In [1]:
```
empty_dictionary = {}
print(type(empty_dictionary))
```

```
<class 'dict'>
```

In [2]:
```
bio_data = {'Name': 'Bob Marley', 'Age':35, 'Height':"5.6 ft", 'Hobby': 'Music'}
print(bio_data)
```

```
{'Name': 'Bob Marley', 'Age': 35, 'Height': '5.6 ft', 'Hobby': 'Music'}
```

In [4]:
```
credentials = { 'UserA' : 'wkliopnc' , 'UserB': 98760 , 'UserC' :98760 }
```

In [5]:
```
hobby = bio_data['Hobby']
print(hobby)
```

```
Music
```

In [6]:
```
age = bio_data['Age']
print(age)
```

```
35
```

In [7]:
```
age = bio_data.get('Age')
print(age)
```

```
35
```

In [8]:
```
profession  =  bio_data.get('Profession','NA')
print(profession)
```

```
NA
```

In [9]:
```
bio_data['Age'] = 36
print(bio_data)
```

```
{'Name': 'Bob Marley', 'Age': 36, 'Height': '5.6 ft', 'Hobby': 'Music'}
```

In [10]:
```
#add a key, val
bio_data['Profession'] = 'Singer'
print(bio_data)
```

```
{'Name': 'Bob Marley', 'Age': 36, 'Height': '5.6 ft', 'Hobby': 'Music', 'Profes
sion': 'Singer'}
```

In [11]:
```
print('Profession' in bio_data)
```

```
True
```

In [12]:

```
#get list of keys
print(list(bio_data.keys()))

#get list of values
print(list(bio_data.values()))
```

```
['Name', 'Age', 'Height', 'Hobby', 'Profession']
['Bob Marley', 36, '5.6 ft', 'Music', 'Singer']
```

In [13]:

```
new_dictionary = dict(Country='Jamaica', Songs=['One Love','Misty Morning'])
```

In [14]:

```
bio_data.update(new_dictionary)
print(bio_data)
```

```
{'Name': 'Bob Marley', 'Age': 36, 'Height': '5.6 ft', 'Hobby': 'Music', 'Profes
sion': 'Singer', 'Country': 'Jamaica', 'Songs': ['One Love', 'Misty Morning']}
```

In [15]:

```
del bio_data['Songs']
print(bio_data)
```

```
{'Name': 'Bob Marley', 'Age': 36, 'Height': '5.6 ft', 'Hobby': 'Music', 'Profes
sion': 'Singer', 'Country': 'Jamaica'}
```

In [16]:

```
students_data = { 1:['Shivam Bansal', 24] , 2:['Udit Bansal',25], 3:['Sonam Gupta

print(students_data)
```

```
{1: ['Shivam Bansal', 24], 2: ['Udit Bansal', 25], 3: ['Sonam Gupta', 26], 4:
['Saif Ansari', 24], 5: ['Huzefa Calcuttawala', 27]}
```

In [17]:

```
print(len(students_data))
```

```
5
```

In [18]:

```
#see all the details of students.
print(list(students_data.values()))
```

```
[['Shivam Bansal', 24], ['Udit Bansal', 25], ['Sonam Gupta', 26], ['Saif Ansar
i', 24], ['Huzefa Calcuttawala', 27]]
```

In [19]:

```
students_data[6] = ['Manasi Sharma', 22]
print(students_data)
```

```
{1: ['Shivam Bansal', 24], 2: ['Udit Bansal', 25], 3: ['Sonam Gupta', 26], 4:
['Saif Ansari', 24], 5: ['Huzefa Calcuttawala', 27], 6: ['Manasi Sharma', 22]}
```

In [20]:

```
del students_data[2]
print(students_data)
```

```
{1: ['Shivam Bansal', 24], 3: ['Sonam Gupta', 26], 4: ['Saif Ansari', 24], 5:
['Huzefa Calcuttawala', 27], 6: ['Manasi Sharma', 22]}
```

In [21]:

```
print(list(students_data.keys()))
```

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

In [22]:

```
import math
math.sqrt(81)
```

Out[22]:

```
9.0
```

In [23]:

```
math.factorial(5)
```

Out[23]:

```
120
```

In [24]:

```
n=5
k=3
math.factorial(n)/(math.factorial(k)*math.factorial(n-k))
math.factorial(n)//(math.factorial(k)*math.factorial(n-k))
```

Out[24]:

```
10
```

In [25]:

```
from math import factorial as fac
fac(15)
```

Out[25]:

```
1307674368000
```

In [26]:

```
h = 50
if h > 50:
    print ('No')
else:
    print('Yes')
```

```
Yes
```

In [1]:

```
c = 5
while c != 0:
    print(c)
    c -= 1
```

```
5
4
3
2
1
```

In [ ]:
```
while True:
    response = input()
    if int(response) % 7 == 0:
        break
```

In [ ]:
```
def hello(who):
    print('hello {}'.format(who))
```

In [3]:
```
print("first" "second")
```

```
firstsecond
```

In [4]:
```
s ='PARROT'
s[4]
```

Out[4]:
```
'O'
```

In [5]:
```
help(str)
```

```
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |  
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |  
 |  Methods defined here:
 |  
 |  __add__(self, value, /)
 |      Return self+value.
 |  
```

In [7]:
```
s = 'parrot'
s.capitalize()
```

Out[7]:
```
'Parrot'
```

In [8]:
```
list("Characters")
```

Out[8]:
```
['C', 'h', 'a', 'r', 'a', 'c', 't', 'e', 'r', 's']
```

In [9]:

```
dict("characters")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-03b814add21b> in <module>()
----> 1 dict("characters")

ValueError: dictionary update sequence element #0 has length 1; 2 is required
```

In [3]:

```
from urllib.request import urlopen
with urlopen('http://sixty-north.com/c/t.txt') as story:
    story_words = []
    for line in story:
        line_words = line.split()
        for word in line_words:
            story_words.append(word)

print(story_words)
```

```
---------------------------------------------------------------------------
TimeoutError                              Traceback (most recent call last)
~\AppData\Local\anaconda3\lib\urllib\request.py in do_open(self, http_class, re
q, **http_conn_args)
   1317                 h.request(req.get_method(), req.selector, req.data, hea
ders,
-> 1318                           encode_chunked=req.has_header('Transfer-encod
ing'))
   1319             except OSError as err: # timeout error

~\AppData\Local\anaconda3\lib\http\client.py in request(self, method, url, bod
y, headers, encode_chunked)
   1238         """Send a complete request to the server."""
-> 1239         self._send_request(method, url, body, headers, encode_chunked)
   1240

~\AppData\Local\anaconda3\lib\http\client.py in _send_request(self, method, ur
l, body, headers, encode_chunked)
   1284             body = _encode(body, 'body')
-> 1285         self.endheaders(body, encode_chunked=encode_chunked)
   1286

~\AppData\Local\anaconda3\lib\http\client.py in endheaders(self, message_body,
 encode_chunked)
   1233             raise CannotSendHeader()
-> 1234         self._send_output(message_body, encode_chunked=encode_chunked)
   1235

~\AppData\Local\anaconda3\lib\http\client.py in _send_output(self, message_bod
y, encode_chunked)
   1025         del self._buffer[:]
-> 1026         self.send(msg)
   1027

~\AppData\Local\anaconda3\lib\http\client.py in send(self, data)
    963             if self.auto_open:
--> 964                 self.connect()
    965             else:

~\AppData\Local\anaconda3\lib\http\client.py in connect(self)
    935         self.sock = self._create_connection(
--> 936             (self.host,self.port), self.timeout, self.source_address)
    937         self.sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)

~\AppData\Local\anaconda3\lib\socket.py in create_connection(address, timeout,
 source_address)
```

```
    723     if err is not None:
--> 724         raise err
    725     else:

~\AppData\Local\anaconda3\lib\socket.py in create_connection(address, timeout,
 source_address)
    712                 sock.bind(source_address)
--> 713             sock.connect(sa)
    714             # Break explicitly a reference cycle

TimeoutError: [WinError 10060] A connection attempt failed because the connecte
d party did not properly respond after a period of time, or established connect
ion failed because connected host has failed to respond

During handling of the above exception, another exception occurred:

URLError                                  Traceback (most recent call last)
<ipython-input-3-b58b5af38bcc> in <module>()
      1 from urllib.request import urlopen
----> 2 with urlopen('http://sixty-north.com/c/t.txt') as story:
      3     story_words = []
      4     for line in story:
      5         line_words = line.split()

~\AppData\Local\anaconda3\lib\urllib\request.py in urlopen(url, data, timeout,
 cafile, capath, cadefault, context)
    221     else:
    222         opener = _opener
--> 223     return opener.open(url, data, timeout)
    224
    225 def install_opener(opener):

~\AppData\Local\anaconda3\lib\urllib\request.py in open(self, fullurl, data, ti
meout)
    524             req = meth(req)
    525
--> 526         response = self._open(req, data)
    527
    528         # post-process response

~\AppData\Local\anaconda3\lib\urllib\request.py in _open(self, req, data)
    542         protocol = req.type
    543         result = self._call_chain(self.handle_open, protocol, protocol
 +
--> 544                                   '_open', req)
    545         if result:
    546             return result

~\AppData\Local\anaconda3\lib\urllib\request.py in _call_chain(self, chain, kin
d, meth_name, *args)
    502         for handler in handlers:
    503             func = getattr(handler, meth_name)
--> 504             result = func(*args)
    505             if result is not None:
    506                 return result

~\AppData\Local\anaconda3\lib\urllib\request.py in http_open(self, req)
```

```
   1344 
   1345     def http_open(self, req):
-> 1346         return self.do_open(http.client.HTTPConnection, req)
   1347 
   1348     http_request = AbstractHTTPHandler.do_request_

~\AppData\Local\anaconda3\lib\urllib\request.py in do_open(self, http_class, re
q, **http_conn_args)
   1318                                   encode_chunked=req.has_header('Transfer-encod
ing'))
   1319             except OSError as err: # timeout error
-> 1320                 raise URLError(err)
   1321             r = h.getresponse()
   1322         except:

URLError: <urlopen error [WinError 10060] A connection attempt failed because t
he connected party did not properly respond after a period of time, or establis
hed connection failed because connected host has failed to respond>
```

In [2]:

```
# creating a 1d arraya using a single list
import numpy as np
array_1d = np.array ([2, 4, 5, 6, 7, 9])
print(array_1d)
print(type(array_1d))
```

```
[2 4 5 6 7 9]
<class 'numpy.ndarray'>
```

In [3]:

```
# creating a 1d arraya using a single list
import numpy as np
array_2d = np.array ([[2, 4, 5],[ 6, 7, 9]])
print(array_2d)
print(type(array_2d))
```

```
[[2 4 5]
 [6 7 9]]
<class 'numpy.ndarray'>
```

In [6]:

```
# Create a 3*3 array using list_1 = [1,2,3] list_2 = [4,5,6] list_3 = [7,8,9]
import ast,sys



import numpy as np
array_1 = np.array ([[1, 2, 3],[4, 5, 6],[7, 8, 9]])

print(array_1)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [15]:

```
# Create a 3*3 array using list_1 = [1,2,3] list_2 = [4,5,6] list_3 = [7,8,9]
import ast,sys
#input_str = sys.stdin.read()
#input_list = ast.literal_eval(input_str)
#list_1 = input_list[0]
#list_2 = input_list[1]
#list_3 = input_list[2]

list_1 = [1,2,3]
list_2 = [4,5,6]
list_3 = [7,8,9]

import numpy as np
array_1 = np.array([list_1,list_2,list_3])
array_2 = np.array(list_1)
array_3 = np.array(list_2)
array_4 = array_2*array_3
array_5 = array_2**2
array_6 = array_2+array_3
print(array_1)
print(array_2)
print(array_3)
print(array_4)
print(array_5)
print(array_6)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[1 2 3]
[4 5 6]
[ 4 10 18]
[1 4 9]
[5 7 9]
```

In [19]:

```
#Perform an element-wise multiplication using list_1 = [2,3,4,5] list_2 = [7,8,9,
#Hint: Convert the list to an array and after multiplication convert it back to a

list_1 = [2,3,4,5]
list_2 = [7,8,9,6]
import numpy as np
array_1 = np.array(list_1)#Type your answer here
array_2 = np.array(list_2)#Type your answer here
array_3 = array_1*array_2#Type your answer here

print(list(array_3))

print(array_3)
```

```
[14, 24, 36, 30]
[14 24 36 30]
```

In [38]:

```
import numpy as np
# create 5X3 2 dimensional array of ones with default datatype float64.
array_1 = np.ones((5,3))
# create 5X3 2 dimensional array of ones with default datatype integer.
array_6 = np.ones((5,3),dtype = np.int)
array_2 = np.zeros(3)
array_3 = np.random.random(5)
array_4 = np.arange(10,100,5)
# array of 25 between 15 and 18
array_5 = np.linspace(15,18,25)
print(array_1)
print(array_2)
print(array_3)
print(array_4)
print(array_6)
print(array_5)
#Create an array of first 10 multiples of 5 using the 'arange' function.
array_7 = np.arange(5,55,5)
print(array_7)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[0. 0. 0.]
[0.83275204 0.86647743 0.29780628 0.979333   0.70985028]
[10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95]
[[1 1 1]
 [1 1 1]
 [1 1 1]
 [1 1 1]
 [1 1 1]]
[15.     15.125 15.25   15.375 15.5    15.625 15.75   15.875 16.     16.125
 16.25   16.375 16.5    16.625 16.75   16.875 17.     17.125 17.25   17.375
 17.5    17.625 17.75   17.875 18.   ]
[ 5 10 15 20 25 30 35 40 45 50]
```

In [45]:

```
#Create an array using list list_1 = [10,11,12,13] and list_2 = [15,12,13,14]
#and print the shape and dimension of the array created.

list_1 = [10,11,12,13]
list_2 = [15,12,13,14]
import numpy as np
array_1 = np.array([list_1,list_2])
print(array_1)
#array_1.shape
print(array_1.ndim)
print(array_1.shape)
```

```
[[10 11 12 13]
 [15 12 13 14]]
2
(2, 4)
```

In [54]:
```
# Create a random large array and process it.
import numpy as np
array_1= np.random.random((1000,300))

# print first row
print(array_1[1,])
print(array_1.shape)
print(array_1.dtype)
print(array_1.itemsize)
print(array_1.ndim)
```

```
[0.80649677 0.55162442 0.44651401 0.47701209 0.42859912 0.77975405
 0.46591274 0.27689779 0.80947984 0.21662684 0.3655462  0.54769068
 0.52908255 0.05099134 0.39619353 0.51834591 0.0426613  0.57611871
 0.96735961 0.47821094 0.98598753 0.69511954 0.7586764  0.85732279
 0.88428747 0.3557488  0.15012623 0.72632942 0.03036386 0.12867723
 0.75116441 0.32593737 0.01207448 0.03678673 0.47678584 0.12768942
 0.58214378 0.54127876 0.79091173 0.59456168 0.86049421 0.49093159
 0.56046316 0.30818558 0.53320614 0.43469991 0.39360042 0.07827278
 0.68091167 0.70007023 0.59198383 0.7159765  0.94152995 0.20837323
 0.88920517 0.73631415 0.51757972 0.86433867 0.99739276 0.87181265
 0.60934817 0.1415654  0.26862255 0.86607782 0.8037377  0.15251813
 0.57528889 0.71268161 0.59574215 0.14614738 0.8869052  0.79427184
 0.62951339 0.1960688  0.14721482 0.67329766 0.69120368 0.57920432
 0.43638952 0.42105551 0.43623329 0.45970772 0.0164101  0.82716031
 0.04376496 0.17994072 0.7406025  0.13037203 0.90631653 0.15836997
 0.92749593 0.14770067 0.92960967 0.25145331 0.70695548 0.19555101
 0.39257045 0.13575597 0.28789077 0.07375115 0.96403943 0.80313771
 0.02917173 0.48666191 0.77912268 0.51756297 0.4639424  0.45322914
 0.38812763 0.15560133 0.86403352 0.78385157 0.11912181 0.414019
 0.15632493 0.60048972 0.9935398  0.06402706 0.99163941 0.02229378
 0.10392711 0.01960803 0.6115714  0.9259645  0.69804233 0.10492756
 0.42785002 0.72796694 0.24008469 0.74157092 0.71052113 0.62358871
 0.91327252 0.0223508  0.08373651 0.91913787 0.26580305 0.5797042
 0.36105558 0.43275102 0.98064654 0.85529366 0.23352471 0.92815969
 0.38710368 0.17094735 0.84450817 0.21908917 0.66711474 0.13754708
 0.49009395 0.40602442 0.62919072 0.83398323 0.17790809 0.04064466
 0.31187587 0.03330163 0.79893053 0.99728246 0.01805598 0.98859583
 0.80320987 0.13442005 0.46710442 0.45831797 0.13123517 0.29542599
 0.49542796 0.51880929 0.37296186 0.76090064 0.7548383  0.48058113
 0.83465054 0.52248016 0.21956368 0.46589712 0.75598859 0.42812441
 0.73145622 0.16918261 0.02445512 0.00221758 0.36042122 0.8490867
 0.33369163 0.67030894 0.38665223 0.16246456 0.45532514 0.74233386
 0.96521169 0.36725452 0.29358193 0.46125555 0.53211547 0.26290122
 0.65179942 0.25847337 0.81724057 0.99261451 0.21992123 0.89211556
 0.29526797 0.17077962 0.54908317 0.91248904 0.97213494 0.63959052
 0.14703749 0.60297332 0.66603429 0.49020522 0.87938218 0.62886067
 0.44875409 0.52478597 0.38884389 0.02894025 0.90383319 0.21177767
 0.43254025 0.93294878 0.27725741 0.34579388 0.53539685 0.79673216
 0.94900626 0.77829025 0.35582298 0.23463353 0.00695873 0.52320125
 0.57403479 0.35387197 0.39661362 0.515138   0.40797052 0.50839455
 0.27935224 0.62498251 0.47124265 0.08278059 0.32522849 0.52843611
 0.10311503 0.71629296 0.2100147  0.89140145 0.44107502 0.10752843
 0.4129794  0.47232796 0.26068975 0.04262106 0.93320837 0.71532016
 0.77673294 0.82747674 0.65518846 0.23316578 0.57558667 0.02203924
 0.35149204 0.44548199 0.28056153 0.86991441 0.07413055 0.26056994
 0.55678097 0.17042038 0.1422051  0.46194247 0.04764662 0.23542336
 0.16805661 0.57474717 0.74215953 0.96927519 0.53903366 0.68734613
 0.56036842 0.34194987 0.10460768 0.95553562 0.46436383 0.68579989
 0.9399565  0.82128614 0.81091975 0.11823248 0.16111312 0.64620289
 0.91379759 0.98562967 0.67133502 0.64514114 0.83379166 0.02825724]
(1000, 300)
float64
8
2
```

In [57]:

```
#create a 3D array and rearrange it
import numpy as np
array_4 = np.arange(24).reshape(2,3,4)
print(array_4)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

In [66]:

```
import numpy as np
array_1 = np.arange(10)
print(array_1)
print(array_1[5])
print(array_1[[2,5,6]])
print(array_1[2:])
print(array_1[:2])
print(array_1[3:7])
print(array_1[2::3])
```

```
[0 1 2 3 4 5 6 7 8 9]
5
[2 5 6]
[2 3 4 5 6 7 8 9]
[0 1]
[3 4 5 6]
[2 5 8]
```

In [68]:

```
import numpy as np
list1 = [1, 2, 3, 5, 4, 6, 7, 8, 5, 3, 2]
array = np.array(list1)
print(array)
print(array[0::2])
```

```
[1 2 3 5 4 6 7 8 5 3 2]
[1 3 4 7 5 2]
```

In [73]:

```python
# indesxing in multidimensional arrays.Create a 2D array
import numpy as np
array = np.array([[2,5,7,5],[4,6,8,10],[10,12,15,19]])
print(array)
# print 3rd row second column
print(array[2,1])
# print 2nd row second column
print(array[1:])
```

```
[[ 2  5  7  5]
 [ 4  6  8 10]
 [10 12 15 19]]
12
[[ 4  6  8 10]
 [10 12 15 19]]
```

In [102]:

```python
#From a 2D array extract all the rows of the 2 column.
import numpy as np
array = np.array([[5,6,7],[7,6,5],[0,8,7]])
print(array)
print(array[ : ,1])
```

```
[[5 6 7]
 [7 6 5]
 [0 8 7]]
[6 6 8]
```

In [5]:

```python
import numpy as np

# Reshape a 1-D array to a 3 x 4 array
some_array = np.arange(0, 12).reshape(3, 4)
print(some_array)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [6]:

```python
import numpy as np

# Reshape a 1-D array to a 3 x 4 array
some_array = np.arange(0, 12).reshape(2, 6)
print(some_array)
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
```

In [7]:

```
# If you specify -1 as a dimension, the dimensions are automatically calculated
# -1 means "whatever dimension is needed"
import numpy as np

# Reshape a 1-D array to a 3 x 4 array
some_array = np.arange(0, 12).reshape(4, -1)
print(some_array)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

In [9]:

```
#transposing an array
import numpy as np

# Reshape a 1-D array to a 3 x 4 array
some_array = np.arange(0, 12).reshape(2, 6)
print(some_array.T)
```

```
[[ 0  6]
 [ 1  7]
 [ 2  8]
 [ 3  9]
 [ 4 10]
 [ 5 11]]
```

In [16]:

```
# stacking.In this you can vstack as number of columns is same (4)

# Creating two arrays
array_1 = np.arange(12).reshape(3, 4)
array_2 = np.arange(20).reshape(5, 4)

array_3 = np.arange(15).reshape(5, 3)
array_4 = np.arange(20).reshape(5, 4)

print(array_1)
print("\n")
print(array_2)
print("\n")
print(np.vstack((array_1,array_2)))
print("\n")
print(np.hstack((array_3,array_4)))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]


[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]


[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]


[[ 0  1  2  0  1  2  3]
 [ 3  4  5  4  5  6  7]
 [ 6  7  8  8  9 10 11]
 [ 9 10 11 12 13 14 15]
 [12 13 14 16 17 18 19]]
```

In [18]:

```
# Basic mathematical operations
a = np.arange(1, 20)

# sin, cos, exp, log
print(np.sin(a))
print(np.cos(a))
print(np.exp(a))
print(np.log(a))
print("\n")
print(np.sqrt(a))
```

```
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155
  0.6569866   0.98935825  0.41211849 -0.54402111 -0.99999021 -0.53657292
  0.42016704  0.99060736  0.65028784 -0.28790332 -0.96139749 -0.75098725
  0.14987721]
[ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219  0.96017029
  0.75390225 -0.14550003 -0.91113026 -0.83907153  0.0044257   0.84385396
  0.90744678  0.13673722 -0.75968791 -0.95765948 -0.27516334  0.66031671
  0.98870462]
[2.71828183e+00 7.38905610e+00 2.00855369e+01 5.45981500e+01
 1.48413159e+02 4.03428793e+02 1.09663316e+03 2.98095799e+03
 8.10308393e+03 2.20264658e+04 5.98741417e+04 1.62754791e+05
 4.42413392e+05 1.20260428e+06 3.26901737e+06 8.88611052e+06
 2.41549528e+07 6.56599691e+07 1.78482301e+08]
[0.         0.69314718 1.09861229 1.38629436 1.60943791 1.79175947
 1.94591015 2.07944154 2.19722458 2.30258509 2.39789527 2.48490665
 2.56494936 2.63905733 2.7080502  2.77258872 2.83321334 2.89037176
 2.94443898]


[1.         1.41421356 1.73205081 2.         2.23606798 2.44948974
 2.64575131 2.82842712 3.         3.16227766 3.31662479 3.46410162
 3.60555128 3.74165739 3.87298335 4.         4.12310563 4.24264069
 4.35889894]
```

In [24]:

```
#vectorizing a user function or operation
a = np.arange(1, 20)
f = np.vectorize(lambda x: x/(x+1))
print(f(a))
```

```
[0.5        0.66666667 0.75       0.8        0.83333333 0.85714286
 0.875      0.88888889 0.9        0.90909091 0.91666667 0.92307692
 0.92857143 0.93333333 0.9375     0.94117647 0.94444444 0.94736842
 0.95      ]
```

In [27]:

```
#Given an array, 'array_3' divide each element with 5.
#Hint: Create a vectorized function, then apply it to the array_3
a = np.arange(5, 21,5)
f = np.vectorize(lambda x: x/5)
print(f(a))
```

```
[1. 2. 3. 4.]
```

In [29]:

```
# linera algerbra functions
# Creating arrays
a = np.arange(1, 10).reshape(3, 3)
b= np.arange(1, 13).reshape(3, 4)
print(a)
print(b)

# Inverse
np.linalg.inv(a)
# Determinant
np.linalg.det(a)
# Eigenvalues and eigenvectors
np.linalg.eig(a)
# Multiply matrices
np.dot(a, b)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

Out[29]:

```
array([[ 38,  44,  50,  56],
       [ 83,  98, 113, 128],
       [128, 152, 176, 200]])
```

In [30]:

```
# Inverse
np.linalg.inv(a)
```

Out[30]:

```
array([[ 3.15251974e+15, -6.30503948e+15,  3.15251974e+15],
       [-6.30503948e+15,  1.26100790e+16, -6.30503948e+15],
       [ 3.15251974e+15, -6.30503948e+15,  3.15251974e+15]])
```

In [32]:

```
# Determinant
np.linalg.det(a)
```

Out[32]:

```
-9.51619735392994e-16
```

In [33]:

```
# Eigenvalues and eigenvectors
np.linalg.eig(a)
```

Out[33]:

```
(array([ 1.61168440e+01, -1.11684397e+00, -9.75918483e-16]),
 array([[-0.23197069, -0.78583024,  0.40824829],
        [-0.52532209, -0.08675134, -0.81649658],
        [-0.8186735 ,  0.61232756,  0.40824829]]))
```

In [1]:

```
# import pandas, pd is an alias
import pandas as pd

# Creating a numeric pandas series
s = pd.Series([2, 4, 5, 6, 9])
print(s)
print(type(s))
```

```
0    2
1    4
2    5
3    6
4    9
dtype: int64
<class 'pandas.core.series.Series'>
```

In [2]:

```
# creating a series of characters
# notice that the 'dtype' here is 'object'
char_series = pd.Series(['a', 'b', 'af'])
char_series
```

Out[2]:

```
0     a
1     b
2    af
dtype: object
```

In [4]:

```
# creating a series of type datetime
date_series = pd.date_range(start = '11-09-2017', end = '12-12-2017')
date_series
type(date_series)
```

Out[4]:

```
pandas.core.indexes.datetimes.DatetimeIndex
```

In [5]:

```
# Indexing pandas series: Same as indexing 1-d numpy arrays or lists
# accessing the fourth element
s[3]

# accessing elements starting index = 2 till the end
s[2:]
```

Out[5]:

```
2    5
3    6
4    9
dtype: int64
```

In [6]:

```
# accessing the fourth element
s[3]
```

Out[6]:

```
6
```

In [7]:
```
# accessing the second and the fourth elements
# note that s[1, 3] will not work, you need to pass the indices [1, 3] as a list
s[[1, 3]]
```

Out[7]:
```
1    4
3    6
dtype: int64
```

In [10]:
```
# using own function with apply statement
import pandas as pd

# Creating a numeric pandas series
s = pd.Series([2, 4, 5, 6, 9])
f = s.apply(lambda x: x**2)
print(f)
```

```
0     4
1    16
2    25
3    36
4    81
dtype: int64
```

In [11]:
```
#dataframes
# keys become column names
df = pd.DataFrame({'name': ['Vinay', 'Kushal', 'Aman', 'Saif'],
                   'age': [22, 25, 24, 28],
                    'occupation': ['engineer', 'doctor', 'data analyst', 'teacher
df
```

Out[11]:

	age	name	occupation
0	22	Vinay	engineer
1	25	Kushal	doctor
2	24	Aman	data analyst
3	28	Saif	teacher

In [12]:

```
# reading a CSV file as a dataframe
market_df = pd.read_csv("../global_sales_data/market_fact.csv")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-12-b22d4a109963> in <module>()
      1 # reading a CSV file as a dataframe
----> 2 market_df = pd.read_csv("../global_sales_data/market_fact.csv")

~\AppData\Local\anaconda3\lib\site-packages\pandas\io\parsers.py in parser_f(fi
lepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, p
refix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values,
 skipinitialspace, skiprows, nrows, na_values, keep_default_na, na_filter, verb
ose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_
parser, dayfirst, iterator, chunksize, compression, thousands, decimal, lineter
minator, quotechar, quoting, escapechar, comment, encoding, dialect, tupleize_c
ols, error_bad_lines, warn_bad_lines, skipfooter, skip_footer, doublequote, del
im_whitespace, as_recarray, compact_ints, use_unsigned, low_memory, buffer_line
s, memory_map, float_precision)
    707                     skip_blank_lines=skip_blank_lines)
    708 
--> 709         return _read(filepath_or_buffer, kwds)
    710 
    711     parser_f.__name__ = name

~\AppData\Local\anaconda3\lib\site-packages\pandas\io\parsers.py in _read(filep
ath_or_buffer, kwds)
    447 
    448     # Create the parser.
--> 449     parser = TextFileReader(filepath_or_buffer, **kwds)
    450 
    451     if chunksize or iterator:

~\AppData\Local\anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(se
lf, f, engine, **kwds)
    816             self.options['has_index_names'] = kwds['has_index_names']
    817 
--> 818         self._make_engine(self.engine)
    819 
    820     def close(self):

~\AppData\Local\anaconda3\lib\site-packages\pandas\io\parsers.py in _make_engin
e(self, engine)
   1047     def _make_engine(self, engine='c'):
   1048         if engine == 'c':
-> 1049             self._engine = CParserWrapper(self.f, **self.options)
   1050         else:
   1051             if engine == 'python':

~\AppData\Local\anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(se
lf, src, **kwds)
   1693         kwds['allow_leading_cols'] = self.index_col is not False
   1694 
-> 1695         self._reader = parsers.TextReader(src, **kwds)
   1696 
   1697         # XXX
```

```
pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_sourc
e()

FileNotFoundError: File b'../global_sales_data/market_fact.csv' does not exist
```

In [17]:

```
series = pd.Series([6,7,8,9,2,3,4,5])
series1 = series.apply(lambda x: x**2)
print(series)
print(series1)
```

```
0    6
1    7
2    8
3    9
4    2
5    3
6    4
7    5
dtype: int64
0    36
1    49
2    64
3    81
4     4
5     9
6    16
7    25
dtype: int64
```

In [23]:
```
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
#Sort the dataframe on 'month' and 'day' in ascending order in the dataframe 'df'
df_2 = df.sort_values(by=['month','day'])
print(df_2.head(20))
```

```
     X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain   area
241  4  4   apr  fri  83.0   23.3   85.3   2.3  16.7  20   3.1   0.0   0.00
442  6  5   apr  mon  87.9   24.9   41.6   3.7  10.9  64   3.1   0.0   3.35
19   6  4   apr  sat  86.3   27.4   97.1   5.1   9.3  44   4.5   0.0   0.00
239  7  5   apr  sun  81.9    3.0    7.9   3.5  13.4  75   1.8   0.0   0.00
469  6  3   apr  sun  91.0   14.6   25.6  12.3  13.7  33   9.4   0.0  61.13
470  5  4   apr  sun  91.0   14.6   25.6  12.3  17.6  27   5.8   0.0   0.00
176  6  5   apr  thu  81.5    9.1   55.2   2.7   5.8  54   5.8   0.0   4.61
196  6  5   apr  thu  81.5    9.1   55.2   2.7   5.8  54   5.8   0.0  10.93
240  6  3   apr  wed  88.0   17.2   43.5   3.8  15.2  51   2.7   0.0   0.00
12   6  5   aug  fri  63.5   70.8  665.3   0.8  17.0  72   6.7   0.0   0.00
78   1  2   aug  fri  90.1  108.0  529.8  12.5  14.7  66   2.7   0.0   0.00
142  8  6   aug  fri  90.1  108.0  529.8  12.5  21.2  51   8.9   0.0   0.61
184  8  6   aug  fri  93.9  135.7  586.7  15.1  20.8  34   4.9   0.0   6.96
195  2  5   aug  fri  93.9  135.7  586.7  15.1  23.5  36   5.4   0.0  10.02
261  3  4   aug  fri  91.6  112.4  573.0   8.9  11.2  84   7.6   0.0   3.30
262  2  4   aug  fri  91.6  112.4  573.0   8.9  21.4  42   3.1   0.0   4.25
263  6  3   aug  fri  91.1  141.1  629.1   7.1  19.3  39   3.6   0.0   1.56
264  4  4   aug  fri  94.3  167.6  684.4  13.0  21.8  53   3.1   0.0   6.54
388  6  4   aug  fri  94.8  227.0  706.7  12.0  23.3  34   3.1   0.0  28.74
389  7  4   aug  fri  94.8  227.0  706.7  12.0  23.3  34   3.1   0.0   0.00
```

In [6]:
```
#selecting and indexing data in dataframe
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df[2:7])
print('\n')
print(df[5::2].head())
type(df)
```

```
   X  Y month  day  FFMC   DMC     DC   ISI  temp  RH  wind  rain  area
2  7  4   oct  sat  90.6  43.7  686.9   6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7  33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3  51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5  8  6   aug  sun  92.3  85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6  8  6   aug  mon  92.3  88.9  495.6   8.5  24.1  27   3.1   0.0   0.0


    X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
5   8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
7   8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
9   7  5   sep  sat  92.5   88.0  698.6   7.1  22.8  40   4.0   0.0   0.0
11  7  5   sep  sat  92.8   73.2  713.0  22.6  19.3  38   4.0   0.0   0.0
13  6  5   sep  mon  90.9  126.5  686.5   7.0  21.3  42   2.2   0.0   0.0
```

Out[6]:
```
pandas.core.frame.DataFrame
```

In [10]:
```
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
sales = df['ISI']
sales.head()
```

Out[10]:
```
0    5.1
1    6.7
2    6.7
3    9.0
4    9.6
Name: ISI, dtype: float64
```

In [12]:
```
# Using df.column
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
sales = df.ISI
sales.head()
```

Out[12]:
```
0    5.1
1    6.7
2    6.7
3    9.0
4    9.6
Name: ISI, dtype: float64
```

In [14]:
```
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
sales = df[['ISI','day','month']]
print(sales.head())
```

```
   ISI  day month
0  5.1  fri   mar
1  6.7  tue   oct
2  6.7  sat   oct
3  9.0  fri   mar
4  9.6  sun   mar
```

In [17]:
```
import pandas as pd
df = pd.read_csv('C:\Users\Z001MC7\Downloads\Introduction_to_Pandas\global_sales_
print(df)
```

```
  File "<ipython-input-17-e5720112804d>", line 2
    df = pd.read_csv('C:\Users\Z001MC7\Downloads\Introduction_to_Pandas\global_
sales_data\market_fact')
                                       ^
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in positi
on 2-3: truncated \UXXXXXXXX escape
```

In [20]:
```
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.head())

market_df = pd.read_csv("../global_sales_data/market_fact.csv")
market_df.head()
```

```
   X  Y month  day  FFMC   DMC     DC  ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2  26.2   94.3  5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6  35.4  669.1  6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6  43.7  686.9  6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7  33.3   77.5  9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3  51.3  102.2  9.6  11.4  99   1.8   0.0   0.0

---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-20-7bf23c6e9e00> in <module>()
      3 print(df.head())
      4 
----> 5 market_df = pd.read_csv("../global_sales_data/market_fact.csv")
      6 market_df.head()

~\AppData\Local\anaconda3\lib\site-packages\pandas\io\parsers.py in parser_f(fi
lepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, p
refix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values,
 skipinitialspace, skiprows, nrows, na_values, keep_default_na, na_filter, verb
ose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_
parser, dayfirst, iterator, chunksize, compression, thousands, decimal, lineter
minator, quotechar, quoting, escapechar, comment, encoding, dialect, tupleize_c
ols, error_bad_lines, warn_bad_lines, skipfooter, skip_footer, doublequote, del
im_whitespace, as_recarray, compact_ints, use_unsigned, low_memory, buffer_line
s, memory_map, float_precision)
    707                     skip_blank_lines=skip_blank_lines)
    708 
--> 709         return _read(filepath_or_buffer, kwds)
    710 
    711     parser_f.__name__ = name

~\AppData\Local\anaconda3\lib\site-packages\pandas\io\parsers.py in _read(filep
ath_or_buffer, kwds)
    447 
    448     # Create the parser.
--> 449     parser = TextFileReader(filepath_or_buffer, **kwds)
    450 
    451     if chunksize or iterator:

~\AppData\Local\anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(se
lf, f, engine, **kwds)
    816             self.options['has_index_names'] = kwds['has_index_names']
    817 
--> 818         self._make_engine(self.engine)
    819 
    820     def close(self):

~\AppData\Local\anaconda3\lib\site-packages\pandas\io\parsers.py in _make_engin
e(self, engine)
   1047     def _make_engine(self, engine='c'):
```

```
   1048         if engine == 'c':
-> 1049             self._engine = CParserWrapper(self.f, **self.options)
   1050         else:
   1051             if engine == 'python':

~\AppData\Local\anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(se
lf, src, **kwds)
   1693         kwds['allow_leading_cols'] = self.index_col is not False
   1694 
-> 1695         self._reader = parsers.TextReader(src, **kwds)
   1696 
   1697         # XXX

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_sourc
e()

FileNotFoundError: File b'../global_sales_data/market_fact.csv' does not exist
```

In [25]:

```
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.head(10))
print('\n')

# Selecting a single element
# Note that 2, 4 corresponds to the third row and fifth column
print(df.iloc[2, 4])

print('\n')

# Selecting a single row, and all columns
# Select the 6th row, with label (and index) = 5
print(df.iloc[5])

print('\n')
# Select multiple rows using a list of indices

print(df.iloc[[3, 7, 8]])
```

```
   X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5  8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6  8  6   aug  mon  92.3   88.9  495.6   8.5  24.1  27   3.1   0.0   0.0
7  8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
8  8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0   0.0
9  7  5   sep  sat  92.5   88.0  698.6   7.1  22.8  40   4.0   0.0   0.0


90.6


X           8
Y           6
month     aug
day       sun
FFMC     92.3
DMC      85.3
DC        488
ISI      14.7
temp     22.2
RH         29
wind      5.4
rain        0
area        0
Name: 5, dtype: object


   X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
3  8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
7  8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
8  8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0   0.0
```

In [29]:

```
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.head(10))
print('\n')
# Selecting rows using a range of integer indices
# Notice that 4 is included, 8 is not
print(df.iloc[4:8])
print('\n')
# Selecting a single column
# Notice that the column index starts at 0, and 2 represents the third column (Cu
print((df.iloc[:, 2]).head())
print('\n')
# Selecting multiple columns
print(df.iloc[:, 3:8])
```

```
   X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5  8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6  8  6   aug  mon  92.3   88.9  495.6   8.5  24.1  27   3.1   0.0   0.0
7  8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
8  8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0   0.0
9  7  5   sep  sat  92.5   88.0  698.6   7.1  22.8  40   4.0   0.0   0.0


   X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
4  8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5  8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6  8  6   aug  mon  92.3   88.9  495.6   8.5  24.1  27   3.1   0.0   0.0
7  8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0


0    mar
1    oct
2    oct
3    mar
4    mar
Name: month, dtype: object


     day  FFMC    DMC     DC   ISI
0    fri  86.2   26.2   94.3   5.1
1    tue  90.6   35.4  669.1   6.7
2    sat  90.6   43.7  686.9   6.7
3    fri  91.7   33.3   77.5   9.0
4    sun  89.3   51.3  102.2   9.6
5    sun  92.3   85.3  488.0  14.7
6    mon  92.3   88.9  495.6   8.5
7    mon  91.5  145.4  608.2  10.7
8    tue  91.0  129.5  692.6   7.0
9    sat  92.5   88.0  698.6   7.1
10   sat  92.5   88.0  698.6   7.1
11   sat  92.8   73.2  713.0  22.6
```

```
12   fri  63.5   70.8  665.3   0.8
13   mon  90.9  126.5  686.5   7.0
14   wed  92.9  133.3  699.6   9.2
15   fri  93.3  141.2  713.9  13.9
16   sat  91.7   35.8   80.8   7.8
17   mon  84.9   32.8  664.2   3.0
18   wed  89.2   27.9   70.8   6.3
19   sat  86.3   27.4   97.1   5.1
20   tue  91.0  129.5  692.6   7.0
21   mon  91.8   78.5  724.3   9.2
22   sun  94.3   96.3  200.0  56.1
23   sat  90.2  110.9  537.4   6.2
24   sat  93.5  139.4  594.2  20.3
25   sun  91.4  142.4  601.4  10.6
26   fri  92.4  117.9  668.0  12.2
27   mon  90.9  126.5  686.5   7.0
28   sat  93.4  145.4  721.4   8.1
29   sun  93.5  149.3  728.6   8.1
..   ...   ...    ...    ...   ...
487  tue  95.1  141.3  605.8  17.7
488  tue  95.1  141.3  605.8  17.7
489  wed  95.1  141.3  605.8  17.7
490  wed  95.1  141.3  605.8  17.7
491  thu  95.8  152.0  624.1  13.8
492  fri  95.9  158.0  633.6  11.3
493  fri  95.9  158.0  633.6  11.3
494  sat  96.0  164.0  643.0  14.0
495  mon  96.2  175.5  661.8  16.8
496  mon  96.2  175.5  661.8  16.8
497  tue  96.1  181.1  671.2  14.3
498  tue  96.1  181.1  671.2  14.3
499  tue  96.1  181.1  671.2  14.3
500  tue  96.1  181.1  671.2  14.3
501  tue  96.1  181.1  671.2  14.3
502  tue  96.1  181.1  671.2  14.3
503  wed  94.5  139.4  689.1  20.0
504  wed  94.5  139.4  689.1  20.0
505  thu  91.0  163.2  744.4  10.1
506  fri  91.0  166.9  752.6   7.1
507  fri  91.0  166.9  752.6   7.1
508  fri  91.0  166.9  752.6   7.1
509  fri  91.0  166.9  752.6   7.1
510  fri  91.0  166.9  752.6   7.1
511  sun  81.6   56.7  665.6   1.9
512  sun  81.6   56.7  665.6   1.9
513  sun  81.6   56.7  665.6   1.9
514  sun  81.6   56.7  665.6   1.9
515  sat  94.4  146.0  614.7  11.3
516  tue  79.5    3.0  106.7   1.1

[517 rows x 5 columns]
```

In [30]:

```
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.head(10))
print('\n')
# Selecting multiple rows and columns
print(df.iloc[3:6, 2:5])
```

```
   X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5  8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6  8  6   aug  mon  92.3   88.9  495.6   8.5  24.1  27   3.1   0.0   0.0
7  8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
8  8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0   0.0
9  7  5   sep  sat  92.5   88.0  698.6   7.1  22.8  40   4.0   0.0   0.0


  month  day  FFMC
3   mar  fri  91.7
4   mar  sun  89.3
5   aug  sun  92.3
```

In [37]:

```python
#Dataframe iloc
#Description
#Using iloc index the dataframe to print all the rows of the columns at index 3,4
#Hint: Use 3,4,5 not 2,3,4
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.head(10))
print('\n')
print((df.iloc[:, 3:6]).head())

# or
print('\n')
print((df.iloc[:,[3,4,5]]).head())
```

```
   X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5  8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6  8  6   aug  mon  92.3   88.9  495.6   8.5  24.1  27   3.1   0.0   0.0
7  8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
8  8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0   0.0
9  7  5   sep  sat  92.5   88.0  698.6   7.1  22.8  40   4.0   0.0   0.0


   day  FFMC   DMC
0  fri  86.2  26.2
1  tue  90.6  35.4
2  sat  90.6  43.7
3  fri  91.7  33.3
4  sun  89.3  51.3


   day  FFMC   DMC
0  fri  86.2  26.2
1  tue  90.6  35.4
2  sat  90.6  43.7
3  fri  91.7  33.3
4  sun  89.3  51.3
```

In [40]:

```
#label indexing

import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.head(10))
print('\n')

# Selecting a single element
# Select row label = 3 and column label = 'FFMC
print(df.loc[3, 'FFMC'])

print('\n')
# Selecting a single row using a single label
# df.loc reads 5 as a label, not index
print((df.loc[5]).head())
```

```
   X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5  8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6  8  6   aug  mon  92.3   88.9  495.6   8.5  24.1  27   3.1   0.0   0.0
7  8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
8  8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0   0.0
9  7  5   sep  sat  92.5   88.0  698.6   7.1  22.8  40   4.0   0.0   0.0


91.7


X            8
Y            6
month      aug
day        sun
FFMC      92.3
Name: 5, dtype: object
```

In [42]:
```
#Using loc function print out all the columns and rows from 2 to 20 of the 'df' d
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.head(25))
print('\n')
print(df.loc[2:20])
```

```
    X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
0   7  5   mar  fri  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0   0.0
1   7  4   oct  tue  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0   0.0
2   7  4   oct  sat  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.0
3   8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
4   8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5   8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6   8  6   aug  mon  92.3   88.9  495.6   8.5  24.1  27   3.1   0.0   0.0
7   8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
8   8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0   0.0
9   7  5   sep  sat  92.5   88.0  698.6   7.1  22.8  40   4.0   0.0   0.0
10  7  5   sep  sat  92.5   88.0  698.6   7.1  17.8  51   7.2   0.0   0.0
11  7  5   sep  sat  92.8   73.2  713.0  22.6  19.3  38   4.0   0.0   0.0
12  6  5   aug  fri  63.5   70.8  665.3   0.8  17.0  72   6.7   0.0   0.0
13  6  5   sep  mon  90.9  126.5  686.5   7.0  21.3  42   2.2   0.0   0.0
14  6  5   sep  wed  92.9  133.3  699.6   9.2  26.4  21   4.5   0.0   0.0
15  6  5   sep  fri  93.3  141.2  713.9  13.9  22.9  44   5.4   0.0   0.0
16  5  5   mar  sat  91.7   35.8   80.8   7.8  15.1  27   5.4   0.0   0.0
17  8  5   oct  mon  84.9   32.8  664.2   3.0  16.7  47   4.9   0.0   0.0
18  6  4   mar  wed  89.2   27.9   70.8   6.3  15.9  35   4.0   0.0   0.0
19  6  4   apr  sat  86.3   27.4   97.1   5.1   9.3  44   4.5   0.0   0.0
20  6  4   sep  tue  91.0  129.5  692.6   7.0  18.3  40   2.7   0.0   0.0
21  5  4   sep  mon  91.8   78.5  724.3   9.2  19.1  38   2.7   0.0   0.0
22  7  4   jun  sun  94.3   96.3  200.0  56.1  21.0  44   4.5   0.0   0.0
23  7  4   aug  sat  90.2  110.9  537.4   6.2  19.5  43   5.8   0.0   0.0
24  7  4   aug  sat  93.5  139.4  594.2  20.3  23.7  32   5.8   0.0   0.0


    X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
2   7  4   oct  sat  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.0
3   8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
4   8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5   8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6   8  6   aug  mon  92.3   88.9  495.6   8.5  24.1  27   3.1   0.0   0.0
7   8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
8   8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0   0.0
9   7  5   sep  sat  92.5   88.0  698.6   7.1  22.8  40   4.0   0.0   0.0
10  7  5   sep  sat  92.5   88.0  698.6   7.1  17.8  51   7.2   0.0   0.0
11  7  5   sep  sat  92.8   73.2  713.0  22.6  19.3  38   4.0   0.0   0.0
12  6  5   aug  fri  63.5   70.8  665.3   0.8  17.0  72   6.7   0.0   0.0
13  6  5   sep  mon  90.9  126.5  686.5   7.0  21.3  42   2.2   0.0   0.0
14  6  5   sep  wed  92.9  133.3  699.6   9.2  26.4  21   4.5   0.0   0.0
15  6  5   sep  fri  93.3  141.2  713.9  13.9  22.9  44   5.4   0.0   0.0
16  5  5   mar  sat  91.7   35.8   80.8   7.8  15.1  27   5.4   0.0   0.0
17  8  5   oct  mon  84.9   32.8  664.2   3.0  16.7  47   4.9   0.0   0.0
18  6  4   mar  wed  89.2   27.9   70.8   6.3  15.9  35   4.0   0.0   0.0
19  6  4   apr  sat  86.3   27.4   97.1   5.1   9.3  44   4.5   0.0   0.0
20  6  4   sep  tue  91.0  129.5  692.6   7.0  18.3  40   2.7   0.0   0.0
```

In [52]:
```
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.head(25))
print('\n')
# Select all rows where Sales > 3000
# First, we get a boolean array where True corresponds to rows having Sales > 300
print(df.loc[(df.DMC > 100)])
```

```
    X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
0   7  5   mar  fri  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0   0.0
1   7  4   oct  tue  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0   0.0
2   7  4   oct  sat  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.0
3   8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
4   8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5   8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6   8  6   aug  mon  92.3   88.9  495.6   8.5  24.1  27   3.1   0.0   0.0
7   8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
8   8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0   0.0
9   7  5   sep  sat  92.5   88.0  698.6   7.1  22.8  40   4.0   0.0   0.0
10  7  5   sep  sat  92.5   88.0  698.6   7.1  17.8  51   7.2   0.0   0.0
11  7  5   sep  sat  92.8   73.2  713.0  22.6  19.3  38   4.0   0.0   0.0
12  6  5   aug  fri  63.5   70.8  665.3   0.8  17.0  72   6.7   0.0   0.0
13  6  5   sep  mon  90.9  126.5  686.5   7.0  21.3  42   2.2   0.0   0.0
14  6  5   sep  wed  92.9  133.3  699.6   9.2  26.4  21   4.5   0.0   0.0
15  6  5   sep  fri  93.3  141.2  713.9  13.9  22.9  44   5.4   0.0   0.0
16  5  5   mar  sat  91.7   35.8   80.8   7.8  15.1  27   5.4   0.0   0.0
17  8  5   oct  mon  84.9   32.8  664.2   3.0  16.7  47   4.9   0.0   0.0
18  6  4   mar  wed  89.2   27.9   70.8   6.3  15.9  35   4.0   0.0   0.0
19  6  4   apr  sat  86.3   27.4   97.1   5.1   9.3  44   4.5   0.0   0.0
20  6  4   sep  tue  91.0  129.5  692.6   7.0  18.3  40   2.7   0.0   0.0
21  5  4   sep  mon  91.8   78.5  724.3   9.2  19.1  38   2.7   0.0   0.0
22  7  4   jun  sun  94.3   96.3  200.0  56.1  21.0  44   4.5   0.0   0.0
23  7  4   aug  sat  90.2  110.9  537.4   6.2  19.5  43   5.8   0.0   0.0
24  7  4   aug  sat  93.5  139.4  594.2  20.3  23.7  32   5.8   0.0   0.0


     X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain    area
7    8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0    0.00
8    8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0    0.00
13   6  5   sep  mon  90.9  126.5  686.5   7.0  21.3  42   2.2   0.0    0.00
14   6  5   sep  wed  92.9  133.3  699.6   9.2  26.4  21   4.5   0.0    0.00
15   6  5   sep  fri  93.3  141.2  713.9  13.9  22.9  44   5.4   0.0    0.00
20   6  4   sep  tue  91.0  129.5  692.6   7.0  18.3  40   2.7   0.0    0.00
23   7  4   aug  sat  90.2  110.9  537.4   6.2  19.5  43   5.8   0.0    0.00
24   7  4   aug  sat  93.5  139.4  594.2  20.3  23.7  32   5.8   0.0    0.00
25   7  4   aug  sun  91.4  142.4  601.4  10.6  16.3  60   5.4   0.0    0.00
26   7  4   sep  fri  92.4  117.9  668.0  12.2  19.0  34   5.8   0.0    0.00
27   7  4   sep  mon  90.9  126.5  686.5   7.0  19.4  48   1.3   0.0    0.00
28   6  3   sep  sat  93.4  145.4  721.4   8.1  30.2  24   2.7   0.0    0.00
29   6  3   sep  sun  93.5  149.3  728.6   8.1  22.8  39   3.6   0.0    0.00
42   4  4   aug  tue  94.8  108.3  647.1  17.0  16.6  54   5.4   0.0    0.00
46   5  6   sep  mon  90.9  126.5  686.5   7.0  14.7  70   3.6   0.0    0.00
50   4  4   sep  thu  92.9  137.0  706.4   9.2  20.8  17   1.3   0.0    0.00
52   4  3   aug  wed  92.1  111.2  654.1   9.6  20.4  42   4.9   0.0    0.00
53   4  3   aug  wed  92.1  111.2  654.1   9.6  20.4  42   4.9   0.0    0.00
54   4  3   aug  thu  91.7  114.3  661.3   6.3  17.6  45   3.6   0.0    0.00
```

```
55   4  3   sep  thu  92.9  137.0  706.4   9.2  27.7  24   2.2  0.0   0.00
64   2  2   aug  mon  91.1  103.2  638.8   5.8  23.1  31   3.1  0.0   0.00
65   2  2   aug  thu  91.7  114.3  661.3   6.3  18.6  44   4.5  0.0   0.00
66   2  2   sep  fri  92.4  117.9  668.0  12.2  23.0  37   4.5  0.0   0.00
67   2  2   sep  fri  92.4  117.9  668.0  12.2  19.6  33   5.4  0.0   0.00
68   2  2   sep  fri  92.4  117.9  668.0  12.2  19.6  33   6.3  0.0   0.00
73   5  4   aug  tue  88.8  147.3  614.5   9.0  17.3  43   4.5  0.0   0.00
74   5  4   sep  fri  93.3  141.2  713.9  13.9  27.6  30   1.3  0.0   0.00
78   1  2   aug  fri  90.1  108.0  529.8  12.5  14.7  66   2.7  0.0   0.00
79   1  2   aug  tue  91.0  121.2  561.6   7.0  21.6  19   6.7  0.0   0.00
80   1  2   aug  sun  91.4  142.4  601.4  10.6  19.5  39   6.3  0.0   0.00
..  .. ..   ...  ...   ...    ...    ...   ...   ...  ..   ...  ...    ...
482  3  4   aug  sun  94.9  130.3  587.1  14.1  23.4  40   5.8  0.0   1.29
483  8  6   aug  sun  94.9  130.3  587.1  14.1  31.0  27   5.4  0.0   0.00
484  2  5   aug  sun  94.9  130.3  587.1  14.1  33.1  25   4.0  0.0  26.43
485  2  4   aug  mon  95.0  135.5  596.3  21.3  30.6  28   3.6  0.0   2.07
486  5  4   aug  tue  95.1  141.3  605.8  17.7  24.1  43   6.3  0.0   2.00
487  5  4   aug  tue  95.1  141.3  605.8  17.7  26.4  34   3.6  0.0  16.40
488  4  4   aug  tue  95.1  141.3  605.8  17.7  19.4  71   7.6  0.0  46.70
489  4  4   aug  wed  95.1  141.3  605.8  17.7  20.6  58   1.3  0.0   0.00
490  4  4   aug  wed  95.1  141.3  605.8  17.7  28.7  33   4.0  0.0   0.00
491  4  4   aug  thu  95.8  152.0  624.1  13.8  32.4  21   4.5  0.0   0.00
492  1  3   aug  fri  95.9  158.0  633.6  11.3  32.4  27   2.2  0.0   0.00
493  1  3   aug  fri  95.9  158.0  633.6  11.3  27.5  29   4.5  0.0  43.32
494  6  6   aug  sat  96.0  164.0  643.0  14.0  30.8  30   4.9  0.0   8.59
495  6  6   aug  mon  96.2  175.5  661.8  16.8  23.9  42   2.2  0.0   0.00
496  4  5   aug  mon  96.2  175.5  661.8  16.8  32.6  26   3.1  0.0   2.77
497  3  4   aug  tue  96.1  181.1  671.2  14.3  32.3  27   2.2  0.0  14.68
498  6  5   aug  tue  96.1  181.1  671.2  14.3  33.3  26   2.7  0.0  40.54
499  7  5   aug  tue  96.1  181.1  671.2  14.3  27.3  63   4.9  6.4  10.82
500  8  6   aug  tue  96.1  181.1  671.2  14.3  21.6  65   4.9  0.8   0.00
501  7  5   aug  tue  96.1  181.1  671.2  14.3  21.6  65   4.9  0.8   0.00
502  4  4   aug  tue  96.1  181.1  671.2  14.3  20.7  69   4.9  0.4   0.00
503  2  4   aug  wed  94.5  139.4  689.1  20.0  29.2  30   4.9  0.0   1.95
504  4  3   aug  wed  94.5  139.4  689.1  20.0  28.9  29   4.9  0.0  49.59
505  1  2   aug  thu  91.0  163.2  744.4  10.1  26.7  35   1.8  0.0   5.80
506  1  2   aug  fri  91.0  166.9  752.6   7.1  18.5  73   8.5  0.0   0.00
507  2  4   aug  fri  91.0  166.9  752.6   7.1  25.9  41   3.6  0.0   0.00
508  1  2   aug  fri  91.0  166.9  752.6   7.1  25.9  41   3.6  0.0   0.00
509  5  4   aug  fri  91.0  166.9  752.6   7.1  21.1  71   7.6  1.4   2.17
510  6  5   aug  fri  91.0  166.9  752.6   7.1  18.2  62   5.4  0.0   0.43
515  1  4   aug  sat  94.4  146.0  614.7  11.3  25.6  42   4.0  0.0   0.00

[289 rows x 13 columns]
```

In [55]:

```
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.head())
print('\n')
# Select all rows where Sales > 3000
# First, we get a boolean array where True corresponds to rows having Sales > 300
print(df.loc[(df.DMC > 100) & (df.DC > 800)])
```

```
   X  Y month  day  FFMC   DMC     DC  ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2  26.2   94.3  5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6  35.4  669.1  6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6  43.7  686.9  6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7  33.3   77.5  9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3  51.3  102.2  9.6  11.4  99   1.8   0.0   0.0


     X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain   area
315  3  4   sep  wed  91.2  134.7  817.5   7.2  18.5  30   2.7   0.0   0.00
323  3  5   sep  thu  90.7  136.9  822.8   6.8  12.9  39   2.7   0.0   2.18
342  6  3   sep  mon  91.5  130.1  807.1   7.5  20.6  37   1.8   0.0   0.00
343  8  6   sep  mon  91.5  130.1  807.1   7.5  15.9  51   4.5   0.0   2.18
344  6  3   sep  mon  91.5  130.1  807.1   7.5  12.2  66   4.9   0.0   6.10
345  2  2   sep  mon  91.5  130.1  807.1   7.5  16.8  43   3.1   0.0   5.83
346  1  4   sep  mon  91.5  130.1  807.1   7.5  21.3  35   2.2   0.0  28.19
366  4  5   sep  tue  91.1  132.3  812.1  12.5  15.9  38   5.4   0.0   1.75
367  4  5   sep  tue  91.1  132.3  812.1  12.5  16.4  27   3.6   0.0   0.00
369  4  5   sep  sun  91.0  276.3  825.1   7.1  13.8  77   7.6   0.0   0.00
370  7  4   sep  sun  91.0  276.3  825.1   7.1  13.8  77   7.6   0.0  11.06
374  6  5   sep  fri  90.3  290.0  855.3   7.4  10.3  78   4.0   0.0  18.30
384  8  4   aug  sat  91.6  273.8  819.1   7.7  21.3  44   4.5   0.0  12.18
392  1  3   sep  sun  91.0  276.3  825.1   7.1  21.9  43   4.0   0.0  70.76
406  6  5   sep  sat  87.1  291.3  860.6   4.0  17.0  67   4.9   0.0   3.95
408  4  3   sep  fri  90.3  290.0  855.3   7.4  19.9  44   3.1   0.0   7.80
430  7  4   sep  thu  89.7  287.2  849.3   6.8  19.4  45   3.6   0.0   0.00
434  1  4   aug  fri  90.6  269.8  811.2   5.5  22.2  45   3.6   0.0   0.00
440  5  4   sep  fri  90.3  290.0  855.3   7.4  16.2  58   3.6   0.0   0.00
444  2  5   sep  fri  90.3  290.0  855.3   7.4  16.2  58   3.6   0.0   9.96
448  7  4   sep  wed  89.7  284.9  844.0  10.1  10.5  77   4.0   0.0   0.00
453  4  5   aug  thu  89.4  266.2  803.3   5.6  17.4  54   3.1   0.0   0.00
459  7  4   aug  sat  91.6  273.8  819.1   7.7  15.5  72   8.0   0.0   1.94
462  1  4   sep  sun  91.0  276.3  825.1   7.1  14.5  76   7.6   0.0   3.71
```

In [57]:
```
#Print all the columns and the rows where 'area' is greater than 0, 'wind' is gre

import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.loc[(df.area > 0) & (df.wind > 1) & (df.temp > 15)])
```

```
     X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain    area
138  9  9   jul  tue  85.8   48.3  313.4   3.9  18.0  42   2.7   0.0    0.36
139  1  4   sep  tue  91.0  129.5  692.6   7.0  21.7  38   2.2   0.0    0.43
140  2  5   sep  mon  90.9  126.5  686.5   7.0  21.9  39   1.8   0.0    0.47
141  1  2   aug  wed  95.5   99.9  513.3  13.2  23.3  31   4.5   0.0    0.55
142  8  6   aug  fri  90.1  108.0  529.8  12.5  21.2  51   8.9   0.0    0.61
143  1  2   jul  sat  90.0   51.3  296.3   8.7  16.6  53   5.4   0.0    0.71
144  2  5   aug  wed  95.5   99.9  513.3  13.2  23.8  32   5.4   0.0    0.77
145  6  5   aug  thu  95.2  131.7  578.8  10.4  27.4  22   4.0   0.0    0.90
147  8  3   sep  tue  84.4   73.4  671.9   3.2  24.2  28   3.6   0.0    0.96
148  2  2   aug  tue  94.8  108.3  647.1  17.0  17.4  43   6.7   0.0    1.07
149  8  6   sep  thu  93.7   80.9  685.2  17.9  23.7  25   4.5   0.0    1.12
150  6  5   jun  fri  92.5   56.4  433.3   7.1  23.2  39   5.4   0.0    1.19
151  9  9   jul  sun  90.1   68.6  355.2   7.2  24.8  29   2.2   0.0    1.36
152  3  4   jul  sat  90.1   51.2  424.1   6.2  24.6  43   1.8   0.0    1.43
153  5  4   sep  fri  94.3   85.1  692.3  15.9  20.1  47   4.9   0.0    1.46
154  1  5   sep  sat  93.4  145.4  721.4   8.1  29.6  27   2.7   0.0    1.46
155  7  4   aug  sun  94.8  108.3  647.1  17.0  16.4  47   1.3   0.0    1.56
156  2  4   sep  sat  93.4  145.4  721.4   8.1  28.6  27   2.2   0.0    1.61
157  2  2   aug  wed  92.1  111.2  654.1   9.6  18.4  45   3.6   0.0    1.63
158  2  4   aug  wed  92.1  111.2  654.1   9.6  20.5  35   4.0   0.0    1.64
159  7  4   sep  fri  92.4  117.9  668.0  12.2  19.0  34   5.8   0.0    1.69
160  7  4   mar  mon  90.1   39.7   86.6   6.2  16.1  29   3.1   0.0    1.75
161  6  4   aug  thu  95.2  131.7  578.8  10.4  20.3  41   4.0   0.0    1.90
162  6  3   mar  sat  90.6   50.1  100.4   7.8  15.2  31   8.5   0.0    1.94
163  8  6   sep  sat  92.5  121.1  674.4   8.6  17.8  56   1.8   0.0    1.95
164  8  5   sep  sun  89.7   90.0  704.4   4.8  17.8  67   2.2   0.0    2.01
167  6  5   aug  wed  96.0  127.1  570.5  16.5  23.4  33   4.5   0.0    2.51
169  8  6   aug  thu  95.2  131.7  578.8  10.4  20.7  45   2.2   0.0    2.55
170  5  4   sep  wed  92.9  133.3  699.6   9.2  21.9  35   1.8   0.0    2.57
171  8  6   aug  wed  85.6   90.4  609.6   6.6  17.4  50   4.0   0.0    2.69
..  .. ..   ...  ...   ...    ...    ...   ...   ...  ..   ...   ...     ...
459  7  4   aug  sat  91.6  273.8  819.1   7.7  15.5  72   8.0   0.0    1.94
471  4  3   may  fri  89.6   25.4   73.7   5.7  18.0  40   4.0   0.0   38.48
473  9  4   jun  sat  90.5   61.1  252.6   9.4  24.5  50   3.1   0.0   70.32
474  4  3   jun  thu  93.0  103.8  316.7  10.8  26.4  35   2.7   0.0   10.08
475  2  5   jun  thu  93.7  121.7  350.2  18.0  22.7  40   9.4   0.0    3.19
476  4  3   jul  thu  93.5   85.3  395.0   9.9  27.2  28   1.3   0.0    1.76
477  4  3   jul  sun  93.7  101.3  423.4  14.7  26.1  45   4.0   0.0    7.36
478  7  4   jul  sun  93.7  101.3  423.4  14.7  18.2  82   4.5   0.0    2.21
479  7  4   jul  mon  89.2  103.9  431.6   6.4  22.6  57   4.9   0.0  278.53
480  9  9   jul  thu  93.2  114.4  560.0   9.5  30.2  25   4.5   0.0    2.75
482  3  4   aug  sun  94.9  130.3  587.1  14.1  23.4  40   5.8   0.0    1.29
484  2  5   aug  sun  94.9  130.3  587.1  14.1  33.1  25   4.0   0.0   26.43
485  2  4   aug  mon  95.0  135.5  596.3  21.3  30.6  28   3.6   0.0    2.07
486  5  4   aug  tue  95.1  141.3  605.8  17.7  24.1  43   6.3   0.0    2.00
487  5  4   aug  tue  95.1  141.3  605.8  17.7  26.4  34   3.6   0.0   16.40
488  4  4   aug  tue  95.1  141.3  605.8  17.7  19.4  71   7.6   0.0   46.70
493  1  3   aug  fri  95.9  158.0  633.6  11.3  27.5  29   4.5   0.0   43.32
494  6  6   aug  sat  96.0  164.0  643.0  14.0  30.8  30   4.9   0.0    8.59
```

```
496  4  5   aug  mon  96.2  175.5  661.8  16.8  32.6  26   3.1   0.0    2.77
497  3  4   aug  tue  96.1  181.1  671.2  14.3  32.3  27   2.2   0.0   14.68
498  6  5   aug  tue  96.1  181.1  671.2  14.3  33.3  26   2.7   0.0   40.54
499  7  5   aug  tue  96.1  181.1  671.2  14.3  27.3  63   4.9   6.4   10.82
503  2  4   aug  wed  94.5  139.4  689.1  20.0  29.2  30   4.9   0.0    1.95
504  4  3   aug  wed  94.5  139.4  689.1  20.0  28.9  29   4.9   0.0   49.59
505  1  2   aug  thu  91.0  163.2  744.4  10.1  26.7  35   1.8   0.0    5.80
509  5  4   aug  fri  91.0  166.9  752.6   7.1  21.1  71   7.6   1.4    2.17
510  6  5   aug  fri  91.0  166.9  752.6   7.1  18.2  62   5.4   0.0    0.43
512  4  3   aug  sun  81.6   56.7  665.6   1.9  27.8  32   2.7   0.0    6.44
513  2  4   aug  sun  81.6   56.7  665.6   1.9  21.9  71   5.8   0.0   54.29
514  7  4   aug  sun  81.6   56.7  665.6   1.9  21.2  70   6.7   0.0   11.16

[212 rows x 13 columns]
```

In [6]:

```
#Merging and concatenating dataframes
import pandas as pd
market_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/global_
customer_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/glob
product_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/globa
shipping_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/glob
orders_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/global_
print(market_df.head())
print('\n')
print(customer_df.head())
print('\n')
print(product_df.head())
print('\n')
print(shipping_df.head())
print('\n')
print(orders_df.head())
print('\n')
```

```
     Ord_id   Prod_id   Ship_id    Cust_id    Sales  Discount  Order_Quantity  \
0  Ord_5446  Prod_16  SHP_7609  Cust_1818   136.81      0.01              23
1  Ord_5406  Prod_13  SHP_7549  Cust_1818    42.27      0.01              13
2  Ord_5446   Prod_4  SHP_7610  Cust_1818  4701.69      0.00              26
3  Ord_5456   Prod_6  SHP_7625  Cust_1818  2337.89      0.09              43
4  Ord_5485  Prod_17  SHP_7664  Cust_1818  4233.15      0.08              35

    Profit  Shipping_Cost  Product_Base_Margin
0   -30.51           3.60                 0.56
1     4.56           0.93                 0.54
2  1148.90           2.50                 0.59
3   729.34          14.30                 0.37
4  1219.87          26.30                 0.38


        Customer_Name Province   Region Customer_Segment Cust_id
0  MUHAMMED MACINTYRE  NUNAVUT  NUNAVUT   SMALL BUSINESS  Cust_1
1        BARRY FRENCH  NUNAVUT  NUNAVUT         CONSUMER  Cust_2
2       CLAY ROZENDAL  NUNAVUT  NUNAVUT        CORPORATE  Cust_3
3      CARLOS SOLTERO  NUNAVUT  NUNAVUT         CONSUMER  Cust_4
4        CARL JACKSON  NUNAVUT  NUNAVUT        CORPORATE  Cust_5


  Product_Category            Product_Sub_Category Prod_id
0  OFFICE SUPPLIES           STORAGE & ORGANIZATION  Prod_1
1  OFFICE SUPPLIES                       APPLIANCES  Prod_2
2  OFFICE SUPPLIES  BINDERS AND BINDER ACCESSORIES  Prod_3
3       TECHNOLOGY    TELEPHONES AND COMMUNICATION  Prod_4
4        FURNITURE               OFFICE FURNISHINGS  Prod_5


   Order_ID       Ship_Mode    Ship_Date Ship_id
```

```
0          3     REGULAR AIR  20-10-2010   SHP_1
1        293  DELIVERY TRUCK  02-10-2012   SHP_2
2        293     REGULAR AIR  03-10-2012   SHP_3
3        483     REGULAR AIR  12-07-2011   SHP_4
4        515     REGULAR AIR  30-08-2010   SHP_5


   Order_ID  Order_Date Order_Priority Ord_id
0         3  13-10-2010            LOW  Ord_1
1       293  01-10-2012           HIGH  Ord_2
2       483  10-07-2011           HIGH  Ord_3
3       515  28-08-2010  NOT SPECIFIED  Ord_4
4       613  17-06-2011           HIGH  Ord_5
```

In [11]:

```
#Merging and concatenating dataframes
import pandas as pd
market_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/global_
customer_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/glob
product_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/globa
shipping_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/glob
orders_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/global_

# Merging the dataframes
# Note that Cust_id is the common column/key, which is provided to the 'on' argum
# how = 'inner' makes sure that only the customer ids present in both dfs are inc
merge_df = pd.merge(market_df, customer_df, how='inner', on='Cust_id')
print(merge_df.head())
```

```
     Ord_id  Prod_id   Ship_id    Cust_id    Sales  Discount  Order_Quantity  \
0  Ord_5446  Prod_16  SHP_7609  Cust_1818   136.81      0.01              23
1  Ord_5406  Prod_13  SHP_7549  Cust_1818    42.27      0.01              13
2  Ord_5446   Prod_4  SHP_7610  Cust_1818  4701.69      0.00              26
3  Ord_5456   Prod_6  SHP_7625  Cust_1818  2337.89      0.09              43
4  Ord_5485  Prod_17  SHP_7664  Cust_1818  4233.15      0.08              35

    Profit  Shipping_Cost  Product_Base_Margin  Customer_Name Province Region
\
0   -30.51           3.60                 0.56  AARON BERGMAN  ALBERTA   WEST

1     4.56           0.93                 0.54  AARON BERGMAN  ALBERTA   WEST

2  1148.90           2.50                 0.59  AARON BERGMAN  ALBERTA   WEST

3   729.34          14.30                 0.37  AARON BERGMAN  ALBERTA   WEST

4  1219.87          26.30                 0.38  AARON BERGMAN  ALBERTA   WEST


  Customer_Segment
0        CORPORATE
1        CORPORATE
2        CORPORATE
3        CORPORATE
4        CORPORATE
```

In [14]:
```
#Merging and concatenating dataframes
import pandas as pd
market_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/global_
customer_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/glob
product_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/globa
shipping_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/glob
orders_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/global_
df_1 = pd.merge(market_df, customer_df, how='inner', on='Cust_id')
# Example 2: Select all orders from product category = office supplies and from th
# We now need to merge the product_df

df_2 = pd.merge(df_1, product_df, how='inner', on='Prod_id')
df_2.head()
print('\n')
# Select all orders from product category = office supplies and from the corporate
df_2.loc[(df_2['Product_Category']=='OFFICE SUPPLIES') & (df_2['Customer_Segment'
```

In [18]:

```
# dataframes having the same columns
df1 = pd.DataFrame({'Name': ['Aman', 'Joy', 'Rashmi', 'Saif'],
                    'Age': ['34', '31', '22', '33'],
                    'Gender': ['M', 'M', 'F', 'M']}
                  )

df2 = pd.DataFrame({'Name': ['Akhil', 'Asha', 'Preeti'],
                    'Age': ['31', '22', '23'],
                    'Gender': ['M', 'F', 'F']}
                  )
df1
print(df1)
print('\n')
print(df2)
print('\n')
# To concatenate them, one on top of the other, you can use pd.concat
# The first argument is a sequence (list) of dataframes
# axis = 0 indicates that we want to concat along the row axis
print(pd.concat([df1, df2], axis = 0))
print('\n')
# A useful and intuitive alternative to concat along the rows is the append() fun
# It concatenates along the rows
df1.append(df2)
```

```
  Age Gender    Name
0  34      M    Aman
1  31      M     Joy
2  22      F  Rashmi
3  33      M    Saif


  Age Gender    Name
0  31      M   Akhil
1  22      F    Asha
2  23      F  Preeti


  Age Gender    Name
0  34      M    Aman
1  31      M     Joy
2  22      F  Rashmi
3  33      M    Saif
0  31      M   Akhil
1  22      F    Asha
2  23      F  Preeti
```

Out[18]:

	Age	Gender	Name
0	34	M	Aman
1	31	M	Joy
2	22	F	Rashmi
3	33	M	Saif

	Age	Gender	Name
0	31	M	Akhil
1	22	F	Asha
2	23	F	Preeti

In [19]:

```
df1 = pd.DataFrame({'Name': ['Aman', 'Joy', 'Rashmi', 'Saif'],
                    'Age': ['34', '31', '22', '33'],
                    'Gender': ['M', 'M', 'F', 'M']}
                  )
df1
df2 = pd.DataFrame({'School': ['RK Public', 'JSP', 'Carmel Convent', 'St. Paul'],
                    'Graduation Marks': ['84', '89', '76', '91']}
                  )
df2
```

Out[19]:

	Graduation Marks	School
0	84	RK Public
1	89	JSP
2	76	Carmel Convent
3	91	St. Paul

In [22]:

```
# Loading libraries and files for Grouping and summarization
import numpy as np

import pandas as pd
market_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/global_
customer_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/glob
product_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/globa
shipping_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/glob
orders_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/global_

#First, we will merge all the dataframes, so we have all the data in one master_df
df_1 = pd.merge(market_df, customer_df, how='inner', on='Cust_id')
df_2 = pd.merge(df_1, product_df, how='inner', on='Prod_id')
df_3 = pd.merge(df_2, shipping_df, how='inner', on='Ship_id')
master_df = pd.merge(df_3, orders_df, how='inner', on='Ord_id')

master_df.head()
```

Out[22]:

```
     Ord_id   Prod_id    Ship_id    Cust_id    Sales  Discount  Order_Quantity    Profit  Shipping_
0  Ord_5446   Prod_16   SHP_7609  Cust_1818   136.81      0.01              23    -30.51
1  Ord_5446    Prod_4   SHP_7610  Cust_1818  4701.69      0.00              26   1148.90
2  Ord_5446    Prod_6   SHP_7608  Cust_1818   164.02      0.03              23    -47.64
3  Ord_2978   Prod_16   SHP_4112  Cust_1088   305.05      0.04              27     23.12
4  Ord_5484   Prod_16   SHP_7663  Cust_1820   322.82      0.05              35    -17.58

5 rows × 22 columns
```

In [56]:

```
#Step 1. Grouping using df.groupby()
#Typically, you group the data using a categorical variable, such as customer segi

#For example, in this case, we will group the data along Customer_Segment.
df_by_segment = master_df.groupby('Customer_Segment')
df_by_segment
print('\n')

#Step 2. Applying a Function
#After grouping, you apply a function to a numeric variable, such as mean(Sales),
# Step 2. Applying a function
# We can choose aggregate functions such as sum, mean, median, etc.
df_by_segment['Profit'].sum()
print('\n')
# Alternatively
df_by_segment.Profit.sum()
# For better readability, you may want to sort the summarised series:
df_by_segment.Profit.sum().sort_values(ascending = False)

#Step 3. Combining the results into a Data Structure
#You can optionally show the results as a dataframe.
# Converting to a df
print(pd.DataFrame(df_by_segment['Profit'].sum().sort_values(ascending = False)))
print('\n')
print(pd.DataFrame(df_by_segment['Profit'].mean().sort_values(ascending = False))
print('\n')
print(pd.DataFrame(df_by_segment['Profit'].count().sort_values(ascending = False)
print('\n')
print(pd.DataFrame(df_by_segment['Profit'].describe()))
```

```


                     Profit
Customer_Segment
CORPORATE         599746.00
HOME OFFICE       318354.03
SMALL BUSINESS    315708.01
CONSUMER          287959.94


                      Profit
Customer_Segment
CORPORATE         194.975943
SMALL BUSINESS    192.270408
CONSUMER          174.627010
HOME OFFICE       156.670290


                  Profit
Customer_Segment
CORPORATE           3076
HOME OFFICE         2032
CONSUMER            1649
```

```
SMALL BUSINESS      1642


                  count        mean          std       min      25%    50%  \
Customer_Segment                                                              
CONSUMER         1649.0  174.627010  1370.392654 -14140.70 -83.6300  0.260   
CORPORATE        3076.0  194.975943  1224.120222 -11861.46 -83.3075 -2.840   
HOME OFFICE      2032.0  156.670290  1054.269294 -12558.00 -88.7225 -0.430   
SMALL BUSINESS   1642.0  192.270408  1121.383478 -11984.40 -76.8825 -0.725   

                      75%       max  
Customer_Segment                     
CONSUMER         154.6500  27220.69  
CORPORATE        165.3350  14440.39  
HOME OFFICE      156.6775  10521.33  
SMALL BUSINESS   173.9525  13340.26  
```

In [44]:

```
   X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.0
5  8  6   aug  sun  92.3   85.3  488.0  14.7  22.2  29   5.4   0.0   0.0
6  8  6   aug  mon  92.3   88.9  495.6   8.5  24.1  27   3.1   0.0   0.0
7  8  6   aug  mon  91.5  145.4  608.2  10.7   8.0  86   2.2   0.0   0.0
8  8  6   sep  tue  91.0  129.5  692.6   7.0  13.1  63   5.4   0.0   0.0
9  7  5   sep  sat  92.5   88.0  698.6   7.1  22.8  40   4.0   0.0   0.0
```

In [69]:

```
#Group the data 'df' by 'month' and 'day' and find the mean value for column 'rai
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
print(df.head())
print('\n')
df_2 = df.groupby(['month','day'])
df_2




df_1 = pd.DataFrame(df_2['rain','wind'].mean())

print(df_1.head(20))
```

```
   X  Y month  day  FFMC   DMC     DC  ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2  26.2   94.3  5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6  35.4  669.1  6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6  43.7  686.9  6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7  33.3   77.5  9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3  51.3  102.2  9.6  11.4  99   1.8   0.0   0.0


               rain      wind
month day                    
apr   fri  0.000000  3.100000
      mon  0.000000  3.100000
      sat  0.000000  4.500000
      sun  0.000000  5.666667
      thu  0.000000  5.800000
      wed  0.000000  2.700000
aug   fri  0.066667  4.766667
      mon  0.000000  2.873333
      sat  0.000000  4.310345
      sun  0.025000  4.417500
      thu  0.000000  3.503846
      tue  0.300000  4.567857
      wed  0.000000  3.520000
dec   fri  0.000000  4.900000
      mon  0.000000  8.500000
      sun  0.000000  8.500000
      thu  0.000000  4.900000
      tue  0.000000  8.500000
      wed  0.000000  8.000000
feb   fri  0.000000  4.820000
```

In [75]:

```python
# lamda and pivot
# Loading libraries and files
import numpy as np
import pandas as pd

market_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/global_
customer_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/glob
product_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/globa
shipping_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/glob
orders_df = pd.read_csv("C:/Users/Z001MC7/Downloads/Introduction_to_Pandas/global_

# Merging the dataframes to create a master_df
df_1 = pd.merge(market_df, customer_df, how='inner', on='Cust_id')
df_2 = pd.merge(df_1, product_df, how='inner', on='Prod_id')
df_3 = pd.merge(df_2, shipping_df, how='inner', on='Ship_id')
master_df = pd.merge(df_3, orders_df, how='inner', on='Ord_id')



# Create a function to be applied
def is_positive(x):
    return x > 0

# Create a new column
master_df['is_profitable'] = master_df['Profit'].apply(is_positive)
master_df.head()

print('\n')

#or we can use a simpler way

# Create a new column using a lambda function
master_df['is_profitable'] = master_df['Profit'].apply(lambda x: x > 0)
master_df.head()
```

Out[75]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipping_
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51	
1	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90	
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.64	
3	Ord_2978	Prod_16	SHP_4112	Cust_1088	305.05	0.04	27	23.12	
4	Ord_5484	Prod_16	SHP_7663	Cust_1820	322.82	0.05	35	-17.58	

5 rows × 23 columns

In [74]:

```
# Comparing percentage of profitable orders across customer segments using the ne
by_segment = master_df.groupby('Customer_Segment')
by_segment.is_profitable.mean()
```

Out[74]:

```
Customer_Segment
CONSUMER          0.500910
CORPORATE         0.481469
HOME OFFICE       0.498524
SMALL BUSINESS    0.496346
Name: is_profitable, dtype: float64
```

In [76]:

```
# You can also use apply and lambda to alter existing columns
# E.g. you want to see Profit as one decimal place
# apply the round() function
master_df['Profit'] = master_df['Profit'].apply(lambda x: round(x, 1))
master_df.head()
```

Out[76]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipping_
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.5	
1	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.9	
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.6	
3	Ord_2978	Prod_16	SHP_4112	Cust_1088	305.05	0.04	27	23.1	
4	Ord_5484	Prod_16	SHP_7663	Cust_1820	322.82	0.05	35	-17.6	

5 rows × 23 columns

In [77]:

```
# Creating a column Profit / Order_Quantity
master_df['profit_per_qty'] = master_df['Profit'] / master_df['Order_Quantity']
master_df.head()
```

Out[77]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipping_
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.5	
1	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.9	
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.6	
3	Ord_2978	Prod_16	SHP_4112	Cust_1088	305.05	0.04	27	23.1	
4	Ord_5484	Prod_16	SHP_7663	Cust_1820	322.82	0.05	35	-17.6	

5 rows × 24 columns

In [78]:

```
#pivot table.
# E.g. Compare average Sales across customer segments
master_df.pivot_table(values = 'Sales', index = 'Customer_Segment', aggfunc = 'me
```

Out[78]:

	Sales
Customer_Segment	
CONSUMER	1857.859965
CORPORATE	1787.680389
HOME OFFICE	1754.312931
SMALL BUSINESS	1698.124841

In [79]:

```
# E.g. compare total number of profitable orders across regions
# Note that since is_profitable is 1/0, we can directly compute the sum
master_df.pivot_table(values = 'is_profitable', index = 'Region', aggfunc = 'sum'
```

Out[79]:

	is_profitable
Region	
ATLANTIC	544.0
NORTHWEST TERRITORIES	194.0
NUNAVUT	38.0
ONTARIO	916.0
PRARIE	852.0
QUEBEC	360.0
WEST	969.0
YUKON	262.0

In [80]:

```
# Grouping by both rows and columns
# Compare the total profit across product categories and customer segments
# Since there are two categorical variables, we use both rows (index) and columns
master_df.pivot_table(values = 'Profit',
                      index = 'Product_Category',
                      columns = 'Customer_Segment',
                      aggfunc = 'sum')
```

Out[80]:

Customer_Segment	CONSUMER	CORPORATE	HOME OFFICE	SMALL BUSINESS
Product_Category				
FURNITURE	42728.5	22008.3	23978.6	28717.5
OFFICE SUPPLIES	88532.4	203038.8	121145.6	105306.8
TECHNOLOGY	156700.1	374701.1	173230.6	181684.1

In [83]:

```
#Group the data 'df' by 'month' and 'day' and find the mean value for column 'rai
import numpy as np
import pandas as pd
df = pd.read_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')
df_1 = df.pivot_table(values = ['rain','wind'],
                      index = ['month','day'],
                      aggfunc = 'mean')
print(df_1.head(20))
```

```
               rain      wind
month day                    
apr   fri  0.000000  3.100000
      mon  0.000000  3.100000
      sat  0.000000  4.500000
      sun  0.000000  5.666667
      thu  0.000000  5.800000
      wed  0.000000  2.700000
aug   fri  0.066667  4.766667
      mon  0.000000  2.873333
      sat  0.000000  4.310345
      sun  0.025000  4.417500
      thu  0.000000  3.503846
      tue  0.300000  4.567857
      wed  0.000000  3.520000
dec   fri  0.000000  4.900000
      mon  0.000000  8.500000
      sun  0.000000  8.500000
      thu  0.000000  4.900000
      tue  0.000000  8.500000
      wed  0.000000  8.000000
feb   fri  0.000000  4.820000
```

In [ ]:

```
#below this we start new topic "Getting and Cleaning Data"
```

In [2]:

```
#reading txt file
import numpy as np
import pandas as pd
df = pd.read_csv("C:/Users/Z001MC7/Downloads/3_Getting_and_Cleaning_Data/companie
df.head()
```

Out[2]:

	permalink	name	homepage_url	category_list	status	country_co
0	/Organization/-Fame	#fame	http://livfame.com	Media	operating	IN
1	/Organization/-Qounter	:Qounter	http://www.qounter.com	Application Platforms|Real Time|Social Network...	operating	US
2	/Organization/-The-One-Of-Them-Inc-	(THE) ONE of THEM,Inc.	http://oneofthem.jp	Apps|Games|Mobile	operating	Na
3	/Organization/0-6-Com	0-6.com	http://www.0-6.com	Curated Web	operating	CH
4	/Organization/004-Technologies	004 Technologies	http://004gmbh.de/en/004-interact	Software	operating	US

In [6]:

```
import numpy as np
import pandas as pd
import pymysql

# create a connection object 'conn'
conn = pymysql.connect(host="localhost", # your host, localhost for your local ma
                     user="root", # your username, usually "root" for localhost
                      passwd="Shaunak@2011", # your password
                      db="world") # name of the data base; world comes inbuilt wi

# create a cursor object c
c = conn.cursor()

# execute a query using c.execute
c.execute("select * from city;")

# getting the first row of data as a tuple
all_rows = c.fetchall()

# to get only the first row, use c.fetchone() instead
# notice that it returns a tuple of tuples: each row is a tuple
print(type(all_rows))

# printing the first few rows
print(all_rows[:5])

#Now, it would be useful to convert the list into a dataframe, since you can now u

#pd.DataFrame(list_of_tuples) converts each tuple in the list to a row in the DF.
df = pd.DataFrame(list(all_rows), columns=["ID", "Name", "Country", "District", "
df.head()
```

```
<class 'tuple'>
((1, 'Kabul', 'AFG', 'Kabol', 1780000), (2, 'Qandahar', 'AFG', 'Qandahar', 2375
00), (3, 'Herat', 'AFG', 'Herat', 186800), (4, 'Mazar-e-Sharif', 'AFG', 'Balk
h', 127800), (5, 'Amsterdam', 'NLD', 'Noord-Holland', 731200))
```

Out[6]:

	ID	Name	Country	District	Population
0	1	Kabul	AFG	Kabol	1780000
1	2	Qandahar	AFG	Qandahar	237500
2	3	Herat	AFG	Herat	186800
3	4	Mazar-e-Sharif	AFG	Balkh	127800
4	5	Amsterdam	NLD	Noord-Holland	731200

In [8]:

```
# website data
import requests, bs4

# getting HTML from the Google Play web page
url = "https://play.google.com/store/apps/details?id=com.facebook.orca&hl=en"
req = requests.get(url)

# create a bs4 object
# To avoid warnings, provide "html5lib" explicitly
soup = bs4.BeautifulSoup(req.text, "html5lib")
# getting all the text inside class = "review-body"
reviews = soup.select('.review-body')
print(type(reviews))
print(len(reviews))
print("\n")

# printing an element of the reviews list
print(reviews[6])
```

```
<class 'list'>
0



---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-8-091da7e33ea4> in <module>()
     16 
     17 # printing an element of the reviews list
---> 18 print(reviews[6])

IndexError: list index out of range
```

In [9]:

```python
# Practice problem: Scraping amazon.in to get shoe price data
import pprint

url = "https://www.amazon.in/s/ref=nb_sb_noss?url=search-alias%3Daps&field-keywor
req = requests.get(url)

# create a bs4 object
# To avoid warnings, provide "html5lib" explicitly
soup = bs4.BeautifulSoup(req.text, "html5lib")

# get shoe names
# shoe_data = soup.select('.a-size-medium')
# shoe_data = soup.select('.a-size-small.a-link-normal.a-text-normal')
# print(len(shoe_data))
# print(pprint.pprint(shoe_data))

# get shoe prices
shoe_prices = soup.select('.a-price-whole')
print(len(shoe_prices))
pprint.pprint(shoe_prices)
```

```
0
[]
```

In [10]:

```python
import numpy as np
import pandas as pd

# Need requests to connect to the URL, json to convert JSON to dict
import requests, json
import pprint

# joining words in the address by a "+"
add = "UpGrad, Nishuvi building, Anne Besant Road, Worli, Mumbai"
split_address = add.split(" ")
address = "+".join(split_address)
print(address)
```

```
UpGrad,+Nishuvi+building,+Anne+Besant+Road,+Worli,+Mumbai
```

In [11]:
```
#JASON file
api_key = "AIzaSyBXrK8md7uaOcpRpaluEGZAtdXS4pcI5xo"

url = "https://maps.googleapis.com/maps/api/geocode/json?address={0}&key={1}".for
r = requests.get(url)

# The r.text attribute contains the text in the response object
print(type(r.text))
print(r.text)
```

```
<class 'str'>
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "75",
               "short_name" : "75",
               "types" : [ "street_number" ]
            },
            {
               "long_name" : "Doctor Annie Besant Road",
               "short_name" : "Dr Annie Besant Rd",
               "types" : [ "route" ]
            },
            {
               "long_name" : "Bhim Nagar",
               "short_name" : "Bhim Nagar",
               "types" : [ "political", "sublocality", "sublocality_level_2" ]
            },
            {
               "long_name" : "Worli",
               "short_name" : "Worli",
               "types" : [ "political", "sublocality", "sublocality_level_1" ]
            },
            {
               "long_name" : "Mumbai",
               "short_name" : "Mumbai",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "Mumbai",
               "short_name" : "Mumbai",
               "types" : [ "administrative_area_level_2", "political" ]
            },
            {
               "long_name" : "Maharashtra",
               "short_name" : "MH",
               "types" : [ "administrative_area_level_1", "political" ]
            },
            {
               "long_name" : "India",
               "short_name" : "IN",
               "types" : [ "country", "political" ]
            },
            {
               "long_name" : "400018",
               "short_name" : "400018",
               "types" : [ "postal_code" ]
            }
         ],
         "formatted_address" : "Ground Floor, Nishuvi Building, 75, Dr Annie Besant Rd, Bhim Nagar, Worli, Mumbai, Maharashtra 400018, India",
         "geometry" : {
            "location" : {
               "lat" : 18.9947946,
               "lng" : 72.81638699999999
            },
            "location_type" : "ROOFTOP",
            "viewport" : {
               "northeast" : {
                  "lat" : 18.9961435802915,
                  "lng" : 72.81773598029149
               },
               "southwest" : {
                  "lat" : 18.9934456197085,
                  "lng" : 72.81503801970848
               }
            }
         },
         "place_id" : "ChIJZyC-Y4_O5zsRhmWdnZDvxUg",
         "types" : [ "establishment", "point_of_interest" ]
      }
   ],
   "status" : "OK"
}
```

In [12]:

```
# converting the json object to a dict using json.loads()
r_dict = json.loads(r.text)

# the pretty printing library pprint makes it easy to read large dictionaries
pprint.pprint(r_dict)
```

```
{'results': [{'address_components': [{'long_name': '75',
                                      'short_name': '75',
                                      'types': ['street_number']},
                                     {'long_name': 'Doctor Annie Besant Road',
                                      'short_name': 'Dr Annie Besant Rd',
                                      'types': ['route']},
                                     {'long_name': 'Bhim Nagar',
                                      'short_name': 'Bhim Nagar',
                                      'types': ['political',
                                                'sublocality',
                                                'sublocality_level_2']},
                                     {'long_name': 'Worli',
                                      'short_name': 'Worli',
                                      'types': ['political',
                                                'sublocality',
                                                'sublocality_level_1']},
                                     {'long_name': 'Mumbai',
                                      'short_name': 'Mumbai',
                                      'types': ['locality', 'political']},
                                     {'long_name': 'Mumbai',
                                      'short_name': 'Mumbai',
                                      'types': ['administrative_area_level_2',
                                                'political']},
                                     {'long_name': 'Maharashtra',
                                      'short_name': 'MH',
                                      'types': ['administrative_area_level_1',
                                                'political']},
                                     {'long_name': 'India',
                                      'short_name': 'IN',
                                      'types': ['country', 'political']},
                                     {'long_name': '400018',
                                      'short_name': '400018',
                                      'types': ['postal_code']}],
              'formatted_address': 'Ground Floor, Nishuvi Building, 75, Dr '
                                   'Annie Besant Rd, Bhim Nagar, Worli, '
                                   'Mumbai, Maharashtra 400018, India',
              'geometry': {'location': {'lat': 18.9947946,
                                        'lng': 72.81638699999999},
                           'location_type': 'ROOFTOP',
                           'viewport': {'northeast': {'lat': 18.9961435802915,
                                                      'lng': 72.8177359802914
9},
                                        'southwest': {'lat': 18.9934456197085,
                                                      'lng': 72.8150380197084
8}}},
              'place_id': 'ChIJZyC-Y4_O5zsRhmWdnZDvxUg',
              'types': ['establishment', 'point_of_interest']}],
 'status': 'OK'}
```

In [13]:

```
# The dict has two main keys - status and results
r_dict.keys()
```

Out[13]:

```
dict_keys(['results', 'status'])
```

In [14]:

```
lat = r_dict['results'][0]['geometry']['location']['lat']
lng = r_dict['results'][0]['geometry']['location']['lng']

print((lat, lng))
```

```
(18.9947946, 72.81638699999999)
```

In [15]:

```
#Writing a Function for this Procedure
# Input to the fn: Address in standard human-readable form
# Output: Tuple (lat, lng)

api_key = "AIzaSyBXrK8md7uaOcpRpaluEGZAtdXS4pcI5xo"

def address_to_latlong(address):
    # convert address to the form x+y+z
    split_address = address.split(" ")
    address = "+".join(split_address)

    # pass the address to the URL
    url = "https://maps.googleapis.com/maps/api/geocode/json?address={0}&key={1}"

    # connect to the URL, get response and convert to dict
    r = requests.get(url)
    r_dict = json.loads(r.text)
    lat = r_dict['results'][0]['geometry']['location']['lat']
    lng = r_dict['results'][0]['geometry']['location']['lng']

    return (lat, lng)


# getting some coordinates
print(address_to_latlong("UpGrad, Nishuvi Building, Worli, Mumbai"))
print(address_to_latlong("IIIT Bangalore, Electronic City, Bangalore"))
```

```
(18.9947946, 72.81638699999999)
(12.8447512, 77.6632317)
```

In [18]:

```
import PyPDF2

# reading the pdf file
pdf_object = open('C:/Users/Z001MC7/Downloads/3_Getting_and_Cleaning_Data/animal_
pdf_reader = PyPDF2.PdfFileReader(pdf_object)

# Number of pages in the PDF file
print(pdf_reader.numPages)

# get a certain page's text
page_object = pdf_reader.getPage(5)

# Extract text from the page_object
print(page_object.extractText())
```

```
55
Cowsandhorses,geeseandturkeys,
Allmusttoilforfreedom'ssake.
BeastsofEngland,beastsofIreland,
Beastsofeverylandandclime,
Hearkenwellandspreadmytidings
Ofthegoldenfuturetime.
Thesingingofthissongthrewtheanimalsintothewildestexcitement.
AlmostbeforeMajorhadreachedtheend,theyhadbegunsingingitforthem-
selves.Eventhestupidestofthemhadalreadypickedupthetuneandafewof
thewords,andasforthecleverones,suchasthepigsanddogs,theyhadthe
entiresongbyheartwithinafewminutes.Andthen,afterafewpreliminary
tries,thewholefarmburstoutinto
BeastsofEngland
intremendousunison.
Thecowslowedit,thedogswhinedit,thesheepbleatedit,thehorseswhinnied
it,theducksquackedit.Theyweresodelightedwiththesongthattheysang
itrightthroughetimesinsuccession,andmighthavecontinuedsingingitall
nightiftheyhadnotbeeninterrupted.
Unfortunately,theuproarawokeMr.Jones,whosprangoutofbed,making
surethattherewasafoxintheyard.Heseizedthegunwhichalwaysstoodina
cornerofhisbedroom,andletyachargeofnumber6shotintothedarkness.
Thepelletsburiedthemselvesinthewallofthebarnandthemeetingbroke
uphurriedly.Everyonetohisownsleeping-place.Thebirdsjumpedonto
theirperches,theanimalssettleddowninthestraw,andthewholefarmwas
asleepinamoment.
5


PdfReadWarning: Xref table not zero-indexed. ID numbers for objects will be cor
rected. [pdf.py:1736]
```

In [2]:

```
# cleaning data sets
import numpy as np
import pandas as pd
df=pd.read_csv("C:/Users/Z001MC7/Downloads/3_Getting_and_Cleaning_Data/melbourne.
df.head()
```

Out[2]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...
0	Abbotsford	68 Studley St	2	h	NaN	SS	Jellis	03-09-2016	2.5	3067.0	...
1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	03-12-2016	2.5	3067.0	...
2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	04-02-2016	2.5	3067.0	...
3	Abbotsford	18/659 Victoria St	3	u	NaN	VB	Rounds	04-02-2016	2.5	3067.0	...
4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	04-03-2017	2.5	3067.0	...

5 rows × 21 columns

In [3]:

```
# approx 23k rows, 21 columns
print('\n')
print(df.shape)
print('\n')
print(df.info())
```

```


(23547, 21)


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23547 entries, 0 to 23546
Data columns (total 21 columns):
Suburb           23547 non-null object
Address          23547 non-null object
Rooms            23547 non-null int64
Type             23547 non-null object
Price            18396 non-null float64
Method           23547 non-null object
SellerG          23547 non-null object
Date             23547 non-null object
Distance         23546 non-null float64
Postcode         23546 non-null float64
Bedroom2         19066 non-null float64
Bathroom         19063 non-null float64
Car              18921 non-null float64
Landsize         17410 non-null float64
BuildingArea     10018 non-null float64
YearBuilt        11540 non-null float64
CouncilArea      15656 non-null object
Lattitude        19243 non-null float64
Longtitude       19243 non-null float64
Regionname       23546 non-null object
Propertycount    23546 non-null float64
dtypes: float64(12), int64(1), object(8)
memory usage: 3.8+ MB
None
```

In [4]:
```
df.isnull()
```

Out[4]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Ba
0	False	False	False	False	True	False	False	False	False	False	...	
1	False	False	False	False	False	False	False	False	False	False	...	
2	False	False	False	False	False	False	False	False	False	False	...	
3	False	False	False	False	True	False	False	False	False	False	...	
4	False	False	False	False	False	False	False	False	False	False	...	
5	False	False	False	False	False	False	False	False	False	False	...	
6	False	False	False	False	False	False	False	False	False	False	...	
7	False	False	False	False	True	False	False	False	False	False	...	
8	False	False	False	False	True	False	False	False	False	False	...	
9	False	False	False	False	True	False	False	False	False	False	...	
10	False	False	False	False	False	False	False	False	False	False	...	
11	False	False	False	False	False	False	False	False	False	False	...	
12	False	False	False	False	True	False	False	False	False	False	...	
13	False	False	False	False	True	False	False	False	False	False	...	
14	False	False	False	False	False	False	False	False	False	False	...	
15	False	False	False	False	False	False	False	False	False	False	...	
16	False	False	False	False	False	False	False	False	False	False	...	
17	False	False	False	False	False	False	False	False	False	False	...	
18	False	False	False	False	False	False	False	False	False	False	...	
19	False	False	False	False	False	False	False	False	False	False	...	
20	False	False	False	False	True	False	False	False	False	False	...	
21	False	False	False	False	False	False	False	False	False	False	...	
22	False	False	False	False	False	False	False	False	False	False	...	
23	False	False	False	False	False	False	False	False	False	False	...	
24	False	False	False	False	False	False	False	False	False	False	...	
25	False	False	False	False	False	False	False	False	False	False	...	
26	False	False	False	False	False	False	False	False	False	False	...	
27	False	False	False	False	False	False	False	False	False	False	...	
28	False	False	False	False	False	False	False	False	False	False	...	
29	False	False	False	False	False	False	False	False	False	False	...	
...	...	...	...	...	...	...	...	...	...	...	...	
23517	False	False	False	False	False	False	False	False	False	False	...	
23518	False	False	False	False	False	False	False	False	False	False	...	
23519	False	False	False	False	False	False	False	False	False	False	...	

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Ba
23520	False	False	False	False	True	False	False	False	False	False	...	
23521	False	False	False	False	False	False	False	False	False	False	...	
23522	False	False	False	False	True	False	False	False	False	False	...	
23523	False	False	False	False	False	False	False	False	False	False	...	
23524	False	False	False	False	False	False	False	False	False	False	...	
23525	False	False	False	False	False	False	False	False	False	False	...	
23526	False	False	False	False	False	False	False	False	False	False	...	
23527	False	False	False	False	False	False	False	False	False	False	...	
23528	False	False	False	False	True	False	False	False	False	False	...	
23529	False	False	False	False	True	False	False	False	False	False	...	
23530	False	False	False	False	True	False	False	False	False	False	...	
23531	False	False	False	False	False	False	False	False	False	False	...	
23532	False	False	False	False	False	False	False	False	False	False	...	
23533	False	False	False	False	True	False	False	False	False	False	...	
23534	False	False	False	False	False	False	False	False	False	False	...	
23535	False	False	False	False	False	False	False	False	False	False	...	
23536	False	False	False	False	False	False	False	False	False	False	...	
23537	False	False	False	False	False	False	False	False	False	False	...	
23538	False	False	False	False	False	False	False	False	False	False	...	
23539	False	False	False	False	False	False	False	False	False	False	...	
23540	False	False	False	False	False	False	False	False	False	False	...	
23541	False	False	False	False	False	False	False	False	False	False	...	
23542	False	False	False	False	True	False	False	False	False	False	...	
23543	False	False	False	False	True	False	False	False	False	False	...	
23544	False	False	False	False	False	False	False	False	False	False	...	
23545	False	False	False	False	False	False	False	False	False	False	...	
23546	False	False	False	False	False	False	False	False	False	False	...	

23547 rows × 21 columns

In [6]:

```
## summing up the missing values (column-wise)
df.isnull().sum()
```

Out[6]:

```
Suburb              0
Address             0
Rooms               0
Type                0
Price            5151
Method              0
SellerG             0
Date                0
Distance            1
Postcode            1
Bedroom2         4481
Bathroom         4484
Car              4626
Landsize         6137
BuildingArea    13529
YearBuilt       12007
CouncilArea      7891
Lattitude        4304
Longtitude       4304
Regionname          1
Propertycount       1
dtype: int64
```

In [7]:

```
# columns having at least one missing value
df.isnull().any()

# above is equivalent to axis=0 (by default, any() operates on columns)
df.isnull().any(axis=0)
```

Out[7]:

```
Suburb           False
Address          False
Rooms            False
Type             False
Price             True
Method           False
SellerG          False
Date             False
Distance          True
Postcode          True
Bedroom2          True
Bathroom          True
Car               True
Landsize          True
BuildingArea      True
YearBuilt         True
CouncilArea       True
Lattitude         True
Longtitude        True
Regionname        True
Propertycount     True
dtype: bool
```

In [9]:

```
# rows having at least one missing value
df.isnull().any(axis =1)
```

Out[9]:

```
0         True
1         True
2        False
3         True
4        False
5         True
6        False
7         True
8         True
9         True
10        True
11       False
12        True
13        True
14       False
15        True
16        True
17        True
18       False
19        True
20        True
21        True
22        True
23        True
24       False
25       False
26        True
27        True
28        True
29        True
         ...
23517     True
23518     True
23519     True
23520     True
23521     True
23522     True
23523     True
23524     True
23525     True
23526     True
23527     True
23528     True
23529     True
23530     True
23531     True
23532     True
23533     True
23534     True
23535     True
23536     True
```

```
23537     True
23538     True
23539     True
23540     True
23541     True
23542     True
23543     True
23544     True
23545     True
23546     True
Length: 23547, dtype: bool
```

In [10]:

```
# sum it up to check how many rows have all missing values
df.isnull().all(axis=1).sum()
```

Out[10]:

```
0
```

In [11]:

```
# sum of misisng values in each row
df.isnull().sum(axis=1)
```

Out[11]:

```
0        3
1        2
2        0
3        3
4        0
5        2
6        0
7        1
8        2
9        2
10       2
11       0
12       1
13       1
14       0
15       9
16       9
17       2
18       0
19       9
20       1
21       9
22       9
23       2
24       0
25       0
26       7
27       9
28       2
29       2
        ..
23517    2
23518    3
23519    3
23520    2
23521    1
23522    4
23523    3
23524    3
23525    1
23526    2
23527    1
23528    5
23529    4
23530    5
23531    9
23532    1
23533    5
23534    2
23535    3
23536    4
23537    2
23538    1
```

```
23539    2
23540    2
23541    1
23542    2
23543    8
23544    4
23545    1
23546    2
Length: 23547, dtype: int64
```

In [12]:

```
# Treat Missing Values in Columns¶
#Let's now treat missing values in columns. Let's look at the number of NaNs in e
#this time as the percentage of missing values in each column. Notice that we cal
# summing up the missing values (column-wise)
round(100*(df.isnull().sum()/len(df.index)), 2)
```

Out[12]:

```
Suburb            0.00
Address           0.00
Rooms             0.00
Type              0.00
Price            21.88
Method            0.00
SellerG           0.00
Date              0.00
Distance          0.00
Postcode          0.00
Bedroom2         19.03
Bathroom         19.04
Car              19.65
Landsize         26.06
BuildingArea     57.46
YearBuilt        50.99
CouncilArea      33.51
Lattitude        18.28
Longtitude       18.28
Regionname        0.00
Propertycount     0.00
dtype: float64
```

In [13]:
```
# removing the three columns
df = df.drop('BuildingArea', axis=1)
df = df.drop('YearBuilt', axis=1)
df = df.drop('CouncilArea', axis=1)

round(100*(df.isnull().sum()/len(df.index)), 2)
```

Out[13]:
```
Suburb            0.00
Address           0.00
Rooms             0.00
Type              0.00
Price            21.88
Method            0.00
SellerG           0.00
Date              0.00
Distance          0.00
Postcode          0.00
Bedroom2         19.03
Bathroom         19.04
Car              19.65
Landsize         26.06
Lattitude        18.28
Longtitude       18.28
Regionname        0.00
Propertycount     0.00
dtype: float64
```

In [14]:
```
#Treating Missing Values in Rows¶
#Now, we need to either delete or impute the missing values. First, let's see if
#significant number of missing values. If so, we can drop those rows, and then ta
#After dropping three columns, we now have 18 columns to work with. Just to inspe
#let's have a look at the rows having more than 5 missing values
df[df.isnull().sum(axis=1) > 5]
```

Out[14]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance
15	Abbotsford	217 Langridge St	3	h	1000000.0	S	Jellis	08-10-2016	2.5
16	Abbotsford	18a Mollison St	2	t	745000.0	S	Jellis	08-10-2016	2.5
19	Abbotsford	403/609 Victoria St	2	u	542000.0	S	Dingle	08-10-2016	2.5
21	Abbotsford	25/84 Trenerry Cr	2	u	760000.0	SP	Biggin	10-12-2016	2.5
22	Abbotsford	106/119 Turner St	1	u	481000.0	SP	Purplebricks	10-12-2016	2.5

In [15]:

```
# count the number of rows having > 5 missing values
# use len(df.index)
len(df[df.isnull().sum(axis=1) > 5].index)
```

Out[15]:

```
4278
```

In [16]:

```
# 4278 rows have more than 5 missing values
# calculate the percentage
100*(len(df[df.isnull().sum(axis=1) > 5].index) / len(df.index))
```

Out[16]:

```
18.16791948018856
```

In [17]:

```
# retaining the rows having <= 5 NaNs
df = df[df.isnull().sum(axis=1) <= 5]

# look at the summary again
round(100*(df.isnull().sum()/len(df.index)), 2)
```

Out[17]:

```
Suburb            0.00
Address           0.00
Rooms             0.00
Type              0.00
Price            21.71
Method            0.00
SellerG           0.00
Date              0.00
Distance          0.00
Postcode          0.00
Bedroom2          1.05
Bathroom          1.07
Car               1.81
Landsize          9.65
Lattitude         0.13
Longtitude        0.13
Regionname        0.00
Propertycount     0.00
dtype: float64
```

In [18]:

```
# removing NaN Price rows
df = df[~np.isnan(df['Price'])]

round(100*(df.isnull().sum()/len(df.index)), 2)
```

Out[18]:

```
Suburb           0.00
Address          0.00
Rooms            0.00
Type             0.00
Price            0.00
Method           0.00
SellerG          0.00
Date             0.00
Distance         0.00
Postcode         0.00
Bedroom2         1.05
Bathroom         1.07
Car              1.76
Landsize         9.83
Lattitude        0.15
Longtitude       0.15
Regionname       0.00
Propertycount    0.00
dtype: float64
```

In [19]:

```
df['Landsize'].describe()
```

Out[19]:

```
count     13603.000000
mean        558.116371
std        3987.326586
min           0.000000
25%         176.500000
50%         440.000000
75%         651.000000
max      433014.000000
Name: Landsize, dtype: float64
```

In [20]:

```
# removing NaNs in Landsize
df = df[~np.isnan(df['Landsize'])]

round(100*(df.isnull().sum()/len(df.index)), 2)
```

Out[20]:

```
Suburb           0.00
Address          0.00
Rooms            0.00
Type             0.00
Price            0.00
Method           0.00
SellerG          0.00
Date             0.00
Distance         0.00
Postcode         0.00
Bedroom2         0.00
Bathroom         0.01
Car              0.46
Landsize         0.00
Lattitude        0.16
Longtitude       0.16
Regionname       0.00
Propertycount    0.00
dtype: float64
```

In [21]:
```
# rows having Lattitude and Longitude missing
df[np.isnan(df['Lattitude'])]
```

Out[21]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	P
2572	Burwood	23 Monica St	3	h	990000.0	VB	Fletchers	17-09-2016	11.7	
3257	Clifton Hill	3/268 Alexandra Pde E	1	u	363000.0	S	hockingstuart	27-06-2016	3.4	
4485	Footscray	483 Barkly St	3	t	781000.0	S	Jas	27-11-2016	6.4	
5170	Hampton East	7 Seafoam St	4	t	1185000.0	S	RT	28-05-2016	14.5	
10745	Williamstown North	4/9 Adeline St	1	u	355000.0	S	Sweeney	27-11-2016	8.9	
13223	Melbourne	1913/228 Abeckett St	3	u	1175000.0	PI	Icon	29-04-2017	2.8	
14008	Brooklyn	9 Richards Ct	3	h	750000.0	S	hockingstuart	20-05-2017	10.9	
14132	North Melbourne	13/201 Abbotsford St	2	t	755000.0	PI	Nelson	29-04-2017	2.3	
14139	Oakleigh South	4 Druitt St	4	h	1205500.0	S	Woodards	22-04-2017	14.7	
14142	Oakleigh South	298 Warrigal Rd	3	h	799999.0	S	Woodards	29-04-2017	14.7	
14143	Oakleigh South	11 Yarra Ct	5	h	1200000.0	PI	Buxton	29-04-2017	14.7	
14976	Essendon	9 Washington St	3	h	1520000.0	S	Brad	03-06-2017	7.5	
15278	Seddon	60 Station Rd	4	h	1500000.0	SP	Sweeney	03-06-2017	5.1	
15561	Croydon	3 Silvergrass Ct	3	h	630000.0	S	McGrath	17-06-2017	23.0	
16358	Kensington	201/102 Rankins Rd	2	u	876000.0	S	Rendina	24-06-2017	3.4	
16564	Strathmore	48 Lebanon St	3	h	950000.0	PI	Considine	24-06-2017	8.2	

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	P
17616	Kensington	109/102 Rankins Rd	1	u	410000.0	PI	Rendina	08-07-2017	3.4	
17620	Keysborough	19 Denmark Rd	5	h	1100000.0	PI	VicHomes	08-07-2017	25.2	
18141	Lalor	83 Rotino Cr	3	t	463000.0	S	HAR	12-08-2017	16.3	
18614	Mickleham	17 Primavera Dr	4	h	610000.0	S	Ray	15-07-2017	20.6	
18793	Wollert	13 Strathalbyn Ch	4	h	631000.0	PI	hockingstuart	15-07-2017	25.5	
18995	Greenvale	40 Frontier Av	3	t	470000.0	SP	Barry	22-07-2017	20.4	

In [22]:

```
df.loc[:, ['Lattitude', 'Longtitude']].describe()
```

Out[22]:

	Lattitude	Longtitude
count	13581.000000	13581.000000
mean	-37.809204	144.995221
std	0.079257	0.103913
min	-38.182550	144.431810
25%	-37.856820	144.929600
50%	-37.802360	145.000100
75%	-37.756400	145.058320
max	-37.408530	145.526350

In [23]:

```
# imputing Lattitude and Longitude by mean values
df.loc[np.isnan(df['Lattitude']), ['Lattitude']] = df['Lattitude'].mean()
df.loc[np.isnan(df['Longtitude']), ['Longtitude']] = df['Longtitude'].mean()

round(100*(df.isnull().sum()/len(df.index)), 2)
```

Out[23]:

```
Suburb           0.00
Address          0.00
Rooms            0.00
Type             0.00
Price            0.00
Method           0.00
SellerG          0.00
Date             0.00
Distance         0.00
Postcode         0.00
Bedroom2         0.00
Bathroom         0.01
Car              0.46
Landsize         0.00
Lattitude        0.00
Longtitude       0.00
Regionname       0.00
Propertycount    0.00
dtype: float64
```

In [24]:

```
df.loc[:, ['Bathroom', 'Car']].describe()
```

Out[24]:

	Bathroom	Car
count	13602.000000	13540.000000
mean	1.534921	1.610414
std	0.691834	0.962244
min	0.000000	0.000000
25%	1.000000	1.000000
50%	1.000000	2.000000
75%	2.000000	2.000000
max	8.000000	10.000000

In [25]:

```
# converting to type 'category'
df['Car'] = df['Car'].astype('category')

# displaying frequencies of each category
df['Car'].value_counts()
```

Out[25]:

```
2.0     5606
1.0     5515
0.0     1026
3.0      748
4.0      507
5.0       63
6.0       54
8.0        9
7.0        8
10.0       3
9.0        1
Name: Car, dtype: int64
```

In [26]:

```
# imputing NaNs by 2.0
df.loc[pd.isnull(df['Car']), ['Car']] = 2
round(100*(df.isnull().sum()/len(df.index)), 2)
```

Out[26]:

```
Suburb           0.00
Address          0.00
Rooms            0.00
Type             0.00
Price            0.00
Method           0.00
SellerG          0.00
Date             0.00
Distance         0.00
Postcode         0.00
Bedroom2         0.00
Bathroom         0.01
Car              0.00
Landsize         0.00
Lattitude        0.00
Longtitude       0.00
Regionname       0.00
Propertycount    0.00
dtype: float64
```

In [27]:

```
# converting to type 'category'
df['Bathroom'] = df['Bathroom'].astype('category')

# displaying frequencies of each category
df['Bathroom'].value_counts()
```

Out[27]:

```
1.0    7517
2.0    4987
3.0     921
4.0     106
0.0      34
5.0      28
6.0       5
8.0       2
7.0       2
Name: Bathroom, dtype: int64
```

In [28]:

```
df.shape
```

Out[28]:

```
(13603, 18)
```

In [29]:

```
# fraction of rows lost
len(df.index)/23547
#Thus, we have lost about 32% observations in cleaning the missing values
```

Out[29]:

```
0.5776956724848176
```

In [33]:

```
#Find out the percentage of missing values in each column in the given dataset.
import pandas as pd
df = pd.read_csv('https://query.data.world/s/Hfu_PsEuD1Z_yJHmGaxWTxvkz7W_b0')
print(round(100*(df.isnull().sum()/len(df.index)), 2))
```

```
Ord_id                 0.00
Prod_id                0.00
Ship_id                0.00
Cust_id                0.00
Sales                  0.24
Discount               0.65
Order_Quantity         0.65
Profit                 0.65
Shipping_Cost          0.65
Product_Base_Margin    1.30
dtype: float64
```

In [36]:

```
#Remove the missing values from the rows having greater than 5 missing values and
#print the percentage of missing values in each column.

import pandas as pd
df = pd.read_csv('https://query.data.world/s/Hfu_PsEuD1Z_yJHmGaxWTxvkz7W_b0')
df = df[df.isnull().sum(axis=1) <= 5]
print(round(100*(df.isnull().sum()/len(df.index)), 2))
```

```
Ord_id                 0.00
Prod_id                0.00
Ship_id                0.00
Cust_id                0.00
Sales                  0.00
Discount               0.42
Order_Quantity         0.42
Profit                 0.42
Shipping_Cost          0.42
Product_Base_Margin    1.06
dtype: float64
```

In [39]:

```
#Impute the mean value at all the missing values of the column 'Product_Base_Marg
# and then print the percentage of missing values in each column.
import numpy as np
import pandas as pd
df = pd.read_csv('https://query.data.world/s/Hfu_PsEuD1Z_yJHmGaxWTxvkz7W_b0')
df.loc[np.isnan(df['Product_Base_Margin']), ['Product_Base_Margin']] = df['Produc
print(round(100*(df.isnull().sum()/len(df.index)), 2))
#Type your code here for mean imputation
#print(round(#Type your code here for percentage of missing values))#Round off to
```

```
Ord_id                 0.00
Prod_id                0.00
Ship_id                0.00
Cust_id                0.00
Sales                  0.24
Discount               0.65
Order_Quantity         0.65
Profit                 0.65
Shipping_Cost          0.65
Product_Base_Margin    0.00
dtype: float64
```

In [ ]: