In [ ]:
```
          "Python Pandas "
```

Pandas is a Python Library .

Pandas are used to analyse data

In [ ]:
```
What is Pandas ?
```

Pandas is a Python Library used for working with data sets.

It has functions for analysing ,cleaning ,exploring and manipulating data.

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

In [ ]:
```
Why Use Pandas?
```

pandas allows us to analyse the big data bases and make some predictions or conclusions based on some statistical theories.

Pandas can clean messy data sets and make them more readble and relevent.

Relevent data is very important in data science

Data Science:--data science is a study of data how we can store and use and analyse data for deriving information from it.

In [ ]:
```
What can Pandas Do?
```

Pandas gives you answers about data:

1.Is there a correlation between two or more columns ?

2.What is average value?

3.Max value?

4.Min value?

Pandas are also able to delete rows that are not relevent or contains wrong values ,like empty or null values .this is called cleaning the data.

In [ ]:
```
"Installation of Pandas"
```

we have already Python and PiP installed on our sysytem,then installation of pandas is very easy.

In [ ]:
```
pip install pandas
```

some other tools like anaconda ,jupyter in this have already built in .

In [ ]:
```
"Import Pandas"
```

Once Pandas is installed, import it in your applications by adding the import keyword:

In [15]:
```
import pandas
```

Now Pandas is imported and ready to use.

In [21]:
```
mydataset={'cars':["Bmw","ford","mercedes"],
           "prices":[100000,200000,300000]
          }
myvar=pandas.DataFrame(mydataset)
print(myvar)
```

```
      cars  prices
0      Bmw  100000
1     ford  200000
2  mercedes  300000
```

In [ ]:
```
Pandas as pd
```

pandas as simply alis with pd ,we can simply to use

alias: InPyhton alias used for give alternative name for reffering to the same thing

In [ ]:
```
import pandas as pd
```

In [ ]:
```
"Checking Pandas Version"
```

The version string is stored under **version** attribute

In [24]:
```
import pandas as pd
print(pd.__version__)
```

```
2.2.2
```

In [ ]:
```
"Pandas Series"
```

In [ ]:
```
What is a Series?
```

A pandas Series is like a column in a table.

It is a one_dimentional array holding data of any type

In [27]:
```
#create a simple Pandas Series from a List:
a=[1,2,4,5]
myvar=pd.Series(a)
print(myvar)
```

```
0    1
1    2
2    4
3    5
dtype: int64
```

In [ ]:
```
"Labels"
```

If nothing else is specified ,the values are labeled with thier index number.first value has index 0,second value has index 1 etc.

this label can be used to access a specified value.

In [34]:
```
a=[1,2,4,5]
myvar=pd.Series(a)
print(myvar[0])
```

```
1
```

In [ ]:
```
"create a Labels"
```

with the "index" arguement ,we can name your own labels.

In [43]:
```
a=[3,56,7,88]
myvar=pd.Series(a,index = ["x","y","z","w"])
print(myvar)
```

```
x     3
y    56
z     7
w    88
dtype: int64
```

when we created that labels we can access the data by reffering to the label

In [46]:
```
print(myvar["x"])
```

```
3
```

In [ ]:
```
"key / Value Object as Series"
```

we can also use a key/value object ,like dictionary ,when creating a Series.

In [49]:
```
calaries={"day1":420,"day2":234,"day3":312}
myvar=pd.Series(calaries)
print(myvar)
```

```
day1    420
day2    234
day3    312
dtype: int64
```

In [ ]:
```
Note: the keys of the dictionary become the labels.
```

To select some of the elements from the dictionary use "index" arguement and specify only items that we want to include in the Series.

In [68]:
```
import pandas as pd
Details={"name":'vasu',"age":21,"gender":'male'}
var=pd.Series(Details,index=["name","age"])
print(var)
```

```
name    vasu
age       21
dtype: object
```

In [ ]:
```
"DataFrames"
```

Data Sets in Pandas are useally multi=dimensional tables ,called DataFrames.

Series is Like a column and a DataFrame is the whole table.

In [74]:
```
#create a DataFrames from two Series.
import pandas as pd

data={
    "details":[12,2,4],
    "duration":[21,24,24]

}
var=pd.DataFrame(data)
print(var)
```

```
   details  duration
0       12        21
1        2        24
2        4        24
```

In [ ]:
```
"DataFrames "
```

In [ ]:
```
What is DataFrame?
```

A pandas DataFrame is a 2 dimensional data structure ,like a 2 dimensional array, or a table with rows and columns.

In [77]:
```
import pandas as pd

data={
    "Details":["vasu",21,"Male"],
    "studies":["datascience",4,2]
}
var=pd.DataFrame(data)
print(var)
```

```
  Details      studies
0    vasu  datascience
1      21            4
2    Male            2
```

In [ ]:
```
"Locate Row"
```

we can see the above table ,the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s).

In [84]:
```
import pandas as pd

data={
    "Details":["vasu",21,"Male"],
    "studies":["datascience",4,2]
}
var=pd.DataFrame(data)
print(var.loc[0])
```

```
Details           vasu
studies    datascience
Name: 0, dtype: object
```

In [ ]:
```
Note: this example returns the Pandas Series.
```

In [88]:
```
import pandas as pd

data={
    "Details":["vasu",21,"Male"],
    "studies":["datascience",4,2]
}
var=pd.DataFrame(data)
print(var.loc[[0,1]]) #return 0 and 1 rows
```

```
  Details      studies
0    vasu  datascience
1      21            4
```

In [ ]:
```
Note: When using [], the result is a Pandas DataFrame.
```

In [ ]:
```

```

In [ ]:
```
"Named Indexes"
```

with the 'index' arguement,you can name your own indexes.

In [91]:
```
import pandas as pd

data={
    "Details":["vasu",21,"Male"],
    "studies":["datascience",4,2]
}
var=pd.DataFrame(data,index=["a","b","c"])
print(var)
```

```
  Details      studies
a    vasu  datascience
b      21            4
c    Male            2
```

In [ ]:
```
"Locate Named Indexes"
```

use the named index in the 'loc' attribute to return the specified row(s).

In [104…
```
import pandas as pd

data={
    "Details":["vasu",21,"Male"],
    "studies":["datascience",4,2]
}
var=pd.DataFrame(data,index=["a","b","c"])
print(var.loc["a"])
```

```
Details           vasu
studies    datascience
Name: a, dtype: object
```

In [102…
```
import pandas as pd

data={
    "Details":["vasu",21,"Male"],
    "studies":["datascience",4,2]
}
var=pd.DataFrame(data,index=["a","b","c"])
print(var.loc[["a","b"]])
```

```
  Details      studies
a    vasu  datascience
b      21            4
```

In [ ]:
```
"Load Files into a DataFrame"
```

If our files data sets are stored in a file ,Pandas can load them into a Data Frame.

Load a comma separated file (CSV file) into a DataFrame:

In [139…
```
import pandas as pd
df=pd.read_csv('Orders.csv')
print(df.head(10))
```

```
         date  product_id  city_id  orders
0  2019-12-10        5628       25       3
1  2018-08-15        3646       14     157
2  2018-10-23        1859       25       1
3  2019-08-17        7292       25       1
4  2019-01-06        4344       25       3
5  2018-08-23        1811       25       4
6  2018-11-21        1282       26       1
7  2019-03-27        5022        2      41
8  2019-06-29        3699        3      15
9  2018-08-30        4373       11       3
```

In [ ]:
```
"Pandas Read CSV"
```

A simple way to store big data sets is to use CSV files (comma seperated files).

CSV file contians plain text and is a well know format that can be read by everyone including Pandas.

In our examples we will be using a CSV file called 'data.csv'.

In [152…
```
import pandas as pd
df=pd.read_csv('Orders.csv')
#print(df.to_string())
```

Tip: use to_string() to print the entire DataFrame.

If we have a large DataFrames with many rows ,Pandas will only return the first 5 rows and last 5 rows

In [155…
```
import pandas as pd
df=pd.read_csv('Orders.csv')
#print(df)
```

```
          date  product_id  city_id  orders
0   2019-12-10        5628       25       3
1   2018-08-15        3646       14     157
2   2018-10-23        1859       25       1
3   2019-08-17        7292       25       1
4   2019-01-06        4344       25       3
..         ...         ...      ...     ...
995 2018-10-08         255       13       1
996 2018-12-06        5521        7       1
997 2019-05-07         487       26      14
998 2019-03-03        1503       21       2
999 2019-10-15        6371        7      22

[1000 rows x 4 columns]
```

In [ ]:
```
"max_rows"
```

the number of rows returned is defined in Pandas option settings.

you can check your systems's maximum rows with the pd.options.display.max_rows' statement.

In [158…
```
print(pd.options.display.max_rows)
```

```
60
```

In many systems the number is 60 ,which means that if the DataFrame contains more than 60 rows ,the print(df) statement will return only the headers and the first and last 5 rows.

we can change the maximum rows number with the same statement.

In [ ]:
```
#pd.options.display.max_rows = 9999
df=pd.read_csv('orders.csv')
print(df.head(10))
```

In [ ]:
```
"Pandas Read JSON"
```

Big data sets are often stored, or extracted as JSON

JSON is a plain text, but has the format of an object,and is well known the world of programming ,including Pandas

In our exapmles we will be using a JSON file called 'data.json'.

In [1]:
```
# Load the JSON file into a DataFrame:
import pandas as pd
df=pd.read_json('Orders.json')
print(df.to_string)
```

```
<bound method DataFrame.to_string of           date  product_id  city_id  orders
0   2019-12-10        5628       25       3
1   2018-08-15        3646       14     157
2   2018-10-23        1859       25       1
3   2019-08-17        7292       25       1
4   2019-01-06        4344       25       3
..         ...         ...      ...     ...
995 2018-10-08         255       13       1
996 2018-12-06        5521        7       1
997 2019-05-07         487       26      14
998 2019-03-03        1503       21       2
999 2019-10-15        6371        7      22

[1000 rows x 4 columns]>
```

Tip: use to_string() to print the entire DataFrame.

In [ ]:
```
"Dictionary as JSON"
```

JSON=Python Dictionary

JSON objects have the same format as python dictionaries

if If your JSON code is not in a file ,but in a Python Dictionary ,you can load it into a DataFrame directly

In [186…
```
import pandas as pd
data = {
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
    "4":45,
    "5":60
  },
  "Pulse":{
    "0":110,
    "1":117,
    "2":103,
    "3":109,
    "4":117,
    "5":102
  },
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135,
    "3":175,
    "4":148,
    "5":127
  },
  "Calories":{
    "0":409,
    "1":479,
    "2":340,
    "3":282,
    "4":406,
    "5":300
  }
}
data=pd.DataFrame(data)
print(data)
```

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130       409
1        60    117       145       479
2        60    103       135       340
3        45    109       175       282
4        45    117       148       406
5        60    102       127       300
```

In [ ]:
```
"Pandas -Analysing DataFrames"
```

In [ ]:
```
"viewing the Data"
```

one of the most used method for getting a quick overviw of the DataFrame,is the head() method.

The "head()" method returns the headers and a specified number of rows ,starting from the top.

In [7]:
```
import pandas as pd
df=pd.read_csv('Orders.csv')
df.head(10)
```

Out[7]:

	date	product_id	city_id	orders
0	2019-12-10	5628	25	3
1	2018-08-15	3646	14	157
2	2018-10-23	1859	25	1
3	2019-08-17	7292	25	1
4	2019-01-06	4344	25	3
5	2018-08-23	1811	25	4
6	2018-11-21	1282	26	1
7	2019-03-27	5022	2	41
8	2019-06-29	3699	3	15
9	2018-08-30	4373	11	3

Note: if the number of rows is not specified, the head() method will return the top 5 rows.

In [5]:
```
# print the first 5 rows of the DataFrame:
df.head()
```

Out[5]:

	date	product_id	city_id	orders
0	2019-12-10	5628	25	3
1	2018-08-15	3646	14	157
2	2018-10-23	1859	25	1
3	2019-08-17	7292	25	1
4	2019-01-06	4344	25	3

In [ ]:
```
"tail"
```

There ia s also a tail() method for viewing the last rows of the DataFrame.

The tail() method returns the headers and a specified number of rows,starting from the bottom

In [10]:
```
df.tail()
```

Out[10]:

	date	product_id	city_id	orders
995	2018-10-08	255	13	1
996	2018-12-06	5521	7	1
997	2019-05-07	487	26	14
998	2019-03-03	1503	21	2
999	2019-10-15	6371	7	22

In [ ]:
```
"Info About the Data"
```

The DataFrame object has a method called info() ,that gives you more information about the data set.

In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   date        1000 non-null   object
 1   product_id  1000 non-null   int64 
 2   city_id     1000 non-null   int64 
 3   orders      1000 non-null   int64 
dtypes: int64(3), object(1)
memory usage: 31.4+ KB
```

In [ ]:
```
"Result Explanation"
```

The result us there are 169 rows and 4 columns

In [ ]:
```
RangeIndex: 1000 entries, 0 to 999

Data columns (total 4 columns):
```

And the name of each column, with the data type:

In [ ]:
```
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   date        1000 non-null   object
 1   product_id  1000 non-null   int64 
 2   city_id     1000 non-null   int64 
 3   orders      1000 non-null   int64 
```

In [ ]:
```
"null values"
```

the "info()" function tells us how many non-null values there are present in each column and in our data set it seems like there are 1000 of 1000 .here we don't have any null values.

whenever analysing our big data null values may have cause to disturb the analysing process .so, before analysing we have to drop that null values is called "cleaning data" .

In [ ]:
```
"Pandas --- Cleaning Data"
```

In [ ]:
```
"Data Cleaning "
```

Data cleaning means filtering the bad or any null values or any uneccessary data will occurs while analysing. Bad means: 1.Empty cells 2.Data in wrong format 3.Wrong data 4.Duplicates

In [ ]:
```
Our Data Set
```

In [ ]:
```
      Duration          Date  Pulse  Maxpulse  Calories
  0         60  '2020/12/01'    110       130     409.1
  1         60  '2020/12/02'    117       145     479.0
  2         60  '2020/12/03'    103       135     340.0
  3         45  '2020/12/04'    109       175     282.4
  4         45  '2020/12/05'    117       148     406.0
  5         60  '2020/12/06'    102       127     300.0
  6         60  '2020/12/07'    110       136     374.0
  # 7        450  '2020/12/08'    104       134     253.3
  8         30  '2020/12/09'    109       133     195.1
  9         60  '2020/12/10'     98       124     269.0
  10        60  '2020/12/11'    103       147     329.3
  # 11        60  '2020/12/12'    100       120     250.7
  # 12        60  '2020/12/12'    100       120     250.7
  13        60  '2020/12/13'    106       128     345.3
  14        60  '2020/12/14'    104       132     379.3
  15        60  '2020/12/15'     98       123     275.0
  16        60  '2020/12/16'     98       120     215.2
  17        60  '2020/12/17'    100       120     300.0
  18        45  '2020/12/18'     90       112       NaN
  19        60  '2020/12/19'    103       123     323.0
  20        45  '2020/12/20'     97       125     243.0
  21        60  '2020/12/21'    108       131     364.2
  22        45           NaN    100       119     282.0
  23        60  '2020/12/23'    130       101     300.0
  24        45  '2020/12/24'    105       132     246.0
  25        60  '2020/12/25'    102       126     334.5
  26        60    2020/12/26    100       120     250.0
  27        60  '2020/12/27'     92       118     241.0
  # 28        60  '2020/12/28'    103       132       NaN
  29        60  '2020/12/29'    100       132     280.0
  30        60  '2020/12/30'    102       129     380.3
  31        60  '2020/12/31'     92       115     243.0
```

In [ ]:
```
The data set contains some empty cells ("Date" in row 22, and "Calories" in row 18 and 28).

The data set contains wrong format ("Date" in row 26).

The data set contains wrong data ("Duration" in row 7).

The data set contains duplicates (row 11 and 12).
```

In [ ]:
```
"Pandas -Cleaning Empty Cells"
```

Empty calls--> Empty cells can potentially give you a wrong result when you anaysing data.

In [ ]:
```
"Remove Rows"
```

One way to deal with empty cells is to remove rows that contain empty cells.

this small removing rows doesn't show any big impact on the result.

In [80]:
```
import pandas as pd

df=pd.read_csv("bigmart_data.csv")
# new_df=df.dropna()
# print(new_df)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8522 entries, 0 to 8521
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   FDA15              8522 non-null   object
 1   9.3                7059 non-null   float64
 2   Low Fat            8522 non-null   object
 3   0.016047301        8522 non-null   float64
 4   Dairy              8522 non-null   object
 5   249.8092           8522 non-null   float64
 6   OUT049             8522 non-null   object
 7   1999               8522 non-null   int64
 8   Medium             6112 non-null   object
 9   Tier 1             8522 non-null   object
 10  Supermarket Type1  8522 non-null   object
 11  3735.138           8522 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.1+ KB
```

In [92]:
```
import pandas as pd

df=pd.read_csv("bigmart_data.csv")
new_df=df.dropna()
new_df
```

Out[92]:

	FDA15	9.3	Low Fat	0.016047301	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.138
0	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
1	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052
4	FDP36	10.395	Regular	0.000000	Baking Goods	51.4008	OUT018	2009	Medium	Tier 3	Supermarket Type2	556.6088
5	FDO10	13.650	Regular	0.012741	Snack Foods	57.6588	OUT013	1987	High	Tier 3	Supermarket Type1	343.5528
...	...	...	...	...	...	...	...	...	...	...	...	...
8516	FDF53	20.750	reg	0.083607	Frozen Foods	178.8318	OUT046	1997	Small	Tier 1	Supermarket Type1	3608.6360
8517	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218	OUT013	1987	High	Tier 3	Supermarket Type1	2778.3834
8519	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	OUT035	2004	Small	Tier 2	Supermarket Type1	1193.1136
8520	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332	OUT018	2009	Medium	Tier 3	Supermarket Type2	1845.5976
8521	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670	OUT046	1997	Small	Tier 1	Supermarket Type1	765.6700

4649 rows × 12 columns

In [94]:
```
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4649 entries, 0 to 8521
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   FDA15              4649 non-null   object 
 1   9.3                4649 non-null   float64
 2   Low Fat            4649 non-null   object 
 3   0.016047301        4649 non-null   float64
 4   Dairy              4649 non-null   object 
 5   249.8092           4649 non-null   float64
 6   OUT049             4649 non-null   object 
 7   1999               4649 non-null   int64  
 8   Medium             4649 non-null   object 
 9   Tier 1             4649 non-null   object 
 10  Supermarket Type1  4649 non-null   object 
 11  3735.138           4649 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 472.2+ KB
```

Note: By default, the dropna() method returns a new DataFrame, and will not change the original.

If we want to change the original DataFrame,use the inplace=True arguement.

In [102…
```
# Remove all rows with NULL values:
df=pd.read_csv('bigmart_data.csv')
df.dropna(inplace=True)
df
```

Out[102…

	FDA15	9.3	Low Fat	0.016047301	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.138
0	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
1	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052
4	FDP36	10.395	Regular	0.000000	Baking Goods	51.4008	OUT018	2009	Medium	Tier 3	Supermarket Type2	556.6088
5	FDO10	13.650	Regular	0.012741	Snack Foods	57.6588	OUT013	1987	High	Tier 3	Supermarket Type1	343.5528
...	...	...	...	...	...	...	...	...	...	...	...	...
8516	FDF53	20.750	reg	0.083607	Frozen Foods	178.8318	OUT046	1997	Small	Tier 1	Supermarket Type1	3608.6360
8517	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218	OUT013	1987	High	Tier 3	Supermarket Type1	2778.3834
8519	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	OUT035	2004	Small	Tier 2	Supermarket Type1	1193.1136
8520	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332	OUT018	2009	Medium	Tier 3	Supermarket Type2	1845.5976
8521	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670	OUT046	1997	Small	Tier 1	Supermarket Type1	765.6700

4649 rows × 12 columns

Note: Now,the dropna(inplace=True) will NOT return a new DataFrame ,but it will removes all rows containing NULL values from the original DataFrame.

In [ ]:
```
"Replace Empty Values"
```

Another way of dealing with empty cells is to insert a new value instead.

This way we do not delete any rows because of just some empty cells.

The 'fillna()' method allows us to replace empty cells with a value.

In [108…
```
# Replace Null values with any number 120:
df=pd.read_csv('data.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Duration  169 non-null    int64  
 1   Pulse     169 non-null    int64  
 2   Maxpulse  169 non-null    int64  
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
```

we have five null values in calarioes so,we can replace the null value with any value

In [113…
```
df.fillna(213,inplace=True)
df
```

Out[113…

	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	300.0
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

169 rows × 4 columns

In [115…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Duration  169 non-null    int64  
 1   Pulse     169 non-null    int64  
 2   Maxpulse  169 non-null    int64  
 3   Calories  169 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
```

here successfully filled empty to values

In [ ]:
```
"Replace Only For Specified Columns"
```

the Above example replaces all empty cells in the whole DataFrame.

To only replace empty values for one column ,specify the column name for the DataFrame

In [131…
```
df=pd.read_csv('data.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Duration  169 non-null    int64  
 1   Pulse     169 non-null    int64  
 2   Maxpulse  169 non-null    int64  
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
```

In [139…
```
df["Calories"].fillna(123,inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Duration  169 non-null    int64  
 1   Pulse     169 non-null    int64  
 2   Maxpulse  169 non-null    int64  
 3   Calories  169 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
```

```
C:\Users\gadam\AppData\Local\Temp\ipykernel_5188\2633696100.py:1: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on w
hich we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)'
or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df["Calories"].fillna(123,inplace=True)
```

In [179…
```
'df.method({col: value}, inplace=True)'
```

Out[179…
```
'df.method({col: value}, inplace=True)'
```

In [195…
```
df=pd.read_csv('data.csv')
df.fillna({"Calories":125},inplace=True)
df.info(["Calories"])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Duration  169 non-null    int64  
 1   Pulse     169 non-null    int64  
 2   Maxpulse  169 non-null    int64  
 3   Calories  169 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
```

In [ ]:
```
"Replace Using Mean ,Median, or Mode"
```

A common way to replace empty cells ,is to calculate the mean ,median or mode value of the column

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

In [198…
```
df.describe()
```

Out[198…

	Duration	Pulse	Maxpulse	Calories
count	169.000000	169.000000	169.000000	169.000000
mean	63.846154	107.461538	134.047337	368.370414
std	42.299949	14.510259	16.450434	265.824988
min	15.000000	80.000000	100.000000	50.300000
25%	45.000000	100.000000	124.000000	250.000000
50%	60.000000	105.000000	131.000000	310.200000
75%	60.000000	111.000000	141.000000	384.000000
max	300.000000	159.000000	184.000000	1860.400000

In [ ]:
```
Calculate the MEAN, and replace any empty values with it:
```

In [224…
```
import pandas as  pd
df=pd.read_csv('data.csv')
x=df["Calories"].mean()
#df["Calories"].fillna(x,inplace=True)
df.fillna({"Calories":x},inplace=True # it's version difference
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     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

In [226…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  169 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
```

Mean = the average value (the sum of all values divided by number of values).

In [231…
```
# Calculate the MEDIAN, and replace any empty values with it:

import pandas as pd

df=pd.read_csv("data.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
```

In [237…
```
import pandas as pd

df=pd.read_csv("data.csv")
x=df["Calories"].median()
df.fillna({"Calories":x},inplace=True)
print(df)
print(df.info())
```

```
     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     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  169 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

Median = the value in the middle, after you have sorted all values ascending.

In [244…
```
# Calculate the MODE, and replace any empty values with it:
df=pd.read_csv("data.csv")
x=df["Calories"].mode()[0]
df.fillna({"Calories":x},inplace=True)
print(df)
print(df.info())
```

```
     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     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  169 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

Mode = the value that appears most frequently.

In [ ]:
```
"Pandas -Cleaning Data of Wrong Format"
```

Cells with data of wrong formatt can amke it difficult,or even impossible ,to analyse data.

To fix this problem ,we have two solution.

1.delete that rows of wrong formatted

2.replace them with same format values

In [ ]:
```
"Convert into Correct Format"
```

In [254…
```
df=pd.read_csv('data.1.csv')
df.head(13)
```

Out[254…

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	NaN	340.0
3	45	234	175	282.4
4	45	117	148	406.0
5	60	102	127	300.0
6	60	110	136	374.0
7	45	104	134	253.3
8	30	109	133	195.1
9	60	98	124	269.0
10	60	103	231'	329.3
11	60	100	120	250.7
12	60	106	128	345.3

In this Data Frame,we have two cells with the wrong format.check out row 2 and 10 the maxpulse column should be int that represents a integer value

In [259…
```
  0         60  '2020/12/01'    110       130     409.1
  1         60  '2020/12/02'    117       145     479.0
  2         60  '2020/12/03'    103       135     340.0
  3         45  '2020/12/04'    109       175     282.4
  4         45  '2020/12/05'    117       148     406.0
  5         60  '2020/12/06'    102       127     300.0
  6         60  '2020/12/07'    110       136     374.0
  7        450  '2020/12/08'    104       134     253.3
  8         30  '2020/12/09'    109       133     195.1
  9         60  '2020/12/10'     98       124     269.0
```

```
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaT    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60  '2020/12/26'    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0, 0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaT    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60  '2020/12/26'    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0]
```

```
---------------------------------------------------------------
ValueError                    Traceback (most recent call last)
Cell In[259], line 2
      1 df=pd.read_csv("data.1.csv")
----> 2 df["Date"]=['2320-24-23']

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:4311, in DataFrame.__setitem__(self, key, value)
   4308     self._setitem_array([key], value)
   4309 else:
   4310     # set column
-> 4311     self._set_item(key, value)

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:4524, in DataFrame._set_item(self, key, value)
   4514 def _set_item(self, key, value) -> None:
   4515     """
   4516     Add series to DataFrame in specified column.
   4517
   (...)
   4522     ensure homogeneity.
   4523     """
-> 4524     value, refs = self._sanitize_column(value)
   4526     if (
   4527         key in self.columns
   4528         and value.ndim == 1
   4529         and not isinstance(value.dtype, ExtensionDtype)
   4530     ):
   4531         # broadcast across multiple columns if necessary
   4532         if not self.columns.is_unique or isinstance(self.columns, MultiIndex):

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:5266, in DataFrame._sanitize_column(self, value)
   5263     return _reindex_for_setitem(value, self.index)
   5265 if is_list_like(value):
-> 5266     com.require_length_match(value, self.index)
   5267 arr = sanitize_array(value, self.index, copy=True, allow_2d=True)
   5268 if (
   5269     isinstance(value, Index)
   5270     and value.dtype == "object"
   (...)
   5273     # TODO: Remove kludge in sanitize_array for string mode when enforcing
   5274     # this deprecation

File ~\anaconda3\Lib\site-packages\pandas\core\common.py:573, in require_length_match(data, index)
    569 """
    570 Check the length of data matches the length of the index.
    571 """
    572 if len(data) != len(index):
--> 573     raise ValueError(
    574         "Length of values "
    575         f"({len(data)}) "
    576         "does not match length of index "
    577         f"({len(index)})"
    578     )

ValueError: Length of values (1) does not match length of index (169)
```

In [23]:

```
# Let's try to create cells in the 'Date' column into dates.

# Pandas has a 'to_datetime()' method for this:
import pandas as pd
df=pd.read_csv('data.vc.csv')
df
```

Out[23]:

	Duration	Pulse	Maxpulse	Calories	date
0	60.0	110.0	130	409.1	2024-08-09
1	60.0	117.0	145	479.0	2024-08-10
2	60.0	103.0	NaN	340.0	2024-08-11
3	45.0	234.0	175	282.4	Null
4	45.0	117.0	148	406.0	2024-08-13
...	...	...	...	...	...
164	NaN	NaN	NaN	NaN	NaN
165	NaN	NaN	NaN	NaN	NaN
166	NaN	NaN	NaN	NaN	NaN
167	75.0	120.0	150	320.4	NaN
168	75.0	125.0	150	330.4	NaN

169 rows × 5 columns

In [31]:
```
df.dropna(inplace=True)
df
```

Out[31]:

	Duration	Pulse	Maxpulse	Calories	date
0	60.0	110.0	130	409.1	2024-08-09
1	60.0	117.0	145	479.0	2024-08-10
3	45.0	234.0	175	282.4	Null
4	45.0	117.0	148	406.0	2024-08-13
5	60.0	102.0	127	300.0	2024-08-14
6	60.0	110.0	136	374.0	2024-08-15
7	45.0	104.0	134	253.3	2024-08-16
8	30.0	109.0	133	195.1	2024-08-17
9	60.0	98.0	124	269.0	2024-08-18
10	60.0	103.0	231'	329.3	2024-08-19
11	60.0	100.0	120	250.7	2024-08-20
12	60.0	106.0	128	345.3	2024-08-21
13	60.0	104.0	132	379.3	2024-08-22
14	60.0	98.0	123	275.0	2024-08-23
15	60.0	98.0	120	215.2	2024-08-24
16	60.0	100.0	120	300.0	2024-08-25
18	60.0	103.0	123	323.0	2024-08-27
19	45.0	97.0	125	243.0	2024-08-28
20	60.0	108.0	131	364.2	2024-08-29
21	45.0	100.0	119	282.0	2024-08-30
22	60.0	130.0	101	300.0	2024-08-31
23	45.0	105.0	132	246.0	2024-08-32
24	60.0	102.0	126	334.5	2024-08-33
25	60.0	100.0	120	250.0	2024-08-34
26	60.0	92.0	118	241.0	2024-08-35
28	60.0	100.0	132	280.0	2024-08-37

In [35]:
```
df=pd.read_csv('dat2.csv')
df['date']=pd.to_datetime(df['date'])
df
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[35], line 2
      1 df=pd.read_csv('dat2.csv')
----> 2 df['date']=pd.to_datetime(df['date'])
      3 df

File ~\anaconda3\Lib\site-packages\pandas\core\tools\datetimes.py:1067, in to_datetime(arg, errors, dayfirst, ye
arfirst, utc, format, exact, unit, infer_datetime_format, origin, cache)
   1065         result = arg.map(cache_array)
   1066     else:
-> 1067         values = convert_listlike(arg._values, format)
   1068         result = arg._constructor(values, index=arg.index, name=arg.name)
   1069 elif isinstance(arg, (ABCDataFrame, abc.MutableMapping)):

File ~\anaconda3\Lib\site-packages\pandas\core\tools\datetimes.py:433, in _convert_listlike_datetimes(arg, forma
t, name, utc, unit, errors, dayfirst, yearfirst, exact)
    431 # `format` could be inferred, or user didn't ask for mixed-format parsing.
    432 if format is not None and format != "mixed":
--> 433     return _array_strptime_with_fallback(arg, name, utc, format, exact, errors)
    435 result, tz_parsed = objects_to_datetime64(
    436     arg,
    437     dayfirst=dayfirst,
   (...)
    441     allow_object=True,
    442 )
    444 if tz_parsed is not None:
    445     # We can take a shortcut since the datetime64 numpy array
    446     # is in UTC

File ~\anaconda3\Lib\site-packages\pandas\core\tools\datetimes.py:467, in _array_strptime_with_fallback(arg, nam
e, utc, fmt, exact, errors)
    456 def _array_strptime_with_fallback(
    457     arg,
    458     name,
   (...)
    462     errors: str,
    463 ) -> Index:
    464     """
    465     Call array_strptime, with fallback behavior depending on 'errors'.
    466     """
--> 467     result, tz_out = array_strptime(arg, fmt, exact=exact, errors=errors, utc=utc)
    468     if tz_out is not None:
    469         unit = np.datetime_data(result.dtype)[0]

File strptime.pyx:501, in pandas._libs.tslibs.strptime.array_strptime()

File strptime.pyx:451, in pandas._libs.tslibs.strptime.array_strptime()

File strptime.pyx:587, in pandas._libs.tslibs.strptime._parse_with_format()

ValueError: unconverted data remains when parsing with format "%Y-%m-%d": "2", at position 23. You might want to
try:
    - passing `format` if your strings have a consistent format;
    - passing `format='ISO8601'` if your strings are all ISO8601 but not necessarily in exactly the same format;
    - passing `format='mixed'`, and the format will be inferred for each element individually. You might want to
use `dayfirst` alongside this.
```

In [ ]:

In [73]:
```
x={ 'name':['vasu','ravi','sai','mani','bhavya','bunny'],
   'age':[10,19,19,23,23,43],
   'DoB':['2004/12/21','2004/04/04','2005/03/03','2003/03/03','2003/02/03','2003/02/03']
  }
df=pd.DataFrame(x)
df['DoB'][2]='Nan'
df
```

```
C:\Users\gadam\AppData\Local\Temp\ipykernel_12248\2265255242.py:6: FutureWarning: ChainedAssignmentError: behavi
our will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  df['DoB'][2]='Nan'
C:\Users\gadam\AppData\Local\Temp\ipykernel_12248\2265255242.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df['DoB'][2]='Nan'
```

Out[73]:

	name	age	DoB
0	vasu	10	2004/12/21
1	ravi	19	2004/04/04
2	sai	19	Nan
3	mani	23	2003/03/03
4	bhavya	23	2003/02/03
5	bunny	43	2003/02/03

In [113…
```
df.loc[6,'DoB']=20020909 #advance vesrion format
df
```

```
C:\Users\gadam\AppData\Local\Temp\ipykernel_12248\3188170503.py:1: FutureWarning: Setting an item of incompatibl
e dtype is deprecated and will raise an error in a future version of pandas. Value '20020909' has dtype incompat
ible with datetime64[ns], please explicitly cast to a compatible dtype first.
  df.loc[6,'DoB']=20020909 #advance vesrion format
```

Out[113…

	name	age	DoB
0	vasu	10.0	2004-12-21 00:00:00
1	ravi	19.0	2004-04-04 00:00:00
2	sai	19.0	NaT
3	mani	23.0	2003-03-03 00:00:00
4	bhavya	23.0	2003-02-03 00:00:00
5	bunny	43.0	2003-02-03 00:00:00
6	NaN	NaN	20020909

In [ ]:
```
Handling Missing Values with errors='coerce'
```

In [115…
```
df['DoB']=pd.to_datetime(df['DoB'],format='mixed',errors='coerce')
df
```

Out[115…

	name	age	DoB
0	vasu	10.0	2004-12-21 00:00:00.000000000
1	ravi	19.0	2004-04-04 00:00:00.000000000
2	sai	19.0	NaT
3	mani	23.0	2003-03-03 00:00:00.000000000
4	bhavya	23.0	2003-02-03 00:00:00.000000000
5	bunny	43.0	2003-02-03 00:00:00.000000000
6	NaN	NaN	1970-01-01 00:00:00.020020909

In [ ]:
```
Solution 2: Specifying the Date Format
```

In [104…
```
df['DoB']=pd.to_datetime(df['DoB'],format='%Y/%m/%d',errors='coerce')
df
```

Out[104…

	name	age	DoB
0	vasu	10.0	2004-12-21 00:00:00.000000000
1	ravi	19.0	2004-04-04 00:00:00.000000000
2	sai	19.0	NaT
3	mani	23.0	2003-03-03 00:00:00.000000000
4	bhavya	23.0	2003-02-03 00:00:00.000000000
5	bunny	43.0	2003-02-03 00:00:00.000000000
6	NaN	NaN	1970-01-01 00:00:00.020020909

In [106…

```
df['DoB'] = pd.to_datetime(df['DoB'], format='mixed', errors='coerce')
df
```

Out[106…

	name	age	DoB
0	vasu	10.0	2004-12-21 00:00:00.000000000
1	ravi	19.0	2004-04-04 00:00:00.000000000
2	sai	19.0	NaT
3	mani	23.0	2003-03-03 00:00:00.000000000
4	bhavya	23.0	2003-02-03 00:00:00.000000000
5	bunny	43.0	2003-02-03 00:00:00.000000000
6	NaN	NaN	1970-01-01 00:00:00.020020909

In [108…

```
df['DoB'] = pd.to_datetime(df['DoB'], format='%Y/%m/%d', errors='coerce')
df
```

Out[108…

	name	age	DoB
0	vasu	10.0	2004-12-21 00:00:00.000000000
1	ravi	19.0	2004-04-04 00:00:00.000000000
2	sai	19.0	NaT
3	mani	23.0	2003-03-03 00:00:00.000000000
4	bhavya	23.0	2003-02-03 00:00:00.000000000
5	bunny	43.0	2003-02-03 00:00:00.000000000
6	NaN	NaN	1970-01-01 00:00:00.020020909

In [ ]:

```
"Removing Rows"
```

The result from the converting in the example above gave us a NaT values,which can be handled as Null value ,and we can remove the row using dropna() method.

In [120…

```
df.dropna(subset=['DoB'],inplace=True)
df
```

Out[120…

	name	age	DoB
0	vasu	10.0	2004-12-21 00:00:00.000000000
1	ravi	19.0	2004-04-04 00:00:00.000000000
3	mani	23.0	2003-03-03 00:00:00.000000000
4	bhavya	23.0	2003-02-03 00:00:00.000000000
5	bunny	43.0	2003-02-03 00:00:00.000000000
6	NaN	NaN	1970-01-01 00:00:00.020020909

In [ ]:

```
"Fixing Wrong Data"
```

"Wrong data" does not have to be "empty cells " or "wrong format" ,it can just be wrong ,like if someone registered '199' instead of 1.199

Sometimes you can spot wrong data by looking at the data set,because we have an expectation of what it should be.

If you take a look at our data set, you can see that in row 7, the duration is 450, but for all the other rows the duration is between 30 and 60.

It doesn't have to be wrong, but taking in consideration that this is the data set of someone's workout sessions, we conclude with the fact

that this person did not work out in 450 minutes.

In [139…
```
data={"id":[1,2,3,4,5],
      "name":["vasu","mani","sai","bhavya","vamsi"],
      "hours":[12,24,24,234,23]}
df=pd.DataFrame(data)
df
```

Out[139…

	id	name	hours
0	1	vasu	12
1	2	mani	24
2	3	sai	24
3	4	bhavya	234
4	5	vamsi	23

In [164…
```
df.loc[2,'Duration']=15
df
```

Out[164…

	Duration	Pulse	Maxpulse	Calories	date
0	60.0	110.0	130	409.1	2024-08-09
1	60.0	117.0	145	479.0	2024-08-10
2	15.0	103.0	34	340.0	2024-08-11
3	45.0	234.0	175	282.4	NaN
4	45.0	117.0	148	406.0	2024-08-13
...	...	...	...	...	...
164	NaN	NaN	NaN	NaN	NaN
165	NaN	NaN	NaN	NaN	NaN
166	NaN	NaN	NaN	NaN	NaN
167	75.0	120.0	150	320.4	NaN
168	75.0	125.0	150	330.4	NaN

169 rows × 5 columns

In [158…
```
df=pd.read_csv('dat2.csv')
df
```

Out[158…

	Duration	Pulse	Maxpulse	Calories	date
0	60.0	110.0	130	409.1	2024-08-09
1	60.0	117.0	145	479.0	2024-08-10
2	60.0	103.0	NaN	340.0	2024-08-11
3	45.0	234.0	175	282.4	NaN
4	45.0	117.0	148	406.0	2024-08-13
...	...	...	...	...	...
164	NaN	NaN	NaN	NaN	NaN
165	NaN	NaN	NaN	NaN	NaN
166	NaN	NaN	NaN	NaN	NaN
167	75.0	120.0	150	320.4	NaN
168	75.0	125.0	150	330.4	NaN

169 rows × 5 columns

In [162…
```
df.loc[2,"Maxpulse"]=34
df
```

Out[162…

	Duration	Pulse	Maxpulse	Calories	date
0	60.0	110.0	130	409.1	2024-08-09
1	60.0	117.0	145	479.0	2024-08-10
2	60.0	103.0	34	340.0	2024-08-11
3	45.0	234.0	175	282.4	NaN
4	45.0	117.0	148	406.0	2024-08-13
...	...	...	...	...	...
164	NaN	NaN	NaN	NaN	NaN
165	NaN	NaN	NaN	NaN	NaN
166	NaN	NaN	NaN	NaN	NaN
167	75.0	120.0	150	320.4	NaN
168	75.0	125.0	150	330.4	NaN

169 rows × 5 columns

for small data sets you can might be able to replace the wrong data one by one,bt not for big data sets.

To replace wrong data for larger data data sets you can create some rules,

e.g. set some boundaries for legal values, and replace any values that are outside of the boundaries.

In [190…

```
for x in df.index:
    if df.loc[x,"Duration"]>50:
        df.loc[x,"Duration"]=120

df
```

Out[190…

	Duration	Pulse	Maxpulse	Calories	date
0	120.0	110.0	130	409.1	2024-08-09
1	120.0	117.0	145	479.0	2024-08-10
2	15.0	103.0	34	340.0	2024-08-11
3	45.0	234.0	175	282.4	NaN
4	45.0	117.0	148	406.0	2024-08-13
...	...	...	...	...	...
162	NaN	NaN	NaN	NaN	NaN
163	NaN	NaN	NaN	NaN	NaN
164	NaN	NaN	NaN	NaN	NaN
165	NaN	NaN	NaN	NaN	NaN
166	NaN	NaN	NaN	NaN	NaN

167 rows × 5 columns

In [ ]:

```
"Removing Rows"
```

Another way of handling wromg data is to remove the rows that contains wrong data.

This way we do not need to find out what to replace them with and there is a good chance you do not need them to do your analysis

In [194…

```
for i in df.index:
    if df.loc[i,"Duration"]>50:
        df.drop(i,inplace=True)

df
```

Out[194…

	Duration	Pulse	Maxpulse	Calories	date
2	15.0	103.0	34	340.0	2024-08-11
3	45.0	234.0	175	282.4	NaN
4	45.0	117.0	148	406.0	2024-08-13
7	45.0	104.0	134	253.3	2024-08-16
8	30.0	109.0	133	195.1	2024-08-17
...	...	...	...	...	...
161	NaN	NaN	NaN	NaN	NaN
162	NaN	NaN	NaN	NaN	NaN
163	NaN	NaN	NaN	NaN	NaN
164	NaN	NaN	NaN	NaN	NaN
165	NaN	NaN	NaN	NaN	NaN

139 rows × 5 columns

In [ ]:
```
"Pandas Removing Duplicates"
```

In [ ]:
```
"Discovering duplicates"
```

Duplicate rows are rows that have been registered more then one time

In [6]:
```
import pandas as pd
df=pd.read_csv('dat2.csv')
df
```

Out[6]:

	Duration	Pulse	Maxpulse	Calories	date
0	60.0	110.0	130	409.1	2024-08-09
1	60.0	117.0	145	479.0	2024-08-10
2	60.0	103.0	NaN	340.0	2024-08-11
3	45.0	234.0	175	282.4	NaN
4	45.0	117.0	148	406.0	2024-08-13
...	...	...	...	...	...
164	NaN	NaN	NaN	NaN	NaN
165	NaN	NaN	NaN	NaN	NaN
166	NaN	NaN	NaN	NaN	NaN
167	75.0	120.0	150	320.4	NaN
168	75.0	125.0	150	330.4	NaN

169 rows × 5 columns

In [ ]:
```
```

In [8]:
```
data={"Duration":[10,20,20,40],
     "Date":['2029/02/09','2029/04/04','2029/04/04','2039/09/08'],
     "Pulse":[120,129,129,140],
     "MaxPulse":[129,121,121,127],
     "Calories":[200.9,900.2,900.2,727.2]}
df=pd.DataFrame(data)
df
```

Out[8]:

	Duration	Date	Pulse	MaxPulse	Calories
0	10	2029/02/09	120	129	200.9
1	20	2029/04/04	129	121	900.2
2	20	2029/04/04	129	121	900.2
3	40	2039/09/08	140	127	727.2

in this we have two same data rows at 2 and 3 are duplicates

To discover duplicates,we can use the duplicated() method.

The duplicated() method returns a boolean values for each row

In [12]:
```
# Returns True for every row that is a duplicate, otherwise False:
df.duplicated()
```

Out[12]:
```
0    False
1    False
2     True
3    False
dtype: bool
```

In [ ]:
```
"Removing Duplicates"
```

To removing duplicates ,use the drop_duplicates() method

In [19]:
```
df.drop_duplicates(inplace=True)
df
```

Out[19]:

	Duration	Date	Pulse	MaxPulse	Calories
0	10	2029/02/09	120	129	200.9
1	20	2029/04/04	129	121	900.2
3	40	2039/09/08	140	127	727.2

Remember: The (inplace = True) will make sure that the method does NOT return a new DataFrame, but it will remove all duplicates from the original DataFrame.

In [ ]:
```
"Pandas -Data Correlations"
```

In [ ]:
```
A great aspect of the pandas module is the 'corr()' method.

The corr() method calculates the relationship between each column in your data set.

The examples in this page uses a CSV file called :'dat2.csv'
```

In [24]:
```
import pandas as pd
df=pd.read_csv('data.csv')
df
```

Out[24]:

	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	300.0
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

169 rows × 4 columns

In [35]:
```
df.corr()
```

Out[35]:

	Duration	Pulse	Maxpulse	Calories
Duration	1.000000	-0.155408	0.009403	0.922717
Pulse	-0.155408	1.000000	0.786535	0.025121
Maxpulse	0.009403	0.786535	1.000000	0.203813
Calories	0.922717	0.025121	0.203813	1.000000

Note: The corr() method ignores the"not numeric" columns

In [ ]:
```
"explanation"
```

In [ ]:
```
The result of corr() method is a table with a lot of numbers that represent how well the relationship is betwee

The number varies from -1 to 1.
```

```
1 means that there is 1 to 1 relationship(a perfect correlation),and for this data set ,each time a value went
0.9 is also a good relationship ,and if you increase one value,the other will probably increase as well.
-0.9 would be just as good relationship as 0.9 ,but if you increase one value,the other will probably go down.
0.2 means NOT a good relationship ,meaning that if one value goes up does not mean that the other will.
```

In [ ]:
```
"good correlation"
```

It depends on our useage,we say their may be like atleast 0.6(or -0.6) to call it a good correlation.

In [ ]:
```
"Perfect Correlation"
```

We can see that "Duration" and "Duaration" got the number 1.00000 which makes sense,each column always has a perfect relationship with itself.

In [ ]:
```
"Good relationship"
```

"Duration" and "Calories" got a 9.922721 correlation,which is very good relationship ,and we can predict that the longer you work out,the more calories we burn ,and the other way around: if we burned a lot of calroies ,you probably had a long wotk out.

In [ ]:
```
"Bad Correlation"
```

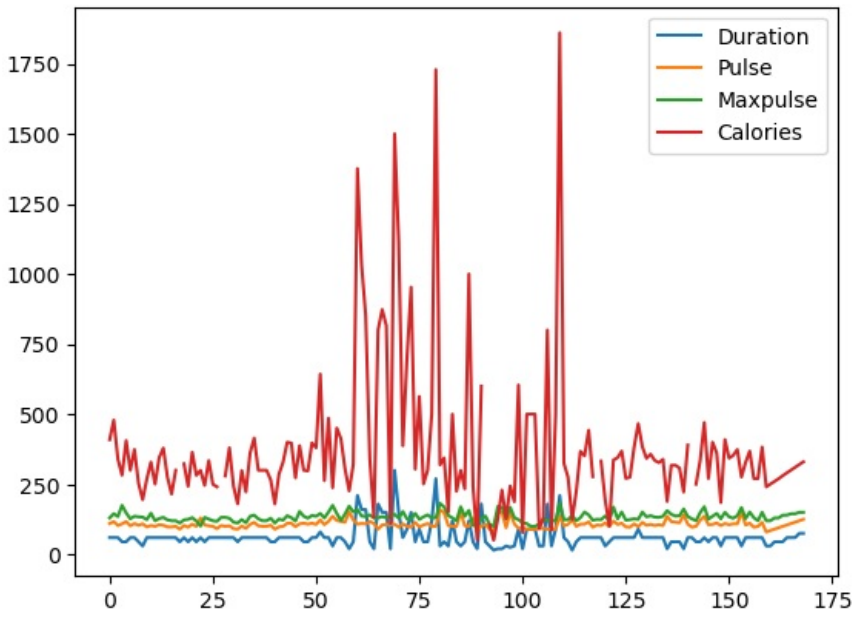
"Duration" and "Maxpulse" got a 0.009403 correlation, which is a very bad correlation, meaning that we can not predict the max pulse by just looking at the duration of the work out, and vice versa.

In [ ]:
```
"Pandas -Plotting"
```

Pandas uses the plot() method to create diagrams.

We can use the Pyplot,a submodulle of the Matplotlib library to visualise the daigram on the screen.

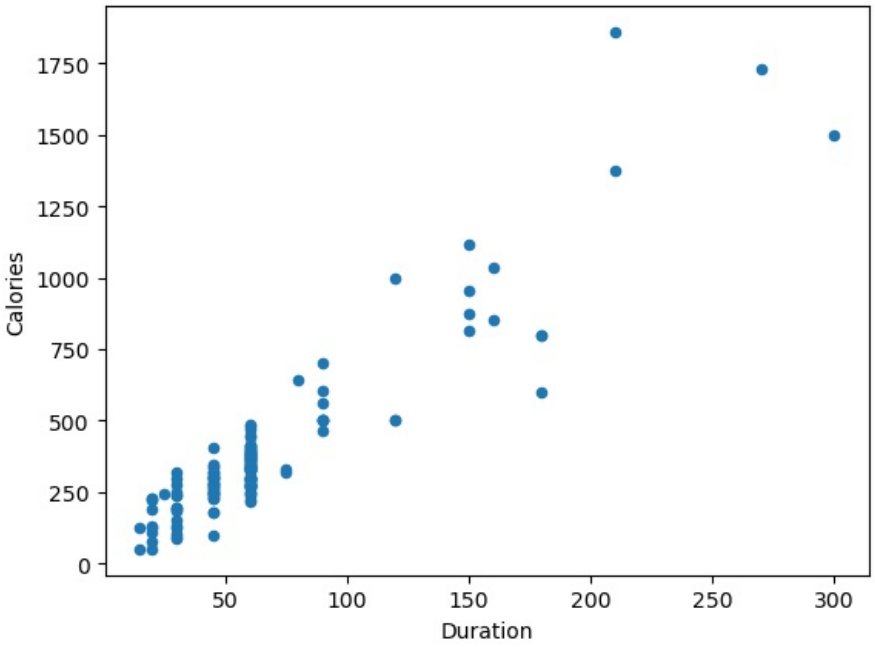
In [48]:
```
import pandas as pd
import matplotlib.pyplot as plt

df=pd.read_csv('data.csv')
df.plot()
plt.show()
```



In [ ]:
```
"Scatter Plot"
```

In [ ]:
```
Specify that you want a scatter plot with the kind arguement:

kind='scatter'

A scatter plot needs an x-and y-axes

In the exapmle below we will use "Duration" for the x_axis and "Calories" for the y_axis.

x = 'Duration', y = 'Calories'
```

In [52]:
```
df.plot(kind='scatter',x='Duration',y='Calories')
plt.show()
```

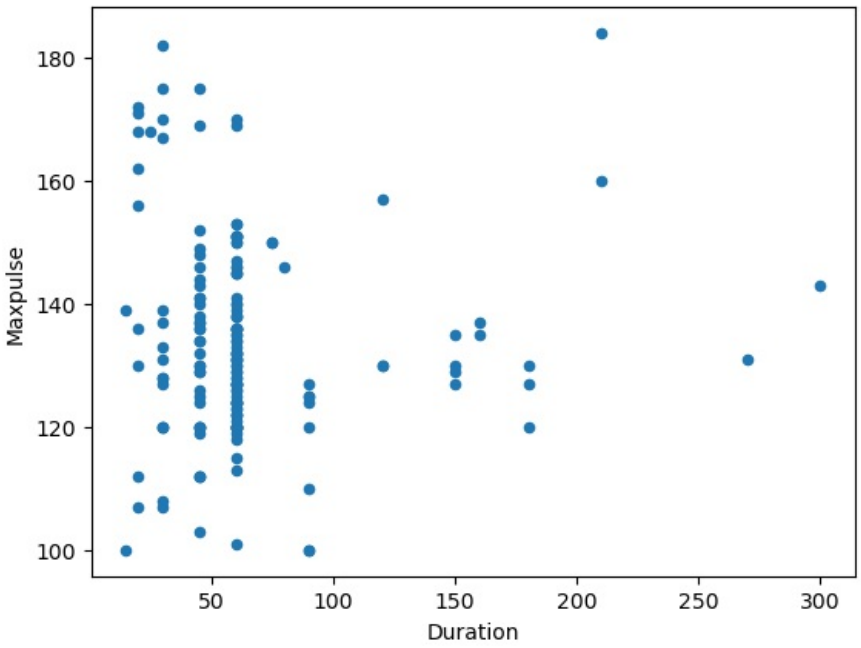


Remember: In the previous example, we learned that the correlation between "Duration" and "Calories" was 0.922721, and we concluded with the fact that higher duration means more calories burned.

By looking at the scatterplot, I will agree.

Let's create another scatterplot, where there is a bad relationship between the columns, like "Duration" and "Maxpulse", with the correlation 0.009403:

In [59]:
```
import pandas as pd
import matplotlib.pyplot as plt

df=pd.read_csv('data.csv')
df.plot(kind='scatter',x='Duration',y='Maxpulse')
plt.show()
```



In [ ]:
```
"Histogram"
```

Use the kind argument to specify that you want a histogram:

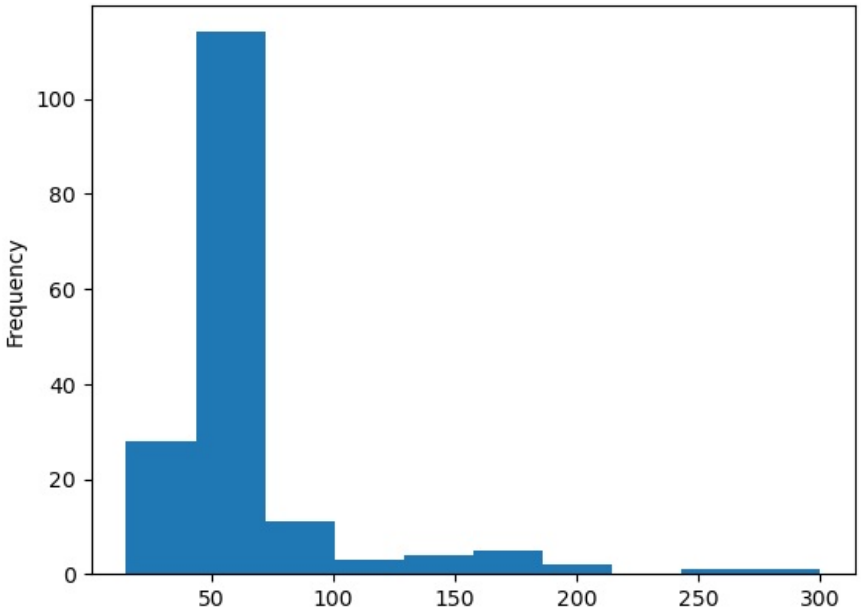
kind = 'hist'

A histogram needs only one column.

A histogram shows us the frequency of each interval, e.g. how many workouts lasted between 50 and 60 minutes?

In the example below we will use the "Duration" column to create the histogram:

In [64]:
```
df['Duration'].plot(kind='hist')
plt.show()
```



In [66]:
```
df
```

Out[66]:

	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	300.0
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

169 rows × 4 columns

Note: The histogram tells us that there were over 100 workouts that lasted between 50 and 60 minutes.

In [78]:
```

```

Out[78]:

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```