In [182]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
import seaborn as sns
import datetime
import scipy.stats as st
```

Python Concepts for Quick reference

1. Python is a high level language.
2. It is an interpreted language which means it isn't compiled directly to machine code and importantly is commonly used in an interactive fashion
3. There is no compilation step
4. Python doesn't require the use of keywords like var to declare a variable name or semicolons at the end of lines which are commonly used in other languages. Python leverages white space to understand the scope of functions and loops and end of line markers to understand the end of statements.

Python Programming: Types and Sequences

In [99]:
```
print(type('Name-Ajay Dhariwal'))
```

```
<class 'str'>
```

In [100]:
```
type(None)
```

Out[100]:
```
NoneType
```

In [101]:
```
type(1)
```

Out[101]:
```
int
```

In [102]:
```
type(1.0)
```

Out[102]: float

In [103]:
```
tup=(1,'a',2,'b')
type(tup)
```

Out[103]: tuple

In [104]:
```
lis=[1,'a',2,'b']
type(lis)
```

Out[104]: list

In [105]:
```
dic={'Name':'Ajay_Dhariwal','Current_Status':'Student of Masters in Business analytics','University':'Univers
ity of Cincinnati'}
```

In [106]:
```
type(dic)
```

Out[106]: dict

Irterating over items in a list, tuple and a dictionary

In [107]:
```
for item in lis:
    print(item)
```

```
1
a
2
b
```

In [108]:
```
for item in tup:
    print(item)
```

```
1
a
2
b
```

The below code is to iterate over keys, values in a dictionary

In [109]:
```
for i in dic:
    print(i)
```

```
Name
Current_Status
University
```

In [110]:
```
for val in dic.values():
    print(val)
```

```
Ajay_Dhariwal
Student of Masters in Business analytics
University of Cincinnati
```

In [111]:
```
for key,val in dic.items():
    print(key)
    print(val)
```

```
Name
Ajay_Dhariwal
Current_Status
Student of Masters in Business analytics
University
University of Cincinnati
```

Unpacking items in a list,tuple with variable assigning

In [112]:
```
lis
```
Out[112]:
```
[1, 'a', 2, 'b']
```

In [113]:
```
v1,v2,v3,v4=lis
```

In [114]:
```
v1
```
Out[114]:
```
1
```

In [115]:
```
v2
```
Out[115]:
```
'a'
```

In [116]:
```
v3
```
Out[116]:
```
2
```

In [117]:
```
v4
```
Out[117]:
```
'b'
```

In [118]:
```
va1,va2,va3,va4=tup
```

In [119]:
```
va1
```
Out[119]:
```
1
```

In [120]:
```
va2
```
Out[120]:
```
'a'
```

In [121]:
```
va3
```
Out[121]:
```
2
```

In [122]:
```
va4
```

Out[122]:
```
'b'
```

Strings also act like a list where you can iterate over the values with a for loop and even do slicing as is done on the list or DataFrames. Let's take a look on formatting strings

In [123]:
```
string1='Ajay has worked in Business intelligence domain prior to joining Masters in Business Analytics at Un
iversity of Cincinnati'
string2='Ajay has cleared both CFA Level-1 and FRM Part-1 in one attempt'
```

In [124]:
```
string1+'        '+string2
# A simple addition cancatenates the strings
```

Out[124]:
```
'Ajay has worked in Business intelligence domain prior to joining Masters in Business Analytics at University
of Cincinnati        Ajay has cleared both CFA Level-1 and FRM Part-1 in one attempt'
```

In [125]:
```
string1*2
# This will double the string
```

Out[125]:
```
'Ajay has worked in Business intelligence domain prior to joining Masters in Business Analytics at University
of CincinnatiAjay has worked in Business intelligence domain prior to joining Masters in Business Analytics a
t University of Cincinnati'
```

String Unicode Transformation Format

The Python string formatting Mini Language allows you to write a string statement indicating place holders for variables to be evaluated. You then pass these variables in either named or ordered arguements and python handles the string manipulation

In [126]:
```
dic
```

Out[126]:
```
{'Name': 'Ajay_Dhariwal',
 'Current_Status': 'Student of Masters in Business analytics',
 'University': 'University of Cincinnati'}
```

Now, we will use the format method on the string

In [127]:
```
statement= ' {} has worked in Business intelligence domain and currently he is a {} at {}'
print(statement.format(dic['Name'],dic['Current_Status'],dic['University']))
```

```
 Ajay_Dhariwal has worked in Business intelligence domain and currently he is a Student of Masters in Busines
s analytics at University of Cincinnati
```

In the above line of code, we made use of format method of string to state the names from the dictionary . This can also be used in dataframes

Data Files and Summary Statistics

In [128]:
```
import csv
```

In [129]:
```
with open ('C:/Users/Ajay Dhariwal/Desktop/Kaggle/Sales Forecasting WOMart Store/train.csv') as file:
    train=list(csv.DictReader(file))
```

1. Dictreader creates an object that operates like a regular reader but maps the information in each row to a dict whose keys are given by the optional fieldnames parameter.
2. The Dictreader has read in each row of the csv file as a dictionary

In [ ]:

In [ ]:

In [130]:
```
train[0].keys()
```

Out[130]:
```
odict_keys(['ID', 'Store_id', 'Store_Type', 'Location_Type', 'Region_Code', 'Date', 'Holiday', 'Discount', '#
Order', 'Sales'])
```

We convert a list or a tuple into a Set to get the unique values in them

1. A set is a collection which is unordered and unindexed. these are written with curly brackets
2. set_1={'A','B','C'}
3. You cannot access the items in a set by referring to anm index since sets are unordered and the items have no index. But you can loop through the set items using a for loop

In [131]:
```
lis1=[1,1,2,3,4,5,5,4,2,2,5,6,3,2,1]
tup1=(1,2,3,4,2,1,1,2,3,4,2,1,3,4,5)
# To get the unique elements in the list, convert the list into a set
unique_elements=set(lis1)
unique_elements_1=set(tup1)
```

In [132]:
```
unique_elements
```

Out[132]:
```
{1, 2, 3, 4, 5, 6}
```

In [133]:
```
unique_elements_1
```

Out[133]:
```
{1, 2, 3, 4, 5}
```

Dates and Times in Python

In [134]:
```
import datetime as dt
import time as tm
```

In [135]:
```
tm.time()
# This gives the seconds from January 1, 1970 til now as a timestamp
```

Out[135]:
```
1633077213.6096325
```

In [136]:
```
dtnow=dt.datetime.fromtimestamp(tm.time())
# we convert the timestamp into datetime using the fromtimestamp class available in the datetime module
```

In [137]:
```
dtnow.isoformat()
# We conver the dtnow into standard iso Format
```

Out[137]:
```
'2021-10-01T04:33:34.000365'
```

In [138]:
```
dtnow.year,dtnow.month,dtnow.day,dtnow.hour,dtnow.minute
# The datetime module can be used to get the current date, year, month, minute etc
```

Out[138]:
```
(2021, 10, 1, 4, 33)
```

In [ ]:
```

```

In [139]:
```
# We can use the timedelta class available in datetime to create a timew delta
# The datetime that we get can be assumed to be as numbers on straight lines where we can do simple calculati
ons such
# as addition and subtraction. The delta can be used to go t days ahead etc as shown below
```

In [140]:
```
delta=dt.timedelta(days=2)
```

In [141]:
```
dt_2daysfromnow=dtnow+delta
```

In [142]:
```
dtnow.day
```

Out[142]:
```
1
```

In [143]:
```
dt_2daysfromnow.day
```

Out[143]:
```
3
```

Advanced Python Objects ,map()

1. Classes in python are generally named using Camel Case which means that First Character of each word is capitalized
2. You don't declare variables within an object. You just start using them
3. All classes have a function called *init* which is always executed when the class is being initiated
4. OBJECT METHODS: Objects can also contain methods. Methods in objects are functions that belong to the object
5. Self Parameter is a reference to the current instance of the class and is used access variables thath belong to the class. It does not have to be named self, you can call it whatever you like but it has to be the first parametet of any function in class.

In [144]:
```
class Person:
    department='Linder School of Business'
    #Department is the class variable here
    def set_name(self,new_name):
        self.name=new_name
#To define a method, you just write it as you would have a function.
##The one change, is that to have access to the instance which a method is being invoked upon
###you must include self, in the method signature.
    def set_location(self,new_location):
        self.location=new_location
#objects in Python do not have private or protected members.
#If you instantiate an object, you have full access to any of the methods or attributes of that object
```

In [145]:
```
person=Person()
person.set_name('Ajay')
person.set_location('Cincinnati,Ohio')
```

In [146]:
```
print('{} lives in {} and studies in {}'.format(person.name,person.location,person.department))
```

```
Ajay lives in Cincinnati,Ohio and studies in Linder School of Business
```

Map function :

The map function return a map object, This is an example of Lazy Evaluation.

In programming language theory, lazy evaluation, or call-by-need, is an evaluation strategy which delays the evaluation of an expression until its value is needed (non-strict evaluation) and which also avoids repeated evaluations (sharing).

The map function signature looks like this. The first parameters of function that you want execute, and the second parameter, and every following parameter, is something which can be iterated upon. All the iterable arguments are unpacked together, and passed into the given function

In [147]:
```
people = ['Dr. Christopher Brooks', 'Dr. Kevyn Collins-Thompson', 'Dr. VG Vinod Vydiswaran', 'Dr. Daniel Rome
ro']

def split_title_and_name(person):
    title = person.split()[0]
    lastname = person.split()[-1]
    return '{} {}'.format(title, lastname)

list(map(split_title_and_name, people))
```

Out[147]:
```
['Dr. Brooks', 'Dr. Collins-Thompson', 'Dr. Vydiswaran', 'Dr. Romero']
```

In [148]:
```
l1=[1,2,3,4,8]
l2=[6,7,9,1,10]
# we want to use the map function to get the minmum value from the two lists above
```

In [149]:
```
a=map(min,l1,l2)
```

In [150]:
```
import numpy as np
```

In [151]:
```
for i in a:
    print(i)
# The values are compared 1 vs 6,2 vs 7 etc
```

```
1
2
3
1
8
```

Python lambda and list comprehensions

lambda input variables: output variable

In [152]:
```
addition=lambda x,y:x+y
```

In [153]:
```
addition(2,1)
```

Out[153]:
```
3
```

In [154]:
```
print(type(addition))
```

```
<class 'function'>
```

List Comprehension

In [155]:
```
my_list=[]
for num in range(1000):
    if num%3==0:
        my_list.append(num)
```

In [156]:
```
my_list
```

Out[156]:
```
[0,
 3,
 6,
 9,
 12,
 15,
 18,
 21,
 24,
 27,
 30,
 33,
 36,
 39,
 42,
 45,
 48,
 51,
 54,
 57,
 60,
 63,
 66,
 69,
 72,
 75,
 78,
 81,
 84,
 87,
 90,
 93,
 96,
 99,
 102,
 105,
 108,
 111,
 114,
 117,
 120,
 123,
 126,
```

```
129,
132,
135,
138,
141,
144,
147,
150,
153,
156,
159,
162,
165,
168,
171,
174,
177,
180,
183,
186,
189,
192,
195,
198,
201,
204,
207,
210,
213,
216,
219,
222,
225,
228,
231,
234,
237,
240,
243,
246,
249,
252,
255,
```

```
258,
261,
264,
267,
270,
273,
276,
279,
282,
285,
288,
291,
294,
297,
300,
303,
306,
309,
312,
315,
318,
321,
324,
327,
330,
333,
336,
339,
342,
345,
348,
351,
354,
357,
360,
363,
366,
369,
372,
375,
378,
381,
384,
```

```
387,
390,
393,
396,
399,
402,
405,
408,
411,
414,
417,
420,
423,
426,
429,
432,
435,
438,
441,
444,
447,
450,
453,
456,
459,
462,
465,
468,
471,
474,
477,
480,
483,
486,
489,
492,
495,
498,
501,
504,
507,
510,
513,
```

```
516,
519,
522,
525,
528,
531,
534,
537,
540,
543,
546,
549,
552,
555,
558,
561,
564,
567,
570,
573,
576,
579,
582,
585,
588,
591,
594,
597,
600,
603,
606,
609,
612,
615,
618,
621,
624,
627,
630,
633,
636,
639,
642,
```

```
645,
648,
651,
654,
657,
660,
663,
666,
669,
672,
675,
678,
681,
684,
687,
690,
693,
696,
699,
702,
705,
708,
711,
714,
717,
720,
723,
726,
729,
732,
735,
738,
741,
744,
747,
750,
753,
756,
759,
762,
765,
768,
771,
```

```
774,
777,
780,
783,
786,
789,
792,
795,
798,
801,
804,
807,
810,
813,
816,
819,
822,
825,
828,
831,
834,
837,
840,
843,
846,
849,
852,
855,
858,
861,
864,
867,
870,
873,
876,
879,
882,
885,
888,
891,
894,
897,
900,
```

```
903,
906,
909,
912,
915,
918,
921,
924,
927,
930,
933,
936,
939,
942,
945,
948,
951,
954,
957,
960,
963,
966,
969,
972,
975,
978,
981,
984,
987,
990,
993,
996,
999]
```

The above given code can be written in a single line using the List comprehension

In [157]:
```
my_list_1=[num for num in range(1000) if num%3==0]
my_list_1
```

Out[157]:
```
[0,
 3,
 6,
 9,
 12,
 15,
 18,
 21,
 24,
 27,
 30,
 33,
 36,
 39,
 42,
 45,
 48,
 51,
 54,
 57,
 60,
 63,
 66,
 69,
 72,
 75,
 78,
 81,
 84,
 87,
 90,
 93,
 96,
 99,
 102,
 105,
 108,
 111,
 114,
 117,
 120,
 123,
 126,
```

```
129,
132,
135,
138,
141,
144,
147,
150,
153,
156,
159,
162,
165,
168,
171,
174,
177,
180,
183,
186,
189,
192,
195,
198,
201,
204,
207,
210,
213,
216,
219,
222,
225,
228,
231,
234,
237,
240,
243,
246,
249,
252,
255,
```

```
258,
261,
264,
267,
270,
273,
276,
279,
282,
285,
288,
291,
294,
297,
300,
303,
306,
309,
312,
315,
318,
321,
324,
327,
330,
333,
336,
339,
342,
345,
348,
351,
354,
357,
360,
363,
366,
369,
372,
375,
378,
381,
384,
```

```
387,
390,
393,
396,
399,
402,
405,
408,
411,
414,
417,
420,
423,
426,
429,
432,
435,
438,
441,
444,
447,
450,
453,
456,
459,
462,
465,
468,
471,
474,
477,
480,
483,
486,
489,
492,
495,
498,
501,
504,
507,
510,
513,
```

```
516,
519,
522,
525,
528,
531,
534,
537,
540,
543,
546,
549,
552,
555,
558,
561,
564,
567,
570,
573,
576,
579,
582,
585,
588,
591,
594,
597,
600,
603,
606,
609,
612,
615,
618,
621,
624,
627,
630,
633,
636,
639,
642,
```

```
645,
648,
651,
654,
657,
660,
663,
666,
669,
672,
675,
678,
681,
684,
687,
690,
693,
696,
699,
702,
705,
708,
711,
714,
717,
720,
723,
726,
729,
732,
735,
738,
741,
744,
747,
750,
753,
756,
759,
762,
765,
768,
771,
```

```
774,
777,
780,
783,
786,
789,
792,
795,
798,
801,
804,
807,
810,
813,
816,
819,
822,
825,
828,
831,
834,
837,
840,
843,
846,
849,
852,
855,
858,
861,
864,
867,
870,
873,
876,
879,
882,
885,
888,
891,
894,
897,
900,
```

```
903,
906,
909,
912,
915,
918,
921,
924,
927,
930,
933,
936,
939,
942,
945,
948,
951,
954,
957,
960,
963,
966,
969,
972,
975,
978,
981,
984,
987,
990,
993,
996,
999]
```

The numerical Python library

Numpy

In [158]:
```
import numpy as np
```

In [159]:
```
my_list_2=[1,2,3,4,5,6,7,8,9,10]
```

In [160]:
```
my_first_array=np.array(my_list_2)
```

In [161]:
```
my_first_array
```

Out[161]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [162]:
```
print(my_first_array.shape)
```

```
(10,)
```

In [163]:
```
x=np.array([2,3,4])
x
```

Out[163]:
```
array([2, 3, 4])
```

In [164]:
```
y=np.array([[2,3,4],[5,6,7]])
y
```

Out[164]:
```
array([[2, 3, 4],
       [5, 6, 7]])
```

In [165]:
```
y.shape
```

Out[165]:
```
(2, 3)
```

In [166]:
```
x.shape
```

Out[166]:
```
(3,)
```

In [167]:
```
z=np.arange(0,30,2)
z
```

Out[167]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28])
```

In [168]:
```
z.reshape(3,5)
```

Out[168]:
```
array([[ 0,  2,  4,  6,  8],
       [10, 12, 14, 16, 18],
       [20, 22, 24, 26, 28]])
```

In [169]:
```
z.reshape(5,3)
```

Out[169]:
```
array([[ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16],
       [18, 20, 22],
       [24, 26, 28]])
```

In [170]:
```
np.ones((2,3))
# an array of shape m,n with all ones
```

Out[170]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

In [171]:
```
np.zeros((2,3))
# an array with all zeros with shape m,n
```

Out[171]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [172]:
```
np.eye(3)
# an array with ones at dioganal with shape m,n
```

Out[172]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [173]:
```
y1=np.array([1,2,3])
np.diag(y1)
# a diaogonal matrix with the elements of the array inputed(y1) on the diagonal and remaining all elements ar
e zero
```

Out[173]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

In [174]:
```
np.array([1,2,3]*3)
# the elements 1,2,3 come m times. m here is 3
```

Out[174]:
```
array([1, 2, 3, 1, 2, 3, 1, 2, 3])
```

In [175]:
```
p=np.ones((2,3),int)
```

In [176]:
```
np.vstack([p,2*p])
# the vertical stacking of the ekements of the array is done.
#The elements to be vertically stacked are passed as a list to the vstack
```

Out[176]:
```
array([[1, 1, 1],
       [1, 1, 1],
       [2, 2, 2],
       [2, 2, 2]])
```

In [177]:
```
np.hstack([p,2*p])
```

Out[177]:
```
array([[1, 1, 1, 2, 2, 2],
       [1, 1, 1, 2, 2, 2]])
```

To get the Transpose

In [178]:
```
zz=np.array([[1,2,3],[4,5,6]])
```

In [179]:
```
zz
```

Out[179]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [180]:
```
zz.T
#array.T gives the transpose of the array
```

Out[180]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [181]:
```
zz.dtype
# to get the datatype
```

Out[181]:
```
dtype('int32')
```

In [182]:
```
zz.astype('f')
# to cast the elements of the array as float
```

Out[182]:
```
array([[1., 2., 3.],
       [4., 5., 6.]], dtype=float32)
```

In [183]:
```
zz.min()
```

Out[183]:
```
1
```

In [184]:
```
zz.max()
```

Out[184]:
```
6
```

In [185]:
```
zz.sum()
```

Out[185]:
```
21
```

In [186]:
```
zz.mean()
```

Out[186]:
```
3.5
```

In [187]:
```
zz.std()
```

Out[187]:
```
1.707825127659933
```

To get the index of the maximum or minimum values in the array

In [188]:
```
zz.argmax()
```

Out[188]:
```
5
```

In [189]:
```
zz.argmin()
```

Out[189]:
```
0
```

In [190]:
```
aa=np.array([[1,2,3],[4,5,6],[7,8,9]])
aa
```

Out[190]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [191]:
```
for i in aa:
    print(i)
```

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

Series: A data structure in python. The items are stored in a order and there are labels that are attached with which we can retrieve them

In [192]:
```
sports={'A':'a','B':'b','C':'c'}
```

In [194]:
```
import pandas as pd
```

In [195]:
```
pd.Series(sports)
# A pandas series can be queued either by index position or the index label
```

Out[195]:
```
A    a
B    b
C    c
dtype: object
```

To querry by numeric location attribute,use the iloc attribute

To query by the index label, use the loc attribute

In [196]:
```
#Broadcasting
#With Broadcasting,you can apply an operation to every value in the series thereby changing the series
```

In [197]:
```
lis_4=[1,2,3,4,5,56]
ser1=pd.Series(lis_4)
```

In [198]:
```
ser1
```

Out[198]:
```
0     1
1     2
2     3
3     4
4     5
5    56
dtype: int64
```

In [199]:
```
ser1+=2
#Here broadcasting has been done. every element of the series has been incremented by 2 in a single line of c
ode
```

In [200]:
```
ser1
```

Out[200]:
```
0     3
1     4
2     5
3     6
4     7
5    58
dtype: int64
```

iteritems()

Iterates over the DataFrame columns, returning a tuple with the column name and the content as a Series.

In [201]:
```
df_1 = pd.DataFrame({'species': ['bear', 'bear', 'marsupial'],
                     'population': [1864, 22000, 80000]},
                     index=['panda', 'polar', 'koala'])
```

In [202]:
```
df_1
```

Out[202]:

	species	population
panda	bear	1864
polar	bear	22000
koala	marsupial	80000

In [203]:
```
for label,content in df_1.iteritems():
    print(f'label: {label}')
    print(f'content: {content}', sep='\n')
```

```
label: species
content: panda         bear
polar         bear
koala    marsupial
Name: species, dtype: object
label: population
content: panda     1864
polar    22000
koala    80000
Name: population, dtype: int64
```

With pandas Module, we can convert the csv file imported into a DataFrame

In [204]:
```
import pandas as pd
df=pd.read_csv('C:/Users/Ajay Dhariwal/Desktop/Kaggle/Sales Forecasting WOMart Store/train.csv')
df.set_index('Store_id',inplace=True)
```

In [205]:
```
df.head()
```

Out[205]:

	ID	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales
Store_id									
1	T1000001	S1	L3	R1	2018-01-01	1	Yes	9	7011.84
253	T1000002	S4	L2	R1	2018-01-01	1	Yes	60	51789.12
252	T1000003	S3	L2	R1	2018-01-01	1	Yes	42	36868.20
251	T1000004	S2	L3	R1	2018-01-01	1	Yes	23	19715.16
250	T1000005	S2	L3	R4	2018-01-01	1	Yes	62	45614.52

In [206]:
```
purchase_1 = pd.Series({'Name': 'Chris',
                        'Item Purchased': 'Dog Food',
                        'Cost': 22.50})
purchase_2 = pd.Series({'Name': 'Kevyn',
                        'Item Purchased': 'Kitty Litter',
                        'Cost': 2.50})
purchase_3 = pd.Series({'Name': 'Vinod',
                        'Item Purchased': 'Bird Seed',
                        'Cost': 5.00})

df_2 = pd.DataFrame([purchase_1, purchase_2, purchase_3], index=['Store 1', 'Store 1', 'Store 2'])
```

In [207]:
```
df_2.T
```

Out[207]:

	Store 1	Store 1	Store 2
Name	Chris	Kevyn	Vinod
Item Purchased	Dog Food	Kitty Litter	Bird Seed
Cost	22.5	2.5	5

In [208]:
```
df_2.T.loc['Item Purchased']
```

Out[208]:
```
Store 1        Dog Food
Store 1    Kitty Litter
Store 2       Bird Seed
Name: Item Purchased, dtype: object
```

In [209]:
```
df_2.T.loc['Name']
```

Out[209]:
```
Store 1    Chris
Store 1    Kevyn
Store 2    Vinod
Name: Name, dtype: object
```

In [210]:
```
df_2.T.loc['Name']['Store 1']
```

Out[210]:
```
Store 1    Chris
Store 1    Kevyn
Name: Name, dtype: object
```

In [211]:
```
# To delete a column
del df_2['Name']
df_2
```

Out[211]:

	Item Purchased	Cost
Store 1	Dog Food	22.5
Store 1	Kitty Litter	2.5
Store 2	Bird Seed	5.0

In [212]:
```
df_2
```

Out[212]:

	Item Purchased	Cost
Store 1	Dog Food	22.5
Store 1	Kitty Litter	2.5
Store 2	Bird Seed	5.0

IDXMAX

This function returns the index of the maximum value. By default, it returns the index of the first maximum value along the column

In [221]:
```
df_4 = pd.DataFrame({'consumption': [10.51, 103.11, 55.48],
                     'co2_emissions': [37.25, 19.66, 1712]},
                     index=['Pork', 'Wheat Products', 'Beef'])
```

In [230]:
```
df_5 = pd.DataFrame({'time': [100, 103, 55],
                     'co2_emissions': [37.2,np.nan,  1712]},
                     index=['Pork', 'Wheat Products', 'Beef'])
```

Joins

In [231]:
```
df_4
```

Out[231]:

	consumption	co2_emissions
Pork	10.51	37.25
Wheat Products	103.11	19.66
Beef	55.48	1712.00

In [232]:
```
df_5
```

Out[232]:

	time	co2_emissions
Pork	100	37.2
Wheat Products	103	NaN
Beef	55	1712.0

In [235]:
```
df_6=pd.merge(df_4,df_5,how='inner',left_index=True,right_index=True)
```

In [237]:
```
df_6
```

Out[237]:

	consumption	co2_emissions_x	time	co2_emissions_y
Pork	10.51	37.25	100	37.2
Wheat Products	103.11	19.66	103	NaN
Beef	55.48	1712.00	55	1712.0

Groupby Object

This function takes some column name or names and splits the dataframe up into chunks based on those names. It returns a dataframe group by object. Which can be iterated upon, and then returns a tuple where the first item is the group condition,and the second item is the data frame reduced by that grouping. Since it's made up of two values, you can unpack this, and project just the column that you're interested in .

In [5]:
```
df_new=pd.read_csv('C:/Users/Ajay Dhariwal/Desktop/Kaggle/NBA Referee Stats.csv')
```

In [6]:
```
df_new.head(2)
```

Out[6]:

	Season	Season type	Referee	Games	Total fouls	Shooting	Personal	Loose ball	Personal take	Offensive charge	...	Delay technical	Lane violation	Jump ball violation	Flagrai
0	2016-17	Playoffs	Bennie Adams	3	48	18	20	0	1	0	...	0	0	0	
1	2016-17	Playoffs	Bill Kennedy	8	101	42	40	6	1	2	...	0	0	0	

2 rows × 34 columns

In [264]:
```
for name,df__ in df_new.groupby('Season type'):
    total=np.sum(df__['Games'])
    print('Season_type ' + name +' '+ 'had ' + str(total) + ' Games')
```

```
Season_type Playoffs had 979 Games
Season_type Regular Season had 14227 Games
```

In [308]:
```
for i,j in df_new.groupby(['Season','Season type']):
    total=np.sum(j['Games'])
    print(i,total)
```

```
('2016-17', 'Playoffs') 237
('2016-17', 'Regular Season') 3689
('2017-18', 'Playoffs') 247
('2017-18', 'Regular Season') 3673
('2018-19', 'Playoffs') 246
('2018-19', 'Regular Season') 3689
('2019-20', 'Playoffs') 249
('2019-20', 'Regular Season') 3176
```

Scales

In [14]:
```
df___ = pd.DataFrame(['A+', 'B', 'C', 'D'],
                     index=['excellent',  'good', 'ok',  'poor'])
df___.rename(columns={0: 'Grades'}, inplace=True)
df___
```

Out[14]:

	Grades
excellent	A+
good	B
ok	C
poor	D

There is a method called astype which can be used to convert the datatype of the column in the dataframe

In [15]:
```
df___['Grades'].astype('category')
# There is no order. Only the category has been created. We can convert into ordered with a proper order also
```

Out[15]:
```
excellent    A+
good          B
ok            C
poor          D
Name: Grades, dtype: category
Categories (4, object): [A+, B, C, D]
```

In [19]:
```
grades=df___['Grades'].astype('category',categories=['D','C','B','A+'],ordered=True)
# The order to be passed must be in ascending form i.e. The one that you think is the worst here  D should co
me first in the list
```

```
C:\Users\Ajay Dhariwal\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: specifying 'catego
ries' or 'ordered' in .astype() is deprecated; pass a CategoricalDtype instead
  """Entry point for launching an IPython kernel.
```

In [21]:
```
grades>'C'
# since the grades has been made a category, it can be used to do boolean masking and all other related stuf
f.
```

Out[21]:
```
excellent     True
good          True
ok           False
poor         False
Name: Grades, dtype: bool
```

Agg method

In [22]:
```
dfff = pd.DataFrame([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9],
                    [np.nan, np.nan, np.nan]],
                   columns=['A', 'B', 'C'])
```

In [23]:
```
dfff
```

Out[23]:

	A	B	C
0	1.0	2.0	3.0
1	4.0	5.0	6.0
2	7.0	8.0	9.0
3	NaN	NaN	NaN

In [27]:
```
dfff.agg('sum',axis=0)
# This applies the function sum to the columns. That is for each coluumn we get the funcition applied
```

Out[27]:
```
A    12.0
B    15.0
C    18.0
dtype: float64
```

In [28]:
```
dfff.agg('sum')
# This is same as above
```

Out[28]:
```
A    12.0
B    15.0
C    18.0
dtype: float64
```

In [30]:
```
dfff.agg('sum',axis=1)
# this applies the function sum to each of the rows. i.e. We get the sum of values along each row
```

Out[30]:
```
0     6.0
1    15.0
2    24.0
3     0.0
dtype: float64
```

In [35]:
```
dfff.agg(['sum','min','mean'])
# When we pass a list to the agg method, it applies the list of the functions to each of the column. i.e. we
 get the
# function applied to each column value
```

Out[35]:

	A	B	C
sum	12.0	15.0	18.0
min	1.0	2.0	3.0
mean	4.0	5.0	6.0

Pandas cut function

Use cut when you need to segment and sort data values into bins. This function is also useful for going from a continuous variable to a categorical variable. For example, cut could convert ages to groups of age ranges. Supports binning into an equal number of bins, or a pre-specified array of bins.

Pivot Table

A pivot table is a way of summarizing data in a data frame for a particular purpose. It makes heavy use of the aggregation function. A pivot table is itself a data frame, where the rows represent one variable that you're interested in, the columns another, and the cell's some aggregate value. A pivot table also tends to includes marginal values as well, which are the sums for each column and row. This allows you to be able to see the relationship between two variables at just a glance

In [43]:
```
data=pd.read_csv(r'C:/Users/Ajay Dhariwal/Desktop/Semester 1/stat methos/assignment.csv',sep=';')
data.head()
```

Out[43]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

In [55]:
```
pivot_data=data.pivot_table(index='alcohol',columns='quality',values='residual sugar',aggfunc=np.mean)
pivot_data.iloc[:7,:]
```

Out[55]:

quality	3	4	5	6	7	8
alcohol						
8.40	2.1	NaN	NaN	1.800000	NaN	NaN
8.50	NaN	NaN	1.600000	NaN	NaN	NaN
8.70	NaN	NaN	NaN	2.100000	NaN	NaN
8.80	NaN	NaN	13.800000	NaN	NaN	NaN
9.00	2.2	3.00	2.090909	3.787500	NaN	NaN
9.05	NaN	1.90	NaN	NaN	NaN	NaN
9.10	NaN	1.65	2.464286	1.928571	NaN	NaN

In [61]:
```
pivot=data.pivot_table(index='alcohol',values='residual sugar',aggfunc=[np.mean,np.min],margins=True)
# this applies the funcitons passed as a list to all the values in the pivot table as shown below
```

In [62]:
```
pivot.iloc[:6,:]
```

Out[62]:

	mean	amin
	residual sugar	residual sugar
alcohol		
8.40	1.95	1.8
8.50	1.60	1.6
8.70	2.10	1.6
8.80	13.80	13.8
9.00	3.06	1.6
9.05	1.90	1.9

Binomial Distribution

Sampling the distribution

We pass the number of trials as n, the probability of success as p and the third arguement as the number of times we want the experiment to run. That is, here the probability of success is p and one event is tossing the coin n times. We do this event 1000 times. The np.random.binomial returns an array of size n in which each number is sum of the successes in every event(10 trials here)

In [94]:
```
n,p=10,0.5
sample=np.random.binomial(n,p,3)
sample
```

Out[94]:
```
array([5, 8, 3])
```

Here, we got an array of size 3 which is the number of times the experiment will run. In every trial of the experiment, we will toss the coin 10 times. out of those 10 times, we get sometimes success which is 1 for python and sometimes we fail, which is 0 for python. The number 8 in the array represents that in trial number 2 of the experiment, we got 8 successes and 2 failures that is why 8.

A company drills 9 wild-cat oil exploration wells, each with an estimated probability of success of 0.1. All nine wells fail. What is the probability of that happening?

In [98]:
```
n_c,p_s=9,0.1
sampling_array=np.random.binomial(n,p,1000)
expected_value=(sum(sampling_array)/1000)
expected_value
```

Out[98]:
```
0.905
```

Uniform Distribution

In [104]:
```
np.random.uniform(0,3,10)
# This code draws random samples from a uniform distribution with low as 0 ,high as 3 and 10 samples are draw
n here
```

Out[104]:
```
array([2.63536113, 0.83997364, 1.79953908, 0.90022203, 0.75524784,
       1.83604179, 0.44670402, 0.51836399, 0.46712908, 2.18251463])
```

In [108]:
```
np.random.randint(0,3,10)
# This draws radnom samples between 0 and 3, all integars and 10 samples are randomly drawn
```

Out[108]:
```
array([1, 0, 0, 0, 1, 1, 1, 1, 0, 2])
```

In [145]:
```
mu, sigma = 0, 0.1 # mean and standard deviation
s = np.random.normal(mu, sigma, 1000)
np.random.seed(123)
# an array is returned with 1000 samples drawns randomly from the normal distribution.
```

In [156]:
```
m,s=0,1
ss=np.random.normal(m,s,1000)
np.random.seed(122)
np.std(ss)
```

Out[156]:
```
1.0233392718534031
```

In [140]:
```
import scipy.stats as st
st.kurtosis(s)
```

Out[140]:
```
-0.16302347619128676
```

In [144]:
```
st.skew(s)
```

Out[144]:
```
0.03097623160394668
```

Hypothesis Testing

Hypothesis: A statement we can test Alternative Hypothesis: Our idea. The statement that researcher wants to establish. Null Hypothesis: The idea researches wants to reject.

We can never be 100% sure about our guess about something. But we can have some confidence about our statment.

Critical Value: Alpha >> The threshold as to how much you are willing to accept. Type 1 error: Rejecting the null when it is falsw Type 2 error: Failing to reject the null when it is False

We use the scipy library to do the hypothesis testing

Question on Hypothesis Testing

When your company's gizmo machine is working properly, the mean length of gizmos is 2.5 inches. However, from time to time the machine gets out of alignment and produces gizmos that are either too long or too short. When this happens, production is stopped and the machine is adjusted. To check the machine, the quality control department takes a gizmo sample each day. Today, a random sample of 49 gizmos showed a mean length of 2.49 inches. The population standard deviation is known to be 0.021 inches. Using a 5% significance level, determine if the machine should be shut down and adjusted.

In [170]:
```
mean_length_gizmo=2.5
sample_size=49
sample_mean1=2.49
pop_std=0.021
significance_level=0.05
confidence_interval=1-significance_level
```

In [174]:
```
#Null Hypothesis: Mean_length of gizmo=2.5. that is machine does not need any adjustment
#ALternate Hypothesis: Mean_length of gizmo <>2.5. That is machine needs an adjustment
#We will test at 5% significance level. Since this a test of equality, we do a 2-tailed test of hypothesis.
#Since population variance is known and sample_size>30, we use the Z distribution for hypothesis testing
```

Method-1

For 95% level, the critical values are -1.96 sigma and +1.96 sigma. We find the Z statistic using the formula x-mu/stderr

In [167]:
```
stder=pop_std/np.sqrt(sample_size)
stder
```

Out[167]:
```
0.003
```

In [171]:
```
z=(sample_mean1-mean_length_gizmo)/stder
```

In [172]:
```
z
```

Out[172]:
```
-3.3333333333332624
```

Since the Z Value is greater than the citical value of -1.96, thus we can reject the null hypothesis and We accept the alternate hypothesis.

Thus the machine needs an adjustment

Method- 2

Finding the confidence interval. We find the confidence interval and if the sample lies in the confidence interval, we accept the null else we fail to reject the null hypothesis

In [179]:
```
import scipy.stats as st
lower_range,upper_range=st.norm.interval(0.95,loc=mean_length_gizmo,scale=stder,)
```

In [180]:
```
def check_hypothesis(value):
    if value >=lower_range and value<=upper_range:
        print('The {} value lies in the 95% confidence interval'.format(value))
    else:
        print('The {} value does not lies in the 95% confidence interval'.format(value))
```

In [181]:
```
check_hypothesis(sample_mean)
```

```
The 2.49 value does not lies in the 95% confidence interval
```

Since the value does not lies in the confidence interval, hence we reject the null hypothesis and we fail to reject the Alternate Hypothesis. Thus we need the machine to be stopped for repair

Zip function

This function is used to zip two list items together and this gives a zip pbject

In [223]:
```
list_1=[2,4,6,8,10]
list_2=[1,3,5,7,9]
zipped=zip(list_1,list_2)
```

In [224]:
```
zipped
```

Out[224]:
```
<zip at 0x22a2cc7ff48>
```