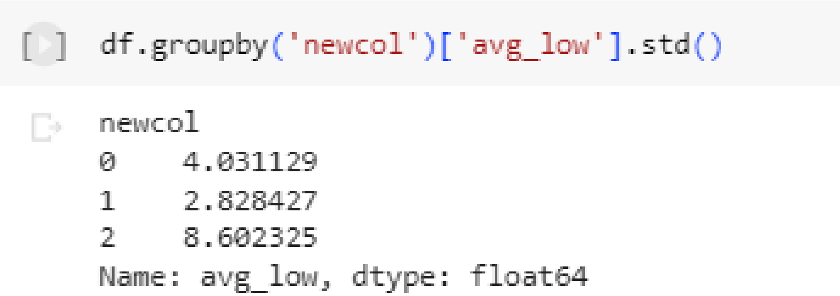
|  | avg_low | avg_high | record_high | record_low | avg_preci |
|---|---|---|---|---|---|
| newcol |  |  |  |  |  |
| 0 | 251 | 185 | 328 | 101 | 9.33 |
| 1 | 300 | 226 | 412 | 163 | 0.62 |
| 2 | 272 | 201 | 356 | 125 | 5.24 |

# Pandas Data frame – groupby

```
df['newcol']=[0,0,0,0,1,1,1,1,2,2,2,2]
df.head()
df.groupby('newcol').sum()
```

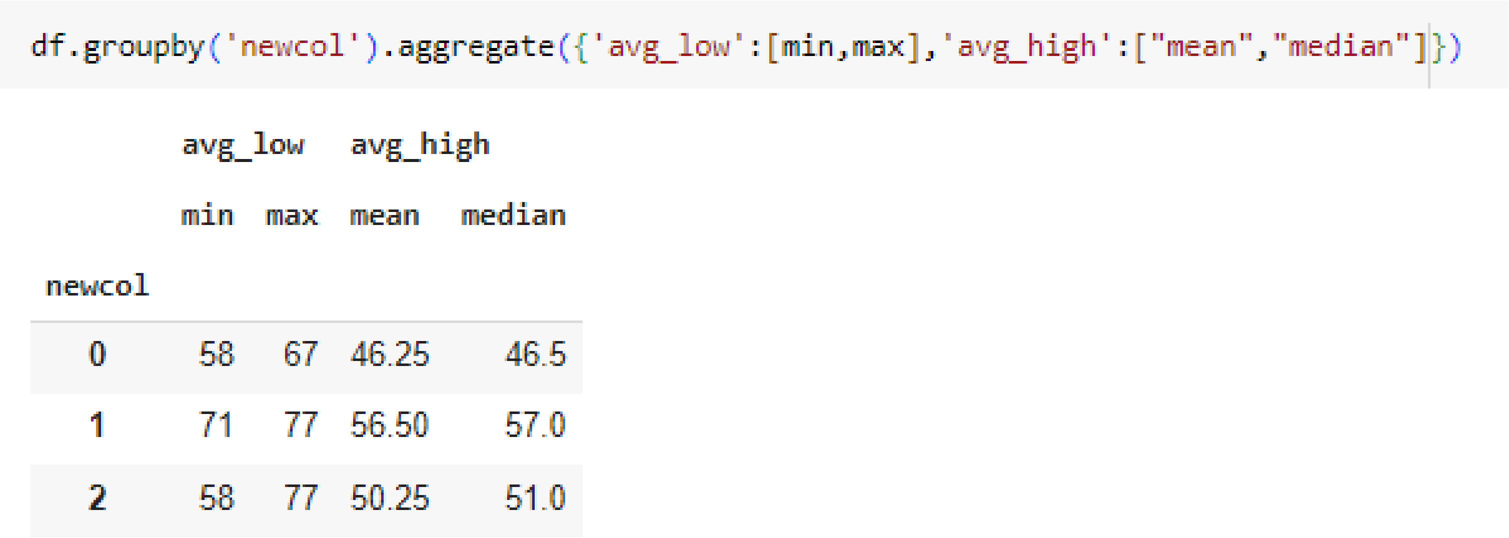


| | avg_low | avg_high | record_high | record_low | avg_preci |
|---|---|---|---|---|---|
| **newcol** | | | | | |
| **0** | 251 | 185 | 328 | 101 | 9.33 |
| **1** | 300 | 226 | 412 | 163 | 0.62 |
| **2** | 272 | 201 | 356 | 125 | 5.24 |



```
[ ] df.groupby('newcol')['avg_low'].std()

    newcol
    0    4.031129
    1    2.828427
    2    8.602325
    Name: avg_low, dtype: float64
```

# Pandas Data frame – groupby



```
df.groupby('newcol').aggregate({'avg_low':[min,max],'avg_high':["mean","median"]})
```

| | avg_low | | avg_high | |
|---|---|---|---|---|
| | min | max | mean | median |
| newcol | | | | |
| 0 | 58 | 67 | 46.25 | 46.5 |
| 1 | 71 | 77 | 56.50 | 57.0 |
| 2 | 58 | 77 | 50.25 | 51.0 |

# Data Wrangling

Data Wrangling is the process of gathering, collecting, and transforming Raw data into another format for better understanding, decision-making, accessing, and analysis in less time.

Data Wrangling is also known as Data Munging.

Data wrangling in Python deals with the below functionalities:

1.Data exploration: In this process, the data is studied, analyzed, and understood by visualizing representations of data.

2.Dealing with missing values: The datasets that contain missing values of NaN, are needed to be taken care by replacing them with mean, mode, the most frequent value of the column, or simply by dropping the row having a NaN value.

3.Reshaping data: In this process, data is manipulated according to the requirements, where new data can be added or pre-existing data can be modified.

4.Filtering data: Some times datasets are comprised of unwanted rows or columns which are required to be removed or filtered.

5.Other: After dealing with the raw dataset with the above functionalities we get an efficient dataset as per the requirements and then it can be used for a required purpose like data analyzing, machine learning, data visualization, model training etc.

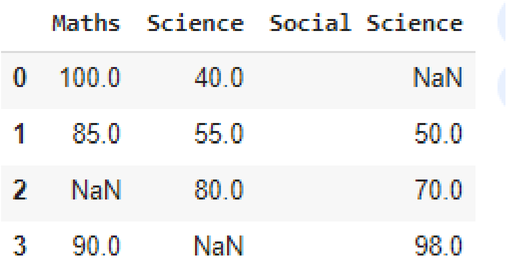
# Working with Missing Data

- Missing data can occur when no information is provided for one or more items or for a whole unit.
- Missing Data can also refer to as NA(Not Available) values in pandas.

## Checking for missing values using isnull() and notnull() :

To check missing values in Pandas DataFrame, a function isnull() and notnull() is used.
Both function help in checking whether a value is NaN or not.
These function can also be used in Pandas Series in order to find null values in a series.

```
import pandas as pd
import numpy as np
# dictionary of lists
dict1 = {'Maths':[100, 85, np.nan, 90],'Science': [40, 55, 80, np.nan],'Social Science':[np.nan, 50, 70, 98]}
# creating a dataframe from list
df = pd.DataFrame(dict1)
# using isnull() function
df.isnull()
# using notnull() function
df.notnull()
```



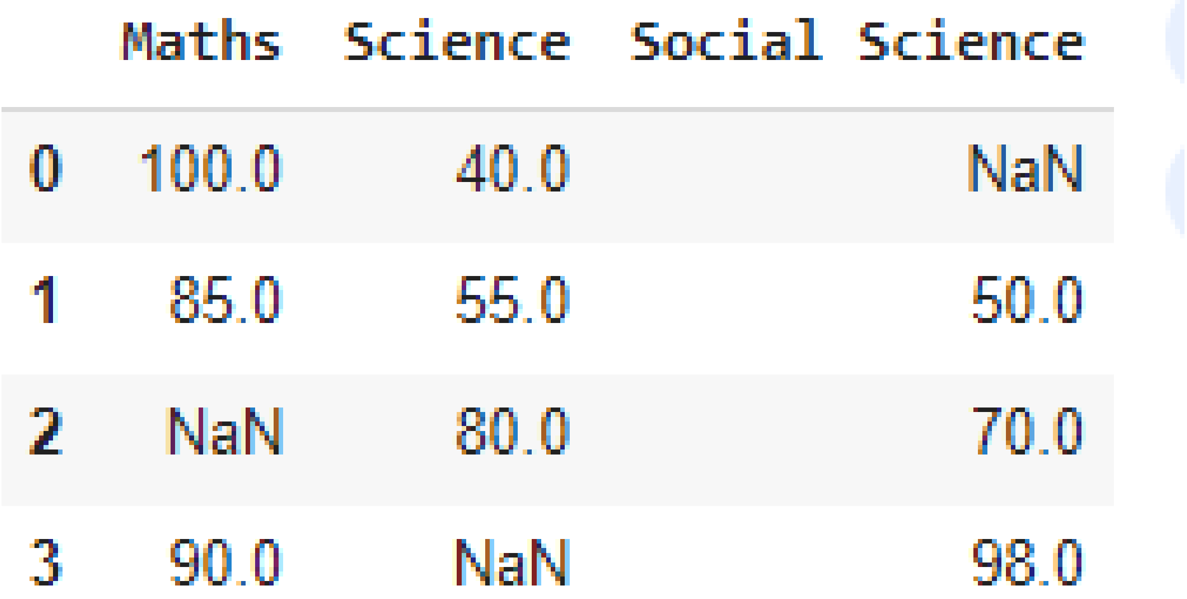
| | Maths | Science | Social Science |
|---|---|---|---|
| 0 | 100.0 | 40.0 | NaN |
| 1 | 85.0 | 55.0 | 50.0 |
| 2 | NaN | 80.0 | 70.0 |
| 3 | 90.0 | NaN | 98.0 |

# Working with Missing Data

- Missing data can occur when no information is provided for one or more items or for a whole unit.
- Missing Data can also refer to as NA(Not Available) values in pandas.

**Checking for missing values using isnull() and notnull() :**

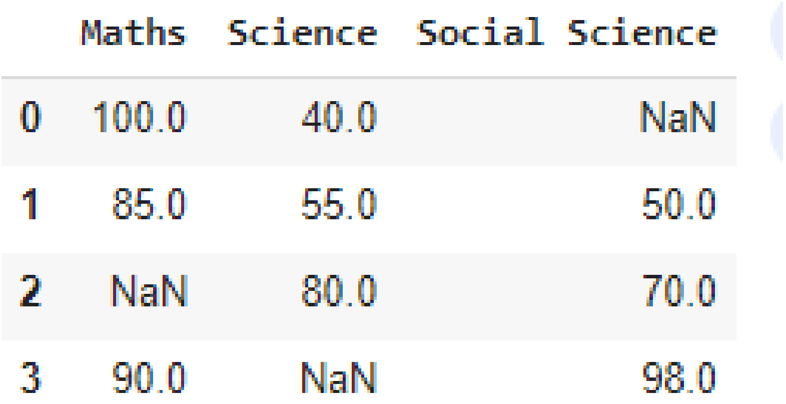
To check missing values in Pandas DataFrame, a function isnull() and notnull() is used.
Both function help in checking whether a value is NaN or not.
These function can also be used in Pandas Series in order to find null values in a series.



| | Maths | Science | Social Science |
|---|---|---|---|
| 0 | 100.0 | 40.0 | NaN |
| 1 | 85.0 | 55.0 | 50.0 |
| 2 | NaN | 80.0 | 70.0 |
| 3 | 90.0 | NaN | 98.0 |

# Working with Missing Data

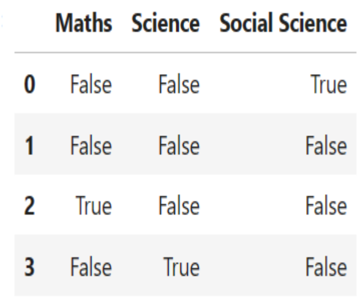
```
import pandas as pd
import numpy as np

dict1 = {'Maths':[100, 85, np.nan, 90],'Science': [40, 55, 80, np.nan],'Social Science':[np.nan, 50, 70, 98]}

df = pd.DataFrame(dict1)
```



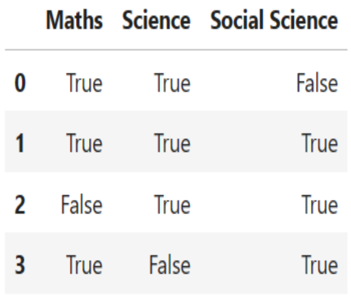
| | Maths | Science | Social Science |
|---|---|---|---|
| 0 | 100.0 | 40.0 | NaN |
| 1 | 85.0 | 55.0 | 50.0 |
| 2 | NaN | 80.0 | 70.0 |
| 3 | 90.0 | NaN | 98.0 |

# Working with Missing Data

```
import pandas as pd
import numpy as np
dict1 = {'Maths':[100, 85, np.nan, 90],'Science': [40, 55,
80, np.nan],'Social Science':[np.nan, 50, 70, 98]}

df = pd.DataFrame(dict1)
# using isnull() function
df.isnull()
# using notnull() function
df.notnull()
```



| | Maths | Science | Social Science |
|---|---|---|---|
| 0 | False | False | True |
| 1 | False | False | False |
| 2 | True | False | False |
| 3 | False | True | False |



| | Maths | Science | Social Science |
|---|---|---|---|
| 0 | True | True | False |
| 1 | True | True | True |
| 2 | False | True | True |
| 3 | True | False | True |

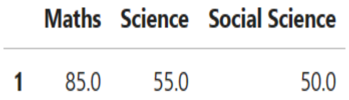
# Working with Missing Data

## Dropping missing values using dropna():

To drop null values from a dataframe, dropna() function is used.

This fuction drop Rows/Columns of datasets with Null values in different ways.

```
import pandas as pd
import numpy as np
# dictionary of lists
dict1 = {'Maths':[100, 85, np.nan, 90],'Science': [40, 55, 80,
np.nan],'Social Science':[np.nan, 50, 70, 98]}
# creating a dataframe from list
df = pd.DataFrame(dict1)
# dropping missing value using droppingna()
df.dropna()
```



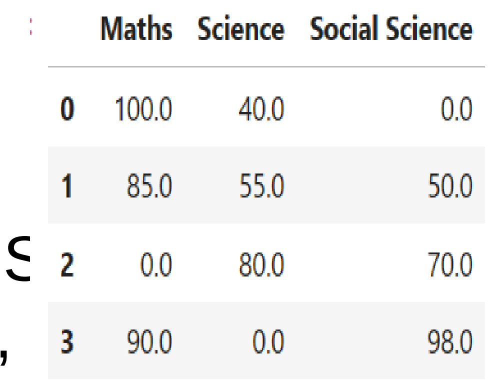
|   | Maths | Science | Social Science |
|---|---|---|---|
| 1 | 85.0 | 55.0 | 50.0 |

# Working with Missing Data

## Filling missing values using fillna() and replace()

To fill null values in a datasets, we use fillna() and replace() function.
These function replace NaN values with some value of their own.

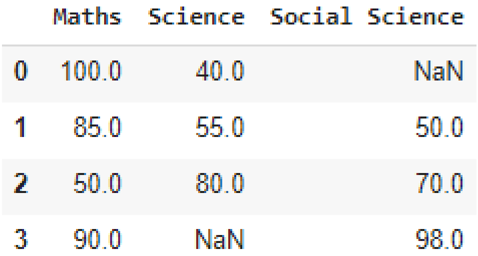
```
import pandas as pd
import numpy as np
# dictionary of lists
dict1 = {'Maths':[100, 85, np.nan, 90],'S                          , 80,
np.nan],'Social Science':[np.nan, 50, 70,
# creating a dataframe from list
df = pd.DataFrame(dict1)
# filling missing value using fillna()
df.fillna(0)
```



|   | Maths | Science | Social Science |
|---|---|---|---|
| 0 | 100.0 | 40.0 | 0.0 |
| 1 | 85.0 | 55.0 | 50.0 |
| 2 | 0.0 | 80.0 | 70.0 |
| 3 | 90.0 | 0.0 | 98.0 |

# Working with Missing Data

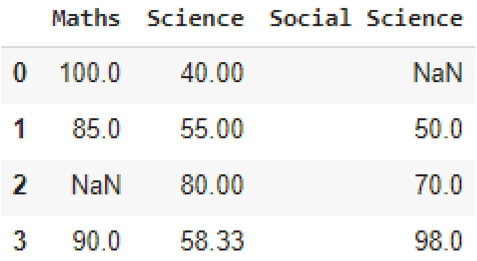
```
df['Maths'].replace(np.nan,50,inplace=True)
df
```



| | Maths | Science | Social Science |
|---|---|---|---|
| 0 | 100.0 | 40.0 | NaN |
| 1 | 85.0 | 55.0 | 50.0 |
| 2 | 50.0 | 80.0 | 70.0 |
| 3 | 90.0 | NaN | 98.0 |

# Working with Missing Data

```
df['Science'].fillna(round(df['Science'].mean(),2),inplace=True)
df
```



|   | Maths | Science | Social Science |
|---|---:|---:|---:|
| **0** | 100.0 | 40.00 | NaN |
| **1** | 85.0 | 55.00 | 50.0 |
| **2** | NaN | 80.00 | 70.0 |
| **3** | 90.0 | 58.33 | 98.0 |

Given a weather dataset with NA values, write the python code to process all NA values and analyze its statistical measures.

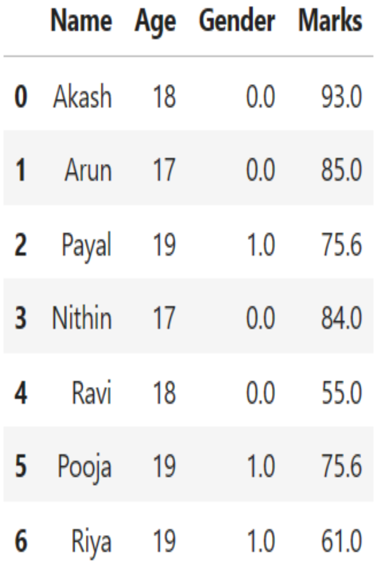
- Count the number of NA values in each attribute
- Drop all the NA values. How many records are retained?
- Replace NA values in ‘record_high’ with its mean value
- Replace NA values in avg_low with its minimum value.
- Replace NA values in avg_high with its maximum value.
- After replacing, compare its summary statistics with the original dataset.

[ ]:

# Data Replacing

In the *GENDER* column, we can replace the Gender column data by categorizing them into different numbers.

```
df['Gender'] = df['Gender'].map({'M': 0,'F': 1, }).astype(float)
# Display data
df
```



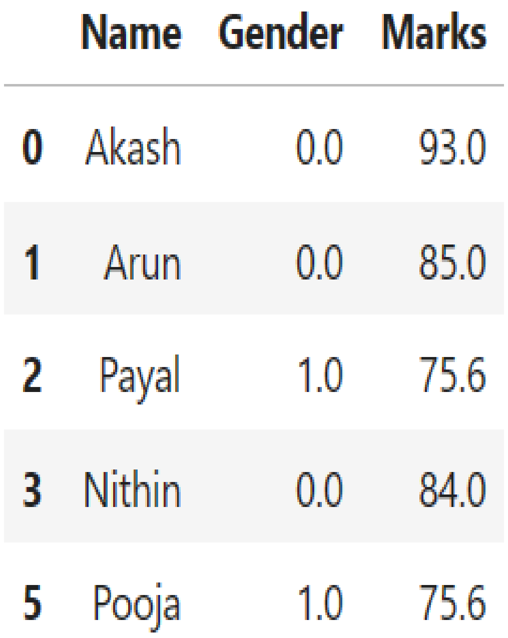
|   | Name | Age | Gender | Marks |
|---|---|---|---|---|
| **0** | Akash | 18 | 0.0 | 93.0 |
| **1** | Arun | 17 | 0.0 | 85.0 |
| **2** | Payal | 19 | 1.0 | 75.6 |
| **3** | Nithin | 17 | 0.0 | 84.0 |
| **4** | Ravi | 18 | 0.0 | 55.0 |
| **5** | Pooja | 19 | 1.0 | 75.6 |
| **6** | Riya | 19 | 1.0 | 61.0 |

[ ]:

# Filtering Data

If we need the details regarding name, gender, and marks of the top-scoring students, then filtering of data can be used.

```
# Filter top scoring students
df = df[df['Marks'] >= 75].copy()
# Remove age column from filtered DataFra
df.drop('Age', axis=1, inplace=True)
# Display data
df
```



|   | Name | Gender | Marks |
|---|---|---|---|
| 0 | Akash | 0.0 | 93.0 |
| 1 | Arun | 0.0 | 85.0 |
| 2 | Payal | 1.0 | 75.6 |
| 3 | Nithin | 0.0 | 84.0 |
| 5 | Pooja | 1.0 | 75.6 |

[ ]:

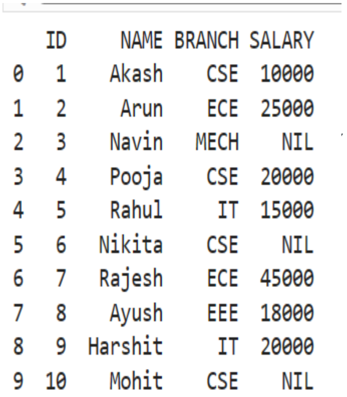
# Data Wrangling using Merge Operation

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

**Syntax:** pd.merge( data_frame1,data_frame2, on=”field “)

Here the field is the name of the column which is similar in both data-frame.

```
import pandas as pd
# creating DataFrame for Employee Details
details = pd.DataFrame({'ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'NAME': ['Akash', 'Arun', 'Navin', 'Pooja', 'Rahul', 'Nikita', 'Rajesh', 'Ayush', 'Harshit', "Mohit"], 'BRANCH': ['CSE', 'ECE', 'MECH', 'CSE', 'IT', 'CSE', 'ECE', 'EEE', 'IT', 'CSE’]})
# Creating Dataframe for Salary
salary = pd.DataFrame({'ID': [1, 2, 3, 4, 5, 6, 7, 8, 9 ':
['10000', '25000', 'NIL', '20000', '15000', 'NIL','45000', ' ',
'NIL’]})
print(pd.merge(details, salary, on='ID'))
```



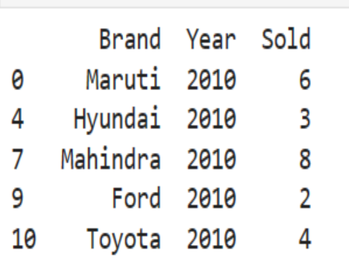
```
   ID     NAME BRANCH SALARY
0   1    Akash    CSE  10000
1   2     Arun    ECE  25000
2   3    Navin   MECH    NIL
3   4    Pooja    CSE  20000
4   5    Rahul     IT  15000
5   6   Nikita    CSE    NIL
6   7   Rajesh    ECE  45000
7   8    Ayush    EEE  18000
8   9  Harshit     IT  20000
9  10    Mohit    CSE    NIL
```

[ ]:

# Data Wrangling using Grouping

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

```
import pandas as pd
# Creating Data
car_selling_data = {'Brand': ['Maruti', 'Honda', 'Maruti','Honda', 'Hyundai', 'Hyundai','Toyota', 'Mahindra', 'Mahindra','Ford', 'Toyota', 'Ford'], 'Year': [2010, 2011, 2009, 2013, 2010, 2011, 2011, 2010, 2013, 2010, 2010, 2011], 'Sold': [6, 7, 9, 8, 3, 5,2,
# Creating Dataframe of car_selling_data
df = pd.DataFrame(car_selling_data)
df1 = df.groupby('Year’)
print(df1.get_group(2010))
```



```
       Brand  Year  Sold
0     Maruti  2010     6
4    Hyundai  2010     3
7   Mahindra  2010     8
9       Ford  2010     2
10    Toyota  2010     4
```

# Data Wrangling by removing Duplication

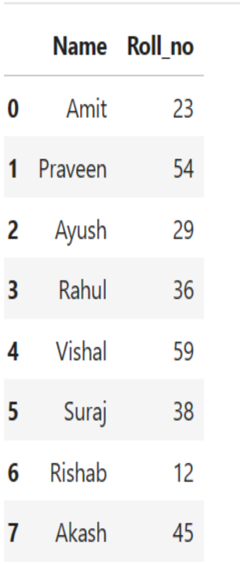
Duplicates method helps to remove duplicate values from Large Data.

**Syntax:** DataFrame.duplicated(subset=None, keep='first')
Here subset is the column value where we want to remove the Duplicate value.

In *keeping*, we have 3 options :
If *keep ='first'* then the first value is marked as the original rest of all values if occur will be removed as it is considered duplicate.
If *keep='last'* then the last value is marked as the original rest of the above values will be removed as it is considered duplicate.
If *keep ='false'* all the values which occur more than once will be removed as all are considered duplicate values.

# Data Wrangling by removing Duplication

```
import pandas as pd
# initializing Data
student_data = {'Name': ['Amit', 'Praveen', 'Ayush','Rahul', 'Vishal',
'Suraj','Rishab', 'Akash', 'Amit', 'Rahul', 'Praveen', 'Amit'], 'Roll_no': [23,
54, 29, 36, 59, 38, 12, 45, 34, 36, 54, 23]}
# creating dataframe
df = pd.DataFrame(student_data)
result = df[~df.duplicated('Name’)]
result
```



|  | Name | Roll_no |
|---|---|---|
| 0 | Amit | 23 |
| 1 | Praveen | 54 |
| 2 | Ayush | 29 |
| 3 | Rahul | 36 |
| 4 | Vishal | 59 |
| 5 | Suraj | 38 |
| 6 | Rishab | 12 |
| 7 | Akash | 45 |

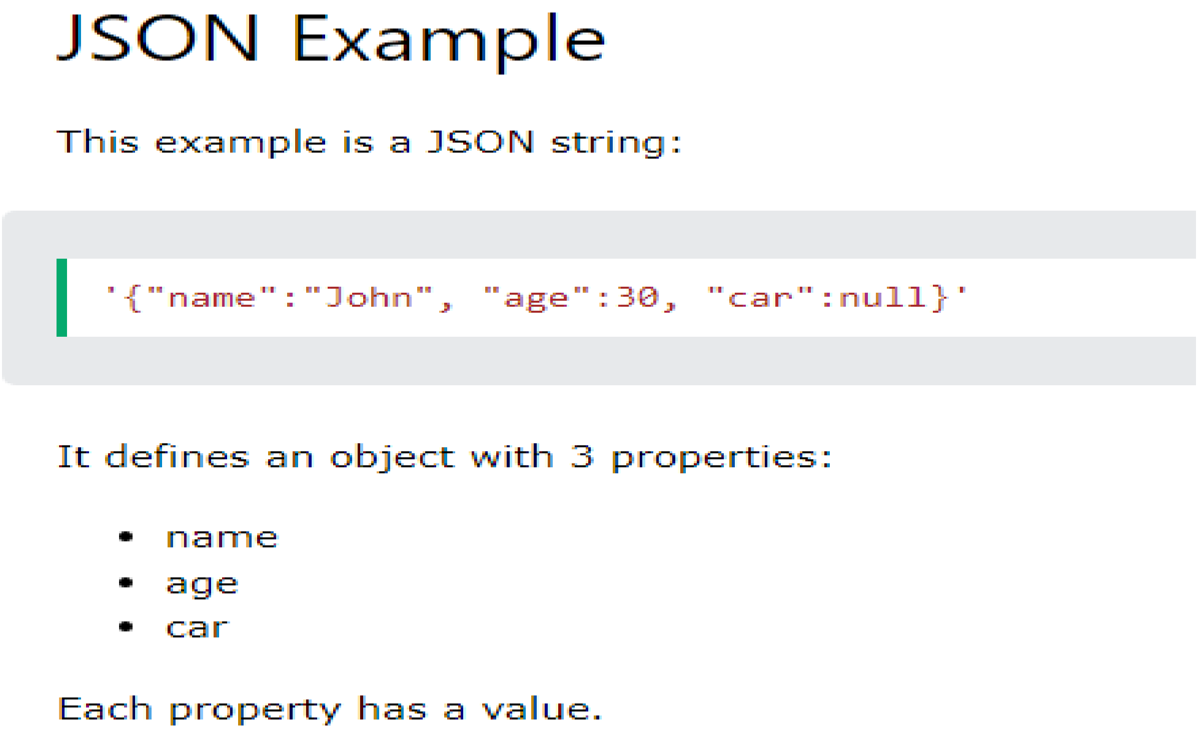
# JSON

JSON stands for JavaScript Object Notation

JSON is a text format for storing and transporting data

JSON is "self-describing" and easy to understand

Python JSON JavaScript Object Notation is a format for structuring data. It is mainly used for storing and transferring data between the browser and the server. Python too supports JSON with a built-in package called JSON



## JSON Example

This example is a JSON string:

```
'{"name":"John", "age":30, "car":null}'
```

It defines an object with 3 properties:

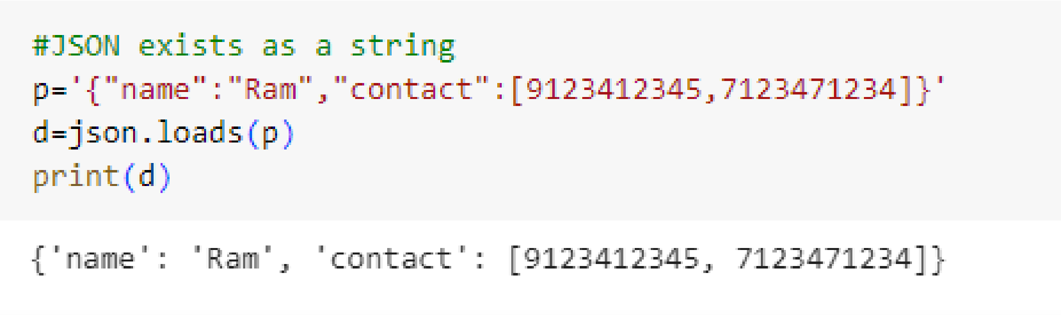
- name
- age
- car

Each property has a value.

# JSON

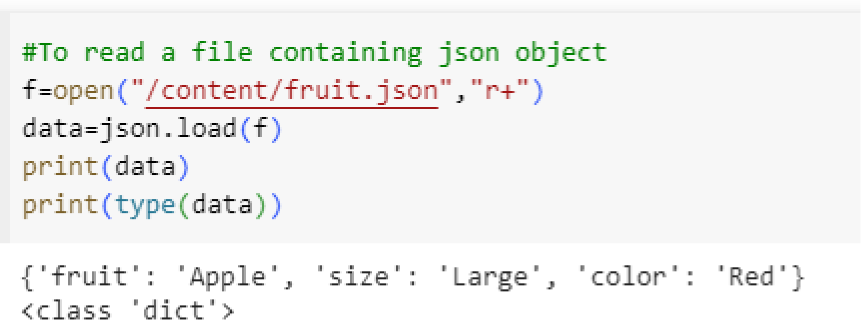
```
#To work with JSON import json module
import json
```

json.loads() method can be used to parse a valid JSON string and convert it into a Python Dictionary.



```
#JSON exists as a string
p='{"name":"Ram","contact":[9123412345,7123471234]}'
d=json.loads(p)
print(d)
```

```
{'name': 'Ram', 'contact': [9123412345, 7123471234]}
```

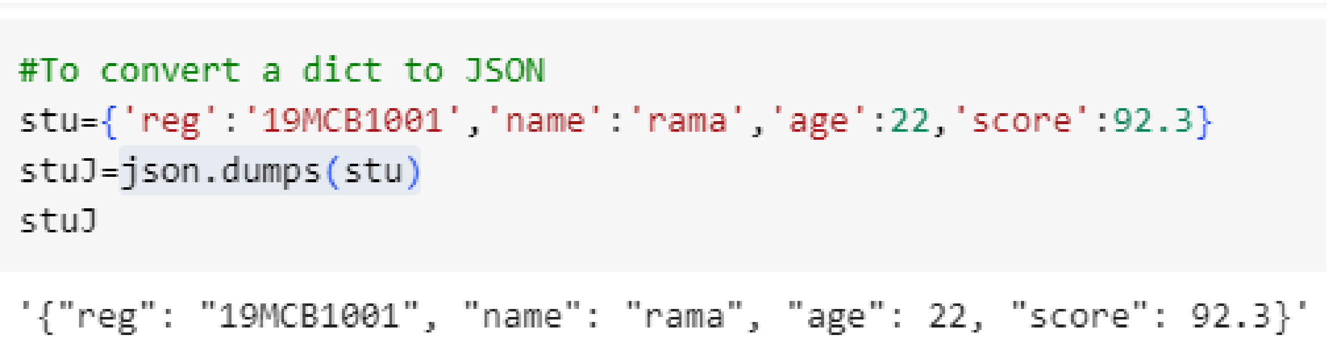


```
#To read a file containing json object
f=open("/content/fruit.json","r+")
data=json.load(f)
print(data)
print(type(data))
```

```
{'fruit': 'Apple', 'size': 'Large', 'color': 'Red'}
<class 'dict'>
```

# JSON

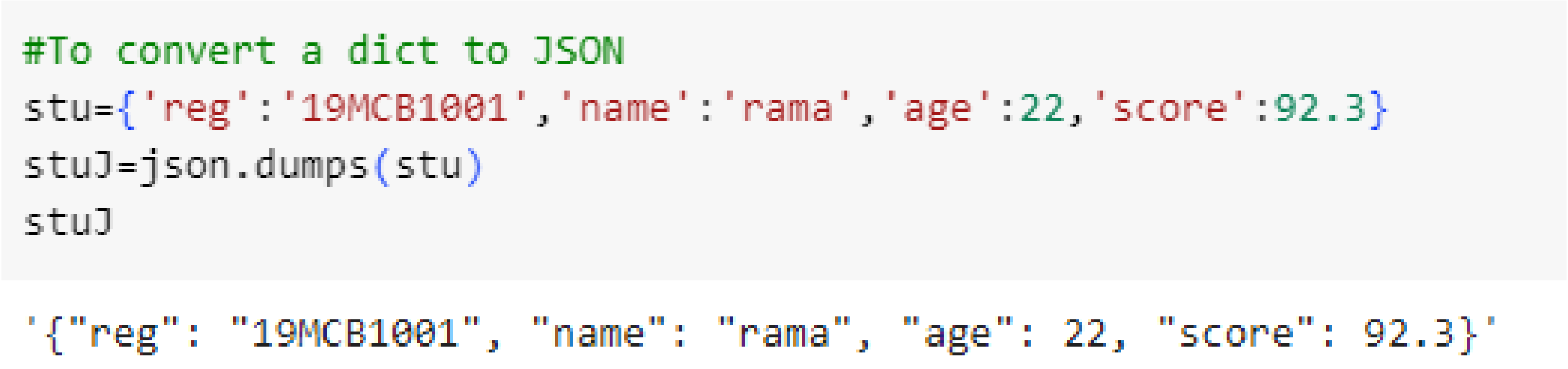
json.dumps()



```
#To convert a dict to JSON
stu={'reg':'19MCB1001','name':'rama','age':22,'score':92.3}
stuJ=json.dumps(stu)
stuJ
```

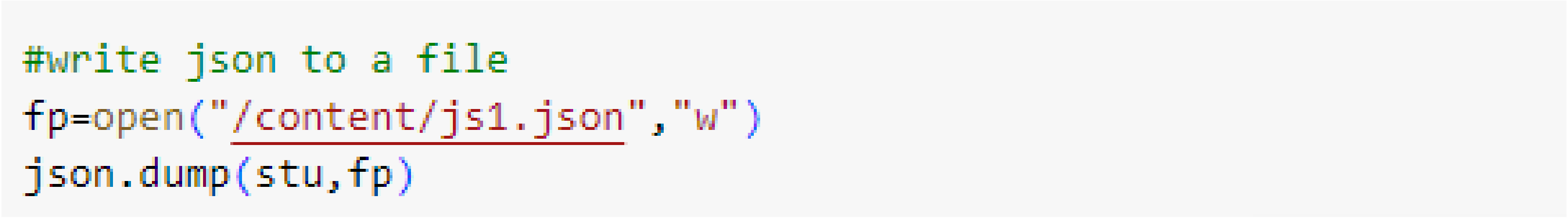
```
'{"reg": "19MCB1001", "name": "rama", "age": 22, "score": 92.3}'
```

# JSON



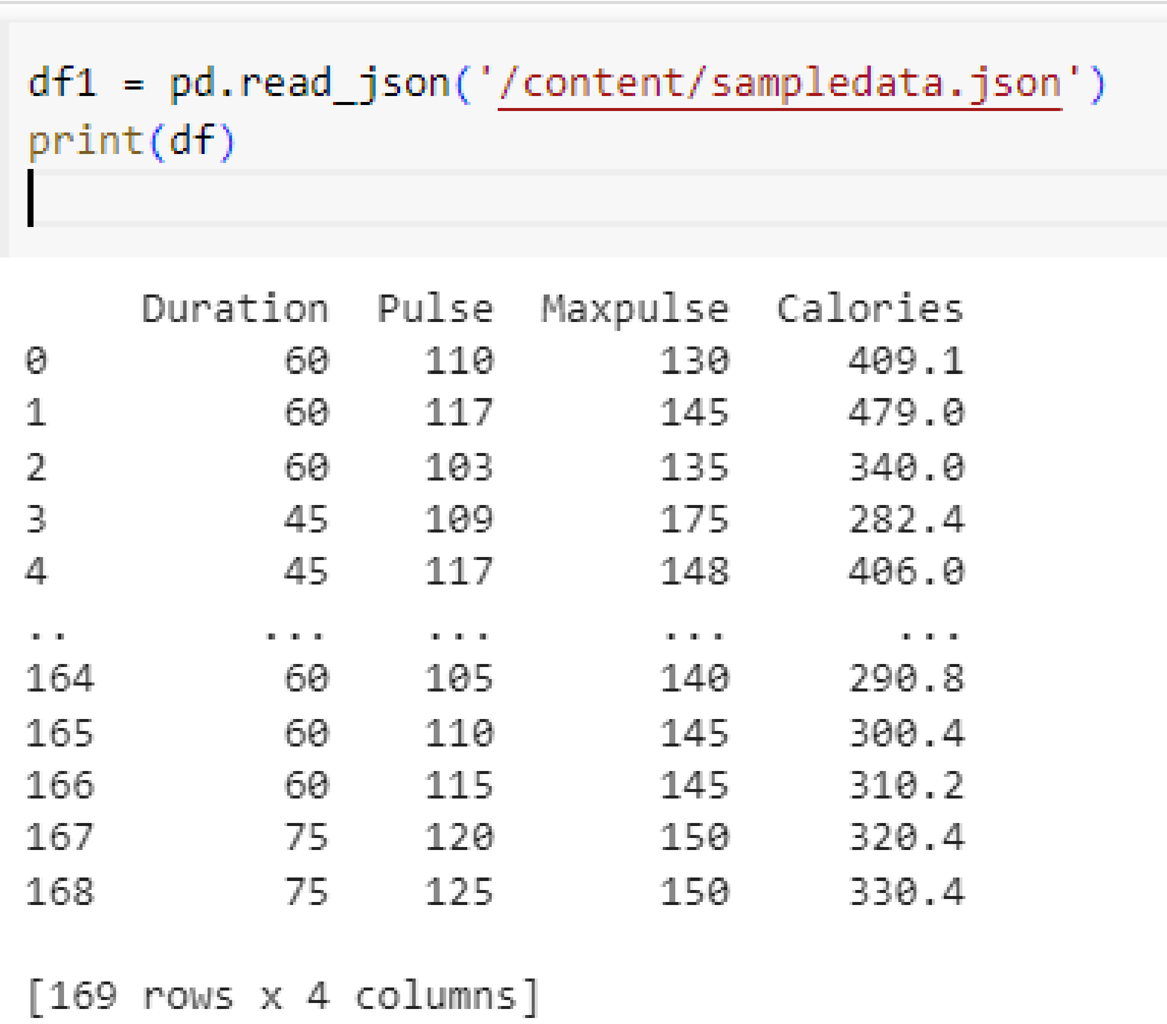
```
#To convert a dict to JSON
stu={'reg':'19MCB1001','name':'rama','age':22,'score':92.3}
stuJ=json.dumps(stu)
stuJ
```

```
'{"reg": "19MCB1001", "name": "rama", "age": 22, "score": 92.3}'
```



```
#write json to a file
fp=open("/content/js1.json","w")
json.dump(stu,fp)
```

# JSON to DataFrame



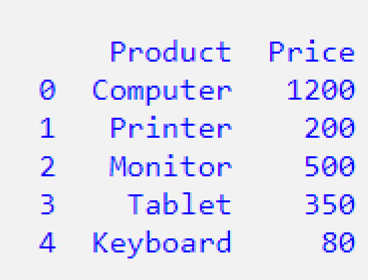
```
df1 = pd.read_json('/content/sampledata.json')
print(df)
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

# Exporting Pandas DataFrame to JSON File

df.to_json(filename.json)

```
import pandas as pd
data1 = {'Product': ['Computer', 'Printer', 'Monitor', 'Tablet', 
'Keyboard'],
         'Price': [1200, 200, 500, 350, 80]
         }

df = pd.DataFrame(data1)
print(df)

df.to_json('/content/export_dataframe.json')
```



```
    Product  Price
0  Computer   1200
1   Printer    200
2   Monitor    500
3    Tablet    350
4  Keyboard     80
```

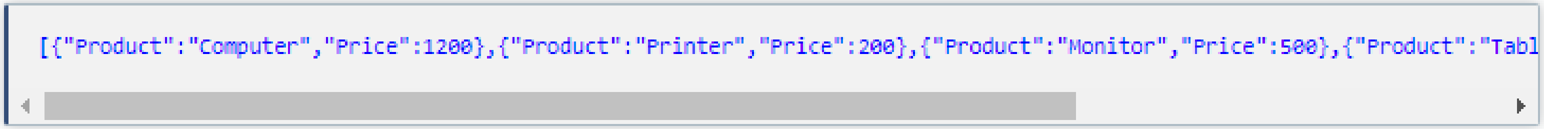
# JSON Formats

There are different ways to format the JSON string. You’ll need to set the **orient** to your desired format. Here are the options:

- split
- records
- index
- values
- table
- columns (the default format)

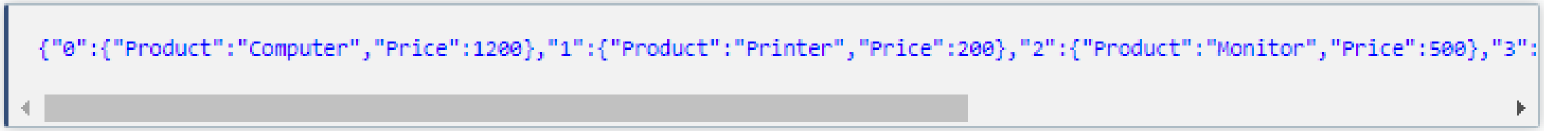
```python
import pandas as pd

data = {'Product': ['Computer', 'Printer', 'Monitor', 'Tablet', 
'Keyboard'],
        'Price': [1200, 200, 500, 350, 80]
        }

df = pd.DataFrame(data)

df.to_json('/content/export_dataframe.json', orient='split')
```

orient='records'



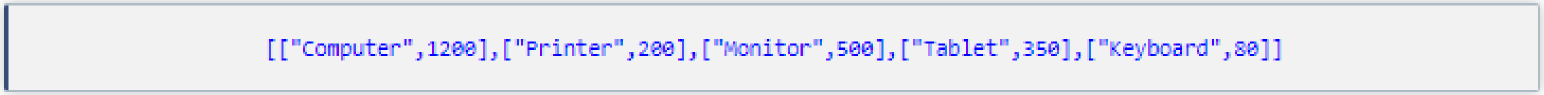
```
[{"Product":"Computer","Price":1200},{"Product":"Printer","Price":200},{"Product":"Monitor","Price":500},{"Product":"Tabl
```

orient='index'



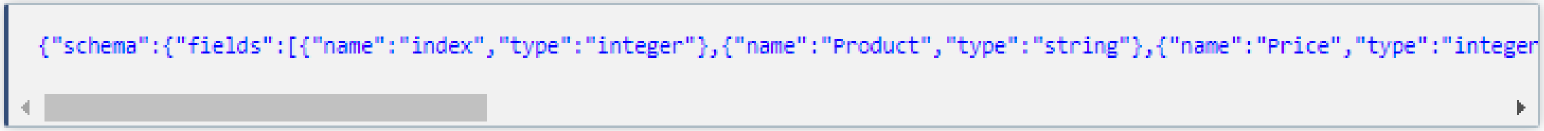
```
{"0":{"Product":"Computer","Price":1200},"1":{"Product":"Printer","Price":200},"2":{"Product":"Monitor","Price":500},"3":
```

orient='values'



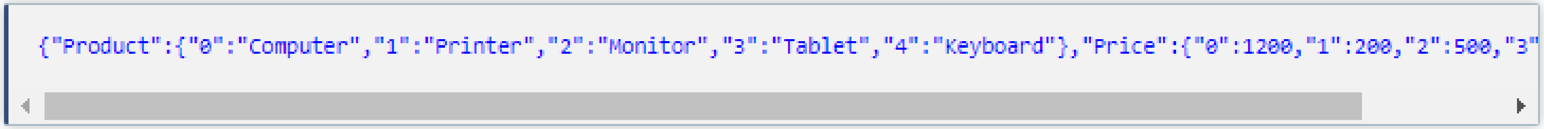
```
[["Computer",1200],["Printer",200],["Monitor",500],["Tablet",350],["Keyboard",80]]
```

orient='table'



```
{"schema":{"fields":[{"name":"index","type":"integer"},{"name":"Product","type":"string"},{"name":"Price","type":"integer
```

orient='columns' (default)



```
{"Product":{"0":"Computer","1":"Printer","2":"Monitor","3":"Tablet","4":"Keyboard"},"Price":{"0":1200,"1":200,"2":500,"3"
```