# NUMPY PROGRAM

In [1]:
```
import numpy as np
a = [1,2,3,4]
sum = np.sum(a)
print("Sum of array elements is",sum)
min = np.min(a)
print("Minimum of array elements is",min)
max = np.max(a)
print("Maximum of array elements is",max)
mean = np.mean(a)
print("Mean of array elements is",mean)
med = np.median(a)
print("Median of array elements is",med)
cor = np.corrcoef(a)
print("Correaltion coefficiant of array elements is",cor)
std = np.std(a)
print("standard deviation of array elements is",std)
```

```
Sum of array elements is 10
Minimum of array elements is 1
Maximum of array elements is 4
Mean of array elements is 2.5
Median of array elements is 2.5
Correaltion coefficiant of array elements is 1.0
standard deviation of array elements is 1.118033988749895
```

In [2]:
```
mat = [[1,2,3,4],[3,5,6,8],[8,9,3,4]]
sum = np.sum(mat)
print("Sum of matrix elements is",sum)
min = np.min(mat)
print("Minimum of the matrix:",min)
max = np.max(mat)
print("Maximum of the matrix:",max)
mean = np.mean(mat)
print("Mean of the matrix:",mean)
med = np.median(a)
print("Median of the matrix:",med)
cor = np.corrcoef(mat)
print("Correaltion coefficiants of the matrix elements:\n",cor)
std = np.std(mat)
print("standard deviation of array elements is",std)
```

```
Sum of matrix elements is 56
Minimum of the matrix: 1
Maximum of the matrix: 9
Mean of the matrix: 4.666666666666667
Median of the matrix: 2.5
Correaltion coefficiants of the matrix elements:
 [[ 1.          0.99227788 -0.78935222]
 [ 0.99227788  1.         -0.70710678]
 [-0.78935222 -0.70710678  1.        ]]
standard deviation of array elements is 2.4608038433722337
```

In [3]:
```
mat = np.array([[1,7,3,4],[3,5,6,8],[10,9,3,4]])
print(mat)
sum = np.sum(mat, axis=1)
print("Sum of array elements row-wise",sum)
sum = np.sum(mat, axis=0)
print("Sum of array elements column-wise",sum)
min = np.min(mat, axis=1)
```

```
print("Row-wise minimum of the matrix:",min)
max = np.max(mat,axis=0)
print("Column-wise maximum of the matrix:",max)
```

```
[[ 1  7  3  4]
 [ 3  5  6  8]
 [10  9  3  4]]
Sum of array elements row-wise [15 22 26]
Sum of array elements column-wise [14 21 12 16]
Row-wise minimum of the matrix: [1 3 3]
Column-wise maximum of the matrix: [10  9  6  8]
```

# lambda

In [4]:
```
def sum(a,b):
    return a+b

sum(4,5)
```

Out[4]:
```
9
```

In [5]:
```
def cube(x):
    return x*x*x
cube(4)
```

Out[5]:
```
64
```

In [6]:
```
lambda_cube = lambda y: y*y*y
lambda_cube(5)
```

Out[6]:
```
125
```

In [7]:
```
sum = lambda a,b:a+b
sum(4,5)
```

Out[7]:
```
9
```

In [8]:
```
add = lambda num: num + 4
print( add(6) )
```

```
10
```

In [9]:
```
def greater(a,b):
    if a>b:
        return a
    else:
        return b

greater(4,5)
```

Out[9]:
```
5
```

In [10]:
```
Max = lambda a, b : a if(a > b) else b

Max(4,5)
```

Out[10]:
```
5
```

In [11]:
```
my_list= [5,7,2,8,6]
my_list_squared = []
for i in my_list:
    i_squared = i**2
    my_list_squared.append(i_squared)

my_list_squared
```

Out[11]:
```
[25, 49, 4, 64, 36]
```

In [12]:
```
my_list_squared = [i**2 for i in my_list]
my_list_squared
```

Out[12]:
```
[25, 49, 4, 64, 36]
```

In [13]:
```
my_list_squared = list(map(lambda i: i**2, my_list))
my_list_squared
```

Out[13]:
```
[25, 49, 4, 64, 36]
```

# Map

In [14]:
```
def add4(x):
    return x+4
list1 = [4,6,7,8,9]

list2 = list(map(add4,list1))
list2
```

Out[14]:
```
[8, 10, 11, 12, 13]
```

In [15]:
```
list3 = list(map(lambda x:x+4,list1))
list3
```

Out[15]:
```
[8, 10, 11, 12, 13]
```

In [16]:
```
set_of_strings = ['abc','def','xyz']
string_map_22 = list(map(lambda my_string: my_string + '_2022', set_of_strings))
string_map_22
```

Out[16]:
```
['abc_2022', 'def_2022', 'xyz_2022']
```

# Filter()

In [17]:
```
def oddeven(x):
    if x%2 == 0:
        return True
    else:
        return False

list1 = [4,5,6,7,8,9]
evenlist = list(filter(oddeven,list1))
evenlist
```

Out[17]:
```
[4, 6, 8]
```

In [18]:
```
list1 = [4,5,6,7,8,9]
evenlist = list(filter(lambda x:True if x%2==0 else False,list1))
evenlist
```
Out[18]:
```
[4, 6, 8]
```

# Reduce

In [19]:
```
from functools import reduce
def sum(x,y):
    return x+y

list1 = [6,7,8,9]
s = reduce(sum,list1)
s
```
Out[19]:
```
30
```

# Write a NumPy program to create a 3x3 matrix with values ranging from 2 to 10

In [24]:
```
x =  np.arange(2, 11).reshape(3,3)
print(x)
```
```
[[ 2  3  4]
 [ 5  6  7]
 [ 8  9 10]]
```

# Write a NumPy program to create a null vector of size 10 and update sixth value to 11

In [25]:
```
x = np.zeros(10)
print(x)
print("Update sixth value to 11")
x[6] = 11
print(x)
```
```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Update sixth value to 11
[ 0.  0.  0.  0.  0.  0. 11.  0.  0.  0.]
```

In [7]:
```
item_list = ['Bread', 'Milk', 'Eggs', 'Butter', 'Cocoa']
student_marks = [78, 47, 96, 55, 34]
hetero_list =  [ 1,2,3.0, 'text', True, 3+2j ]
```

In [8]:
```
student_marks = [78, 47, 96, 55, 34]
for i in range(len(student_marks)):
    student_marks[i]+=5
print(student_marks)
```
```
[83, 52, 101, 60, 39]
```

In [9]:
```
%%time
#Used to calculate total operation time
list1 = list(range(1,1000000))
list2 = list(range(2,1000001))
list3 = []
for i in range(len(list1)):
    list3.append(list1[i]+list2[i])
```

```
Wall time: 533 ms
```

In [10]:
```
import numpy as np
student_marks_arr = np.array([78, 92, 36, 64, 89])
student_marks_arr
```

Out[10]:
```
array([78, 92, 36, 64, 89])
```

In [11]:
```
car_attributes = [[18, 15, 18, 16, 17],[130, 165, 150, 150, 140],[307, 350, 318, 3(

#creating a numpy array from car_attributes list
car_attributes_arr = np.array(car_attributes)
car_attributes_arr
```

Out[11]:
```
array([[ 18,  15,  18,  16,  17],
       [130, 165, 150, 150, 140],
       [307, 350, 318, 304, 302]])
```

In [12]:
```
car_attributes_arr.shape
```

Out[12]:
```
(3, 5)
```

In [13]:
```
car_attributes_arr.dtype
```

Out[13]:
```
dtype('int32')
```

In [14]:
```
car_attributes = [[18, 15, 18, 16, 17],[130, 165, 150, 150, 140],[307, 350, 318, 3(
#converting dtype
car_attributes_arr = np.array(car_attributes, dtype = 'float')
print(car_attributes_arr)
print(car_attributes_arr.dtype)
```

```
[[ 18.  15.  18.  16.  17.]
 [130. 165. 150. 150. 140.]
 [307. 350. 318. 304. 302.]]
float64
```

# Accessing element from 1D array.

In [15]:
```
cars = np.array(['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satel
#accessing the second car from the array
cars[1]
```

Out[15]:
```
'buick skylark 320'
```

# Accessing elements from a 2D array

In [16]:
```
car_names = ['chevrolet', 'buick', 'ply', 'amc', 'ford']
horsepower = [130, 165, 150, 150, 140]
```

```
car_hp_arr = np.array([car_names, horsepower])
car_hp_arr
```

Out[16]:
```
array([['chevrolet', 'buick', 'ply', 'amc', 'ford'],
       ['130', '165', '150', '150', '140']], dtype='<U11')
```

In [17]:
```
car_hp_arr[0]
```

Out[17]:
```
array(['chevrolet', 'buick', 'ply', 'amc', 'ford'], dtype='<U11')
```

In [18]:
```
car_hp_arr[1]
```

Out[18]:
```
array(['130', '165', '150', '150', '140'], dtype='<U11')
```

In [19]:
```
car_hp_arr[0,1]
```

Out[19]:
```
'buick'
```

In [20]:
```
car_hp_arr[0,-1]
```

Out[20]:
```
'ford'
```

# Slicing from 1D array

In [21]:
```
#creating an array of cars
cars = np.array(['chevrolet', 'buick', 'ply', 'amc', 'ford'])
#accessing a subset of cars from the array
cars[1:4]
```

Out[21]:
```
array(['buick', 'ply', 'amc'], dtype='<U9')
```

# Slicing from a 2D array

In [22]:
```
car_names = ['chevrolet', 'buick', 'ply', 'amc', 'ford']
horsepower = [130, 165, 150, 150, 140]
acceleration = [18, 15, 18, 16, 17]
car_hp_acc_arr = np.array([car_names, horsepower, acceleration])
car_hp_acc_arr
```

Out[22]:
```
array([['chevrolet', 'buick', 'ply', 'amc', 'ford'],
       ['130', '165', '150', '150', '140'],
       ['18', '15', '18', '16', '17']], dtype='<U11')
```

In [23]:
```
car_hp_acc_arr[0:2]
```

Out[23]:
```
array([['chevrolet', 'buick', 'ply', 'amc', 'ford'],
       ['130', '165', '150', '150', '140']], dtype='<U11')
```

In [24]:
```
car_hp_acc_arr[0:2, 3:5]
```

Out[24]:
```
array([['amc', 'ford'],
       ['150', '140']], dtype='<U11')
```

In [25]:
```
car_hp_acc_arr[0:3, 0:3]
```

Out[25]:
```
array([['chevrolet', 'buick', 'ply'],
       ['130', '165', '150'],
       ['18', '15', '18']], dtype='<U11')
```

# The engineers at XYZ Custom Cars want to know about the mean and median of horsepower

In [26]:
```
#creating a list of 5 horsepower values
horsepower = [130, 165, 150, 150, 140]
#creating a numpy array from horsepower list
horsepower_arr = np.array(horsepower)
#mean horsepower
print("Mean horsepower = ",np.mean(horsepower_arr))
```

```
Mean horsepower =  147.0
```

In [27]:
```
print("Minimum horsepower: ", np.min(horsepower_arr))
print("Maximum horsepower: ", np.max(horsepower_arr))
```

```
Minimum horsepower:  130
Maximum horsepower:  165
```

# Finding the index of minimum and maximum values:

In [28]:
```
#creating a list of 5 horsepower values
horsepower = [130, 165, 150, 150, 140]
#creating a numpy array from horsepower list
horsepower_arr = np.array(horsepower)
print("Index of Minimum horsepower: ", np.argmin(horsepower_arr))
print("Index of Maximum horsepower: ", np.argmax(horsepower_arr))
```

```
Index of Minimum horsepower:  0
Index of Maximum horsepower:  1
```

# The engineers at XYZ Custom Cars want to know the horsepower of cars that are greater than or equal to 150

In [29]:
```
#creating a list of 5 horsepower values
horsepower = [130, 165, 150, 150, 140]
#creating a numpy array from horsepower list
horsepower_arr = np.array(horsepower)
x = np.where(horsepower_arr >= 150)
print(x) # gives the indices
# With the indices , we can find those values
horsepower_arr[x]
```

```
(array([1, 2, 3], dtype=int64),)
```

Out[29]:
```
array([165, 150, 150])
```

In [30]:
```
horsepower_arr[3]
```

Out[30]:
```
150
```

In [31]:
```
horsepower_arr[[1,4]]
```

Out[31]:
```
array([165, 140])
```

# The Engineers at XYZ Custom Cars want to create a separate array consisting of filtered values of horsepower greater than 135.

In [32]:
```
#creating a list of 5 horsepower values
horsepower = [130, 165, 150, 150, 140]
#creating a numpy array from horsepower list
horsepower_arr = np.array(horsepower)
#creating filter array
x = horsepower_arr > 135
print(x.dtype)
newarr = horsepower_arr[x]
print(x)
print(newarr)
```

```
bool
[False  True  True  True  True]
[165 150 150 140]
```

# The engineers at XYZ Custom Cars want the horsepower in sorted order.

In [33]:
```
#creating a list of 5 horsepower values
horsepower = [130, 165, 150, 150, 140]
#creating a numpy array from horsepower list
horsepower_arr = np.array(horsepower)
#using sort(array)
print('original array: ', horsepower_arr)
print('Sorted array: ', np.sort(horsepower_arr))
print('original array after sorting: ', horsepower_arr)
sortedarray = np.sort(horsepower_arr)
print(sortedarray)
```

```
original array:  [130 165 150 150 140]
Sorted array:  [130 140 150 150 165]
original array after sorting:  [130 165 150 150 140]
[130 140 150 150 165]
```

# array.sort() function modifies the original array by default, whereas the sort(array) function does not

In [34]:
```
horsepower = [130, 165, 150, 150, 140]
horsepower_arr = np.array(horsepower)
np.sort(horsepower_arr)
print(horsepower_arr)
```

```
horsepower_arr.sort()
print(horsepower_arr)
```

```
[130 165 150 150 140]
[130 140 150 150 165]
```

In [35]:

```
#creating a list of 5 horsepower values
horsepower = [130, 165, 150, 150, 140]
#creating a numpy array from horsepower list
horsepower_arr = np.array(horsepower)
#using sort(array)
print('original array: ', horsepower_arr)
horsepower_arr.sort()
print('original array after sorting: ', horsepower_arr)
```

```
original array:  [130 165 150 150 140]
original array after sorting:  [130 140 150 150 165]
```

# The mathematical operations can be performed on Numpy arrays. Numpy makes use of optimized, pre-compiled code to perform mathematical operations on each array element. This eliminates the need of using loops, thereby enhancing the performance. This process is called vectorization. Numpy provides various mathematical functions such as sum(), add(), sub(), log(), sin() etc. which uses vectorization.

In [36]:

```
student_marks_arr = np.array([78, 92, 36, 64, 89])
print(np.sum(student_marks_arr))
```

```
359
```

In [37]:

```
l1 = [2,3,4,5]
l2 = [4,5,6,7]
l3 = l1+l2
l3
```

Out[37]:

```
[2, 3, 4, 5, 4, 5, 6, 7]
```

In [38]:

```
additional_marks = [2, 2, 5, 10, 1]
student_marks_arr =student_marks_arr+additional_marks
student_marks_arr
```

Out[38]:

```
array([80, 94, 41, 74, 90])
```

In [39]:

```
student_marks_arr = np.array([78, 92, 36, 64, 89])
student_marks_arr = np.add(student_marks_arr, additional_marks)
student_marks_arr
```

Out[39]:

```
array([80, 94, 41, 74, 90])
```

# "Broadcasting" refers to the term on how Numpy handles arrays with different shapes during arithmetic operations. Array of smaller size is stretched or copied across the larger array.

In [40]:
```
# Array 1
array1=np.array([5, 10, 15])
# Array 2
array2=np.array([5])
array3= array1 * array2
array3
```

Out[40]:
```
array([25, 50, 75])
```

In [41]:
```
# Array 1
array1=np.array([0,1,2])
# Array 2
array2=np.array([5])
array3= array1 + array2
array3
```

Out[41]:
```
array([5, 6, 7])
```

In [42]:
```
# Array 1
array1=np.array([[1,1,1],[1,1,1],[1,1,1]])
#array1 = np.ones([3,3])
# Array 2
array2=np.array([0,1,2])
array3= array1 + array2
array3
```

Out[42]:
```
array([[1, 2, 3],
       [1, 2, 3],
       [1, 2, 3]])
```

In [43]:
```
# Array 1
array1=np.array([[1,1,1],[1,1,1],[1,1,1]])
#array1 = np.ones([3,3])
# Array 2
array2=np.array([0,1,2])
array3= array1 + array2
array3
```

Out[43]:
```
array([[1, 2, 3],
       [1, 2, 3],
       [1, 2, 3]])
```

In [44]:
```
#Students marks in 4 subjects
students_marks = np.array([[67, 45],[90, 92],[66, 72],[32, 40]])
students_marks
```

Out[44]:
```
array([[67, 45],
       [90, 92],
       [66, 72],
       [32, 40]])
```

In [45]:
```
#Broadcasting
students_marks += [5,10]
students_marks
```

Out[45]:
```
array([[ 72,  55],
       [ 95, 102],
       [ 71,  82],
       [ 37,  50]])
```

# Represent the above data in a 10x2 array. In each row, the first element should contain day number and second element should contain steps walked.

In [47]:
```
import numpy as np
#Creating a 2D array
Day_number = np.arange(1,11)
Steps_walked = [6012,7079,6886,7230,4598,5564,6971,7763,8032,9569]
arr = np.array([Day_number, Steps_walked])
arr = arr.T
arr
```

Out[47]:
```
array([[   1, 6012],
       [   2, 7079],
       [   3, 6886],
       [   4, 7230],
       [   5, 4598],
       [   6, 5564],
       [   7, 6971],
       [   8, 7763],
       [   9, 8032],
       [  10, 9569]])
```

# Lee notices that the tracker's battery dies every day at 7 pm. Lee discovers that on an average, he walks 2000 steps every day after 7 pm. Perform an appropriate operation on your array to add 2000 steps to all the observations.

In [48]:
```
new_arr= arr[:,1] + 2000
arr[:,1]=new_arr
arr
```

Out[48]:
```
array([[    1,  8012],
       [    2,  9079],
       [    3,  8886],
       [    4,  9230],
       [    5,  6598],
       [    6,  7564],
       [    7,  8971],
       [    8,  9763],
       [    9, 10032],
       [   10, 11569]])
```

# Write a program that returns the steps walked if the steps walked are more than 9000.

In [49]:
```
matched = arr[:,1]>9000
matched
new_arr = arr[matched]
new_arr
```

Out[49]:
```
array([[    2,  9079],
       [    4,  9230],
       [    8,  9763],
       [    9, 10032],
       [   10, 11569]])
```

# Print an array containing steps walked in sorted order.

In [50]:
```
sortedArr = arr[arr[:,1].argsort()]
print('Sorted 2D Numpy Array')
print(sortedArr)
```

```
Sorted 2D Numpy Array
[[    5  6598]
 [    6  7564]
 [    1  8012]
 [    3  8886]
 [    7  8971]
 [    2  9079]
 [    4  9230]
 [    8  9763]
 [    9 10032]
 [   10 11569]]
```

# Vectorized Operations

In [51]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import time
import matplotlib.pyplot as plt
```

In [52]:
```
v1 = np.random.rand(1000000, 1)
v2 = np.random.rand(1000000, 1)
```

# Multiplication by a Scalar

In [53]:
```
start = time.process_time()
v1_scaled = np.zeros((1000000, 1))

for i in range(len(v1)):
    v1_scaled[i] = 2 * v1[i]

end = time.process_time()

print("Scaling vector Answer = " + str(v1_scaled))
print("Time taken = " + str(1000*(end - start)) + " ms")
```

```
Scaling vector Answer = [[0.01606641]
 [1.18871072]
 [1.37933358]
 ...
 [0.91803805]
 [0.72178821]
 [0.60712981]]
Time taken = 3703.125 ms
```

In [54]:
```
start = time.process_time()
v1_scaled = np.zeros((1000000, 1))

v1_scaled = 2 * v1

end = time.process_time()

print("Scaling vector Answer = " + str(v1_scaled))
print("Time taken = " + str(1000*(end - start)) + " ms")
```

```
Scaling vector Answer = [[0.01606641]
 [1.18871072]
 [1.37933358]
 ...
 [0.91803805]
 [0.72178821]
 [0.60712981]]
Time taken = 0.0 ms
```

# Dot Products

In [55]:
```
start = time.process_time()
product = 0


for i in range(len(v1)):
    product += v1[i] * v2[i]

end = time.process_time()

print("Dot product Answer = " + str(product))
print("Time taken = " + str(1000*(end - start)) + " ms")
```

```
Dot product Answer = [250052.02591175]
Time taken = 4234.375 ms
```

In [56]:
```
start = time.process_time()
product = 0
```

```
product = np.dot(v1.T, v2)

end = time.process_time()

print("Dot product Answer = " + str(product))
print("Time taken = " + str(1000*(end - start)) + " ms")
```

```
Dot product Answer = [[250052.02591175]]
Time taken = 0.0 ms
```

# Element Wise multiplication

In [57]:

```
start = time.process_time()

answer = np.zeros((1000000, 1))

for i in range(len(v1)):
    answer[i] = v1[i] * v2[i]

end = time.process_time()

print("Element Wise answer = " + str(answer))
print("Time Taken = " + str(1000*(end - start)) + " ms")
```

```
Element Wise answer = [[0.00738031]
 [0.25075942]
 [0.47414855]
 ...
 [0.15277981]
 [0.10664533]
 [0.16544372]]
Time Taken = 2734.375 ms
```

In [58]:

```
start = time.process_time()

answer = np.zeros((1000000, 1))

answer = v1 * v2

end = time.process_time()

print("Element Wise answer = " + str(answer))
print("Time Taken = " + str(1000*(end - start)) + " ms")
```

```
Element Wise answer = [[0.00738031]
 [0.25075942]
 [0.47414855]
 ...
 [0.15277981]
 [0.10664533]
 [0.16544372]]
Time Taken = 0.0 ms
```

# Element Wise Matrix Multiplication

In [ ]:

```
m1 = np.random.rand(10000, 10000)
m2 = np.random.rand(10000, 10000)
answer = np.zeros((10000, 10000))
```

```
start = time.process_time()

for i in range(m1.shape[0]):
    for j in range(m1.shape[1]):
        answer[i, j] = m1[i, j] * m2[i, j]

end = time.process_time()

print("Element Wise Matrix answer = " + str(answer))
print("Time Taken = " + str(1000*(end - start)) + " ms")
```

In [60]:

```
answer = np.zeros((10000, 10000))

start = time.process_time()

answer = np.multiply(m1, m2)

end = time.process_time()

print("Element Wise Matrix answer = " + str(answer))
print("Time Taken = " + str(1000*(end - start)) + " ms")
```

```
Element Wise Matrix answer = [[0.29585707 0.47037833 0.28423336 ... 0.05270216 0.4
9971171 0.01541464]
 [0.08171883 0.48599659 0.51932325 ... 0.00679934 0.09085834 0.38656662]
 [0.44280843 0.52113339 0.04118208 ... 0.12256153 0.16887633 0.73663934]
 ...
 [0.01070294 0.00467872 0.03780787 ... 0.7297074  0.07283657 0.0353923 ]
 [0.1659375  0.01745508 0.09423007 ... 0.04291333 0.30375337 0.16461444]
 [0.27243572 0.89540811 0.58656314 ... 0.0940314  0.41941846 0.06509439]]
Time Taken = 375.0 ms
```

# Time-complexity Plot

In [62]:

```
sizes = [10, 100, 1000, 10000, 100000, 1000000, 10000000]
complexity = pd.DataFrame(columns=['sizes', 'for_loop', 'numpy'])
complexity['sizes'] = sizes
```

In [63]:

```
for_loops = []
numpy = []

for size in sizes:
    v1 = np.random.rand(size, 1)
    v2 = np.random.rand(size, 1)

    #For loop implementation
    start = time.process_time()
    product = 0

    for i in range(len(v1)):
        product += v1[i] * v2[i]

    end = time.process_time()

    for_loops.append(1000*(end-start))

    #Vectorized implementation

    start = time.process_time()
    product = 0
```

```
    product = np.dot(v1.T, v2)

    end = time.process_time()
    numpy.append(1000*(end - start))
```

In [64]:

```
complexity['for_loop'] = for_loops
complexity['numpy'] = numpy
complexity
```

Out[64]:

| | sizes | for_loop | numpy |
|---|---|---|---|
| **0** | 10 | 0.000 | 0.000 |
| **1** | 100 | 0.000 | 0.000 |
| **2** | 1000 | 15.625 | 0.000 |
| **3** | 10000 | 46.875 | 0.000 |
| **4** | 100000 | 703.125 | 0.000 |
| **5** | 1000000 | 4656.250 | 0.000 |
| **6** | 10000000 | 43843.750 | 15.625 |

In [ ]:

In [ ]:

# WEEK-3

# PANDAS PROGRAMS(1)

In [1]:
```
import pandas as pd
import numpy as np
marks = {'Chemistry': [67,90,66,32],
        'Physics': [45,92,72,40],
        'Mathematics': [50,87,81,12],
        'English': [19,90,72,68]}
marks_df = pd.DataFrame(marks, index = ['Subodh', 'Ram', 'Abdul', 'John'])
marks_df
```

Out[1]:

| | Chemistry | Physics | Mathematics | English |
|---|---|---|---|---|
| **Subodh** | 67 | 45 | 50 | 19 |
| **Ram** | 90 | 92 | 87 | 90 |
| **Abdul** | 66 | 72 | 81 | 72 |
| **John** | 32 | 40 | 12 | 68 |

# The teacher wants to create a new column called total and the value of each row in total column should be the sum of all marks of each student

In [2]:
```
marks_df['Total'] = marks_df['Chemistry'] + marks_df['Physics'] + marks_df['Mathematics'] + m
marks_df
```

Out[2]:

| | Chemistry | Physics | Mathematics | English | Total |
|---|---|---|---|---|---|
| **Subodh** | 67 | 45 | 50 | 19 | 181 |
| **Ram** | 90 | 92 | 87 | 90 | 359 |
| **Abdul** | 66 | 72 | 81 | 72 | 291 |
| **John** | 32 | 40 | 12 | 68 | 152 |

# Drop the Total column

In [3]:
```
marks_df.drop(columns = 'Total', inplace = True)
marks_df
```

Out[3]:

| | Chemistry | Physics | Mathematics | English |
|---|---|---|---|---|
| **Subodh** | 67 | 45 | 50 | 19 |
| **Ram** | 90 | 92 | 87 | 90 |
| **Abdul** | 66 | 72 | 81 | 72 |
| **John** | 32 | 40 | 12 | 68 |

# The teacher wants to award five bonus marks to all the students.

In [5]:
```
new_marks = marks_df + 5
new_marks
```

Out[5]:

| | Chemistry | Physics | Mathematics | English |
|---|---|---|---|---|
| **Subodh** | 72 | 50 | 55 | 24 |
| **Ram** | 95 | 97 | 92 | 95 |
| **Abdul** | 71 | 77 | 86 | 77 |
| **John** | 37 | 45 | 17 | 73 |

# The teacher wants to increase the marks of all the students as follows-

Chemistry: + 5, Physics: + 10, Mathematics: +10, English: + 2,

In [6]:
```
new_marks = marks_df + [5,10,10,2]
new_marks
```

Out[6]:

| | Chemistry | Physics | Mathematics | English |
|---|---|---|---|---|
| **Subodh** | 72 | 55 | 60 | 21 |
| **Ram** | 95 | 102 | 97 | 92 |
| **Abdul** | 71 | 82 | 91 | 74 |
| **John** | 37 | 50 | 22 | 70 |

# The teacher wants to get the total marks scored in each subject

In [7]:
```
marks_df.apply(np.sum, axis = 0)
```

Out[7]:
```
Chemistry      255
Physics        249
Mathematics    230
English        249
dtype: int64
```

# The teacher wants to get the total marks scored by each student.

In [8]:
```
marks_df.apply(np.sum, axis = 1)
```

Out[8]:
```
Subodh    181
Ram       359
Abdul     291
John      152
dtype: int64
```

# The teacher wants to hide the marks of the students who scored less than 35 marks and display Fail in place of those marks

In [9]:
```
f = marks_df < 35
marks_df.mask(f, 'Fail')
```

Out[9]:

| | Chemistry | Physics | Mathematics | English |
|---|---|---|---|---|
| **Subodh** | 67 | 45 | 50 | Fail |
| **Ram** | 90 | 92 | 87 | 90 |
| **Abdul** | 66 | 72 | 81 | 72 |
| **John** | Fail | 40 | Fail | 68 |

# PANDAS PROGRAMS(2)

# Perform the following operation on Autompg.csv of XYZ Custom Cars company using Pandas

In [10]:
```
import numpy as np
import pandas as pd
```

# Read data from an existing file

In [11]:
```
import pandas as pd
import numpy as np
df = pd.read_csv('auto_mpg.csv')
df.head()
```

Out[11]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | usa | chevrolet chevelle malibu |
| **1** | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | usa | buick skylark 320 |
| **2** | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | usa | plymouth satellite |
| **3** | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | usa | amc rebel sst |
| **4** | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | usa | ford torino |

# Engineers at XYZ Custom Cars want to know how many cars are Fuel efficient

MPG > 29, Horsepower < 93.5, Weight < 2500

In [12]:
```
df.loc[(df['mpg'] > 29) & (df['horsepower'] < 93.5) & (df['weight'] < 2500)]
```

Out[12]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **51** | 30.0 | 4 | 79.0 | 70.0 | 2074 | 19.5 | 71 | europe | peugeot 304 |
| **52** | 30.0 | 4 | 88.0 | 76.0 | 2065 | 14.5 | 71 | europe | fiat 124b |
| **53** | 31.0 | 4 | 71.0 | 65.0 | 1773 | 19.0 | 71 | japan | toyota corolla 1200 |
| **54** | 35.0 | 4 | 72.0 | 69.0 | 1613 | 18.0 | 71 | japan | datsun 1200 |
| **129** | 31.0 | 4 | 79.0 | 67.0 | 1950 | 19.0 | 74 | japan | datsun b210 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **384** | 32.0 | 4 | 91.0 | 67.0 | 1965 | 15.7 | 82 | japan | honda civic (auto) |
| **385** | 38.0 | 4 | 91.0 | 67.0 | 1995 | 16.2 | 82 | japan | datsun 310 gx |
| **391** | 36.0 | 4 | 135.0 | 84.0 | 2370 | 13.0 | 82 | usa | dodge charger 2.2 |
| **394** | 44.0 | 4 | 97.0 | 52.0 | 2130 | 24.6 | 82 | europe | vw pickup |
| **395** | 32.0 | 4 | 135.0 | 84.0 | 2295 | 11.6 | 82 | usa | dodge rampage |

81 rows × 9 columns

# Engineers at XYZ Custom Cars want to know how many cars are Muscle cars

Displacement >262, Horsepower > 126, Weight in range[2800, 3600]

In [13]:
```
df.loc[(df['displacement'] > 262) & (df['horsepower'] > 126) & (df['weight'] >=2800) & (df['w
```

Out[13]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | usa | chevrolet chevelle malibu |
| **2** | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | usa | plymouth satellite |
| **3** | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | usa | amc rebel sst |
| **4** | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | usa | ford torino |
| **10** | 15.0 | 8 | 383.0 | 170.0 | 3563 | 10.0 | 70 | usa | dodge challenger se |
| **13** | 14.0 | 8 | 455.0 | 225.0 | 3086 | 10.0 | 70 | usa | buick estate wagon (sw) |
| **121** | 15.0 | 8 | 318.0 | 150.0 | 3399 | 11.0 | 73 | usa | dodge dart custom |
| **166** | 13.0 | 8 | 302.0 | 129.0 | 3169 | 12.0 | 75 | usa | ford mustang ii |
| **251** | 20.2 | 8 | 302.0 | 139.0 | 3570 | 12.8 | 78 | usa | mercury monarch ghia |
| **262** | 19.2 | 8 | 305.0 | 145.0 | 3425 | 13.2 | 78 | usa | chevrolet monte carlo landau |
| **264** | 18.1 | 8 | 302.0 | 139.0 | 3205 | 11.2 | 78 | usa | ford futura |

# Engineers at XYZ Custom Cars want to know how many cars are SUVs

Horsepower > 140 , Weight > 4500

In [14]:
```
df.loc[(df['horsepower'] > 140) & (df['weight'] >=4500)]
```

Out[14]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **25** | 10.0 | 8 | 360.0 | 215.0 | 4615 | 14.0 | 70 | usa | ford f250 |
| **28** | 9.0 | 8 | 304.0 | 193.0 | 4732 | 18.5 | 70 | usa | hi 1200d |
| **42** | 12.0 | 8 | 383.0 | 180.0 | 4955 | 11.5 | 71 | usa | dodge monaco (sw) |
| **43** | 13.0 | 8 | 400.0 | 170.0 | 4746 | 12.0 | 71 | usa | ford country squire (sw) |
| **44** | 13.0 | 8 | 400.0 | 175.0 | 5140 | 12.0 | 71 | usa | pontiac safari (sw) |
| **67** | 11.0 | 8 | 429.0 | 208.0 | 4633 | 11.0 | 72 | usa | mercury marquis |
| **68** | 13.0 | 8 | 350.0 | 155.0 | 4502 | 13.5 | 72 | usa | buick lesabre custom |
| **90** | 12.0 | 8 | 429.0 | 198.0 | 4952 | 11.5 | 73 | usa | mercury marquis brougham |
| **94** | 13.0 | 8 | 440.0 | 215.0 | 4735 | 11.0 | 73 | usa | chrysler new yorker brougham |
| **95** | 12.0 | 8 | 455.0 | 225.0 | 4951 | 11.0 | 73 | usa | buick electra 225 custom |
| **103** | 11.0 | 8 | 400.0 | 150.0 | 4997 | 14.0 | 73 | usa | chevrolet impala |
| **104** | 12.0 | 8 | 400.0 | 167.0 | 4906 | 12.5 | 73 | usa | ford country |
| **105** | 13.0 | 8 | 360.0 | 170.0 | 4654 | 13.0 | 73 | usa | plymouth custom suburb |
| **137** | 13.0 | 8 | 350.0 | 150.0 | 4699 | 14.5 | 74 | usa | buick century luxus (sw) |
| **156** | 16.0 | 8 | 400.0 | 170.0 | 4668 | 11.5 | 75 | usa | pontiac catalina |
| **159** | 14.0 | 8 | 351.0 | 148.0 | 4657 | 13.5 | 75 | usa | ford ltd |

# Engineers at XYZ Custom Cars want to know how many cars are Racecars

Weight <2223, acceleration > 17

In [15]:
```
df.loc[(df['acceleration'] > 17) & (df['weight'] < 2223)]
```

Out[15]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **19** | 26.0 | 4 | 97.0 | 46.0 | 1835 | 20.5 | 70 | europe | volkswagen 1131 deluxe sedan |
| **32** | 25.0 | 4 | 98.0 | NaN | 2046 | 19.0 | 71 | usa | ford pinto |
| **51** | 30.0 | 4 | 79.0 | 70.0 | 2074 | 19.5 | 71 | europe | peugeot 304 |
| **53** | 31.0 | 4 | 71.0 | 65.0 | 1773 | 19.0 | 71 | japan | toyota corolla 1200 |
| **54** | 35.0 | 4 | 72.0 | 69.0 | 1613 | 18.0 | 71 | japan | datsun 1200 |
| **55** | 27.0 | 4 | 97.0 | 60.0 | 1834 | 19.0 | 71 | europe | volkswagen model 111 |
| **56** | 26.0 | 4 | 91.0 | 70.0 | 1955 | 20.5 | 71 | usa | plymouth cricket |
| **79** | 26.0 | 4 | 96.0 | 69.0 | 2189 | 18.0 | 72 | europe | renault 12 (sw) |
| **102** | 26.0 | 4 | 97.0 | 46.0 | 1950 | 21.0 | 73 | europe | volkswagen super beetle |
| **117** | 29.0 | 4 | 68.0 | 49.0 | 1867 | 19.5 | 73 | europe | fiat 128 |
| **129** | 31.0 | 4 | 79.0 | 67.0 | 1950 | 19.0 | 74 | japan | datsun b210 |
| **131** | 32.0 | 4 | 71.0 | 65.0 | 1836 | 21.0 | 74 | japan | toyota corolla 1200 |
| **145** | 32.0 | 4 | 83.0 | 61.0 | 2003 | 19.0 | 74 | japan | datsun 710 |
| **181** | 33.0 | 4 | 91.0 | 53.0 | 1795 | 17.5 | 75 | japan | honda civic cvcc |
| **195** | 29.0 | 4 | 85.0 | 52.0 | 2035 | 22.2 | 76 | usa | chevrolet chevette |
| **196** | 24.5 | 4 | 98.0 | 60.0 | 2164 | 22.1 | 76 | usa | chevrolet woody |
| **198** | 33.0 | 4 | 91.0 | 53.0 | 1795 | 17.4 | 76 | japan | honda civic |
| **216** | 31.5 | 4 | 98.0 | 68.0 | 2045 | 18.5 | 77 | japan | honda accord cvcc |
| **218** | 36.0 | 4 | 79.0 | 58.0 | 1825 | 18.6 | 77 | europe | renault 5 gtl |
| **244** | 43.1 | 4 | 90.0 | 48.0 | 1985 | 21.5 | 78 | europe | volkswagen rabbit custom diesel |
| **246** | 32.8 | 4 | 78.0 | 52.0 | 1985 | 19.4 | 78 | japan | mazda glc deluxe |
| **247** | 39.4 | 4 | 85.0 | 70.0 | 2070 | 18.6 | 78 | japan | datsun b210 gx |
| **303** | 31.8 | 4 | 85.0 | 65.0 | 2020 | 19.2 | 79 | japan | datsun 210 |
| **310** | 38.1 | 4 | 89.0 | 60.0 | 1968 | 18.8 | 80 | japan | toyota corolla tercel |
| **322** | 46.6 | 4 | 86.0 | 65.0 | 2110 | 17.9 | 80 | japan | mazda glc |
| **324** | 40.8 | 4 | 85.0 | 65.0 | 2110 | 19.2 | 80 | japan | datsun 210 |
| **325** | 44.3 | 4 | 90.0 | 48.0 | 2085 | 21.7 | 80 | europe | vw rabbit c (diesel) |
| **330** | 40.9 | 4 | 85.0 | NaN | 1835 | 17.3 | 80 | europe | renault lecar deluxe |
| **331** | 33.8 | 4 | 97.0 | 67.0 | 2145 | 18.0 | 80 | japan | subaru dl |

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **346** | 32.3 | 4 | 97.0 | 67.0 | 2065 | 17.8 | 81 | japan | subaru |
| **347** | 37.0 | 4 | 85.0 | 65.0 | 1975 | 19.4 | 81 | japan | datsun 210 mpg |
| **348** | 37.7 | 4 | 89.0 | 62.0 | 2050 | 17.3 | 81 | japan | toyota tercel |
| **376** | 37.0 | 4 | 91.0 | 68.0 | 2025 | 18.2 | 82 | japan | mazda glc custom l |
| **377** | 31.0 | 4 | 91.0 | 68.0 | 1970 | 17.6 | 82 | japan | mazda glc custom |
| **379** | 36.0 | 4 | 98.0 | 70.0 | 2125 | 17.3 | 82 | usa | mercury lynx l |
| **394** | 44.0 | 4 | 97.0 | 52.0 | 2130 | 24.6 | 82 | europe | vw pickup |

# XYZ Custom cars want the data sorted according to the number of cylinders.

In [16]:
```
df.sort_values(by = 'cylinders')
```

Out[16]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **111** | 18.0 | 3 | 70.0 | 90.0 | 2124 | 13.5 | 73 | japan | maxda rx3 |
| **71** | 19.0 | 3 | 70.0 | 97.0 | 2330 | 13.5 | 72 | japan | mazda rx2 coupe |
| **334** | 23.7 | 3 | 70.0 | 100.0 | 2420 | 12.5 | 80 | japan | mazda rx-7 gs |
| **243** | 21.5 | 3 | 80.0 | 110.0 | 2720 | 13.5 | 77 | japan | mazda rx-4 |
| **267** | 27.5 | 4 | 134.0 | 95.0 | 2560 | 14.2 | 78 | japan | toyota corona |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **86** | 14.0 | 8 | 304.0 | 150.0 | 3672 | 11.5 | 73 | usa | amc matador |
| **285** | 17.0 | 8 | 305.0 | 130.0 | 3840 | 15.4 | 79 | usa | chevrolet caprice classic |
| **286** | 17.6 | 8 | 302.0 | 129.0 | 3725 | 13.4 | 79 | usa | ford ltd landau |
| **92** | 13.0 | 8 | 351.0 | 158.0 | 4363 | 13.0 | 73 | usa | ford ltd |
| **0** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | usa | chevrolet chevelle malibu |

398 rows × 9 columns

# There is a requirement in which the cars that have lowest acceleration must be assessed. It is also to be checked that which cars have higher horsepower despite having lower acceleration.

In [17]:
```
df.sort_values(['acceleration', 'horsepower'], ascending = (1,0))
```

Out[17]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| **11** | 14.0 | 8 | 340.0 | 160.0 | 3609 | 8.0 | 70 | usa | plymouth 'cuda 340 |
| **7** | 14.0 | 8 | 440.0 | 215.0 | 4312 | 8.5 | 70 | usa | plymouth fury iii |
| **9** | 15.0 | 8 | 390.0 | 190.0 | 3850 | 8.5 | 70 | usa | amc ambassador dpl |
| **6** | 14.0 | 8 | 454.0 | 220.0 | 4354 | 9.0 | 70 | usa | chevrolet impala |
| **116** | 16.0 | 8 | 400.0 | 230.0 | 4278 | 9.5 | 73 | usa | pontiac grand prix |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **195** | 29.0 | 4 | 85.0 | 52.0 | 2035 | 22.2 | 76 | usa | chevrolet chevette |
| **59** | 23.0 | 4 | 97.0 | 54.0 | 2254 | 23.5 | 72 | europe | volkswagen type 3 |
| **326** | 43.4 | 4 | 90.0 | 48.0 | 2335 | 23.7 | 80 | europe | vw dasher (diesel) |
| **394** | 44.0 | 4 | 97.0 | 52.0 | 2130 | 24.6 | 82 | europe | vw pickup |
| **299** | 27.2 | 4 | 141.0 | 71.0 | 3190 | 24.8 | 79 | europe | peugeot 504 |

398 rows × 9 columns

# PANDAS PROGRAMS(3)

# Consider the rainfall dataset. This data contains region(district) wise rainfall across India. Perform the following operations for the dataset

In [18]:

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

In [20]:

```
import pandas as pd
import numpy as np
df = pd.read_csv('rainfall.csv')
df
```

Out[20]:

| | STATE_UT_NAME | DISTRICT | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ANDAMAN And NICOBAR ISLANDS | NICOBAR | 107.3 | 57.9 | 65.2 | 117.0 | 358.5 | 295.5 | 285.0 | 271.9 | 354.8 | 326.0 | 3 |
| **1** | ANDAMAN And NICOBAR ISLANDS | SOUTH ANDAMAN | 43.7 | 26.0 | 18.6 | 90.5 | 374.4 | 457.2 | 421.3 | 423.1 | 455.6 | 301.2 | 2 |
| **2** | ANDAMAN And NICOBAR ISLANDS | N & M ANDAMAN | 32.7 | 15.9 | 8.6 | 53.4 | 343.6 | 503.3 | 465.4 | 460.9 | 454.8 | 276.1 | 1 |
| **3** | ARUNACHAL PRADESH | LOHIT | 42.2 | 80.8 | 176.4 | 358.5 | 306.4 | 447.0 | 660.1 | 427.8 | 313.6 | 167.1 | |
| **4** | ARUNACHAL PRADESH | EAST SIANG | 33.3 | 79.5 | 105.9 | 216.5 | 323.0 | 738.3 | 990.9 | 711.2 | 568.0 | 206.9 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **636** | KERALA | IDUKKI | 13.4 | 22.1 | 43.6 | 150.4 | 232.6 | 651.6 | 788.9 | 527.3 | 308.4 | 343.2 | 1 |
| **637** | KERALA | KASARGOD | 2.3 | 1.0 | 8.4 | 46.9 | 217.6 | 999.6 | 1108.5 | 636.3 | 263.1 | 234.9 | |
| **638** | KERALA | PATHANAMTHITTA | 19.8 | 45.2 | 73.9 | 184.9 | 294.7 | 556.9 | 539.9 | 352.7 | 266.2 | 359.4 | 2 |
| **639** | KERALA | WAYANAD | 4.8 | 8.3 | 17.5 | 83.3 | 174.6 | 698.1 | 1110.4 | 592.9 | 230.7 | 213.1 | |
| **640** | LAKSHADWEEP | LAKSHADWEEP | 20.8 | 14.7 | 11.8 | 48.9 | 171.7 | 330.2 | 287.7 | 217.5 | 163.1 | 157.1 | 1 |

641 rows × 19 columns

# Check for missing values, if any and drop the corresponding rows.

In [21]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 641 entries, 0 to 640
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   STATE_UT_NAME  641 non-null    object 
 1   DISTRICT       641 non-null    object 
 2   JAN            641 non-null    float64
 3   FEB            641 non-null    float64
 4   MAR            641 non-null    float64
 5   APR            641 non-null    float64
 6   MAY            641 non-null    float64
 7   JUN            641 non-null    float64
 8   JUL            641 non-null    float64
 9   AUG            641 non-null    float64
 10  SEP            641 non-null    float64
 11  OCT            641 non-null    float64
 12  NOV            641 non-null    float64
 13  DEC            641 non-null    float64
 14  ANNUAL         641 non-null    float64
 15  Jan-Feb        641 non-null    float64
 16  Mar-May        641 non-null    float64
 17  Jun-Sep        641 non-null    float64
 18  Oct-Dec        641 non-null    float64
dtypes: float64(17), object(2)
memory usage: 95.3+ KB
```

# Find the district that gets the highest annual rainfall.

In [22]:
```
sorted_df = df.sort_values(by = 'ANNUAL', ascending=False)
highest = sorted_df.iloc[0,1]
print("District that gets the highest annual rainfall:",highest)
```

```
District that gets the highest annual rainfall: TAMENGLONG
```

# Display the top 5 states that get the highest annual rainfall.

In [23]:
```
sorted_df.head(5)
```

Out[23]:

| | STATE_UT_NAME | DISTRICT | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **55** | MANIPUR | TAMENGLONG | 48.5 | 229.6 | 224.5 | 431.5 | 539.9 | 1158.7 | 1820.9 | 1522.1 | 726.3 | 376.1 | 144 |
| **47** | MEGHALAYA | JAINTIA HILLS | 33.8 | 44.1 | 115.1 | 282.3 | 598.8 | 1316.1 | 1591.3 | 933.8 | 826.3 | 517.7 | 110 |
| **46** | MEGHALAYA | EAST KHASI HI | 15.4 | 24.1 | 129.7 | 312.5 | 733.7 | 1476.2 | 1518.4 | 1019.4 | 607.8 | 277.9 | 40 |
| **12** | ARUNACHAL PRADESH | UPPER SIANG | 74.3 | 176.7 | 362.6 | 397.5 | 408.7 | 801.9 | 653.0 | 417.9 | 686.0 | 264.9 | 86 |
| **598** | KARNATAKA | UDUPI | 1.4 | 0.4 | 4.1 | 29.4 | 193.8 | 1081.0 | 1371.6 | 902.2 | 404.9 | 223.8 | 74 |

In [25]:
```
new_df = df.drop(['JAN','FEB', 'MAR','JUN', 'JUL','SEP', 'OCT','DEC'],axis=1)
new_df
```

Out[25]:

| | STATE_UT_NAME | DISTRICT | APR | MAY | AUG | NOV | ANNUAL | Jan-Feb | Mar-May | Jun-Sep | Oct-Dec |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ANDAMAN And NICOBAR ISLANDS | NICOBAR | 117.0 | 358.5 | 271.9 | 315.2 | 2805.2 | 165.2 | 540.7 | 1207.2 | 892.1 |
| **1** | ANDAMAN And NICOBAR ISLANDS | SOUTH ANDAMAN | 90.5 | 374.4 | 423.1 | 275.8 | 3015.7 | 69.7 | 483.5 | 1757.2 | 705.3 |
| **2** | ANDAMAN And NICOBAR ISLANDS | N & M ANDAMAN | 53.4 | 343.6 | 460.9 | 198.6 | 2913.3 | 48.6 | 405.6 | 1884.4 | 574.7 |
| **3** | ARUNACHAL PRADESH | LOHIT | 358.5 | 306.4 | 427.8 | 34.1 | 3043.8 | 123.0 | 841.3 | 1848.5 | 231.0 |
| **4** | ARUNACHAL PRADESH | EAST SIANG | 216.5 | 323.0 | 711.2 | 29.5 | 4034.7 | 112.8 | 645.4 | 3008.4 | 268.1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **636** | KERALA | IDUKKI | 150.4 | 232.6 | 527.3 | 172.9 | 3302.5 | 35.5 | 426.6 | 2276.2 | 564.2 |
| **637** | KERALA | KASARGOD | 46.9 | 217.6 | 636.3 | 84.6 | 3621.6 | 3.3 | 272.9 | 3007.5 | 337.9 |
| **638** | KERALA | PATHANAMTHITTA | 184.9 | 294.7 | 352.7 | 213.5 | 2958.4 | 65.0 | 553.5 | 1715.7 | 624.2 |
| **639** | KERALA | WAYANAD | 83.3 | 174.6 | 592.9 | 93.6 | 3253.1 | 13.1 | 275.4 | 2632.1 | 332.5 |
| **640** | LAKSHADWEEP | LAKSHADWEEP | 48.9 | 171.7 | 217.5 | 117.7 | 1600.0 | 35.5 | 232.4 | 998.5 | 333.6 |

641 rows × 11 columns

# Display the state-wise mean rainfall for all the months using a pivot table

In [26]:

```
new_df = new_df.drop(['ANNUAL'],axis=1)
table = pd.pivot_table(new_df,index=['STATE_UT_NAME'])
table
```

Out[26]:

| STATE_UT_NAME | APR | AUG | Jan-Feb | Jun-Sep | MAY | Mar-May | NOV | Oct-I |
|---|---|---|---|---|---|---|---|---|
| ANDAMAN And NICOBAR ISLANDS | 86.966667 | 385.300000 | 94.500000 | 1616.266667 | 358.833333 | 476.600000 | 263.200000 | 724.033 |
| ANDHRA PRADESH | 19.873913 | 179.426087 | 13.673913 | 639.534783 | 48.765217 | 78.734783 | 58.965217 | 213.130 |
| ARUNACHAL PRADESH | 275.162500 | 378.600000 | 146.981250 | 1784.037500 | 300.262500 | 740.443750 | 43.187500 | 255.912 |
| ASSAM | 181.266667 | 377.370370 | 47.448148 | 1641.200000 | 333.870370 | 592.900000 | 24.922222 | 172.811 |
| BIHAR | 16.865789 | 289.481579 | 22.413158 | 1022.478947 | 51.673684 | 78.413158 | 6.715789 | 77.250 |
| CHANDIGARH | 14.800000 | 287.500000 | 83.200000 | 844.200000 | 30.100000 | 78.100000 | 9.900000 | 65.100 |
| CHATISGARH | 13.116667 | 375.338889 | 20.850000 | 1145.772222 | 17.483333 | 43.577778 | 8.494444 | 76.150 |
| DADAR NAGAR HAVELI | 0.000000 | 655.900000 | 0.700000 | 2316.900000 | 7.400000 | 7.400000 | 10.500000 | 49.100 |
| DAMAN AND DUI | 0.100000 | 394.600000 | 1.050000 | 1481.800000 | 4.150000 | 4.450000 | 12.400000 | 48.400 |
| DELHI | 8.900000 | 245.500000 | 32.700000 | 636.200000 | 19.300000 | 43.500000 | 5.600000 | 34.700 |
| GOA | 7.800000 | 683.800000 | 0.600000 | 2980.900000 | 87.750000 | 96.100000 | 35.000000 | 200.900 |
| GUJARAT | 0.507692 | 257.630769 | 1.176923 | 879.188462 | 4.803846 | 6.453846 | 10.826923 | 37.523 |
| HARYANA | 7.619048 | 190.909524 | 35.942857 | 511.004762 | 14.642857 | 36.000000 | 5.266667 | 31.609 |
| HIMACHAL | 47.683333 | 322.325000 | 162.375000 | 925.100000 | 54.358333 | 189.675000 | 16.908333 | 94.441 |
| JAMMU AND KASHMIR | 82.268182 | 167.918182 | 169.622727 | 471.868182 | 65.136364 | 267.390909 | 27.159091 | 107.736 |
| JHARKHAND | 18.662500 | 310.316667 | 32.158333 | 1093.904167 | 45.875000 | 81.054167 | 10.212500 | 96.320 |
| KARNATAKA | 36.773333 | 209.256667 | 4.723333 | 858.913333 | 88.166667 | 132.103333 | 44.350000 | 198.876 |
| KERALA | 109.021429 | 417.950000 | 25.742857 | 2046.142857 | 244.728571 | 384.821429 | 151.535714 | 480.685 |
| LAKSHADWEEP | 48.900000 | 217.500000 | 35.500000 | 998.500000 | 171.700000 | 232.400000 | 117.700000 | 333.600 |
| MADHYA PRADESH | 3.270000 | 331.048000 | 22.050000 | 938.396000 | 7.006000 | 17.762000 | 10.042000 | 54.102 |
| MAHARASHTRA | 6.974286 | 314.585714 | 8.265714 | 1135.771429 | 19.925714 | 32.897143 | 18.588571 | 101.654 |
| MANIPUR | 150.766667 | 451.800000 | 77.722222 | 1715.344444 | 213.377778 | 446.555556 | 56.000000 | 257.011 |
| MEGHALAYA | 211.228571 | 584.371429 | 36.585714 | 2654.042857 | 430.042857 | 716.028571 | 39.571429 | 276.185 |
| MIZORAM | 152.600000 | 440.588889 | 41.511111 | 1694.888889 | 321.322222 | 570.177778 | 64.633333 | 309.744 |
| NAGALAND | 134.227273 | 350.872727 | 46.154545 | 1313.854545 | 213.381818 | 410.627273 | 38.554545 | 170.063 |
| ORISSA | 36.653333 | 363.346667 | 33.180000 | 1146.516667 | 70.723333 | 134.830000 | 30.400000 | 151.593 |
| PONDICHERRY | 12.275000 | 116.425000 | 52.175000 | 362.025000 | 40.825000 | 69.825000 | 395.150000 | 894.450 |
| PUNJAB | 12.160000 | 172.415000 | 50.445000 | 502.185000 | 16.165000 | 54.225000 | 6.085000 | 41.690 |
| RAJASTHAN | 3.303030 | 194.554545 | 10.069697 | 530.075758 | 10.627273 | 17.745455 | 6.254545 | 23.706 |
| SIKKIM | 206.900000 | 434.600000 | 124.850000 | 1790.750000 | 323.550000 | 661.050000 | 30.950000 | 261.700 |
| TAMIL NADU | 42.596875 | 91.571875 | 32.928125 | 330.840625 | 67.531250 | 128.196875 | 184.625000 | 468.040 |
| TRIPURA | 220.750000 | 356.475000 | 44.875000 | 1497.225000 | 391.575000 | 705.950000 | 43.300000 | 231.075 |
| UTTAR | 5.318310 | 291.232394 | 30.340845 | 837.145070 | 15.561972 | 30.987324 | 4.576056 | 56.971 |

| STATE_UT_NAME | APR | AUG | Jan-Feb | Jun-Sep | MAY | Mar-May | NOV | Oct-I |
|---|---|---|---|---|---|---|---|---|
| PRADESH | | | | | | | | |
| UTTARANCHAL | 29.815385 | 426.784615 | 99.484615 | 1229.769231 | 58.392308 | 139.876923 | 9.238462 | 88.907 |
| WEST BENGAL | 56.647368 | 361.573684 | 34.115789 | 1401.073684 | 139.489474 | 224.110526 | 19.389474 | 151.126 |

# Display the count of districts in each state.

In [27]:
```
df.groupby(['STATE_UT_NAME']).count()['DISTRICT']
```

Out[27]:
```
STATE_UT_NAME
ANDAMAN And NICOBAR ISLANDS     3
ANDHRA PRADESH                 23
ARUNACHAL PRADESH              16
ASSAM                          27
BIHAR                          38
CHANDIGARH                      1
CHATISGARH                     18
DADAR NAGAR HAVELI              1
DAMAN AND DUI                   2
DELHI                           9
GOA                             2
GUJARAT                        26
HARYANA                        21
HIMACHAL                       12
JAMMU AND KASHMIR              22
JHARKHAND                      24
KARNATAKA                      30
KERALA                         14
LAKSHADWEEP                     1
MADHYA PRADESH                 50
MAHARASHTRA                    35
MANIPUR                         9
MEGHALAYA                       7
MIZORAM                         9
NAGALAND                       11
ORISSA                         30
PONDICHERRY                     4
PUNJAB                         20
RAJASTHAN                      33
SIKKIM                          4
TAMIL NADU                     32
TRIPURA                         4
UTTAR PRADESH                  71
UTTARANCHAL                    13
WEST BENGAL                    19
Name: DISTRICT, dtype: int64
```

# For each state, display the district that gets the highest rainfall in May. Also display the recorded rainfall.

In [28]:
```
pivot = pd.pivot_table(data=df,index='STATE_UT_NAME',values=['DISTRICT','MAY'],aggfunc=['max'
print(pivot)
```

| | max | | sum |
|---|---|---|---|
| | DISTRICT | MAY | MAY |
| STATE_UT_NAME | | | |
| ANDAMAN And NICOBAR ISLANDS | SOUTH ANDAMAN | 374.4 | 1076.5 |
| ANDHRA PRADESH | WEST GODAVARI | 96.6 | 1121.6 |
| ARUNACHAL PRADESH | WEST SIANG | 453.0 | 4804.2 |
| ASSAM | UDALGURI(DARA | 604.0 | 9014.5 |
| BIHAR | WEST CHAMPARAN | 155.7 | 1963.6 |
| CHANDIGARH | CHANDIGARH | 30.1 | 30.1 |
| CHATISGARH | SURGUJA | 38.6 | 314.7 |
| DADAR NAGAR HAVELI | DNH | 7.4 | 7.4 |
| DAMAN AND DUI | DIU | 7.4 | 8.3 |
| DELHI | WEST DELHI | 19.3 | 173.7 |
| GOA | SOUTH GOA | 94.3 | 175.5 |
| GUJARAT | VALSAD | 12.5 | 124.9 |
| HARYANA | YAMUNANAGAR | 27.9 | 307.5 |
| HIMACHAL | UNA | 91.7 | 652.3 |
| JAMMU AND KASHMIR | UDHAMPUR | 111.4 | 1433.0 |
| JHARKHAND | WEST SINGHBHUM | 86.1 | 1101.0 |
| KARNATAKA | YADGIR | 193.8 | 2645.0 |
| KERALA | WAYANAD | 300.4 | 3426.2 |
| LAKSHADWEEP | LAKSHADWEEP | 171.7 | 171.7 |
| MADHYA PRADESH | VIDISHA | 19.9 | 350.3 |
| MAHARASHTRA | YAVATMAL | 60.2 | 697.4 |
| MANIPUR | UKHRUL | 539.9 | 1920.4 |
| MEGHALAYA | WEST GARO HIL | 733.7 | 3010.3 |
| MIZORAM | SERCHHIP | 351.4 | 2891.9 |
| NAGALAND | ZUNHEBOTO | 325.6 | 2347.2 |
| ORISSA | SUNDARGARH | 136.8 | 2121.7 |
| PONDICHERRY | YANAM | 43.6 | 163.3 |
| PUNJAB | TARN TARAN | 25.6 | 323.3 |
| RAJASTHAN | UDAIPUR | 20.7 | 350.7 |
| SIKKIM | WEST SIKKIM | 355.4 | 1294.2 |
| TAMIL NADU | VIRUDHUNAGAR | 141.5 | 2161.0 |
| TRIPURA | WEST TRIPURA | 440.1 | 1566.3 |
| UTTAR PRADESH | VARANASI | 38.6 | 1104.9 |
| UTTARANCHAL | UTTARKASHI | 102.1 | 759.1 |
| WEST BENGAL | WEST MIDNAPOR | 345.4 | 2650.3 |

# PANDAS PROGRAMS(4)

# Reshaping Pandas Data frames with Melt & PivotReshaping Pandas Data frames with Melt & Pivot

# Melt

*Melt is used for converting multiple columns into a single column, which is exactly what I need here.#

In [29]:
```
import pandas as pd
```

In [30]:
```
df = pd.DataFrame(data = {
    'Day' : ['MON', 'TUE', 'WED', 'THU', 'FRI'],
    'Google' : [1129,1132,1134,1152,1152],
    'Apple' : [191,192,190,190,188],
    'Samsung' : [191,192,190,190,188]
})
df
```

Out[30]:

| | Day | Google | Apple | Samsung |
|---|---|---|---|---|
| 0 | MON | 1129 | 191 | 191 |
| 1 | TUE | 1132 | 192 | 192 |
| 2 | WED | 1134 | 190 | 190 |
| 3 | THU | 1152 | 190 | 190 |
| 4 | FRI | 1152 | 188 | 188 |

In [31]:

```
reshaped_df = df.melt(id_vars=['Day'])
reshaped_df
```

Out[31]:

| | Day | variable | value |
|---|---|---|---|
| 0 | MON | Google | 1129 |
| 1 | TUE | Google | 1132 |
| 2 | WED | Google | 1134 |
| 3 | THU | Google | 1152 |
| 4 | FRI | Google | 1152 |
| 5 | MON | Apple | 191 |
| 6 | TUE | Apple | 192 |
| 7 | WED | Apple | 190 |
| 8 | THU | Apple | 190 |
| 9 | FRI | Apple | 188 |
| 10 | MON | Samsung | 191 |
| 11 | TUE | Samsung | 192 |
| 12 | WED | Samsung | 190 |
| 13 | THU | Samsung | 190 |
| 14 | FRI | Samsung | 188 |

In [32]:

```
reshaped_df.columns
```

Out[32]:

```
Index(['Day', 'variable', 'value'], dtype='object')
```

In [33]:

```
reshaped_df.columns = [['Day', 'Company', 'Closing Price']]
reshaped_df
```

Out[33]:

| | Day | Company | Closing Price |
|---|---|---|---|
| **0** | MON | Google | 1129 |
| **1** | TUE | Google | 1132 |
| **2** | WED | Google | 1134 |
| **3** | THU | Google | 1152 |
| **4** | FRI | Google | 1152 |
| **5** | MON | Apple | 191 |
| **6** | TUE | Apple | 192 |
| **7** | WED | Apple | 190 |
| **8** | THU | Apple | 190 |
| **9** | FRI | Apple | 188 |
| **10** | MON | Samsung | 191 |
| **11** | TUE | Samsung | 192 |
| **12** | WED | Samsung | 190 |
| **13** | THU | Samsung | 190 |
| **14** | FRI | Samsung | 188 |

In [34]:

```
reshaped_df = df.melt(id_vars=['Day'], var_name='Company', value_name='Closing Price')
reshaped_df
```

Out[34]:

| | Day | Company | Closing Price |
|---|---|---|---|
| **0** | MON | Google | 1129 |
| **1** | TUE | Google | 1132 |
| **2** | WED | Google | 1134 |
| **3** | THU | Google | 1152 |
| **4** | FRI | Google | 1152 |
| **5** | MON | Apple | 191 |
| **6** | TUE | Apple | 192 |
| **7** | WED | Apple | 190 |
| **8** | THU | Apple | 190 |
| **9** | FRI | Apple | 188 |
| **10** | MON | Samsung | 191 |
| **11** | TUE | Samsung | 192 |
| **12** | WED | Samsung | 190 |
| **13** | THU | Samsung | 190 |
| **14** | FRI | Samsung | 188 |

# Unmelt/Reverse Melt/Pivot

Reverse of the melt operation which is called as Pivoting we convert a column with multiple values into several columns of their own. The pivot() method on the dataframe takes two main arguments index and columns. The index parameter is similar to id_vars we have seen before i.e., It is used to specify the column you don't want to touch. The columns parameter is to specify which column should be used to create the new columns.

In [35]:
```
reshaped_df.pivot(index='Day', columns='Company')
```

Out[35]:

| | Closing Price | | |
|---|---|---|---|
| **Company** | **Apple** | **Google** | **Samsung** |
| **Day** | | | |
| **FRI** | 188 | 1152 | 188 |
| **MON** | 191 | 1129 | 191 |
| **THU** | 190 | 1152 | 190 |
| **TUE** | 192 | 1132 | 192 |
| **WED** | 190 | 1134 | 190 |

In [36]:
```
original_df = reshaped_df.pivot(index='Day', columns='Company')['Closing Price'].reset_index(
original_df.columns.name = None
original_df
```

Out[36]:

| | Day | Apple | Google | Samsung |
|---|---|---|---|---|
| **0** | FRI | 188 | 1152 | 188 |
| **1** | MON | 191 | 1129 | 191 |
| **2** | THU | 190 | 1152 | 190 |
| **3** | TUE | 192 | 1132 | 192 |
| **4** | WED | 190 | 1134 | 190 |

# PANDAS POGRAMS(5)

# Map in Pandas

In [37]:
```
import pandas as pd
import numpy as np
technologies= {
        'Duration':['30days','50days','30days','35days','40days'],
         'Fee' :[22000,25000,23000,np.NaN,26000]
          }
df = pd.DataFrame(technologies)
print(df)
```

```
  Duration      Fee
0   30days  22000.0
1   50days  25000.0
2   30days  23000.0
3   35days      NaN
4   40days  26000.0
```

In [38]:
```
df['Fee'] = df['Fee'].map(lambda x: x - (x*10/100))
print(df)
```

```
  Duration      Fee
0   30days  19800.0
1   50days  22500.0
2   30days  20700.0
3   35days      NaN
4   40days  23400.0
```

In [39]:
```
def fun1(x):
    return x/100

df['Discount'] = df['Fee'].map(lambda x:fun1(x))
print(df)
```

```
  Duration      Fee  Discount
0   30days  19800.0     198.0
1   50days  22500.0     225.0
2   30days  20700.0     207.0
3   35days      NaN       NaN
4   40days  23400.0     234.0
```

In [40]:
```
df['Service'] = df['Fee'].map(lambda x: x - (x*10/100))
df
```

Out[40]:

| | Duration | Fee | Discount | Service |
|---|---|---|---|---|
| **0** | 30days | 19800.0 | 198.0 | 17820.0 |
| **1** | 50days | 22500.0 | 225.0 | 20250.0 |
| **2** | 30days | 20700.0 | 207.0 | 18630.0 |
| **3** | 35days | NaN | NaN | NaN |
| **4** | 40days | 23400.0 | 234.0 | 21060.0 |

In [41]:
```
df['Fee'] = df['Fee'].map('{} RS'.format)
df['Discount'] = df['Discount'].map('{} RS'.format)
print(df)
```

```
  Duration         Fee  Discount  Service
0   30days  19800.0 RS  198.0 RS  17820.0
1   50days  22500.0 RS  225.0 RS  20250.0
2   30days  20700.0 RS  207.0 RS  18630.0
3   35days      nan RS    nan RS      NaN
4   40days  23400.0 RS  234.0 RS  21060.0
```

In [42]:
```
df['Service'] = df['Service'].map('{} RS'.format, na_action='ignore')
print(df)
```

```
  Duration         Fee  Discount     Service
0   30days  19800.0 RS  198.0 RS  17820.0 RS
1   50days  22500.0 RS  225.0 RS  20250.0 RS
2   30days  20700.0 RS  207.0 RS  18630.0 RS
3   35days      nan RS    nan RS         NaN
4   40days  23400.0 RS  234.0 RS  21060.0 RS
```

# PANDAS PROGRAMS(6)

# Aggregate functions in Pandas

In [43]:
```
import numpy as np
import pandas as pd
```

In [44]:
```
marks = {'Chemistry': [67,90,66,32,72,45,60,98],
         'Physics': [45,92,72,92,72,34,72,45],
```

```
            }
marks_df = pd.DataFrame(marks, index = ['Subodh', 'Ram', 'Abdul', 'John','Nandini', 'Zoya', '
marks_df
```

Out[44]:

| | Chemistry | Physics |
|---|---|---|
| **Subodh** | 67 | 45 |
| **Ram** | 90 | 92 |
| **Abdul** | 66 | 72 |
| **John** | 32 | 92 |
| **Nandini** | 72 | 72 |
| **Zoya** | 45 | 34 |
| **Shivam** | 60 | 72 |
| **James** | 98 | 45 |

In [45]:

```
marks_df.Chemistry.sum()
```

Out[45]:

```
530
```

In [46]:

```
marks_df['Chemistry'].sum()
```

Out[46]:

```
530
```

In [47]:

```
marks_df['Physics'].mean()
```

Out[47]:

```
65.5
```

In [48]:

```
marks_df.mean()
```

Out[48]:

```
Chemistry    66.25
Physics      65.50
dtype: float64
```

In [49]:

```
marks_df.sum()
```

Out[49]:

```
Chemistry    530
Physics      524
dtype: int64
```

In [50]:

```
marks_df.count()
```

Out[50]:

```
Chemistry    8
Physics      8
dtype: int64
```

In [51]:

```
marks_df.agg(['min', 'max','sum','mean','median'])
```

Out[51]:

| | Chemistry | Physics |
|---|---|---|
| **min** | 32.00 | 34.0 |
| **max** | 98.00 | 92.0 |
| **sum** | 530.00 | 524.0 |
| **mean** | 66.25 | 65.5 |
| **median** | 66.50 | 72.0 |

In [52]:

```
print(marks_df)
marks_df.groupby("Physics").max()
```

```
          Chemistry  Physics
Subodh           67       45
Ram              90       92
Abdul            66       72
John             32       92
Nandini          72       72
Zoya             45       34
Shivam           60       72
James            98       45
```

Out[52]:

| | Chemistry |
|---|---|
| **Physics** | |
| **34** | 45 |
| **45** | 98 |
| **72** | 72 |
| **92** | 90 |

# PANDAS PROGRAMS(7)

# Generating a date range

In [53]:
```
import pandas as pd
from datetime import datetime
import numpy as np
```

In [54]:
```
range_date1 = pd.date_range(start ='1/1/2019', end ='1/08/2019',freq='D')  #days
print(range_date1)
```

```
DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
               '2019-01-05', '2019-01-06', '2019-01-07', '2019-01-08'],
              dtype='datetime64[ns]', freq='D')
```

In [55]:
```
range_date3 = pd.date_range(start ='1/1/2019', end ='1/08/2020',freq='M') #months
print(range_date3)
```

```
DatetimeIndex(['2019-01-31', '2019-02-28', '2019-03-31', '2019-04-30',
               '2019-05-31', '2019-06-30', '2019-07-31', '2019-08-31',
               '2019-09-30', '2019-10-31', '2019-11-30', '2019-12-31'],
              dtype='datetime64[ns]', freq='M')
```

In [56]:
```
range_date2 = pd.date_range(start ='1/1/2019', end ='1/02/2019',freq='H') #hours
print(range_date2)
```

```
DatetimeIndex(['2019-01-01 00:00:00', '2019-01-01 01:00:00',
               '2019-01-01 02:00:00', '2019-01-01 03:00:00',
               '2019-01-01 04:00:00', '2019-01-01 05:00:00',
               '2019-01-01 06:00:00', '2019-01-01 07:00:00',
               '2019-01-01 08:00:00', '2019-01-01 09:00:00',
               '2019-01-01 10:00:00', '2019-01-01 11:00:00',
               '2019-01-01 12:00:00', '2019-01-01 13:00:00',
               '2019-01-01 14:00:00', '2019-01-01 15:00:00',
               '2019-01-01 16:00:00', '2019-01-01 17:00:00',
               '2019-01-01 18:00:00', '2019-01-01 19:00:00',
               '2019-01-01 20:00:00', '2019-01-01 21:00:00',
               '2019-01-01 22:00:00', '2019-01-01 23:00:00',
               '2019-01-02 00:00:00'],
              dtype='datetime64[ns]', freq='H')
```

In [57]:
```
range_date4= pd.date_range(start ='1/1/2019', end ='1/08/2020',freq='3M') #3months
print(range_date4)
```

```
DatetimeIndex(['2019-01-31', '2019-04-30', '2019-07-31', '2019-10-31'], dtype='datetime64[n
s]', freq='3M')
```

In [58]:
```python
range_date5 = pd.date_range(start ='1/1/2019', end ='1/08/2020',freq=None) #days by default
print(range_date5)
```

```
DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
               '2019-01-05', '2019-01-06', '2019-01-07', '2019-01-08',
               '2019-01-09', '2019-01-10',
               ...
               '2019-12-30', '2019-12-31', '2020-01-01', '2020-01-02',
               '2020-01-03', '2020-01-04', '2020-01-05', '2020-01-06',
               '2020-01-07', '2020-01-08'],
              dtype='datetime64[ns]', length=373, freq='D')
```

In [59]:
```python
range_date6= pd.date_range(start ='1/1/2019', end ='1/2/2019',freq='min') #minutes
print(range_date6)
```

```
DatetimeIndex(['2019-01-01 00:00:00', '2019-01-01 00:01:00',
               '2019-01-01 00:02:00', '2019-01-01 00:03:00',
               '2019-01-01 00:04:00', '2019-01-01 00:05:00',
               '2019-01-01 00:06:00', '2019-01-01 00:07:00',
               '2019-01-01 00:08:00', '2019-01-01 00:09:00',
               ...
               '2019-01-01 23:51:00', '2019-01-01 23:52:00',
               '2019-01-01 23:53:00', '2019-01-01 23:54:00',
               '2019-01-01 23:55:00', '2019-01-01 23:56:00',
               '2019-01-01 23:57:00', '2019-01-01 23:58:00',
               '2019-01-01 23:59:00', '2019-01-02 00:00:00'],
              dtype='datetime64[ns]', length=1441, freq='T')
```

In [60]:
```python
range_date7 = pd.date_range(start ='1/1/2018', periods = 13)
print(range_date7)
```

```
DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04',
               '2018-01-05', '2018-01-06', '2018-01-07', '2018-01-08',
               '2018-01-09', '2018-01-10', '2018-01-11', '2018-01-12',
               '2018-01-13'],
              dtype='datetime64[ns]', freq='D')
```

In [61]:
```python
range_date7 = pd.date_range(end ='1/13/2018', periods = 13)
print(range_date7)
```

```
DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04',
               '2018-01-05', '2018-01-06', '2018-01-07', '2018-01-08',
               '2018-01-09', '2018-01-10', '2018-01-11', '2018-01-12',
               '2018-01-13'],
              dtype='datetime64[ns]', freq='D')
```

In [62]:
```python
print(type(range_date1[1]))
```

```
<class 'pandas._libs.tslibs.timestamps.Timestamp'>
```

# We have first created a time series then converted this data into dataframe and use random function to generate the random data and map over the dataframe.

In [63]:
```python
range_data = pd.date_range(start ='1/1/2019', end ='1/08/2019',
                                            freq ='D')

df = pd.DataFrame(range_data, columns =['date'])


df['values'] = np.random.randint(0, 100, size =(len(range_data)))
```

In [64]:
```
print(df.head(10))
```

```
         date  values
0 2019-01-01      41
1 2019-01-02      22
2 2019-01-03      39
3 2019-01-04      86
4 2019-01-05      17
5 2019-01-06      88
6 2019-01-07      33
7 2019-01-08      70
```

# In order to do time series manipulation, we need to have a datetime index so that dataframe is indexed on the timestamp.

In [65]:
```
df['datetime'] = pd.to_datetime(df['date'])
df = df.set_index('datetime')
df.drop(['date'], axis = 1, inplace = True)
df
```

Out[65]:

| datetime | values |
|---|---|
| **2019-01-01** | 41 |
| **2019-01-02** | 22 |
| **2019-01-03** | 39 |
| **2019-01-04** | 86 |
| **2019-01-05** | 17 |
| **2019-01-06** | 88 |
| **2019-01-07** | 33 |
| **2019-01-08** | 70 |

In [ ]:

# Consider the credit card dataset which contains the following columns:

- CLIENTNUM: Primary key of the dataset
- Attrition_Flag: Indicates if a customer is retained or attrited
- Customer_Age: Age of the customer
- Gender: Gender of the customer
- Dependent_count: Number of people dependent on the customer
- Education_Level: Highest level of education of the customer
- Income_Category: Range of income of the customer
- Credit_Limit: Credit card limit
- Total_Revolving_Bal: Pending balance of the credit
- Avg_Purchase: Amount of purchase made by the customer on credit card
- Total_Trans_Amt: Total transaction amount

In [2]:
```
#Importing the necessary Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

In [3]:
```
#Importing the required dataset

credit_df = pd.read_csv("CreditCard_DV.csv")
credit_df  #showing a single row in the df
```

Out[3]:

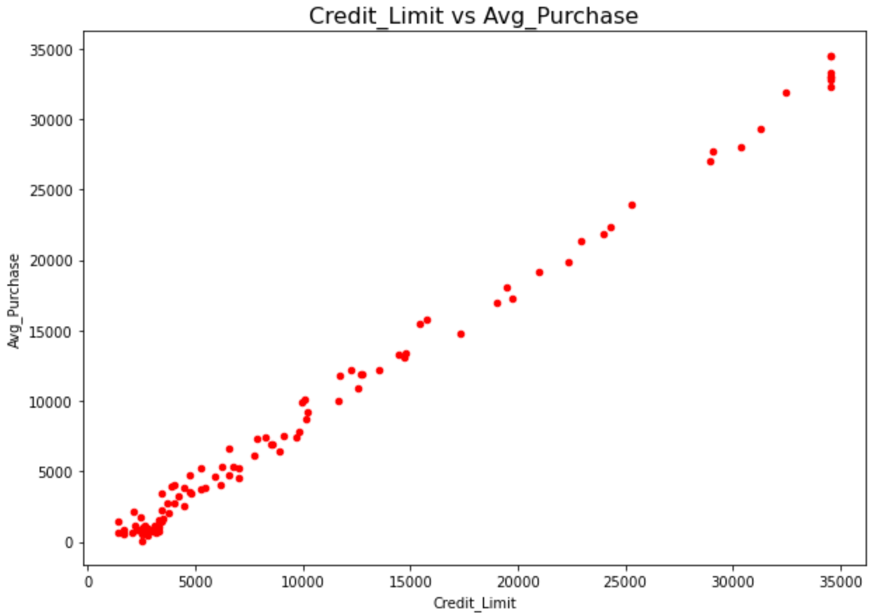
| | CLIENTNUM | Attrition_Flag | Customer_Age | Gender | Dependent_count | Education_Level | Income_Category | Credit_Limit | Total_Revolving_Bal | Avg_Purchase | Total_Trans_Amt |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 768805383 | Existing Customer | 45 | M | 3 | High School | $60K-80$K | 12691.0 | 777 | 11914.0 | 1144 |
| **1** | 818770008 | Existing Customer | 49 | F | 5 | Graduate | Less than $40K | 8256.0 | 864 | 7392.0 | 1291 |
| **2** | 713982108 | Existing Customer | 51 | M | 3 | Graduate | $80K-120$K | 3418.0 | 0 | 3418.0 | 1887 |
| **3** | 769911858 | Existing Customer | 40 | F | 4 | High School | Less than $40K | 3313.0 | 2517 | 796.0 | 1171 |
| **4** | 709106358 | Existing Customer | 40 | M | 3 | Uneducated | $60K-80$K | 4716.0 | 0 | 4716.0 | 816 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **95** | 719712633 | Existing Customer | 64 | M | 1 | Graduate | Less than $40K | 1709.0 | 895 | 814.0 | 1673 |
| **96** | 772629333 | Existing Customer | 45 | M | 3 | Graduate | $40K-60$K | 3454.0 | 1200 | 2254.0 | 1313 |
| **97** | 720336708 | Existing Customer | 53 | M | 3 | Doctorate | $40K-60$K | 3789.0 | 1706 | 2083.0 | 1609 |
| **98** | 802013583 | Existing Customer | 56 | M | 3 | College | $120K + | 9689.0 | 2250 | 7439.0 | 1158 |
| **99** | 711887583 | Attrited Customer | 47 | M | 2 | Unknown | $80K-120$K | 5449.0 | 1628 | 3821.0 | 836 |

100 rows × 11 columns

# Create a bivariate plot to find if there is a correlation between credit card limit and average purchase made on the card.

In [4]:
```
#To Plot the data as a scatter plot

ax = credit_df.plot("Credit_Limit","Avg_Purchase",kind="scatter", color = "red",marker = "o",figsize=(10,7))

#To add labels and title to the output

ax.set_xlabel("Credit_Limit")   #sets label for x-axis
ax.set_ylabel("Avg_Purchase")   #sets label for y-axis
ax.set_title("Credit_Limit vs Avg_Purchase",fontsize=16)    #sets title for the graph
```

Out[4]:
```
Text(0.5, 1.0, 'Credit_Limit vs Avg_Purchase')
```



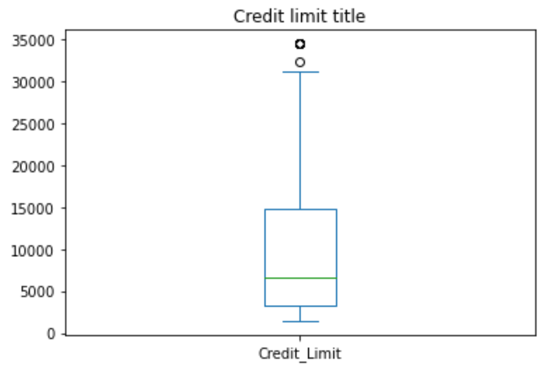

# Visualise the distribution of values for credit card limit and average purchase made on the card. Also, identify the outliers in the data, if any.

In [5]:
```
credit_df["Credit_Limit"].describe()
```

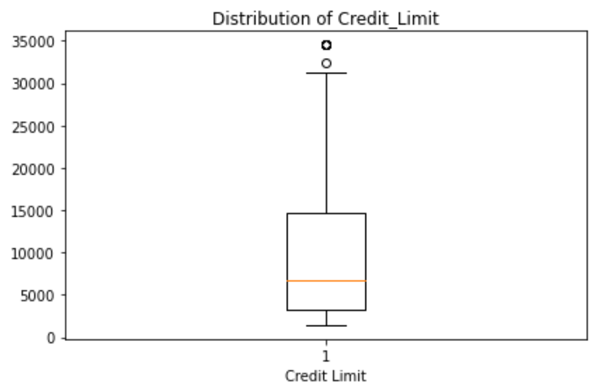
Out[5]:
```
count      100.000000
mean     10881.756000
std      10056.333148
min       1438.300000
25%       3309.250000
50%       6666.000000
75%      14746.500000
max      34516.000000
Name: Credit_Limit, dtype: float64
```

In [6]:
```python
ax = credit_df["Credit_Limit"].plot(kind="box")
ax.set_title("Credit limit title")
```

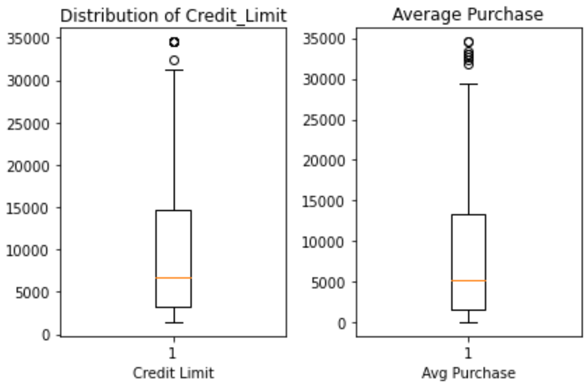
Out[6]: Text(0.5, 1.0, 'Credit limit title')




In [7]:
```python
fig, ax1 = plt.subplots(1, 1)

#The following lines of code change the alignment from vertical to horizontal
ax1.boxplot(credit_df["Credit_Limit"])

#The following lines of code are used to add labels to axes and title to the graph

ax1.set_title('Distribution of Credit_Limit')
ax1.set_xlabel('Credit Limit')



#In case of any superimposition of the subplots, the following functions caters the aesthetics
fig.tight_layout()
```



In [8]:
```python
fig, (ax1, ax2) = plt.subplots(1, 2)

#The following lines of code change the alignment from vertical to horizontal
ax1.boxplot(credit_df["Credit_Limit"])
ax2.boxplot(credit_df["Avg_Purchase"])

#The following lines of code are used to add labels to axes and title to the graph

ax1.set_title('Distribution of Credit_Limit')
ax1.set_xlabel('Credit Limit')

ax2.set_title('Average Purchase')
ax2.set_xlabel("Avg Purchase")

#In case of any superimposition of the subplots, the following functions caters the aesthetics
fig.tight_layout()
```



In [9]:
```
cr_limit_arr = credit_df["Credit_Limit"]
# finding the 1st quartile
q1 = np.quantile(cr_limit_arr, 0.25)

# finding the 3rd quartile
q3 = np.quantile(cr_limit_arr, 0.75)
med = np.median(cr_limit_arr)

# finding the iqr region
iqr = q3-q1

# finding upper and lower whiskers
upper_bound = q3+(1.5*iqr)
lower_bound = q1-(1.5*iqr)
print("IQR:",iqr)
print("upper_bound:",upper_bound)
print("lower_bound:",lower_bound)
```

```
IQR: 11437.25
upper_bound: 31902.375
lower_bound: -13846.625
```

In [10]:
```
outliers = cr_limit_arr[(cr_limit_arr <= lower_bound) | (cr_limit_arr >= upper_bound)]
print('The following are the outliers in the boxplot of Credit Limit:\n',outliers)
```

```
The following are the outliers in the boxplot of Credit Limit:
 6     34516.0
40    32426.0
45    34516.0
61    34516.0
65    34516.0
70    34516.0
81    34516.0
84    34516.0
Name: Credit_Limit, dtype: float64
```

In [11]:
```
x = credit_df['Credit_Limit']
v = x[(x == 34516)]
v
```

Out[11]:
```
6     34516.0
45    34516.0
61    34516.0
65    34516.0
70    34516.0
81    34516.0
84    34516.0
Name: Credit_Limit, dtype: float64
```

In [12]:
```python
avg_purchase = credit_df["Avg_Purchase"]
# finding the 1st quartile
q1 = np.quantile(avg_purchase, 0.25)

# finding the 3rd quartile
q3 = np.quantile(avg_purchase, 0.75)
med = np.median(avg_purchase)

# finding the iqr region
iqr = q3-q1

# finding upper and lower whiskers
upper_bound = q3+(1.5*iqr)
lower_bound = q1-(1.5*iqr)
print("IQR:",iqr)
print("upper_bound:",upper_bound)
print("lower_bound:",lower_bound)
```

```
IQR: 11790.425
upper_bound: 31022.887499999997
lower_bound: -16138.812499999996
```

In [13]:
```python
outliers = avg_purchase[(avg_purchase <= lower_bound) | (avg_purchase >= upper_bound)]
print('The following are the outliers in the boxplot of Average Purchase:\n',outliers)
```

```
The following are the outliers in the boxplot of Average Purchase:
 6     32252.0
40    31848.0
45    34516.0
61    34516.0
65    33001.0
70    32753.0
81    32983.0
84    33297.0
Name: Avg_Purchase, dtype: float64
```

# Provide a visual representation of the number of customers in each income group using a bar chart.

In [14]:
```python
categories = credit_df["Income_Category"].unique()
categories
```

Out[14]:
```
array(['$60K - $80K', 'Less than $40K', '$80K - $120K', '$40K - $60K',
       '$120K +', 'Unknown'], dtype=object)
```

In [15]:
```python
count_df = pd.DataFrame(credit_df[["Income_Category"]].groupby(by= "Income_Category").size().reset_index())
count_df.columns = [["Income_Category","Count"]]
count_df
```

Out[15]:

| | Income_Category | Count |
|---|---|---|
| **0** | $120K + | 11 |
| **1** | 40K−60K | 15 |
| **2** | 60K−80K | 22 |
| **3** | 80K−120K | 23 |
| **4** | Less than $40K | 22 |
| **5** | Unknown | 7 |

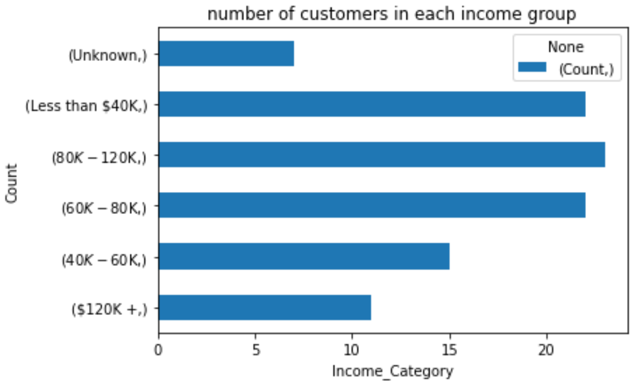
In [16]:
```python
count_df.set_index('Income_Category', inplace = True)
count_df
```

Out[16]:

| Income_Category | Count |
|---|---|
| ($120K +,) | 11 |
| (40K – 60K,) | 15 |
| (60K – 80K,) | 22 |
| (80K – 120K,) | 23 |
| (Less than $40K,) | 22 |
| (Unknown,) | 7 |

In [17]:
```python
count_df['Count'].plot(kind="barh")
plt.title("number of customers in each income group")
plt.xlabel("Income_Category")
plt.ylabel("Count")
```

Out[17]:
```
Text(0, 0.5, 'Count')
```




## Plot the frequency distribution of the total transaction amount.

In [18]:
```python
credit_df["Total_Trans_Amt"].min()
```

Out[18]:
```
602
```

In [19]:
```python
credit_df["Total_Trans_Amt"].max()
```

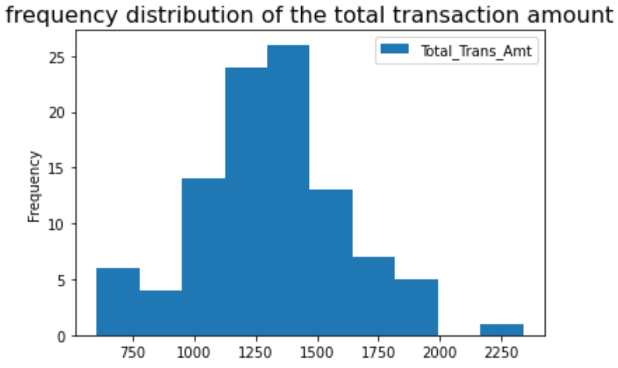
Out[19]:
```
2339
```

In [20]:
```python
credit_df["Total_Trans_Amt"].max() - credit_df["Total_Trans_Amt"].min()
```

Out[20]:
```
1737
```

In [21]:
```
credit_df["Total_Trans_Amt"].plot(kind="hist")
plt.title("frequency distribution of the total transaction amount", fontsize=16)
plt.legend()
```

Out[21]: <matplotlib.legend.Legend at 0x26f979d0d30>




## Graphically represent the percentage of customers retained and those attrited. Highlight the latter by slicing it apart from the main pie

In [22]:
```
Attrition_df= pd.DataFrame(credit_df[["Attrition_Flag"]].groupby(by= ["Attrition_Flag"]).size().reset_index())
Attrition_df.columns = [["Attrition_Flag","Count"]]
Attrition_df
```

Out[22]:

| | Attrition_Flag | Count |
|---|---|---|
| **0** | Attrited Customer | 7 |
| **1** | Existing Customer | 93 |

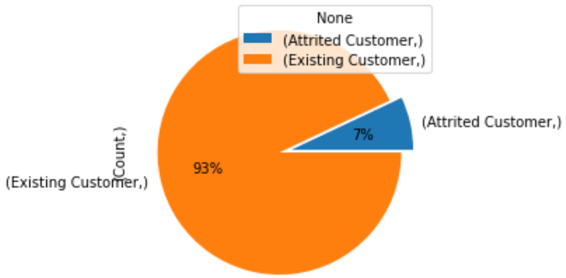
In [23]:
```
Attrition_df.set_index('Attrition_Flag', inplace = True)
Attrition_df
```

Out[23]:

| | Count |
|---|---|
| **Attrition_Flag** | |
| **(Attrited Customer,)** | 7 |
| **(Existing Customer,)** | 93 |

In [24]:
```
explode = (0.05, 0.05)
Attrition_df.plot(kind='pie', y='Count', autopct='%1.0f%%',explode=explode)
```

Out[24]: <AxesSubplot:ylabel='(Count,)'>




# Consider the Cars93 dataset which contains the following columns:

Manufacturer
Model
Type
Price
MPG.city
MPG.highway
Cylinders
EngineSize
Horsepower etc

In [25]:
```
#Importing the required dataset

cars_df = pd.read_csv("Cars93.csv")
columns = ["Manufacturer","Model","Type","Price","MPG.city","MPG.highway","Horsepower","Rear.seat.room","Passengers"]
cars_df[columns].head()
```

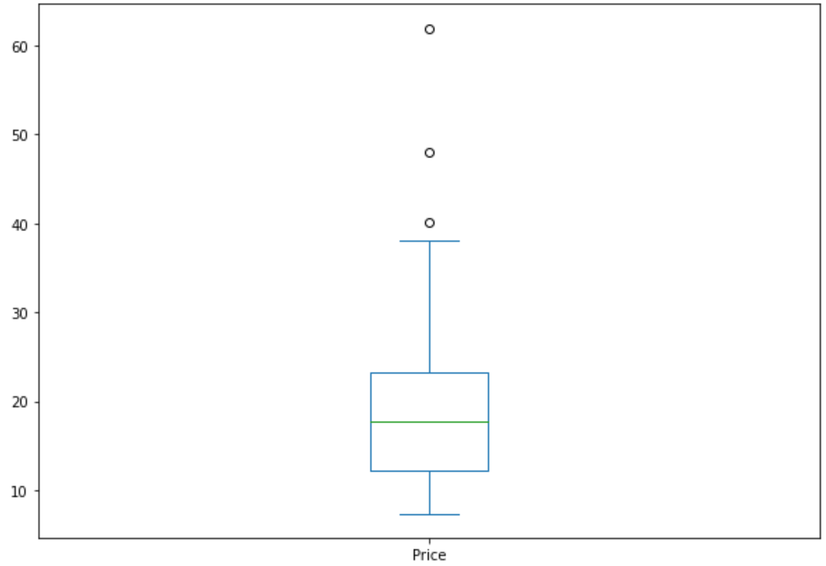
Out[25]:

| | Manufacturer | Model | Type | Price | MPG.city | MPG.highway | Horsepower | Rear.seat.room | Passengers |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Acura | Integra | Small | 15.9 | 25 | 31 | 140 | 26.5 | 5 |
| **1** | Acura | Legend | Midsize | 33.9 | 18 | 25 | 200 | 30.0 | 5 |
| **2** | Audi | 90 | Compact | 29.1 | 20 | 26 | 172 | 28.0 | 5 |
| **3** | Audi | 100 | Midsize | 37.7 | 19 | 26 | 172 | 31.0 | 6 |
| **4** | BMW | 535i | Midsize | 30.0 | 22 | 30 | 208 | 27.0 | 4 |

## Visualize the spread of data for the 'Price' column

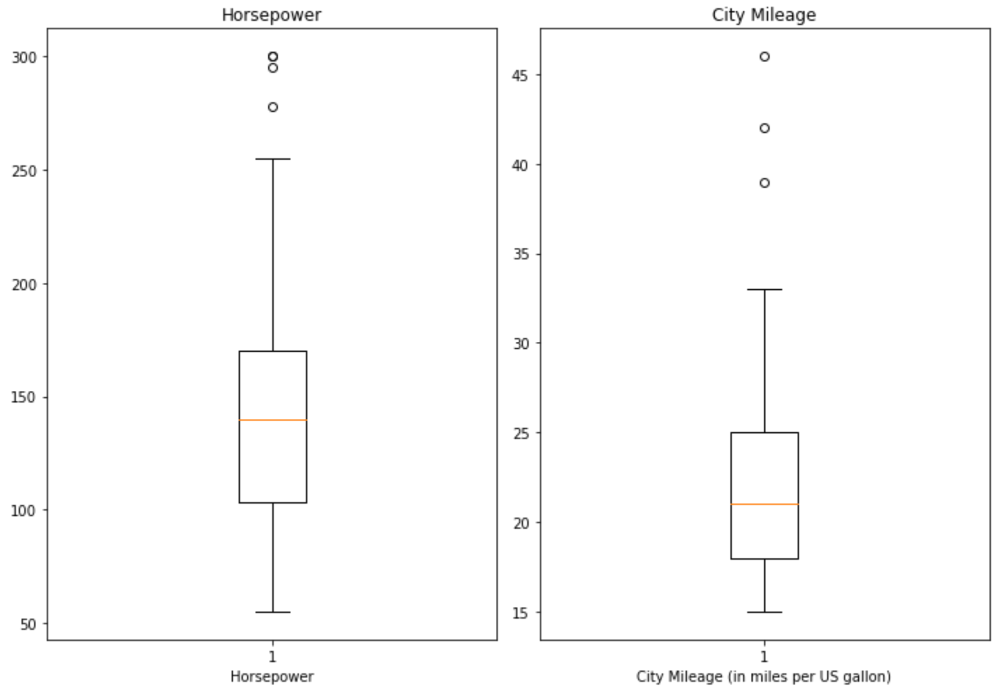
In [26]:
```
cars_df["Price"].plot(kind="box",figsize = (10,7))
```

Out[26]: <AxesSubplot:>




## Visualize the distribution of price for compact and large type of cars

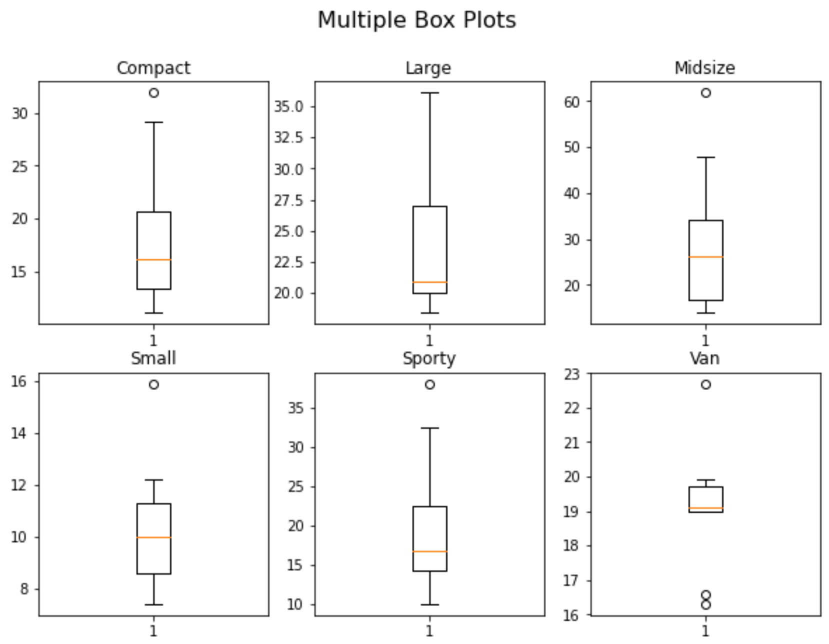
In [27]:
```
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_figwidth(10)
fig.set_figheight(7)

#The following lines of code change the alignment from vertical to horizontal

ax1.boxplot(cars_df["Horsepower"])
ax2.boxplot(cars_df["MPG.city"])

#The following lines of code are used to add labels to axes and title to the graph

ax1.set_title('Horsepower')
ax1.set_xlabel('Horsepower')
ax2.set_title('City Mileage')
ax2.set_xlabel("City Mileage (in miles per US gallon)")
#In case of any superimposition of the subplots, the following functions caters the aesthetics

fig.tight_layout()
```




## Visualize the distribution of price for each type of car

In [28]:
```
fig, ax = plt.subplots(2, 3)
fig.set_figwidth(10)
fig.set_figheight(7)
fig.suptitle("Multiple Box Plots", fontsize=16)

ax[0][0].boxplot(cars_df["Price"][cars_df["Type"]=="Compact"])
ax[0][0].set_title('Compact')

ax[0][1].boxplot(cars_df["Price"][cars_df["Type"]=="Large"])
ax[0][1].set_title('Large')

ax[0][2].boxplot(cars_df["Price"][cars_df["Type"]=="Midsize"])
ax[0][2].set_title('Midsize')

ax[1][0].boxplot(cars_df["Price"][cars_df["Type"]=="Small"])
ax[1][0].set_title('Small')

ax[1][1].boxplot(cars_df["Price"][cars_df["Type"]=="Sporty"])
ax[1][1].set_title('Sporty')

ax[1][2].boxplot(cars_df["Price"][cars_df["Type"]=="Van"])
ax[1][2].set_title('Van')
```

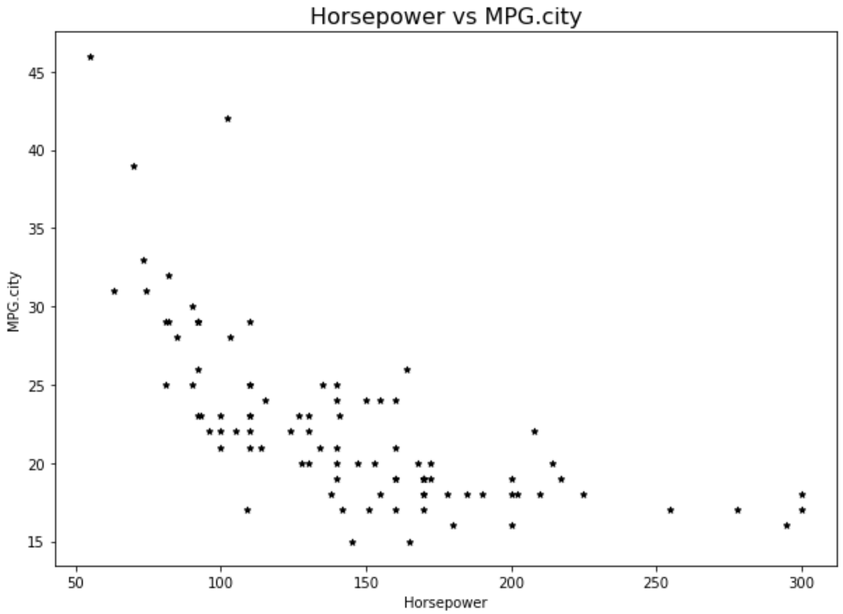
Out[28]:
```
Text(0.5, 1.0, 'Van')
```



## Visualize the correlation between Horsepower and Mileage in the city

In [29]:
```
ax = cars_df.plot(["Horsepower"],["MPG.city"],kind="scatter", color = "black",marker = "*",figsize=(10,7))

#To add labels and title to the output

ax.set_xlabel("Horsepower") #sets label for x-axis
ax.set_ylabel("MPG.city")   #sets label for y-axis
ax.set_title("Horsepower vs MPG.city",fontsize=16)  #sets title for the graph
```

Out[29]:
```
Text(0.5, 1.0, 'Horsepower vs MPG.city')
```



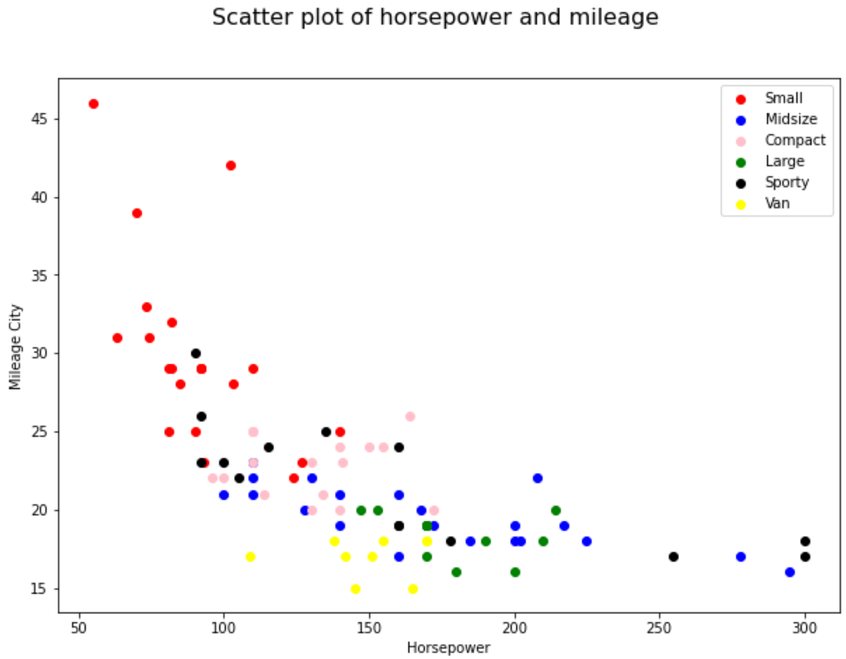

## Visualize the correlation between Horsepower and Mileage in the city for each type of car

In [30]:

```python
fig = plt.figure()
fig.set_figwidth(10)
fig.set_figheight(7)

car_type_list = cars_df["Type"].unique()
print(car_type_list)
colors_list = ['red','blue','pink','green','black','yellow']

for car_type,colr in zip(car_type_list,colors_list):    # for every car type in the car_type_list we plot all the points in the scatter plot
    x = cars_df[cars_df["Type"] == car_type]["Horsepower"]
    y = cars_df[cars_df["Type"] == car_type]["MPG.city"]
    plt.scatter(x,y,color = colr,label=car_type)

plt.suptitle("Scatter plot of horsepower and mileage",fontsize=16)
plt.xlabel("Horsepower")
plt.ylabel("Mileage City")
plt.legend()
```

```
['Small' 'Midsize' 'Compact' 'Large' 'Sporty' 'Van']
```

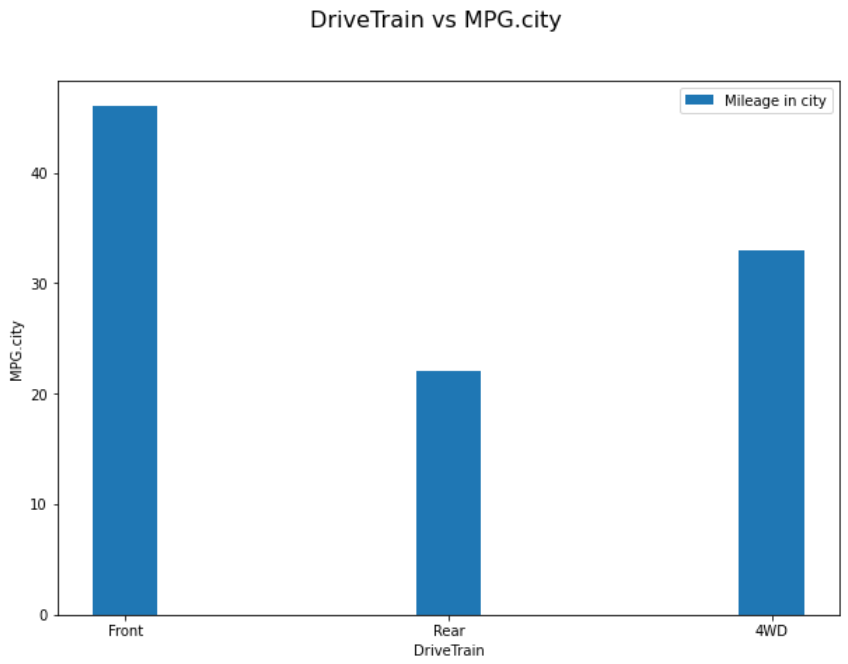
Out[30]: `<matplotlib.legend.Legend at 0x26f992770d0>`



## Visualize and compare Mileage in the city for each type of DriveTrain using a bar chart

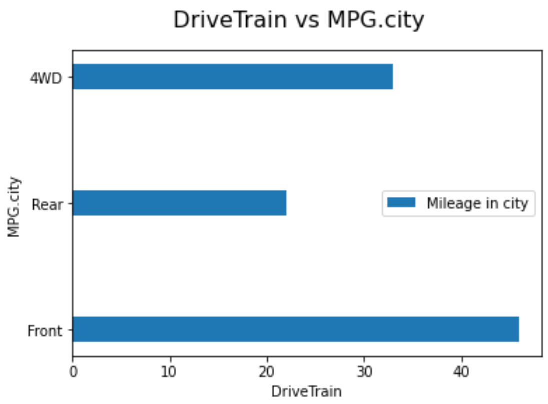
In [31]:
```
fig = plt.figure()
fig.set_figwidth(10)
fig.set_figheight(7)
plt.bar(cars_df["DriveTrain"], cars_df["MPG.city"],width=0.2,label="Mileage in city")
plt.suptitle("DriveTrain vs MPG.city",fontsize=16)
plt.xlabel("DriveTrain")
plt.ylabel("MPG.city")
plt.legend()
```

Out[31]:
```
<matplotlib.legend.Legend at 0x26f991f9040>
```



In [33]:
```
plt.barh(cars_df["DriveTrain"], cars_df["MPG.city"],height=0.2,label="Mileage in city")
plt.suptitle("DriveTrain vs MPG.city",fontsize=16)
plt.xlabel("DriveTrain")
plt.ylabel("MPG.city")
plt.legend()
```

Out[33]:
```
<matplotlib.legend.Legend at 0x26f9947f520>
```




## Visualize the relationship between "No of Passengers" for each "type of car" using a stacked bar chart

In [34]:
```
#Use the following code snippet to filter the unique values of no. of passengers a car can carry
cars_df["Passengers"].unique()
```

Out[34]:
```
array([5, 6, 4, 7, 8, 2], dtype=int64)
```

In [35]:
```
#Use the following code snippet to filter the unique values of Types of car.
cars_df["Type"].unique()
```

Out[35]:
```
array(['Small', 'Midsize', 'Compact', 'Large', 'Sporty', 'Van'],
      dtype=object)
```

In [38]:
```
grouped_data = cars_df[["Passengers","Type"]].groupby(by= ["Passengers","Type"]).size()
grouped_data
```

Out[38]:
```
Passengers  Type
2           Sporty      2
4           Compact     1
            Midsize     2
            Small       8
            Sporty     12
5           Compact    13
            Midsize    15
            Small      13
6           Compact     2
            Large      11
            Midsize     5
7           Van         8
8           Van         1
dtype: int64
```

In [39]:
```
grouped_data = cars_df[["Passengers","Type"]].groupby(by= ["Passengers","Type"]).size().unstack()
grouped_data
```

Out[39]:

| Type | Compact | Large | Midsize | Small | Sporty | Van |
|---|---|---|---|---|---|---|
| **Passengers** | | | | | | |
| **2** | NaN | NaN | NaN | NaN | 2.0 | NaN |
| **4** | 1.0 | NaN | 2.0 | 8.0 | 12.0 | NaN |
| **5** | 13.0 | NaN | 15.0 | 13.0 | NaN | NaN |
| **6** | 2.0 | 11.0 | 5.0 | NaN | NaN | NaN |
| **7** | NaN | NaN | NaN | NaN | NaN | 8.0 |
| **8** | NaN | NaN | NaN | NaN | NaN | 1.0 |

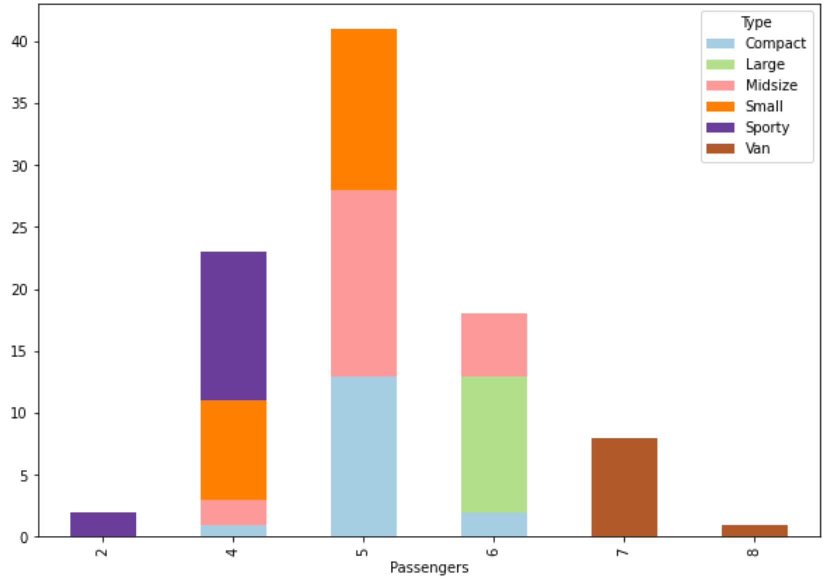
In [40]:
```
#combining the above 2 steps
grouped_data = cars_df[["Passengers","Type"]].groupby(by= ["Passengers","Type"]).size().unstack().reset_index()
grouped_data
```

Out[40]:

| Type | Passengers | Compact | Large | Midsize | Small | Sporty | Van |
|---|---|---|---|---|---|---|---|
| **0** | 2 | NaN | NaN | NaN | NaN | 2.0 | NaN |
| **1** | 4 | 1.0 | NaN | 2.0 | 8.0 | 12.0 | NaN |
| **2** | 5 | 13.0 | NaN | 15.0 | 13.0 | NaN | NaN |
| **3** | 6 | 2.0 | 11.0 | 5.0 | NaN | NaN | NaN |
| **4** | 7 | NaN | NaN | NaN | NaN | NaN | 8.0 |
| **5** | 8 | NaN | NaN | NaN | NaN | NaN | 1.0 |

In [43]:
```
#Stacked Bar Graph can be plotted using the grouped data, as follows:
grouped_data.plot(x="Passengers",kind="bar",stacked=True,colormap=cm.Paired,figsize=(10,7))
#Matplotlib has built-in colormaps. Here, 'Paired' is used.
```

Out[43]: <AxesSubplot:xlabel='Passengers'>




## mtcars.csv dataset

In [4]:
```
mtcar_df = pd.read_csv("mtcars.csv")
mtcar_df
```

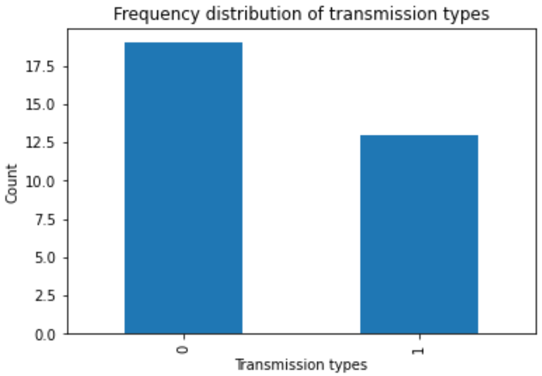
Out[4]:

| | model | mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Mazda RX4 | 21.0 | 6 | 160.0 | 110 | 3.90 | 2.620 | 16.46 | 0 | 1 | 4 | 4 |
| **1** | Mazda RX4 Wag | 21.0 | 6 | 160.0 | 110 | 3.90 | 2.875 | 17.02 | 0 | 1 | 4 | 4 |
| **2** | Datsun 710 | 22.8 | 4 | 108.0 | 93 | 3.85 | 2.320 | 18.61 | 1 | 1 | 4 | 1 |
| **3** | Hornet 4 Drive | 21.4 | 6 | 258.0 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | 3 | 1 |
| **4** | Hornet Sportabout | 18.7 | 8 | 360.0 | 175 | 3.15 | 3.440 | 17.02 | 0 | 0 | 3 | 2 |
| **5** | Valiant | 18.1 | 6 | 225.0 | 105 | 2.76 | 3.460 | 20.22 | 1 | 0 | 3 | 1 |
| **6** | Duster 360 | 14.3 | 8 | 360.0 | 245 | 3.21 | 3.570 | 15.84 | 0 | 0 | 3 | 4 |
| **7** | Merc 240D | 24.4 | 4 | 146.7 | 62 | 3.69 | 3.190 | 20.00 | 1 | 0 | 4 | 2 |
| **8** | Merc 230 | 22.8 | 4 | 140.8 | 95 | 3.92 | 3.150 | 22.90 | 1 | 0 | 4 | 2 |
| **9** | Merc 280 | 19.2 | 6 | 167.6 | 123 | 3.92 | 3.440 | 18.30 | 1 | 0 | 4 | 4 |
| **10** | Merc 280C | 17.8 | 6 | 167.6 | 123 | 3.92 | 3.440 | 18.90 | 1 | 0 | 4 | 4 |
| **11** | Merc 450SE | 16.4 | 8 | 275.8 | 180 | 3.07 | 4.070 | 17.40 | 0 | 0 | 3 | 3 |
| **12** | Merc 450SL | 17.3 | 8 | 275.8 | 180 | 3.07 | 3.730 | 17.60 | 0 | 0 | 3 | 3 |
| **13** | Merc 450SLC | 15.2 | 8 | 275.8 | 180 | 3.07 | 3.780 | 18.00 | 0 | 0 | 3 | 3 |
| **14** | Cadillac Fleetwood | 10.4 | 8 | 472.0 | 205 | 2.93 | 5.250 | 17.98 | 0 | 0 | 3 | 4 |
| **15** | Lincoln Continental | 10.4 | 8 | 460.0 | 215 | 3.00 | 5.424 | 17.82 | 0 | 0 | 3 | 4 |
| **16** | Chrysler Imperial | 14.7 | 8 | 440.0 | 230 | 3.23 | 5.345 | 17.42 | 0 | 0 | 3 | 4 |
| **17** | Fiat 128 | 32.4 | 4 | 78.7 | 66 | 4.08 | 2.200 | 19.47 | 1 | 1 | 4 | 1 |
| **18** | Honda Civic | 30.4 | 4 | 75.7 | 52 | 4.93 | 1.615 | 18.52 | 1 | 1 | 4 | 2 |
| **19** | Toyota Corolla | 33.9 | 4 | 71.1 | 65 | 4.22 | 1.835 | 19.90 | 1 | 1 | 4 | 1 |
| **20** | Toyota Corona | 21.5 | 4 | 120.1 | 97 | 3.70 | 2.465 | 20.01 | 1 | 0 | 3 | 1 |
| **21** | Dodge Challenger | 15.5 | 8 | 318.0 | 150 | 2.76 | 3.520 | 16.87 | 0 | 0 | 3 | 2 |
| **22** | AMC Javelin | 15.2 | 8 | 304.0 | 150 | 3.15 | 3.435 | 17.30 | 0 | 0 | 3 | 2 |
| **23** | Camaro Z28 | 13.3 | 8 | 350.0 | 245 | 3.73 | 3.840 | 15.41 | 0 | 0 | 3 | 4 |
| **24** | Pontiac Firebird | 19.2 | 8 | 400.0 | 175 | 3.08 | 3.845 | 17.05 | 0 | 0 | 3 | 2 |
| **25** | Fiat X1-9 | 27.3 | 4 | 79.0 | 66 | 4.08 | 1.935 | 18.90 | 1 | 1 | 4 | 1 |
| **26** | Porsche 914-2 | 26.0 | 4 | 120.3 | 91 | 4.43 | 2.140 | 16.70 | 0 | 1 | 5 | 2 |
| **27** | Lotus Europa | 30.4 | 4 | 95.1 | 113 | 3.77 | 1.513 | 16.90 | 1 | 1 | 5 | 2 |
| **28** | Ford Pantera L | 15.8 | 8 | 351.0 | 264 | 4.22 | 3.170 | 14.50 | 0 | 1 | 5 | 4 |
| **29** | Ferrari Dino | 19.7 | 6 | 145.0 | 175 | 3.62 | 2.770 | 15.50 | 0 | 1 | 5 | 6 |
| **30** | Maserati Bora | 15.0 | 8 | 301.0 | 335 | 3.54 | 3.570 | 14.60 | 0 | 1 | 5 | 8 |
| **31** | Volvo 142E | 21.4 | 4 | 121.0 | 109 | 4.11 | 2.780 | 18.60 | 1 | 1 | 4 | 2 |

In [16]:
```
temp= mtcar_df['am'].value_counts()
print(temp)
temp.plot.bar()
plt.title("Frequency distribution of transmission types")
plt.xlabel("Transmission types")
plt.ylabel("Count")
```

```
0    19
1    13
Name: am, dtype: int64
```

Out[16]:
```
Text(0, 0.5, 'Count')
```




In [ ]: