Pandas

In [ ]:
```
1. Pandas is a Python library used for working with data sets.
2. It has functions for analyzing, cleaning, exploring, and manipulating data.
3. Pandas can clean messy data sets, and make them readable and relevant.
4. Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values.
   This is called cleaning the data.
```

Installation of Pandas

In [1]:
```
pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\rajee\anaconda3\lib\site-packages (1.5.3)
Requirement already satisfied: pytz>=2020.1 in c:\users\rajee\anaconda3\lib\site-packages (from pandas) (2022.7)
Requirement already satisfied: numpy>=1.21.0 in c:\users\rajee\anaconda3\lib\site-packages (from pandas) (1.23.5)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\rajee\anaconda3\lib\site-packages (from pandas) (2.8.
2)
Requirement already satisfied: six>=1.5 in c:\users\rajee\anaconda3\lib\site-packages (from python-dateutil>=2.8.1->pan
das) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Import Pandas

In [ ]:
```
import : keyword used to import pandas library
pd     : Usually used as Alias name for pandas while importing.

    Syntax : import pandas as pd

Now the Pandas package can be referred to as pd instead of pandas
```

In [2]:
```
import pandas as pd #importing pandas
import numpy as np  #importind numpy
```

Pandas Series

In [ ]:
```
1. A Pandas Series is like a column in a table.
```

```
2. It is a one-dimensional array holding data of any type.

We can create a pandas series using different inputs like:

    1.  list and Nested list.
    2.  Empty series
    3.  ndarray
    4.  ndarray with custom index
    5.  dictionary
    6.  Scalar value
    7.  ndarray method arange()
    8.  range function
    9.  random method

There are several more ways to create a pandas series.
```

Create a Series from a list

In [3]:
```
#Create a Series from a list

l = [1,2,3,4,5]

myseries = pd.Series(l)

print(myseries)
```

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

Create an empty series

In [4]:
```
#Creating an empty Series

s = pd.Series()

print(s)
```

```
Series([], dtype: float64)
C:\Users\rajee\AppData\Local\Temp\ipykernel_28392\1801970861.py:3: FutureWarning: The default dtype for empty Series wi
ll be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.
  s = pd.Series()
```

Create a series using an array

In [3]:
```
#Creating a series using an array

arr = np.array([1,2,3,4,5])
s = pd.Series(arr)
print(s)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int32
```

Create a Series using an array with custom index

In [4]:
```
#Creating Series using an array with custom index

arr = np.array([10,20,30,40,50])
s = pd.Series(arr,index=['a','b','c','d','e'])
print(s)
```

```
a    10
b    20
c    30
d    40
e    50
dtype: int32
```

Create a series from a Nested list

In [5]:
```
#create a simple pandas series from a Nested list
l = [1,[2,3,4],5,6]

s = pd.Series(l)

print(s)
```

```
0           1
1    [2, 3, 4]
2           5
3           6
dtype: object
```

Accessing items from Pandas Series

We can access each item of a pandas series by referring to its index number inside a square brackets. If there is no custom index, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

How to create custom index ?

index :Keyword used to define custom index for each items of a series.

In [6]:
```
#Accessing items from pandas series

l = [1,2,3,4,5]

my_series = pd.Series(l)

print(my_series)
print(my_series[0]) #Accessing 1st item
print(my_series[1]) #Accessing 2nd item
```

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

In [10]:
```
#Accessing each item in a series that has custom index

s =pd.Series([10,20,30,40], index = ['a','b','c','d']) #series with custom index

print (s)
print("First Item : ",s['a']) # acessing first item using custom index label 'a'
print("Last Item:  ",s['d']) # accessing last item using custom index label 'd'
```

```
a    10
b    20
c    30
d    40
dtype: int64
First Item :  10
Last Item:   40
```

Create a Series from a dictionary

In [12]:
```
#Create a Pandas Series from a dictionary
#The keys of the dictionary become the labels(index).

my_dict = {"Name":"Neenu","Age":30,"Place":"USA"}
s = pd.Series(my_dict)
print(s)
```

```
Name     Neenu
Age         30
Place      USA
dtype: object
```

In [13]:
```
#Create a Pandas Series using a specified indexes (keys) from dictionary.

my_dict = {"Name":"Neenu","Age":30,"Place":"USA"}
s = pd.Series(my_dict,index=["Name","Age"])
print(s)
```

```
Name    Neenu
Age        30
dtype: object
```

Create a series from a scalar value

In [15]:
```
# Creating a series from a scalar value

s = pd.Series(20,index=[0,1,2,3,4,5]) #Create a series using scalar value 20
print(s)
```

```
0    20
1    20
2    20
3    20
4    20
5    20
dtype: int64
```

Create a series using arange() method

In [19]:
```
#Creating using built-in method arange()

s = pd.Series(np.arange(1,10,2))
print(s)
```

```
0    1
1    3
2    5
3    7
4    9
dtype: int32
```

In [20]:
```
#Create custom index for above series using argument index

s.index = ['a','b','c','d','e']
print(s)
```

```
a    1
b    3
c    5
d    7
e    9
dtype: int32
```

In [26]:
```
#To reset custom index for above series

s.reset_index() #method to change custom index
```

Out[26]:

	index	0
0	a	1
1	b	3
2	c	5
3	d	7
4	e	9

In [28]:
```
s.reset_index(drop=True) #drop the custom index created using argument index
```

Out[28]:
```
0    1
1    3
2    5
3    7
4    9
dtype: int32
```

Create a Series using range function

In [30]:
```
#Creating a Series using range function

s = pd.Series(range(10))
print(s)
```

```
0    0
1    1
2    2
3    3
4    4
5    5
6    6
7    7
8    8
9    9
dtype: int64
```

Create a series using random method

In [4]:
```
#Create a series with random method

import pandas as pd
import numpy as np
s = pd.Series(np.random.randn(7))
print (s)
```

```
0   -1.509536
1   -0.315707
2    1.866252
3   -0.953432
4   -1.480670
5   -0.918300
6    0.410857
dtype: float64
```

Pandas DataFrames

In [ ]:
```
A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Features of DataFrame:

1. Columns are of different types
2. Size - Mutable
3. Labeled axes (rows and columns)
4. Can Perform Arithmetic operations on rows and columns

A pandas DataFrame can be created using various inputs like :

1. Lists
2. dictionary
3. Series
4. Numpy ndarrays
5. Another DataFrame etc.
```

Create a DataFrame from DataFrame() constructor

In [ ]:
```
A pandas DataFrame can be created using the DataFrame() constructor :

syntax:
pandas.DataFrame( data, index, columns, dtype, copy)

The parameters of the constructor are :-

1. data     --  data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2. 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.
3. columns  --  For column labels, the optional default syntax is - np.arange(n). This is only true if no index is pass
4. dtype    --  Data type of each column.
5. copy     --  The copy() method returns a copy of the DataFrame. By default, the copy is a "deep copy" meaning that
                any changes made in the original DataFrame will NOT be reflected in the copy.
```

Create DataFrame

Create empty DataFrame

In [32]:
```
# Create a empty DataFrame
```

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

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

Create a DataFrame from Lists

In [71]:
```
# Create a DataFrame from Lists (The DataFrame can be created using a single list or a list of lists)

l = [1,2,3,4,5,6]
df = pd.DataFrame(l)
df1 = pd.DataFrame(data=l,columns=["Number"]) #Assigning custom label for column.
print(df)
#print(df1)
```

```
   0
0  1
1  2
2  3
3  4
4  5
5  6
```

Create a DataFrame from Nested list

In [35]:
```
# Create a DataFrame from Nested list.

l = [1,2,3,[4,5,6]]
df = pd.DataFrame(l)
print(df)
```

```
           0
0          1
1          2
2          3
3  [4, 5, 6]
```

Create a DataFrame from two series

In [11]:
```
# create a DataFrame from two series
data = {
  "Name": ["Maria","John","Angel"],
  "Age": [30,40,25]
}
```

```
df = pd.DataFrame(data)

print(df)
```

```
    Name  Age
0  Maria   30
1   John   40
2  Angel   25
```

Access rows from the dataframe

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

In [19]:
```
# Accessing 0th row of a given dataframe (returns a series)
data = {
  "Name": ["Maria","John","Angel"],
  "Age": [30,40,25]
}

df = pd.DataFrame(data)

print(df)
s = df.loc[0]
print("\n",s) #Accessing 0th row , This rerurns a series.

print(type(df))
print(type(s))
```

```
    Name  Age
0  Maria   30
1   John   40
2  Angel   25

 Name    Maria
Age        30
Name: 0, dtype: object
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

In [21]:
```
# Accessing 0th row of a given dataframe (returns a DataFrame)
data = {
  "Name": ["Maria","John","Angel"],
  "Age": [30,40,25]
```

```
}

df = pd.DataFrame(data)

print(df)
df1 = df.loc[[0]]
print("\n",df1) #Accessing 0th row , This rerurns a DataFrame.

print(type(df))
print(type(df1))
```

```
    Name  Age
0  Maria   30
1   John   40
2  Angel   25

     Name  Age
0  Maria   30
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

In [24]:
```
# Accessing 0th and 1st rows of a given dataframe
data = {
  "Name": ["Maria","John","Angel"],
  "Age": [30,40,25]
}

df = pd.DataFrame(data)

print(df)
print("\n",df.loc[[0,1]]) #Accessing 0th and 1st row
```

```
    Name  Age
0  Maria   30
1   John   40
2  Angel   25

     Name  Age
0  Maria   30
1   John   40
```

Named Indexes

index : Argument used to set your own indexes (custom indexes).

In [26]:
```
#Creating your own indexes using index argument

data = {
  "Name": ["Maria","John","Angel"],
  "Age": [30,40,25]
}

df = pd.DataFrame(data, index = ['a','b','c']) #custom indexex a,b and c has been set.

print(df)
```

```
    Name  Age
a  Maria   30
b   John   40
c  Angel   25
```

Locate custom Indexes

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

In [28]:
```
# Accessing first two rows of a given dataframe with custom index

data = {
  "Name": ["Maria","John","Angel"],
  "Age": [30,40,25]
}

df = pd.DataFrame(data, index = ['a','b','c'])

print(df.loc[['a','b']])
```

```
    Name  Age
a  Maria   30
b   John   40
```

Create a DataFrame from dictionary

In [55]:
```
#Using a dictionary
weather_data = {
    'day': ['1/1/2017','1/2/2017','1/3/2017'],
    'temperature': [32,35,28],
    'windspeed': [6,7,2],
    'event': ['Rain', 'Sunny', 'Snow']
}
```

```
df = pd.DataFrame(weather_data)
df
```

Out[55]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow

Create a DataFrame from list of dictionaries

In [59]:

```
#Using a list of dictionaries
weather_data = [
    {'day': '1/1/2017', 'temperature': 32, 'windspeed': 6, 'event': 'Rain'},
    {'day': '1/2/2017', 'temperature': 35, 'windspeed': 7, 'event': 'Sunny'},
    {'day': '1/3/2017', 'temperature': 28, 'windspeed': 2, 'event': 'Snow'},

]
df = pd.DataFrame(weather_data,)
df
```

Out[59]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow

Create a DataFrame from list of tuples

In [57]:

```
# Create a DataFrame using list of tuples
weather_data = [
    ('1/1/2017',32,6,'Rain'),
    ('1/2/2017',35,7,'Sunny'),
    ('1/3/2017',28,2,'Snow')
]
df = pd.DataFrame(data=weather_data, columns=['day','temperature','windspeed','event'])
print(df)
```

```
        day  temperature  windspeed  event
0  1/1/2017           32          6   Rain
1  1/2/2017           35          7  Sunny
2  1/3/2017           28          2   Snow
```

Create a DataFrame from list of lists

In [58]:
```
#Using a list of lists
weather_data = [
    ['1/1/2017',32,6,'Rain'],
    ['1/2/2017',35,7,'Sunny'],
    ['1/3/2017',28,2,'Snow']
]
df = pd.DataFrame(data=weather_data, columns=['day','temperature','windspeed','event'])
print(df)
```

```
        day  temperature  windspeed  event
0  1/1/2017           32          6   Rain
1  1/2/2017           35          7  Sunny
2  1/3/2017           28          2   Snow
```

Load Files as DataFrame

In [ ]:
```
1. If your data sets are stored in a file, Pandas can load them into a DataFrame.

    read_csv()   : Method to load  csv files as dataframe
    read_excel() : Method to load  excel files as dataframe

2. If you have a large DataFrame with many rows, Pandas will only return the first 5 rows, and the last 5 rows.

to_string() : Method to print the entire DataFrame.

max_rows    : The number of rows returned can be defined using "pd.options.display.max_rows" statement.
```

Loading a comma separated file (CSV file) into a DataFrame

In [33]:
```
#Loading a comma separated file (CSV file) into a DataFrame

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

print(df) #returns only the first 5 rows, and the last 5 rows.

#print(df.to_string()) #to_string() method orints entire dataframe
```

```
     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 [34]:
```
#To check the number of maximum returned rows

print(pd.options.display.max_rows)
```

```
60
```

In my system 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 have option to change the maximum number of returned rows using the same attribute max_rows.

In [40]:
```
#Increase the maximum number of rows to display the entire DataFrame:


pd.options.display.max_rows = 9999 #assigning custom value for max_rows
print(pd.options.display.max_rows)
```

```
9999
```

Loading a excel file into a DataFrame

In [54]:
```
#Loading a excel file into a DataFrame

df =pd.read_excel("stack_3_levels.xlsx","Sheet1")
#df =pd.read_excel("stack_3_levels.xlsx","Sheet2")
#df =pd.read_excel("stack_3_levels.xlsx","stock_price")
#df =pd.read_excel("stack_3_levels.xlsx")

print(df)
```

	Unnamed: 0	Company	Price	Price to earnings ratio (P/E)
0	2017-06-05	Facebook	155	37.10
1	NaT	Google	955	32.00
2	NaT	Microsoft	66	30.31
3	2017-06-06	Facebook	150	36.98
4	NaT	Google	987	31.30
5	NaT	Microsoft	69	30.56
6	2017-06-07	Facebook	153	36.78
7	NaT	Google	963	31.70
8	NaT	Microsoft	62	30.46
9	2017-06-08	Facebook	155	36.11
10	NaT	Google	1000	31.20
11	NaT	Microsoft	61	30.11
12	2017-06-09	Facebook	156	37.07
13	NaT	Google	1012	30.00
14	NaT	Microsoft	66	31.00

Analyzing DataFrames

In [ ]:
```
There are several built-in methods in pandas for analyzing DataFrames:

head()     : One of the most used method for getting a quick overview of the DataFrame.
             The head() method returns the headers and a specified number of rows, starting from the top.
             if the number of rows is not specified, the head() method will return the top 5 rows.

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

info()     : The DataFrames object has a method called info(), that gives information about the data set.
             The info() method also tells us how many Non-Null values there are present in each column.

unique()   : This method displays unique values in a specified column

describe() : This method displays statistical properties of all the numerical columns in the dataframe

isin()     : This method retun 'True' or 'False' based on the condition. isin() used to check whether a given value(s)
             present in the specified column of a given dataset. If the value present it return 'True' else return 'Fal

shape      : Return the shape (number rows and columns) of the dataframe.

column     : Return all the columns in the specified dataframe.

index      :Argument used to display the index range of a given dataframe.
```

In [7]:
```
#head() method to return first 5 rows from the dataset.
#We will be using a CSV file called 'data.csv'.

df = pd.read_csv("data.csv")
print(df.head(5))
```

```
   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
```

In [8]:
```
#head() method to return last 5 rows from the dataset.
#We will be using a CSV file called 'data.csv'.

df = pd.read_csv("data.csv")
print(df.tail(5))
```

```
     Duration  Pulse  Maxpulse  Calories
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
```

In [9]:
```
# info() to get the information about the dataset

df = pd.read_csv("data.csv")
print(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
None
```

In [10]:
```
#To get the number columns in the dataset

df = pd.read_csv("data.csv")
print(df.columns)
```

```
Index(['Duration', 'Pulse', 'Maxpulse', 'Calories'], dtype='object')
```

In [11]:
```
#TO display the number of rows and columns

df = pd.read_csv("data.csv")
print(df.shape) #This dataset has 169 rows and 4 columns
```

```
(169, 4)
```

In [12]:
```
#Display statistical properties of all the numerical columns in the dataframe

df = pd.read_csv("data.csv")
print(df.describe())
```

```
         Duration       Pulse    Maxpulse     Calories
count  169.000000  169.000000  169.000000   164.000000
mean    63.846154  107.461538  134.047337   375.790244
std     42.299949   14.510259   16.450434   266.379919
min     15.000000   80.000000  100.000000    50.300000
25%     45.000000  100.000000  124.000000   250.925000
50%     60.000000  105.000000  131.000000   318.600000
75%     60.000000  111.000000  141.000000   387.600000
max    300.000000  159.000000  184.000000  1860.400000
```

In [14]:
```
#Display unique values in a specified column

df = pd.read_csv("data.csv")
print(df['Duration'].unique()) # Display all the unique values in 'Duration' column
```

```
[ 60  45  30  80  20 210 160 180 150 300  90 120 270  15  25  75]
```

In [17]:
```
#Display rows where a condition is satisfied

df =pd.read_csv("data.csv")
df[df['Duration']==60] #return rows that has "Duration"  equal to 60
```

Out[17]:

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
5	60	102	127	300.0
6	60	110	136	374.0
...	...	...	...	...
157	60	100	120	270.4
158	60	114	150	382.8
164	60	105	140	290.8
165	60	110	145	300.0
166	60	115	145	310.2

79 rows × 4 columns

In [23]:
```
#To display the range of indexex in the specified dataset.
df = pd.read_csv("data.csv")
print(df.index)
```

```
RangeIndex(start=0, stop=169, step=1)
```

In [36]:
```
#isin() method to check whether 'Snow' or "Sunny" values present in the 'event' column of Dataframe "Weather_data"

df = pd.read_csv("Weather_data.csv")
print("Display the Dataset 'Weather_data':\n\n",df,"\n")
print("Check whether 'Snow' or 'Sunny' values present in the 'event' column:\n")
print(df['event'].isin(['Snow','Sunny']))
```

Display the Dataset 'Weather_data':

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

Check whether 'Snow' or 'Sunny' values present in the 'event' column:

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

Data Correlations

In [ ]:
```
corr()  : The corr() method calculates the relationship between each column in your data set.
          The corr() method ignores "not numeric" columns.
```

In [19]:
```
#corr() method to find the correlation between columns in the dataset "Weather_data.csv"

df = pd.read_csv("Weather_data.csv")
print(df.corr())
```

```
             temperature  windspeed
temperature     1.000000   0.037226
windspeed       0.037226   1.000000
```

```
C:\Users\rajee\AppData\Local\Temp\ipykernel_11652\3625993116.py:4: FutureWarning: The default value of numeric_only in
DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the v
alue of numeric_only to silence this warning.
  print(df.corr())
```

What is a good correlation

In [ ]:
```
    The Result of the corr() method is a table with a lot of numbers that varies from -1 to 1.Each number represents
the relationship is between two columns.
```

```
Correlation Values                          Meaning

         1                          A perfect correlation (1 : 1 correlation)
                                     (Each time a value went up in the first column, the other one went up as well)

         0.9                        Good Correlation
                                     (if you increase one value, the other will probably increase as well)

        -0.9                        Good relationship
                                      (if you increase one value, the other will probably go down)

         0.2                        NOT a good relationship
                                       (if one value goes up does not mean that the other will)


What is a good correlation?

    It depends on the use, but most of the time if you have at least 0.6 (or -0.6) ,we can call it as good correlation.
```

Cleaning Data

In [ ]:
```
Data cleaning means fixing bad data in your data set.

Bad data could be:

1. Empty cells
2. Data in wrong format
3. Wrong data
4. Duplicates
```

Cleaning Empty Cells

In [ ]:
```
Empty cells can potentially give you a wrong result when you analyze data.

How to deal with empty cell :

Remove the row(s) that contain empty cell. since data sets can be very big, and removing a few rows
will not have a big impact on the result.

How to remove empty cell :
```

```
dropna()          : This method returns a new Data Frame with no empty cells. By default, the dropna() method returns
                    a new DataFrame, and will not change the original.

inplace = True    : Use this argument ,if you want to change the original DataFrame. This argument will NOT return
                    a new DataFrame, but it will remove all rows containing NULL values from the original DataFrame.
```

In [6]:
```
#Remove all rows that contain empty cell( null values) from a given dataframe.

df = pd.read_excel("data.xlsx")
new_df = df.dropna()
print(new_df.to_string())#Return a new dataframe that has no empty cell

#print(df.dropna(inplace=True)) #Remove all the rows that has empty cell from the original dataframe
#print(df.to_string())
```

	Duration	Pulse	Maxpulse	Calories
1	60.0	117.0	145.0	479.0
3	45.0	109.0	175.0	282.4
5	60.0	102.0	127.0	300.0
6	60.0	110.0	136.0	374.0
7	45.0	104.0	134.0	253.3
9	60.0	98.0	124.0	269.0
11	60.0	100.0	120.0	250.7
12	60.0	106.0	128.0	345.3
13	60.0	104.0	132.0	379.3
14	60.0	98.0	123.0	275.0
15	60.0	98.0	120.0	215.2
16	60.0	100.0	120.0	300.0
18	60.0	103.0	123.0	323.0
19	45.0	97.0	125.0	243.0
20	60.0	108.0	131.0	364.2
21	45.0	100.0	119.0	282.0
22	60.0	130.0	101.0	300.0
23	45.0	105.0	132.0	246.0
24	60.0	102.0	126.0	334.5
25	60.0	100.0	120.0	250.0
26	60.0	92.0	118.0	241.0
28	60.0	100.0	132.0	280.0
29	60.0	102.0	129.0	380.3
30	60.0	92.0	115.0	243.0
31	45.0	90.0	112.0	180.1
32	60.0	101.0	124.0	299.0
33	60.0	93.0	113.0	223.0
34	60.0	107.0	136.0	361.0
35	60.0	114.0	140.0	415.0
36	60.0	102.0	127.0	300.0
37	60.0	100.0	120.0	300.0
38	60.0	100.0	120.0	300.0
39	45.0	104.0	129.0	266.0
40	45.0	90.0	112.0	180.1
41	60.0	98.0	126.0	286.0
42	60.0	100.0	122.0	329.4
43	60.0	111.0	138.0	400.0
44	60.0	111.0	131.0	397.0
45	60.0	99.0	119.0	273.0
46	60.0	109.0	153.0	387.6
47	45.0	111.0	136.0	300.0
48	45.0	108.0	129.0	298.0
49	60.0	111.0	139.0	397.6
50	60.0	107.0	136.0	380.2

51	80.0	123.0	146.0	643.1
52	60.0	106.0	130.0	263.0
53	60.0	118.0	151.0	486.0
54	30.0	136.0	175.0	238.0
55	60.0	121.0	146.0	450.7
56	60.0	118.0	121.0	413.0
57	45.0	115.0	144.0	305.0
58	20.0	153.0	172.0	226.4
59	45.0	123.0	152.0	321.0
60	210.0	108.0	160.0	1376.0
61	160.0	110.0	137.0	1034.4
62	160.0	109.0	135.0	853.0
63	45.0	118.0	141.0	341.0
64	20.0	110.0	130.0	131.4
65	180.0	90.0	130.0	800.4
66	150.0	105.0	135.0	873.4
67	150.0	107.0	130.0	816.0
68	20.0	106.0	136.0	110.4
69	300.0	108.0	143.0	1500.2
70	150.0	97.0	129.0	1115.0
71	60.0	109.0	153.0	387.6
72	90.0	100.0	127.0	700.0
73	150.0	97.0	127.0	953.2
74	45.0	114.0	146.0	304.0
75	90.0	98.0	125.0	563.2
76	45.0	105.0	134.0	251.0
77	45.0	110.0	141.0	300.0
78	120.0	100.0	130.0	500.4
79	270.0	100.0	131.0	1729.0
80	30.0	159.0	182.0	319.2
81	45.0	149.0	169.0	344.0
82	30.0	103.0	139.0	151.1
83	120.0	100.0	130.0	500.0
84	45.0	100.0	120.0	225.3
86	45.0	102.0	136.0	234.0
87	120.0	100.0	157.0	1000.1
88	45.0	129.0	103.0	242.0
89	20.0	83.0	107.0	50.3
90	180.0	101.0	127.0	600.1
92	30.0	90.0	107.0	105.3
93	15.0	80.0	100.0	50.5
94	20.0	150.0	171.0	127.4
96	30.0	95.0	128.0	128.2
97	25.0	152.0	168.0	244.2
98	30.0	109.0	131.0	188.2

99	90.0	93.0	124.0	604.1
100	20.0	95.0	112.0	77.7
101	90.0	90.0	110.0	500.0
102	90.0	90.0	100.0	500.0
104	30.0	92.0	108.0	92.7
105	30.0	93.0	128.0	124.0
106	180.0	90.0	120.0	800.3
107	30.0	90.0	120.0	86.2
108	90.0	90.0	120.0	500.3
109	210.0	137.0	184.0	1860.4
111	45.0	107.0	124.0	275.0
112	15.0	124.0	139.0	124.2
113	45.0	100.0	120.0	225.3
114	60.0	108.0	131.0	367.6
115	60.0	108.0	151.0	351.7
116	60.0	116.0	141.0	443.0
117	60.0	97.0	122.0	277.4
119	60.0	103.0	124.0	332.7
120	30.0	112.0	137.0	193.9
122	60.0	119.0	169.0	336.7
124	60.0	111.0	151.0	368.5
125	60.0	98.0	122.0	271.0
126	60.0	97.0	124.0	275.3
127	60.0	109.0	127.0	382.0
128	90.0	99.0	125.0	466.4
129	60.0	114.0	151.0	384.0
130	60.0	104.0	134.0	342.5
131	60.0	107.0	138.0	357.5
132	60.0	103.0	133.0	335.0
133	60.0	106.0	132.0	327.5
134	60.0	103.0	136.0	339.0
135	20.0	136.0	156.0	189.0
136	45.0	117.0	143.0	317.7
137	45.0	115.0	137.0	318.0
138	45.0	113.0	138.0	308.0
139	20.0	141.0	162.0	222.4
140	60.0	108.0	135.0	390.0
142	45.0	100.0	120.0	250.4
143	45.0	122.0	149.0	335.4
144	60.0	136.0	170.0	470.2
145	45.0	106.0	126.0	270.8
146	60.0	107.0	136.0	400.0
147	60.0	112.0	146.0	361.9
148	30.0	103.0	127.0	185.0
149	60.0	110.0	150.0	409.4

```
152      60.0  109.0     138.0     374.0
154      60.0  105.0     128.0     328.0
156      60.0   97.0     131.0     270.4
157      60.0  100.0     120.0     270.4
158      60.0  114.0     150.0     382.8
159      30.0   80.0     120.0     240.9
160      30.0   85.0     120.0     250.4
162      45.0   95.0     130.0     270.0
165      60.0  110.0     145.0     300.0
166      60.0  115.0     145.0     310.2
167      75.0  120.0     150.0     320.4
168      75.0  125.0     150.0     330.4
```

In [9]:
```
#Remove all rows that contain empty cell( null values) from a given dataframe.

df = pd.read_csv("data.csv")
#print(df.dropna()) #Return a new dataframe that has no empty cell
print(df.dropna(inplace=True)) #Remove all the rows that has empty cell from the original dataframe
#print(df.to_string())
```

```
None
```

In [13]:
```
#Remove all rows that contain empty cell( null values) from a given dataframe.

df = pd.read_csv("Stock_data.csv")
#print(df.dropna()) #Return a new dataframe that has no empty cell
print(df.dropna(inplace=True)) #Remove all the rows that has empty cell from the original dataframe
print(df.to_string())
```

```
None
  tickers            eps  revenue price         people
0   GOOGL          27.82       87   845     larry page
1     WMT           4.61      484    65           n.a.
2    MSFT             -1       85    64     bill gates
3     RIL  not available       50  1023  mukesh ambani
4    TATA            5.6       -1  n.a.     ratan tata
```

Replace Empty Values

In [ ]:
```
Another way of dealing with empty cells is to insert a new value instead of removing the cell(s).
This way you do not have to delete entire rows just because of some empty cells.

fillna()     : Method used to replace empty cells with a value.

We can also replace the null values of a specified column by mentioning the colum name inside square brackets.
```

In [10]:
```
#Replace NULL values with the number Null:


df = pd.read_excel('data.xlsx')
df.fillna('null', inplace = True)
print(df.to_string())

#Notice in the result: empty cells will get value 'null'
```

	Duration	Pulse	Maxpulse	Calories
0	null	110.0	130.0	409.1
1	60.0	117.0	145.0	479.0
2	60.0	103.0	null	340.0
3	45.0	109.0	175.0	282.4
4	null	117.0	148.0	406.0
5	60.0	102.0	127.0	300.0
6	60.0	110.0	136.0	374.0
7	45.0	104.0	134.0	253.3
8	30.0	null	133.0	195.1
9	60.0	98.0	124.0	269.0
10	60.0	103.0	147.0	null
11	60.0	100.0	120.0	250.7
12	60.0	106.0	128.0	345.3
13	60.0	104.0	132.0	379.3
14	60.0	98.0	123.0	275.0
15	60.0	98.0	120.0	215.2
16	60.0	100.0	120.0	300.0
17	45.0	90.0	112.0	null
18	60.0	103.0	123.0	323.0
19	45.0	97.0	125.0	243.0
20	60.0	108.0	131.0	364.2
21	45.0	100.0	119.0	282.0
22	60.0	130.0	101.0	300.0
23	45.0	105.0	132.0	246.0
24	60.0	102.0	126.0	334.5
25	60.0	100.0	120.0	250.0
26	60.0	92.0	118.0	241.0
27	60.0	103.0	132.0	null
28	60.0	100.0	132.0	280.0
29	60.0	102.0	129.0	380.3
30	60.0	92.0	115.0	243.0
31	45.0	90.0	112.0	180.1
32	60.0	101.0	124.0	299.0
33	60.0	93.0	113.0	223.0
34	60.0	107.0	136.0	361.0
35	60.0	114.0	140.0	415.0
36	60.0	102.0	127.0	300.0
37	60.0	100.0	120.0	300.0
38	60.0	100.0	120.0	300.0
39	45.0	104.0	129.0	266.0
40	45.0	90.0	112.0	180.1
41	60.0	98.0	126.0	286.0
42	60.0	100.0	122.0	329.4
43	60.0	111.0	138.0	400.0

44	60.0	111.0	131.0	397.0
45	60.0	99.0	119.0	273.0
46	60.0	109.0	153.0	387.6
47	45.0	111.0	136.0	300.0
48	45.0	108.0	129.0	298.0
49	60.0	111.0	139.0	397.6
50	60.0	107.0	136.0	380.2
51	80.0	123.0	146.0	643.1
52	60.0	106.0	130.0	263.0
53	60.0	118.0	151.0	486.0
54	30.0	136.0	175.0	238.0
55	60.0	121.0	146.0	450.7
56	60.0	118.0	121.0	413.0
57	45.0	115.0	144.0	305.0
58	20.0	153.0	172.0	226.4
59	45.0	123.0	152.0	321.0
60	210.0	108.0	160.0	1376.0
61	160.0	110.0	137.0	1034.4
62	160.0	109.0	135.0	853.0
63	45.0	118.0	141.0	341.0
64	20.0	110.0	130.0	131.4
65	180.0	90.0	130.0	800.4
66	150.0	105.0	135.0	873.4
67	150.0	107.0	130.0	816.0
68	20.0	106.0	136.0	110.4
69	300.0	108.0	143.0	1500.2
70	150.0	97.0	129.0	1115.0
71	60.0	109.0	153.0	387.6
72	90.0	100.0	127.0	700.0
73	150.0	97.0	127.0	953.2
74	45.0	114.0	146.0	304.0
75	90.0	98.0	125.0	563.2
76	45.0	105.0	134.0	251.0
77	45.0	110.0	141.0	300.0
78	120.0	100.0	130.0	500.4
79	270.0	100.0	131.0	1729.0
80	30.0	159.0	182.0	319.2
81	45.0	149.0	169.0	344.0
82	30.0	103.0	139.0	151.1
83	120.0	100.0	130.0	500.0
84	45.0	100.0	120.0	225.3
85	30.0	null	170.0	300.0
86	45.0	102.0	136.0	234.0
87	120.0	100.0	157.0	1000.1
88	45.0	129.0	103.0	242.0

89	20.0	83.0	107.0	50.3
90	180.0	101.0	127.0	600.1
91	45.0	107.0	137.0	null
92	30.0	90.0	107.0	105.3
93	15.0	80.0	100.0	50.5
94	20.0	150.0	171.0	127.4
95	20.0	151.0	null	229.4
96	30.0	95.0	128.0	128.2
97	25.0	152.0	168.0	244.2
98	30.0	109.0	131.0	188.2
99	90.0	93.0	124.0	604.1
100	20.0	95.0	112.0	77.7
101	90.0	90.0	110.0	500.0
102	90.0	90.0	100.0	500.0
103	90.0	90.0	null	500.4
104	30.0	92.0	108.0	92.7
105	30.0	93.0	128.0	124.0
106	180.0	90.0	120.0	800.3
107	30.0	90.0	120.0	86.2
108	90.0	90.0	120.0	500.3
109	210.0	137.0	184.0	1860.4
110	60.0	null	124.0	325.2
111	45.0	107.0	124.0	275.0
112	15.0	124.0	139.0	124.2
113	45.0	100.0	120.0	225.3
114	60.0	108.0	131.0	367.6
115	60.0	108.0	151.0	351.7
116	60.0	116.0	141.0	443.0
117	60.0	97.0	122.0	277.4
118	60.0	105.0	125.0	null
119	60.0	103.0	124.0	332.7
120	30.0	112.0	137.0	193.9
121	null	100.0	120.0	100.7
122	60.0	119.0	169.0	336.7
123	60.0	107.0	null	344.9
124	60.0	111.0	151.0	368.5
125	60.0	98.0	122.0	271.0
126	60.0	97.0	124.0	275.3
127	60.0	109.0	127.0	382.0
128	90.0	99.0	125.0	466.4
129	60.0	114.0	151.0	384.0
130	60.0	104.0	134.0	342.5
131	60.0	107.0	138.0	357.5
132	60.0	103.0	133.0	335.0
133	60.0	106.0	132.0	327.5

```
134     60.0  103.0    136.0    339.0
135     20.0  136.0    156.0    189.0
136     45.0  117.0    143.0    317.7
137     45.0  115.0    137.0    318.0
138     45.0  113.0    138.0    308.0
139     20.0  141.0    162.0    222.4
140     60.0  108.0    135.0    390.0
141     60.0   97.0    127.0     null
142     45.0  100.0    120.0    250.4
143     45.0  122.0    149.0    335.4
144     60.0  136.0    170.0    470.2
145     45.0  106.0    126.0    270.8
146     60.0  107.0    136.0    400.0
147     60.0  112.0    146.0    361.9
148     30.0  103.0    127.0    185.0
149     60.0  110.0    150.0    409.4
150     null  106.0    134.0    343.0
151     60.0   null    129.0    353.2
152     60.0  109.0    138.0    374.0
153     30.0   null    167.0    275.8
154     60.0  105.0    128.0    328.0
155     60.0  111.0    151.0     null
156     60.0   97.0    131.0    270.4
157     60.0  100.0    120.0    270.4
158     60.0  114.0    150.0    382.8
159     30.0   80.0    120.0    240.9
160     30.0   85.0    120.0    250.4
161     45.0   null    130.0    260.4
162     45.0   95.0    130.0    270.0
163     null  100.0    140.0    280.9
164     60.0  105.0     null    290.8
165     60.0  110.0    145.0    300.0
166     60.0  115.0    145.0    310.2
167     75.0  120.0    150.0    320.4
168     75.0  125.0    150.0    330.4
```

In [13]:

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

df = pd.read_excel("data.xlsx")
df["Duration"].fillna("NA",inplace=True)
print(df.to_string())

#Notice in the result: empty cells of "Duration" column  will get the value 'NA'
```

	Duration	Pulse	Maxpulse	Calories
0	NA	110.0	130.0	409.1
1	60.0	117.0	145.0	479.0
2	60.0	103.0	NaN	340.0
3	45.0	109.0	175.0	282.4
4	NA	117.0	148.0	406.0
5	60.0	102.0	127.0	300.0
6	60.0	110.0	136.0	374.0
7	45.0	104.0	134.0	253.3
8	30.0	NaN	133.0	195.1
9	60.0	98.0	124.0	269.0
10	60.0	103.0	147.0	NaN
11	60.0	100.0	120.0	250.7
12	60.0	106.0	128.0	345.3
13	60.0	104.0	132.0	379.3
14	60.0	98.0	123.0	275.0
15	60.0	98.0	120.0	215.2
16	60.0	100.0	120.0	300.0
17	45.0	90.0	112.0	NaN
18	60.0	103.0	123.0	323.0
19	45.0	97.0	125.0	243.0
20	60.0	108.0	131.0	364.2
21	45.0	100.0	119.0	282.0
22	60.0	130.0	101.0	300.0
23	45.0	105.0	132.0	246.0
24	60.0	102.0	126.0	334.5
25	60.0	100.0	120.0	250.0
26	60.0	92.0	118.0	241.0
27	60.0	103.0	132.0	NaN
28	60.0	100.0	132.0	280.0
29	60.0	102.0	129.0	380.3
30	60.0	92.0	115.0	243.0
31	45.0	90.0	112.0	180.1
32	60.0	101.0	124.0	299.0
33	60.0	93.0	113.0	223.0
34	60.0	107.0	136.0	361.0
35	60.0	114.0	140.0	415.0
36	60.0	102.0	127.0	300.0
37	60.0	100.0	120.0	300.0
38	60.0	100.0	120.0	300.0
39	45.0	104.0	129.0	266.0
40	45.0	90.0	112.0	180.1
41	60.0	98.0	126.0	286.0
42	60.0	100.0	122.0	329.4
43	60.0	111.0	138.0	400.0

44	60.0	111.0	131.0	397.0
45	60.0	99.0	119.0	273.0
46	60.0	109.0	153.0	387.6
47	45.0	111.0	136.0	300.0
48	45.0	108.0	129.0	298.0
49	60.0	111.0	139.0	397.6
50	60.0	107.0	136.0	380.2
51	80.0	123.0	146.0	643.1
52	60.0	106.0	130.0	263.0
53	60.0	118.0	151.0	486.0
54	30.0	136.0	175.0	238.0
55	60.0	121.0	146.0	450.7
56	60.0	118.0	121.0	413.0
57	45.0	115.0	144.0	305.0
58	20.0	153.0	172.0	226.4
59	45.0	123.0	152.0	321.0
60	210.0	108.0	160.0	1376.0
61	160.0	110.0	137.0	1034.4
62	160.0	109.0	135.0	853.0
63	45.0	118.0	141.0	341.0
64	20.0	110.0	130.0	131.4
65	180.0	90.0	130.0	800.4
66	150.0	105.0	135.0	873.4
67	150.0	107.0	130.0	816.0
68	20.0	106.0	136.0	110.4
69	300.0	108.0	143.0	1500.2
70	150.0	97.0	129.0	1115.0
71	60.0	109.0	153.0	387.6
72	90.0	100.0	127.0	700.0
73	150.0	97.0	127.0	953.2
74	45.0	114.0	146.0	304.0
75	90.0	98.0	125.0	563.2
76	45.0	105.0	134.0	251.0
77	45.0	110.0	141.0	300.0
78	120.0	100.0	130.0	500.4
79	270.0	100.0	131.0	1729.0
80	30.0	159.0	182.0	319.2
81	45.0	149.0	169.0	344.0
82	30.0	103.0	139.0	151.1
83	120.0	100.0	130.0	500.0
84	45.0	100.0	120.0	225.3
85	30.0	NaN	170.0	300.0
86	45.0	102.0	136.0	234.0
87	120.0	100.0	157.0	1000.1
88	45.0	129.0	103.0	242.0

89	20.0	83.0	107.0	50.3
90	180.0	101.0	127.0	600.1
91	45.0	107.0	137.0	NaN
92	30.0	90.0	107.0	105.3
93	15.0	80.0	100.0	50.5
94	20.0	150.0	171.0	127.4
95	20.0	151.0	NaN	229.4
96	30.0	95.0	128.0	128.2
97	25.0	152.0	168.0	244.2
98	30.0	109.0	131.0	188.2
99	90.0	93.0	124.0	604.1
100	20.0	95.0	112.0	77.7
101	90.0	90.0	110.0	500.0
102	90.0	90.0	100.0	500.0
103	90.0	90.0	NaN	500.4
104	30.0	92.0	108.0	92.7
105	30.0	93.0	128.0	124.0
106	180.0	90.0	120.0	800.3
107	30.0	90.0	120.0	86.2
108	90.0	90.0	120.0	500.3
109	210.0	137.0	184.0	1860.4
110	60.0	NaN	124.0	325.2
111	45.0	107.0	124.0	275.0
112	15.0	124.0	139.0	124.2
113	45.0	100.0	120.0	225.3
114	60.0	108.0	131.0	367.6
115	60.0	108.0	151.0	351.7
116	60.0	116.0	141.0	443.0
117	60.0	97.0	122.0	277.4
118	60.0	105.0	125.0	NaN
119	60.0	103.0	124.0	332.7
120	30.0	112.0	137.0	193.9
121	NA	100.0	120.0	100.7
122	60.0	119.0	169.0	336.7
123	60.0	107.0	NaN	344.9
124	60.0	111.0	151.0	368.5
125	60.0	98.0	122.0	271.0
126	60.0	97.0	124.0	275.3
127	60.0	109.0	127.0	382.0
128	90.0	99.0	125.0	466.4
129	60.0	114.0	151.0	384.0
130	60.0	104.0	134.0	342.5
131	60.0	107.0	138.0	357.5
132	60.0	103.0	133.0	335.0
133	60.0	106.0	132.0	327.5

134	60.0	103.0	136.0	339.0
135	20.0	136.0	156.0	189.0
136	45.0	117.0	143.0	317.7
137	45.0	115.0	137.0	318.0
138	45.0	113.0	138.0	308.0
139	20.0	141.0	162.0	222.4
140	60.0	108.0	135.0	390.0
141	60.0	97.0	127.0	NaN
142	45.0	100.0	120.0	250.4
143	45.0	122.0	149.0	335.4
144	60.0	136.0	170.0	470.2
145	45.0	106.0	126.0	270.8
146	60.0	107.0	136.0	400.0
147	60.0	112.0	146.0	361.9
148	30.0	103.0	127.0	185.0
149	60.0	110.0	150.0	409.4
150	NA	106.0	134.0	343.0
151	60.0	NaN	129.0	353.2
152	60.0	109.0	138.0	374.0
153	30.0	NaN	167.0	275.8
154	60.0	105.0	128.0	328.0
155	60.0	111.0	151.0	NaN
156	60.0	97.0	131.0	270.4
157	60.0	100.0	120.0	270.4
158	60.0	114.0	150.0	382.8
159	30.0	80.0	120.0	240.9
160	30.0	85.0	120.0	250.4
161	45.0	NaN	130.0	260.4
162	45.0	95.0	130.0	270.0
163	NA	100.0	140.0	280.9
164	60.0	105.0	NaN	290.8
165	60.0	110.0	145.0	300.0
166	60.0	115.0	145.0	310.2
167	75.0	120.0	150.0	320.4
168	75.0	125.0	150.0	330.4

Replace Using Mean, Median, or Mode

In [ ]:
```
A common way to replace empty cells, is to calculate the mean, median or mode value of the column.

Pandas uses the follwing function:

mean()   : Method to calculate mean [ the average value (the sum of all values divided by number of values)]
```

```
median() : Method to calculate median [the value in the middle, after you have sorted all values ascending]
mode()   : Method to calculate mode  [the value that appears most frequently]
```

In [16]:
```
# Calculate the Median, and replace any empty values with it.

df = pd.read_excel("data.xlsx")
n = df["Calories"].median()
#m = df["Calories"].mean()
df["Duration"].fillna(n,inplace=True) #Replace empty cell with median value
#df["Duration"].fillna(m,inplace=True) #Replace empty cell with mean value
print(df.to_string())

#Notice in the result: empty cells of column "Calories" will get median value '317.85'
```

	Duration	Pulse	Maxpulse	Calories
0	317.85	110.0	130.0	409.1
1	60.00	117.0	145.0	479.0
2	60.00	103.0	NaN	340.0
3	45.00	109.0	175.0	282.4
4	317.85	117.0	148.0	406.0
5	60.00	102.0	127.0	300.0
6	60.00	110.0	136.0	374.0
7	45.00	104.0	134.0	253.3
8	30.00	NaN	133.0	195.1
9	60.00	98.0	124.0	269.0
10	60.00	103.0	147.0	NaN
11	60.00	100.0	120.0	250.7
12	60.00	106.0	128.0	345.3
13	60.00	104.0	132.0	379.3
14	60.00	98.0	123.0	275.0
15	60.00	98.0	120.0	215.2
16	60.00	100.0	120.0	300.0
17	45.00	90.0	112.0	NaN
18	60.00	103.0	123.0	323.0
19	45.00	97.0	125.0	243.0
20	60.00	108.0	131.0	364.2
21	45.00	100.0	119.0	282.0
22	60.00	130.0	101.0	300.0
23	45.00	105.0	132.0	246.0
24	60.00	102.0	126.0	334.5
25	60.00	100.0	120.0	250.0
26	60.00	92.0	118.0	241.0
27	60.00	103.0	132.0	NaN
28	60.00	100.0	132.0	280.0
29	60.00	102.0	129.0	380.3
30	60.00	92.0	115.0	243.0
31	45.00	90.0	112.0	180.1
32	60.00	101.0	124.0	299.0
33	60.00	93.0	113.0	223.0
34	60.00	107.0	136.0	361.0
35	60.00	114.0	140.0	415.0
36	60.00	102.0	127.0	300.0
37	60.00	100.0	120.0	300.0
38	60.00	100.0	120.0	300.0
39	45.00	104.0	129.0	266.0
40	45.00	90.0	112.0	180.1
41	60.00	98.0	126.0	286.0
42	60.00	100.0	122.0	329.4
43	60.00	111.0	138.0	400.0

44	60.00	111.0	131.0	397.0
45	60.00	99.0	119.0	273.0
46	60.00	109.0	153.0	387.6
47	45.00	111.0	136.0	300.0
48	45.00	108.0	129.0	298.0
49	60.00	111.0	139.0	397.6
50	60.00	107.0	136.0	380.2
51	80.00	123.0	146.0	643.1
52	60.00	106.0	130.0	263.0
53	60.00	118.0	151.0	486.0
54	30.00	136.0	175.0	238.0
55	60.00	121.0	146.0	450.7
56	60.00	118.0	121.0	413.0
57	45.00	115.0	144.0	305.0
58	20.00	153.0	172.0	226.4
59	45.00	123.0	152.0	321.0
60	210.00	108.0	160.0	1376.0
61	160.00	110.0	137.0	1034.4
62	160.00	109.0	135.0	853.0
63	45.00	118.0	141.0	341.0
64	20.00	110.0	130.0	131.4
65	180.00	90.0	130.0	800.4
66	150.00	105.0	135.0	873.4
67	150.00	107.0	130.0	816.0
68	20.00	106.0	136.0	110.4
69	300.00	108.0	143.0	1500.2
70	150.00	97.0	129.0	1115.0
71	60.00	109.0	153.0	387.6
72	90.00	100.0	127.0	700.0
73	150.00	97.0	127.0	953.2
74	45.00	114.0	146.0	304.0
75	90.00	98.0	125.0	563.2
76	45.00	105.0	134.0	251.0
77	45.00	110.0	141.0	300.0
78	120.00	100.0	130.0	500.4
79	270.00	100.0	131.0	1729.0
80	30.00	159.0	182.0	319.2
81	45.00	149.0	169.0	344.0
82	30.00	103.0	139.0	151.1
83	120.00	100.0	130.0	500.0
84	45.00	100.0	120.0	225.3
85	30.00	NaN	170.0	300.0
86	45.00	102.0	136.0	234.0
87	120.00	100.0	157.0	1000.1
88	45.00	129.0	103.0	242.0

89	20.00	83.0	107.0	50.3
90	180.00	101.0	127.0	600.1
91	45.00	107.0	137.0	NaN
92	30.00	90.0	107.0	105.3
93	15.00	80.0	100.0	50.5
94	20.00	150.0	171.0	127.4
95	20.00	151.0	NaN	229.4
96	30.00	95.0	128.0	128.2
97	25.00	152.0	168.0	244.2
98	30.00	109.0	131.0	188.2
99	90.00	93.0	124.0	604.1
100	20.00	95.0	112.0	77.7
101	90.00	90.0	110.0	500.0
102	90.00	90.0	100.0	500.0
103	90.00	90.0	NaN	500.4
104	30.00	92.0	108.0	92.7
105	30.00	93.0	128.0	124.0
106	180.00	90.0	120.0	800.3
107	30.00	90.0	120.0	86.2
108	90.00	90.0	120.0	500.3
109	210.00	137.0	184.0	1860.4
110	60.00	NaN	124.0	325.2
111	45.00	107.0	124.0	275.0
112	15.00	124.0	139.0	124.2
113	45.00	100.0	120.0	225.3
114	60.00	108.0	131.0	367.6
115	60.00	108.0	151.0	351.7
116	60.00	116.0	141.0	443.0
117	60.00	97.0	122.0	277.4
118	60.00	105.0	125.0	NaN
119	60.00	103.0	124.0	332.7
120	30.00	112.0	137.0	193.9
121	317.85	100.0	120.0	100.7
122	60.00	119.0	169.0	336.7
123	60.00	107.0	NaN	344.9
124	60.00	111.0	151.0	368.5
125	60.00	98.0	122.0	271.0
126	60.00	97.0	124.0	275.3
127	60.00	109.0	127.0	382.0
128	90.00	99.0	125.0	466.4
129	60.00	114.0	151.0	384.0
130	60.00	104.0	134.0	342.5
131	60.00	107.0	138.0	357.5
132	60.00	103.0	133.0	335.0
133	60.00	106.0	132.0	327.5

134	60.00	103.0	136.0	339.0
135	20.00	136.0	156.0	189.0
136	45.00	117.0	143.0	317.7
137	45.00	115.0	137.0	318.0
138	45.00	113.0	138.0	308.0
139	20.00	141.0	162.0	222.4
140	60.00	108.0	135.0	390.0
141	60.00	97.0	127.0	NaN
142	45.00	100.0	120.0	250.4
143	45.00	122.0	149.0	335.4
144	60.00	136.0	170.0	470.2
145	45.00	106.0	126.0	270.8
146	60.00	107.0	136.0	400.0
147	60.00	112.0	146.0	361.9
148	30.00	103.0	127.0	185.0
149	60.00	110.0	150.0	409.4
150	317.85	106.0	134.0	343.0
151	60.00	NaN	129.0	353.2
152	60.00	109.0	138.0	374.0
153	30.00	NaN	167.0	275.8
154	60.00	105.0	128.0	328.0
155	60.00	111.0	151.0	NaN
156	60.00	97.0	131.0	270.4
157	60.00	100.0	120.0	270.4
158	60.00	114.0	150.0	382.8
159	30.00	80.0	120.0	240.9
160	30.00	85.0	120.0	250.4
161	45.00	NaN	130.0	260.4
162	45.00	95.0	130.0	270.0
163	317.85	100.0	140.0	280.9
164	60.00	105.0	NaN	290.8
165	60.00	110.0	145.0	300.0
166	60.00	115.0	145.0	310.2
167	75.00	120.0	150.0	320.4
168	75.00	125.0	150.0	330.4

Cleaning Data of Wrong Format

In [ ]:
```
Cells with data of wrong format can make it difficult to analyze data.
There are two waus to remove datas with wrong format:

    1. Remove the entire rows that contain data of wrong format.
    2. Convert all cells in the columns into the same format.
```

In [25]:
```
# Convert all cells in the columns into the same format.
# In our Data Frame, we have two cells with the wrong format.
#Check out row 3 and 4, the 'date' column should be a string that represents a date:

df = pd.read_csv("Weather_data.csv")
print(df)
```

```
            Date  Temperature  Windspeed  Event
0   '2020/12/01'           32          6   Rain
1   '2020/12/02'           35          7  Sunny
2   '2020/12/03'           28          2   Snow
3       20201204           24          7   Snow
4            NaN           32          4   Rain
5   '2020/12/06'           31          2  Sunny
```

Convert all cells in the columns into the same format.

In [26]:
```
# To convert all cells in the 'Date' column into dates.
# Pandas has a to_datetime() method for this.


df = pd.read_csv('Weather_data.csv')

df['Date'] = pd.to_datetime(df['Date'])
print(df.to_string())
```

```
        Date  Temperature  Windspeed  Event
0 2020-12-01           32          6   Rain
1 2020-12-02           35          7  Sunny
2 2020-12-03           28          2   Snow
3 2020-12-04           24          7   Snow
4        NaT           32          4   Rain
5 2020-12-06           31          2  Sunny
```

In [ ]:
```
As you can see from the result:

The date in row 3 was fixed,
row 4 got a NaT (Not a Time) value, in other words an empty value.

One way to deal with empty values is simply removing the entire row.
```

Removing Rows with wrong data

dropna() : Method to remove rows having null values.

In [27]:
```
#Remove rows with a NULL value in the "Date" column:

df.dropna(subset=['Date'], inplace = True)
print(df.to_string()) # row with date value null (Nat) has been removed.
```

```
        Date  Temperature  Windspeed  Event
0 2020-12-01           32          6   Rain
1 2020-12-02           35          7  Sunny
2 2020-12-03           28          2   Snow
3 2020-12-04           24          7   Snow
5 2020-12-06           31          2  Sunny
```

Removing Duplicates

In [ ]:
```
Duplicate rows are rows that have been registered more than one time.

duplicated()       : Method to discover duplicate rows. This method returns True for every row that is a duplicate,
                     othwerwise False.

drop_duplicates()  : Method to remove duplicate rows.
```

In [37]:
```
#Discover duplicates from a given dataframe.

df = pd.read_csv("data.csv")
df1 = df.duplicated() # True - duplicated row , False - not a duplicate row
print(df1.to_string())
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7      True
8     False
9     False
10     True
11     True
12     True
13    False
14    False
15    False
16    False
17     True
18    False
19     True
20    False
21     True
22    False
23    False
24    False
25    False
26    False
27     True
28     True
29    False
30     True
31    False
32    False
33    False
34    False
35    False
36    False
37    False
38    False
39    False
40    False
41    False
42    False
43    False
44    False
```

```
45    False
46     True
47     True
48     True
49    False
50     True
51    False
52    False
53    False
54    False
55    False
56    False
57    False
58    False
59    False
60    False
61    False
62    False
63    False
64    False
65    False
66    False
67    False
68    False
69    False
70    False
71    False
72    False
73    False
74    False
75    False
76    False
77    False
78    False
79    False
80    False
81     True
82    False
83    False
84    False
85    False
86    False
87    False
88    False
89    False
```

```
90     False
91     False
92     False
93     False
94     False
95     False
96     False
97     False
98     False
99     False
100    False
101    False
102    False
103    False
104    False
105    False
106    False
107    False
108    False
109    False
110    False
111    False
112    False
113    False
114    False
115    False
116    False
117    False
118    False
119    False
120    False
121    False
122    False
123     True
124    False
125    False
126    False
127    False
128    False
129    False
130    False
131    False
132    False
133    False
134    False
```

```
135    False
136    False
137    False
138    False
139    False
140    False
141    False
142    False
143    False
144    False
145    False
146    False
147    False
148    False
149    False
150    False
151    False
152    False
153    False
154    False
155    False
156    False
157    False
158    False
159    False
160    False
161    False
162    False
163    False
164    False
165     True
166    False
167    False
168    False
169    False
170    False
171    False
172    False
173    False
174    False
175    False
176    False
177    False
178    False
```

In [41]:
```
#Remove duplicated rows

df = pd.read_csv("data.csv")
df.drop_duplicates(inplace = True) # Dropped the duplicated rows
df1 = df.duplicated() # (Check again after dropping the duplicated rows, True - duplicated row , False - not a duplicat
print(df1.to_string()) # output shows only 'False' value means no duplicated rows.
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
8     False
9     False
13    False
14    False
15    False
16    False
18    False
20    False
22    False
23    False
24    False
25    False
26    False
29    False
31    False
32    False
33    False
34    False
35    False
36    False
37    False
38    False
39    False
40    False
41    False
42    False
43    False
44    False
45    False
49    False
51    False
52    False
53    False
54    False
55    False
56    False
57    False
58    False
```

```
59     False
60     False
61     False
62     False
63     False
64     False
65     False
66     False
67     False
68     False
69     False
70     False
71     False
72     False
73     False
74     False
75     False
76     False
77     False
78     False
79     False
80     False
82     False
83     False
84     False
85     False
86     False
87     False
88     False
89     False
90     False
91     False
92     False
93     False
94     False
95     False
96     False
97     False
98     False
99     False
100    False
101    False
102    False
103    False
104    False
```

```
105    False
106    False
107    False
108    False
109    False
110    False
111    False
112    False
113    False
114    False
115    False
116    False
117    False
118    False
119    False
120    False
121    False
122    False
124    False
125    False
126    False
127    False
128    False
129    False
130    False
131    False
132    False
133    False
134    False
135    False
136    False
137    False
138    False
139    False
140    False
141    False
142    False
143    False
144    False
145    False
146    False
147    False
148    False
149    False
150    False
```

```
151    False
152    False
153    False
154    False
155    False
156    False
157    False
158    False
159    False
160    False
161    False
162    False
163    False
164    False
166    False
167    False
168    False
169    False
170    False
171    False
172    False
173    False
174    False
175    False
176    False
177    False
178    False
```

Group By function

Pandas dataframe.groupby() function is used to split the data into groups based on some criteria. This function retuns a GroupBy object.

In [42]:
```
df = pd.read_csv("weather2.csv")
print(df)
```

	date	city	temperature	humidity
0	5/1/2017	new york	65	56
1	5/1/2017	new york	61	54
2	5/2/2017	new york	70	60
3	5/2/2017	new york	72	62
4	5/1/2017	mumbai	75	80
5	5/1/2017	mumbai	78	83
6	5/2/2017	mumbai	82	85
7	5/2/2017	mumbai	80	26

In [46]:
```
#Groupby() method will create a object
df1 = df.groupby("city") #create a groupby based on column 'city'
print(df1)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000017A9F06CEE0>
```

In [48]:
```
# get_group() method to retrieve a specified group from column 'city' from "weather2.csv" dataframe.

print(df1.get_group('mumbai')) #Display row that has 'city' as 'mumbai'
```

	date	city	temperature	humidity
4	5/1/2017	mumbai	75	80
5	5/1/2017	mumbai	78	83
6	5/2/2017	mumbai	82	85
7	5/2/2017	mumbai	80	26

In [50]:
```
#Accessing a group
print(df1.get_group('new york')['temperature']) #This displays only temperatue column value of 'new york' city
```

```
0    65
1    61
2    70
3    72
Name: temperature, dtype: int64
```

In [51]:
```
# sum() method to return the sum values of all numerical columns.
print(df1.sum())
```

city	temperature	humidity
mumbai	315	274
new york	268	232

```
C:\Users\rajee\AppData\Local\Temp\ipykernel_4968\129731223.py:1: FutureWarning: The default value of numeric_only in Da
taFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only
or select only columns which should be valid for the function.
  print(df1.sum())
```

In [52]:
```
#mean() method to return the sum values of all numerical columns.

print(df1.mean())
```

```
          temperature  humidity
city
mumbai          78.75      68.5
new york        67.00      58.0
```

```
C:\Users\rajee\AppData\Local\Temp\ipykernel_4968\281642230.py:3: FutureWarning: The default value of numeric_only in Da
taFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only
or select only columns which should be valid for the function.
  print(df1.mean())
```

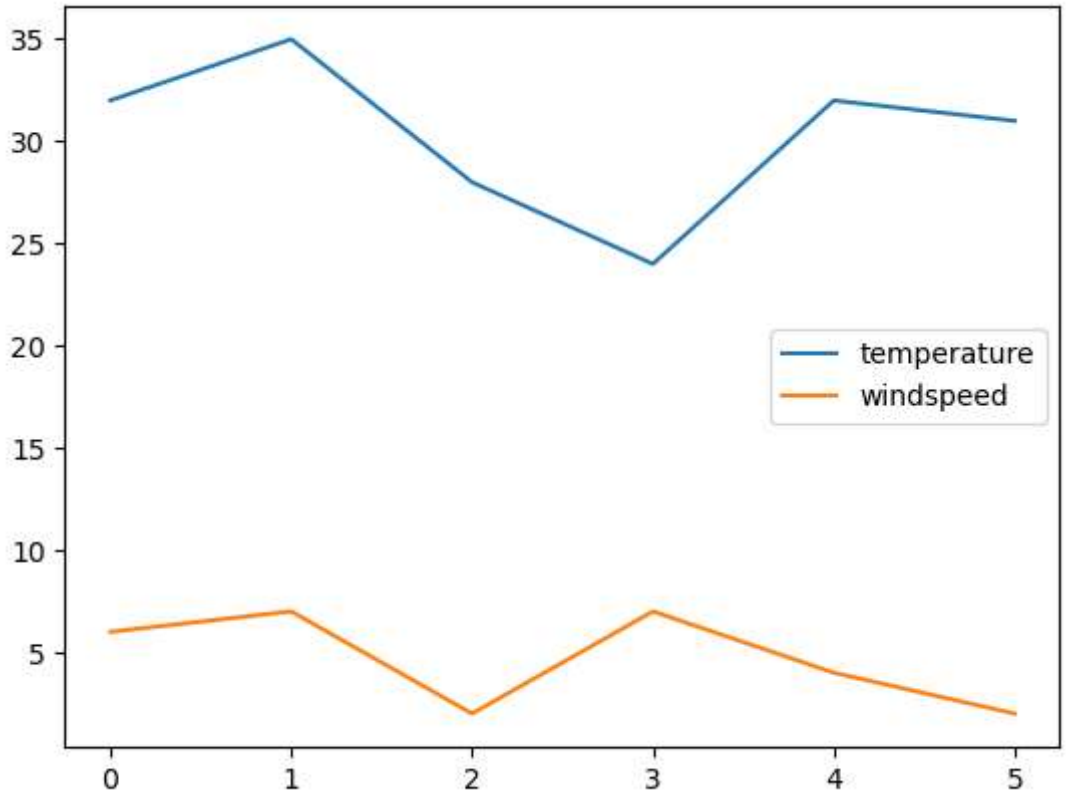
Pandas Plotting

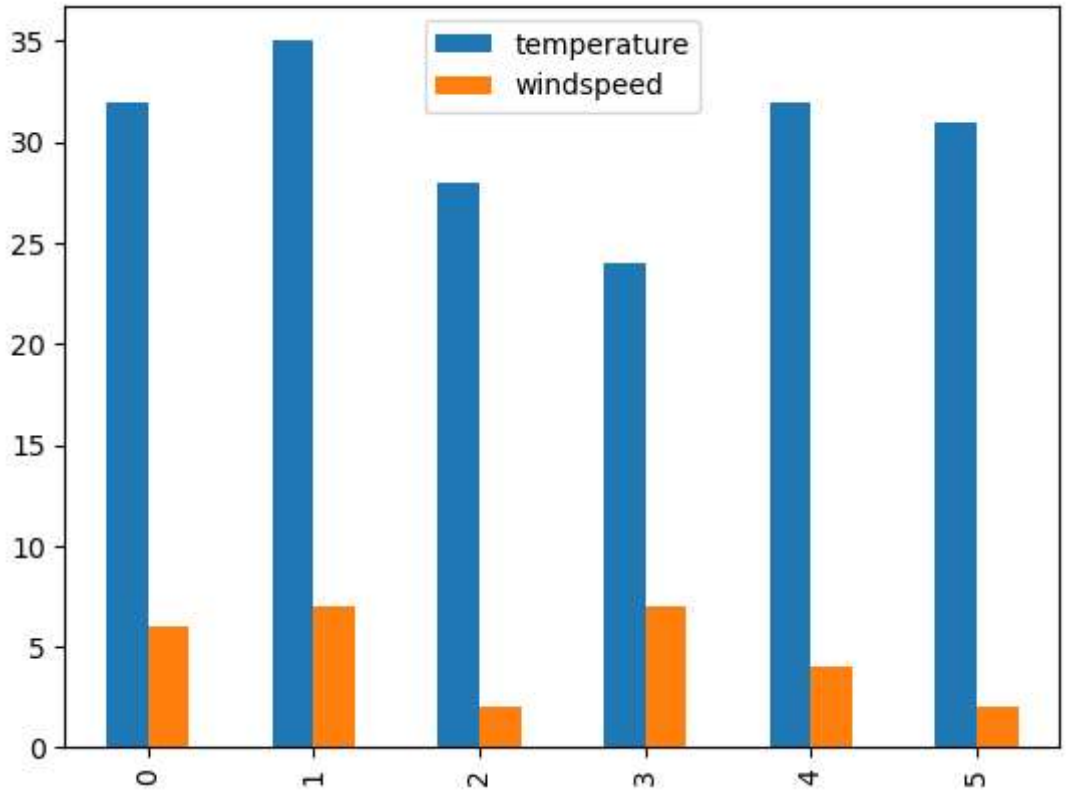
In [ ]:
```
Plot()  :Pandas uses plot() method to plot diagrams.

We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen.


kind    : Argument used to specify the type of chart we want to plot. The values of kind can be scatter plot,Histogram
          It gives Line chart by default.
```

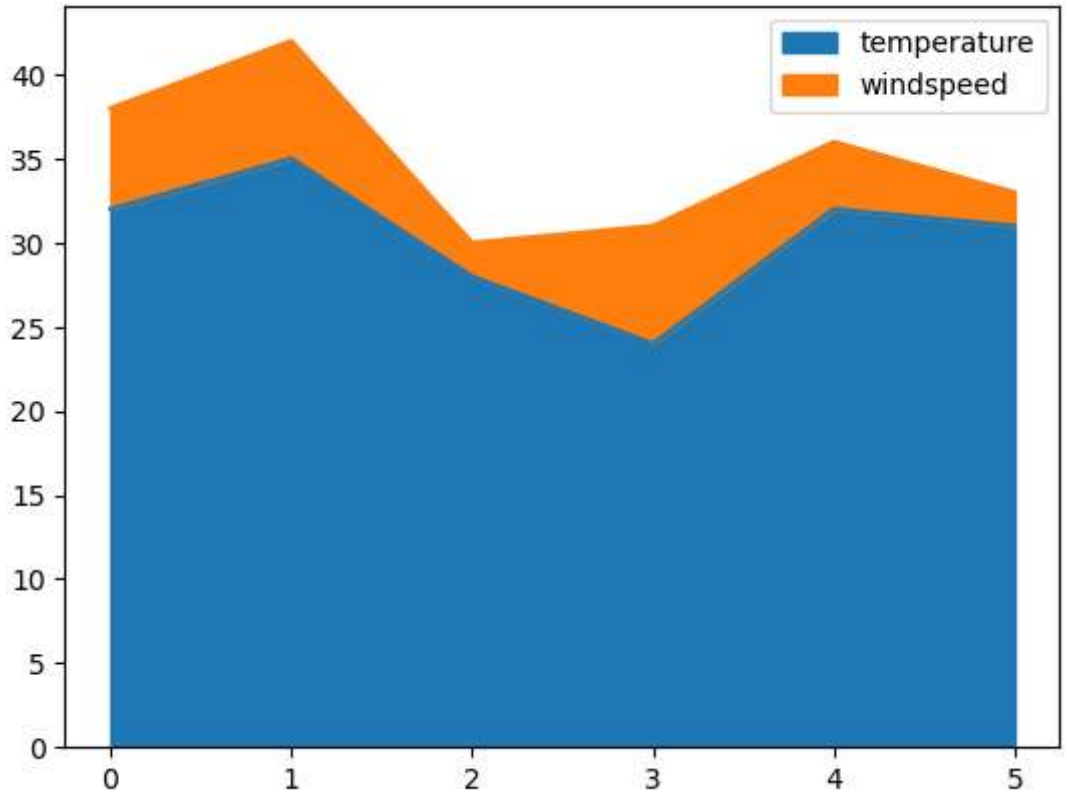
In [2]:
```
#Import pyplot from Matplotlib and visualize our DataFrame:

import matplotlib.pyplot as plt
```

Plot a DataFrame

In [15]:
```
#To plot a DataFrame

df = pd.read_csv("Weather_data.csv")
df.plot() #By default it shows line chart
df.plot(kind="bar") #If you want bar chart
df.plot(kind="area") #If you want area chart
plt.show()
```

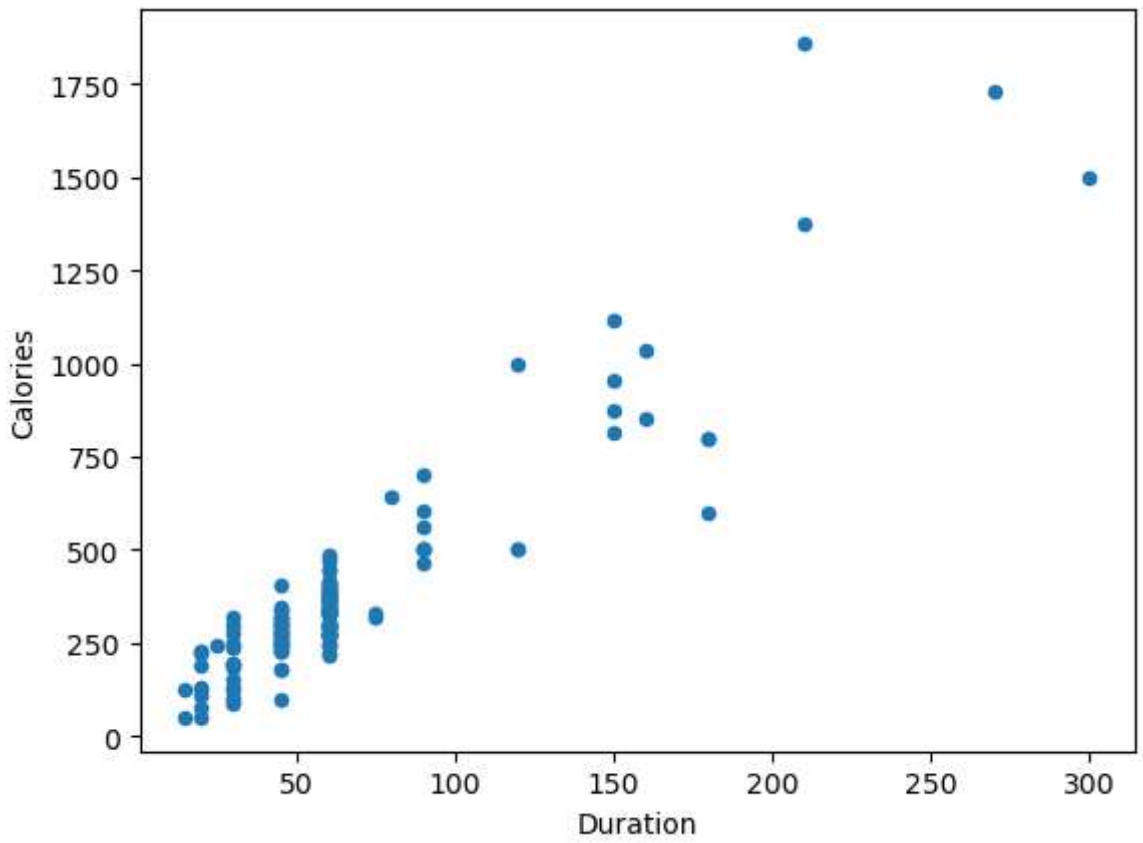

35
30
25
20
15
10
5
0
1
2
3
4
5
temperature
windspeed


temperature
windspeed
35
30
25
20
15
10
5
0
0
1
2
3
4
5



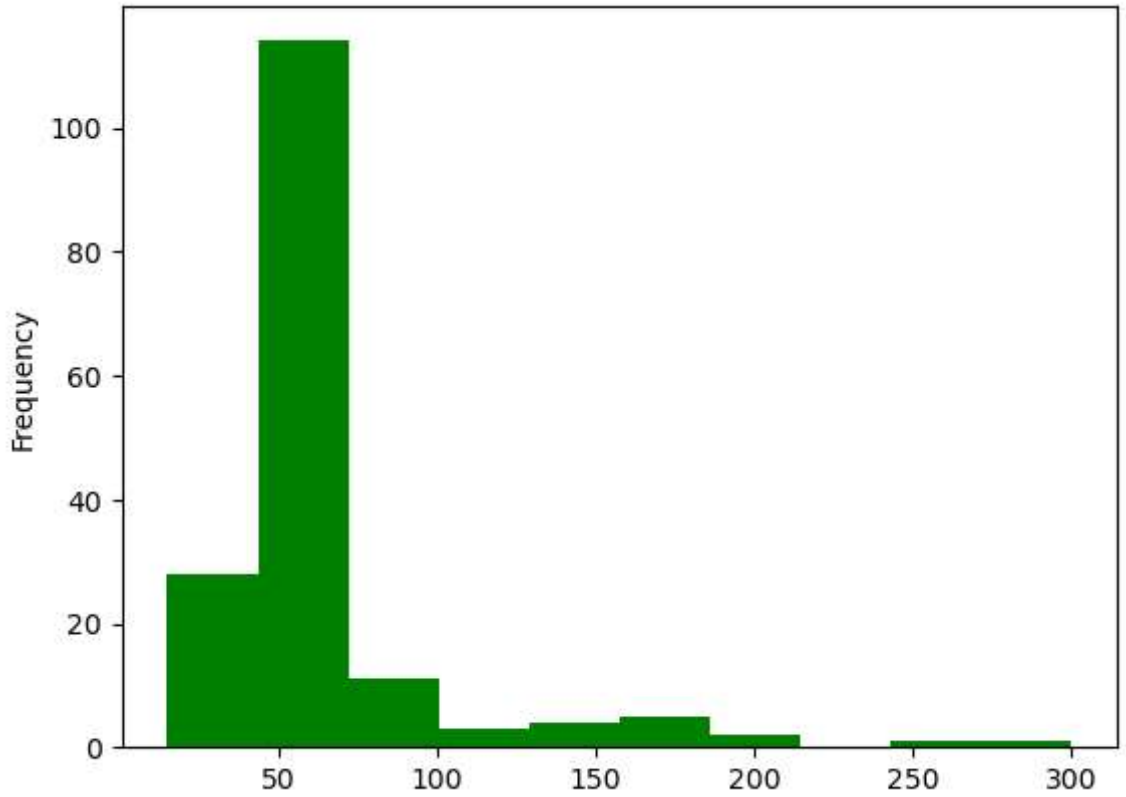
Scatter Plot

In [20]:
```
#Plot a scatter plot by loading csv file as DataFrame
#A scatter plot needs an x- and a y-axis.

df = pd.read_csv('data.csv')

df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')

plt.show()
```



Histogram

In [22]:
```
#Plot a Histogram
# A histogram needs only one column.
# Here,we will use the "Duration" column to create the histogram.

#A histogram shows us the frequency of each interval, e.g. how many workouts lasted between 50 and 60 minutes?
#The histogram tells us that there were over 100 workouts that lasted between 50 and 60 minute

df["Duration"].plot(kind = 'hist',color="green")
plt.show()
```



END