Basics of NumPy

- NumPy - Introduction and Installation
- NumPy - Arrays Data Structure (1D, 2D, ND arrays)
- Creating Arrays
- NumPy - Data Types
- Array Attributes
- Creating Arrays – Alternative Ways
- Sub-setting, Slicing and Indexing Arrays
- Operations on Arrays
- Array Manipulation

NumPy – Introduction and Installation

- NumPy stands for 'Numeric Python'
- Used for mathematical and scientific computations
- NumPy array is the most widely used object of the NumPy library

Installing numpy

!pip install numpy

Importing numpy

In [2]:
```
import numpy as np
```

Arrays Data Structure

An `Array` is combination of homogenous data objects and can be indexed across multiple dimensions

Arrays are –

- ordered sequence/collection of Homogenous data
- multidimensional
- mutable

Creating Arrays – From list/tuple

- `np.array()` is used to create a numpy array from a list

Example on 1-D Array

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

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

Example on 2-D Array

In [4]:
```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
arr
```

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

Array Attributes

- Attributes are the features/characteristics of an object that describes the object
- Some of the attributes of the numpy array are:
  - **shape** - Array dimensions
  - **size** - Number of array elements
  - **dtype** - Data type of array elements
  - **ndim** - Number of array dimensions
  - **dtype.name** - Name of data type
  - **astype** - Convert an array to a different type

In [5]:
```
arr = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
arr
```

Out[5]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [6]:
```
arr.shape
```

Out[6]:
```
(3, 4)
```

In [7]:
```
arr.size
```

Out[7]:
```
12
```

In [8]:
```
arr.ndim
```

Out[8]:
```
2
```

In [9]:
```
arr.dtype
```

Out[9]:
```
dtype('int64')
```

In [10]:
```
arr.astype(float)
```

Out[10]:
```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 9., 10., 11., 12.]])
```

Indexing, Slicing and Boolen Indexing

In [11]:
```
arr = np.random.randint(5, 50, size = 10)
arr
```

Out[11]:
```
array([35, 18, 43, 47, 11, 18, 25, 10, 40, 32], dtype=int32)
```

1-D Arrays

Indexing

- this concept is same for str, list, tuples, arrays

In [12]:
```
arr[0]  # - first element
```

Out[12]:
```
np.int32(35)
```

In [14]:
```
arr[3]  # - fourth element
```

Out[14]:
```
np.int32(47)
```

In [15]:
```
type(arr[3])
```

Out[15]:
```
numpy.int32
```

In [16]:
```
type(32)
```

Out[16]:
```
int
```

In [17]:
```
int(arr[3])
```

Out[17]:
```
47
```

In [19]:
```
arr[-1]  # last element
```

Out[19]:
```
np.int32(32)
```

Slicing

- this concept is same for str, list, tuples, arrays

Ex. Extract first 3 elements

In [21]:
```
arr[0:3]
```

Out[21]:
```
array([35, 18, 43], dtype=int32)
```

Ex. Extract elements from postion 3 to the end if the array

In [22]:
```
arr[3 : ]  # Keep end point as empty if you want to extract till the last element
```

Out[22]:
```
array([47, 11, 18, 25, 10, 40, 32], dtype=int32)
```

Ex. Extract last 5 elements

In [23]:
```
arr[-5 : ]
```

Out[23]:
```
array([18, 25, 10, 40, 32], dtype=int32)
```

Conditional or boolean slicing/indexing - Filtering the arrays

- applicable only to arrays

Ex. Extract elements at index position 2, 5, 9.

In [25]:
```
arr[[9, 2, 5, ]]
```

Out[25]:
```
array([32, 43, 18], dtype=int32)
```

Ex. Extract elements less than 20

In [27]:
```
arr < 20  # returns a boolean array
```

Out[27]:
```
array([False,  True, False, False,  True,  True, False,  True, False,
       False])
```

In [29]:
```
arr[arr < 20] # Extracts the values where condition is True
```

Out[29]:
```
array([18, 11, 18, 10], dtype=int32)
```

In [33]:
```
# Sum of numbers in the array
np.sum(arr)
```

Out[33]:
```
np.int64(279)
```

In [34]:
```
# Count the number of values greater than 20 -
np.sum(arr < 20) # arr < 20 generate a bool array, where True - 1 and False - 0 - s
```

Out[34]:
```
np.int64(4)
```

In [35]:
```
# Are there any numbers less than 20
np.sum(arr < 20) >= 1
```

Out[35]:
```
np.True_
```

In [37]:
```
# Are there any numbers less than 20
np.any(arr < 20) # checks if any 1 value in the bool array is True
```

Out[37]:
```
np.True_
```

2-D Arrays

In [38]:
```
arr = np.random.randint(5, 50, size = (6,4))
arr
```

Out[38]:
```
array([[28, 19, 24, 14],
       [28, 43, 37, 20],
       [40, 37, 33, 17],
       [19, 11,  5, 20],
       [11, 11, 16, 13],
       [11, 47,  9,  9]], dtype=int32)
```

Ex. Extract first 3 rows

In [39]:
```
arr[0:3]
```

Out[39]:
```
array([[28, 19, 24, 14],
       [28, 43, 37, 20],
       [40, 37, 33, 17]], dtype=int32)
```

Ex. Extract last 2 rows

In [40]:
```
arr[-2:]
```

Out[40]:
```
array([[11, 11, 16, 13],
       [11, 47,  9,  9]], dtype=int32)
```

Ex. Extract second column - count wise

In [42]:
```
arr[:, 1]  # Extracting single row or col from 2-D arrays will always return output
```

Out[42]:
```
array([19, 43, 37, 11, 11, 47], dtype=int32)
```

Ex. Extract row 2 and 3 and column 2 and 3

In [44]:
```
arr[1:3, 1:3]
```

Out[44]:
```
array([[43, 37],
       [37, 33]], dtype=int32)
```

Ex. Extract values less than 25

In [46]:
```
arr[arr < 25]
```

Out[46]:
```
array([19, 24, 14, 20, 17, 19, 11,  5, 20, 11, 11, 16, 13, 11,  9,  9],
      dtype=int32)
```

Ex. Identify largest value. Extract values less than half of largest values

In [47]:
```
arr[arr < np.max(arr)/2]
```

Out[47]:
```
array([19, 14, 20, 17, 19, 11,  5, 20, 11, 11, 16, 13, 11,  9,  9],
      dtype=int32)
```

Array Operations

Arithmetic operations on Arrays -

- Addition, Substraction, Multiplication, Division, etc.
- Operations on array and a scalar value
- Operations between two arrays
- Matrix Operations - Multiplication(np.dot()), Transpose(np.transpose())

Array and Scalar

In [48]:
```
arr1 = np.random.randint(1,10,size = 5)
arr1
```

Out[48]:
```
array([7, 6, 5, 3, 3], dtype=int32)
```

In [49]:
```
arr1 + 5 # Addition
```

Out[49]:
```
array([12, 11, 10,  8,  8], dtype=int32)
```

In [50]:
```
arr1 - 5 # Substraction
```

Out[50]:
```
array([ 2,  1,  0, -2, -2], dtype=int32)
```

In [51]:
```
arr1 * 5 # Multiplication
```

Out[51]:
```
array([35, 30, 25, 15, 15], dtype=int32)
```

In [52]:
```
arr1 / 5 # Division
```

Out[52]:
```
array([1.4, 1.2, 1. , 0.6, 0.6])
```

In [53]:
```
arr1 // 5 # Floor Division
```

Out[53]:
```
array([1, 1, 1, 0, 0], dtype=int32)
```

In [54]:
```
arr1 % 5 # Modulus
```

Out[54]:
```
array([2, 1, 0, 3, 3], dtype=int32)
```

Two Arrays

In [55]:
```
arr1 = np.random.randint(1,10,size = 5)
arr1
```

Out[55]:
```
array([1, 5, 6, 9, 8], dtype=int32)
```

In [56]:
```
arr2 = np.random.randint(1,10,size = 5)
arr2
```

Out[56]:
```
array([2, 8, 8, 4, 3], dtype=int32)
```

In [57]:
```
arr1 + arr2 # Addition
```

Out[57]:
```
array([ 3, 13, 14, 13, 11], dtype=int32)
```

In [58]:
```
arr1 - arr2 # Substraction
```

Out[58]:
```
array([-1, -3, -2,  5,  5], dtype=int32)
```

In [59]:
```
arr1 * arr2 # Multiplication
```

Out[59]:
```
array([ 2, 40, 48, 36, 24], dtype=int32)
```

In [60]:
```
arr1 / arr2 # Division
```

Out[60]:
```
array([0.5       , 0.625     , 0.75      , 2.25      , 2.66666667])
```

In [61]:
```
arr1 // arr2 # Floor Division
```

Out[61]:
```
array([0, 0, 0, 2, 2], dtype=int32)
```

In [62]:
```
arr1 % arr2 # Modulus
```

Out[62]:
```
array([1, 5, 6, 1, 2], dtype=int32)
```

Relational operations on Arrays -

- ==, !=, <, >, <=, >=
- Operations on array and a scalar value
- Operations between two arrays

Array and Scalar

In [63]:
```
arr1 = np.random.randint(1,10,size = 5)
arr1
```

Out[63]:
```
array([7, 5, 4, 3, 6], dtype=int32)
```

In [64]:
```
arr1 == 5
```

Out[64]:
```
array([False,  True, False, False, False])
```

In [65]:
```
arr1 != 5
```

Out[65]:
```
array([ True, False,  True,  True,  True])
```

In [66]:
```
arr1 < 5
```

Out[66]:
```
array([False, False,  True,  True, False])
```

In [67]:
```
arr1 > 5
```

Out[67]:
```
array([ True, False, False, False,  True])
```

In [68]:
```
arr1 <= 5
```

Out[68]:
```
array([False,  True,  True,  True, False])
```

In [69]:
```
arr1 >= 5
```

Out[69]:
```
array([ True,  True, False, False,  True])
```

Two Arrays

In [70]:
```
arr1 = np.random.randint(1,10,size = 5)
arr1
```

Out[70]:
```
array([5, 1, 8, 8, 2], dtype=int32)
```

In [71]:
```
arr2 = np.random.randint(1,10,size = 5)
arr2
```

Out[71]:
```
array([6, 6, 6, 8, 4], dtype=int32)
```

In [72]:
```
arr1 == arr2
```

Out[72]:
```
array([False, False, False,  True, False])
```

In [73]:
```
arr1 != arr2
```

Out[73]:
```
array([ True,  True,  True, False,  True])
```

In [74]:
```
arr1 < arr2
```

Out[74]:
```
array([ True,  True, False, False,  True])
```

In [75]:
```
arr1 > arr2
```

Out[75]:
```
array([False, False,  True, False, False])
```

In [76]:
```
arr1 <= arr2
```

Out[76]:
```
array([ True,  True, False,  True,  True])
```

In [77]:
```
arr1 >= arr2
```

Out[77]:
```
array([False, False,  True,  True, False])
```

Logical operations on Arrays -

- np.logical_or()

- np.logical_and()
- np.logical_not()
- np.logical_xor()

In [78]:
```
arr1 = np.random.randint(1,10,size = 5)
arr1
```

Out[78]:
```
array([6, 6, 6, 9, 8], dtype=int32)
```

In [79]:
```
arr2 = np.random.randint(1,10,size = 5)
arr2
```

Out[79]:
```
array([3, 2, 7, 3, 5], dtype=int32)
```

In [82]:
```
np.logical_and(arr1 > 5, arr2 > 5)
```

Out[82]:
```
array([False, False,  True, False, False])
```

In [83]:
```
np.logical_or(arr1 > 5, arr2 > 5)
```

Out[83]:
```
array([ True,  True,  True,  True,  True])
```

In [84]:
```
np.logical_not(arr1 > 5)
```

Out[84]:
```
array([False, False, False, False, False])
```

In [85]:
```
np.logical_xor(arr1 > 5, arr2 > 5)
```

Out[85]:
```
array([ True,  True, False,  True,  True])
```

Set Operations on Arrays

Applicable to 1-D Ararys only

- np.unique() - Find the unique elements of an array.
- np.in1d() - Test whether each element of a 1-D array is also present in a second array.
- np.intersect1d() - Find the intersection of two arrays.
- np.setdiff1d() - Find the set difference of two arrays.
- np.union1d() - Find the union of two arrays.

In [87]:
```
arr1 = np.random.randint(1,10,size = 10)
arr1
```

Out[87]:
```
array([8, 7, 2, 3, 7, 4, 7, 7, 3, 4], dtype=int32)
```

In [89]:
```
np.unique(arr1)
```

Out[89]:
```
array([2, 3, 4, 7, 8], dtype=int32)
```

In [91]:
```
np.unique(arr1, return_counts=True, return_index=True)
```

Out[91]:
```
(array([2, 3, 4, 7, 8], dtype=int32),
 array([2, 3, 5, 1, 0]),
 array([1, 2, 2, 4, 1]))
```

In [92]:
```
arr1 = np.random.randint(1,10,size = 10)
arr1
```

Out[92]:
```
array([1, 3, 6, 3, 2, 8, 4, 1, 5, 6], dtype=int32)
```

In [93]:
```
arr2 = np.random.randint(1,10,size = 10)
arr2
```

Out[93]:
```
array([5, 3, 3, 2, 3, 4, 5, 3, 7, 2], dtype=int32)
```

In [94]:
```
np.intersect1d(arr1, arr2) # common elements in 2 arrays
```

Out[94]:
```
array([2, 3, 4, 5], dtype=int32)
```

In [96]:
```
np.union1d(arr1, arr2)
```

Out[96]:
```
array([1, 2, 3, 4, 5, 6, 7, 8], dtype=int32)
```

In [98]:
```
np.setdiff1d(arr1, arr2) # elements of only arr1 removing common
```

Out[98]:
```
array([1, 6, 8], dtype=int32)
```

In [99]:
```
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([1, 3, 5, 7, 9])

# Checks if elements of array1 are present in array 2
np.in1d(arr1, arr2)
```

```
C:\Users\vaide\AppData\Local\Temp\ipykernel_18052\2558066030.py:5: DeprecationWarnin
g: `in1d` is deprecated. Use `np.isin` instead.
  np.in1d(arr1, arr2)
```

Out[99]:
```
array([ True, False,  True, False,  True])
```

In [100…
```
np.isin(arr1, arr2)
```

Out[100…
```
array([ True, False,  True, False,  True])
```

In [101…
```
dict(zip(arr1, np.isin(arr1, arr2)))
```

Out[101…
```
{np.int64(1): np.True_,
 np.int64(2): np.False_,
 np.int64(3): np.True_,
 np.int64(4): np.False_,
 np.int64(5): np.True_}
```

Array Functions/Methods

- np.all(), np.any()

- arr.sum()
- arr.min(), arr.max(), arr.argmin(), arr.agrmax()
- np.round()
- np.mean(), np.median(), np.average(), np.percentile()

In [102…
```
arr2 = np.array([1, 3, 5, 7, 9])
```

In [103…
```
np.max(arr2)
```

Out[103…
```
np.int64(9)
```

In [104…
```
arr2.max()
```

Out[104…
```
np.int64(9)
```

In [107…
```
arr2.argmax() # the index position of largest element
```

Out[107…
```
np.int64(4)
```

Array Manipulations

- **Changing Shape** – np.reshape()
- **Adding/Removing Elements** – np.append(), np.insert(), np.delete()
- **Splitting Arrays** – np.hsplit(), np.vsplit(), arr_obj.flatten()
- **Sorting Arrays** - arr_obj.sort(), arr_obj.argsort()

In [109…
```
arr = np.arange(1, 13)  # generates sequence of numbers
arr
```

Out[109…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

np.reshape()

In [110…
```
arr = np.reshape(arr, (4, 3))
arr
```

Out[110…
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

np.append()

In [111…
```
np.append(arr, 10) # Flattens the array
```

Out[111…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 10])
```

In [112…
```
np.append(arr, [13, 14, 15]) # Flattens the array
```

Out[112…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

In [113…
```
np.append(arr, np.reshape(np.array([13, 14, 15]), (1, 3)), axis=0) # axis = 0 adds
```

Out[113…
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [13, 14, 15]])
```

In [114…
```
np.append(arr, [[10],[20],[30],[40]], axis=1) # axis = 1 adds a 2-D array column-wi
```

Out[114…
```
array([[ 1,  2,  3, 10],
       [ 4,  5,  6, 20],
       [ 7,  8,  9, 30],
       [10, 11, 12, 40]])
```

np.insert()

In [116…
```
arr = np.reshape(np.arange(1,13), (4,3))
arr
```

Out[116…
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [117…
```
np.insert(arr, 1, 5) # Flattens the arr and inserts 5 at index 1
```

Out[117…
```
array([ 1,  5,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [118…
```
np.insert(arr, 1, 5, axis=0) # Inserts [5, 5, 5] as row 1
```

Out[118…
```
array([[ 1,  2,  3],
       [ 5,  5,  5],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [119…
```
np.insert(arr, 1, 5, axis=1) # Inserts [5, 5, 5, 5] as column 1
```

Out[119…
```
array([[ 1,  5,  2,  3],
       [ 4,  5,  5,  6],
       [ 7,  5,  8,  9],
       [10,  5, 11, 12]])
```

In [120…
```
np.insert(arr, 1, [10, 20, 30, 40], axis=1)
```

Out[120…
```
array([[ 1, 10,  2,  3],
       [ 4, 20,  5,  6],
       [ 7, 30,  8,  9],
       [10, 40, 11, 12]])
```

In [121…
```
np.insert(arr, 1, [10, 20, 30], axis=0)
```

Out[121…
```
array([[ 1,  2,  3],
       [10, 20, 30],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

np.delete()

In [122…
```
arr = np.reshape(np.arange(1,13), (4,3))
arr
```

Out[122…
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [123…
```
np.delete(arr, 1) # Flattens the arr and deletes element at index 1
```

Out[123…
```
array([ 1,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [125…
```
np.delete(arr, 1, axis=0) # deletes row 1
```

Out[125…
```
array([[ 1,  2,  3],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [126…
```
np.delete(arr, 1, axis=1) # deletes column 1
```

Out[126…
```
array([[ 1,  3],
       [ 4,  6],
       [ 7,  9],
       [10, 12]])
```

In [127…
```
np.delete(arr,[0,2], axis=0) # deletes selected rows
```

Out[127…
```
array([[ 4,  5,  6],
       [10, 11, 12]])
```

In [128…
```
np.delete(arr,[0,2], axis=1) # deletes selected columns
```

Out[128…
```
array([[ 2],
       [ 5],
       [ 8],
       [11]])
```

Note - All the 3 functions generate a new array

np.hsplit(), np.vsplit()

In [129…
```
arr = np.reshape(np.arange(1, 25), (6,4))
arr
```

Out[129…
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])
```

In [130…
```
np.vsplit(arr, 2)
```

Out[130…
```
[array([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]]),
 array([[13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]])]
```

In [131…
```
np.hsplit(arr, 2)
```

Out[131…
```
[array([[ 1,  2],
        [ 5,  6],
        [ 9, 10],
        [13, 14],
        [17, 18],
        [21, 22]]),
 array([[ 3,  4],
        [ 7,  8],
        [11, 12],
        [15, 16],
        [19, 20],
        [23, 24]])]
```

flatten()

In [132…
```
arr
```

Out[132…
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])
```

In [134…
```
arr.flatten()
```

Out[134…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24])
```

In [133…
```
arr.flatten(order = "F") # returns a 1-D array
```

Out[133…
```
array([ 1,  5,  9, 13, 17, 21,  2,  6, 10, 14, 18, 22,  3,  7, 11, 15, 19,
       23,  4,  8, 12, 16, 20, 24])
```

In [ ]:
```
help(arr.flatten)
```

Sorting Arrays

In [146…
```
products = np.array(["p1", "p2", "p3", "p4"])
prices = np.array([200, 400, 300, 100])

np.sort(prices) # Generates a new array
```

Out[146…
```
array([100, 200, 300, 400])
```

In [138…
```
np.sort(prices)[::-1]  # DESC Sort
```

Out[138…
```
array([400, 300, 200, 100])
```

In [149…
```
np.argsort(prices)  # returns the index position of the elements in sorted order of
```

Out[149…
```
array([3, 0, 2, 1])
```

In [147…
```
prices[np.argsort(prices)]
```

Out[147…
```
array([100, 200, 300, 400])
```

In [150…
```
products[np.argsort(prices)] # Sort the products by prices
```

Out[150…
```
array(['p4', 'p1', 'p3', 'p2'], dtype='<U2')
```

Examples on Coffee Shop Data Set

In [14]:
```
import numpy as np
products = np.array(['Caffe Latte', 'Cappuccino', 'Colombian', 'Darjeeling', 'Decaf
sales = np.array([52248.0, 14068.0, 71060.0, 60014.0, 69925.0, 27711.0, 19231.0, 24
profits = np.array([17444.0, 5041.0, 28390.0, 20459.0, 23432.0, 7691.0, -2954.0, 71
target_profits = np.array([15934.0, 4924.0, 31814.0, 19649.0, 24934.0, 8461.0, 7090
target_sales = np.array([48909.0, 13070.0, 80916.0, 57368.0, 66906.0, 30402.0, 1821
```

Ex. How many products are there in the dataset?

In [152…
```
len(products)
```

Out[152…
```
10
```

In [153…
```
products.size
```

Out[153…
```
10
```

Ex. Which product had the highest sales?

In [156…
```
products[np.argmax(sales)]  # String object using indexing
```

Out[156…
```
np.str_('Colombian')
```

In [157…
```
products[np.max(sales) == sales]  # array object - using filtering/boolean slicing
```

Out[157…
```
array(['Colombian'], dtype='<U17')
```

Ex. Which products were in loss?

In [158…
```
products[profits < 0]
```

Out[158…
```
array(['Green Tea'], dtype='<U17')
```

Ex. Which products had profit margins (profit/sales) greater than 30%?

In [162…
```
products[profits/sales > 0.30]
```

Out[162…
```
array(['Caffe Latte', 'Cappuccino', 'Colombian', 'Darjeeling',
       'Decaf Irish Cream', 'Mint', 'Regular Espresso'], dtype='<U17')
```

Ex. Find products with low sales but high profits (sales < median, profit > median)

In [163…
```
np.sort(sales)
```

Out[163…
```
array([14068., 19231., 24873., 27711., 32825., 44109., 52248., 60014.,
       69925., 71060.])
```

In [164…
```
(32825 + 44109)/2
```

Out[164…
```
38467.0
```

In [165…
```
np.median(sales)  # Central value - 50% of products have sales above 38k and 50% h
```

Out[165…
```
np.float64(38467.0)
```

In [166…
```
sales < np.median(sales)
```

Out[166…
```
array([False,  True, False, False, False,  True,  True,  True,  True,
       False])
```

In [167…
```
profits > np.median(profits)
```

Out[167…
```
array([ True, False,  True,  True,  True, False, False, False, False,
        True])
```

In [168…
```
np.logical_and((sales < np.median(sales)), (profits > np.median(profits)))
```

Out[168…
```
array([False, False, False, False, False, False, False, False, False,
       False])
```

In [169…
```
products[np.logical_and((sales < np.median(sales)), (profits > np.median(profits)))]
```

Out[169…
```
array([], dtype='<U17')
```

Ex. Generate new values of Sales after applying 18% Tax

In [170…
```
sales * 1.18
```

Out[170…
```
array([61652.64, 16600.24, 83850.8 , 70816.52, 82511.5 , 32698.98,
       22692.58, 29350.14, 38733.5 , 52048.62])
```

Ex. Top 3 most profitable products?

In [173…
```
products[np.argsort(profits)[::-1]][0:3]
```

Out[173…
```
array(['Colombian', 'Decaf Irish Cream', 'Darjeeling'], dtype='<U17')
```

Ex. Which products exceeded their sales targets?

In [174…
```
products[sales > target_sales]
```

Out[174…
```
array(['Caffe Latte', 'Cappuccino', 'Darjeeling', 'Decaf Irish Cream',
       'Green Tea', 'Lemon', 'Mint', 'Regular Espresso'], dtype='<U17')
```

Ex. How many products met or exceeded profit targets?

In [176…
```
products[profits >= target_profits].size
```

Out[176…
```
5
```

In [178…
```
np.sum(profits >= target_profits)
```

Out[178…
```
np.int64(5)
```

Ex. Check if all sales achievers also achieveing their profit targets? - Yes/No. If No then display the names of the products which are not achieveing profits.

HINT - use if-else

In [181…
```
sales_ach = products[sales > target_sales]
profit_ach = products[profits >= target_profits]
if np.all(np.isin(sales_ach, profit_ach)) :
    print("Yes")
else:
    print("No.", np.setdiff1d(sales_ach, profit_ach))
```

```
No. ['Decaf Irish Cream' 'Green Tea' 'Regular Espresso']
```

Ex. What is the average sales target achievement rate?

In [182…
```
np.mean(sales/target_sales)
```

Out[182…
```
np.float64(1.0480011291864888)
```

Ex. Find Products Falling Short of Profit Targets and Sort by Achievement Percentage

1. Filter products that did not meet their profit targets
2. Calculate the percentage of target profit achieved for each
3. Display these products in descending order based on their achievement percentage

In [183…
```
off_products = products[target_profits > profits]
off_profits = profits[target_profits > profits]
off_targets = target_profits[target_profits > profits]
```

In [186…
```
percentage = np.round(off_profits / off_targets * 100, 2)
percentage
```

Out[186…
```
array([ 89.24,  93.98,  90.9 , -41.66,  91.39])
```

In [188…
```
# replace negative values with 0
percentage[percentage < 0 ] = 0
percentage
```

Out[188…
```
array([89.24, 93.98, 90.9 ,  0.  , 91.39])
```

In [189…
```
for prod, perct in zip(off_products, percentage):
    print(f"{prod} is achieveing {perct} % of its total profits.")
```

```
Colombian is achieveing 89.24 % of its total profits.
Decaf Irish Cream is achieveing 93.98 % of its total profits.
Earl Grey is achieveing 90.9 % of its total profits.
Green Tea is achieveing 0.0 % of its total profits.
Regular Espresso is achieveing 91.39 % of its total profits.
```

Data Visualisation

Libraries -

1. matplotlib.pyplot - base
2. pandas - plot()
3. seaborn
4. plotly

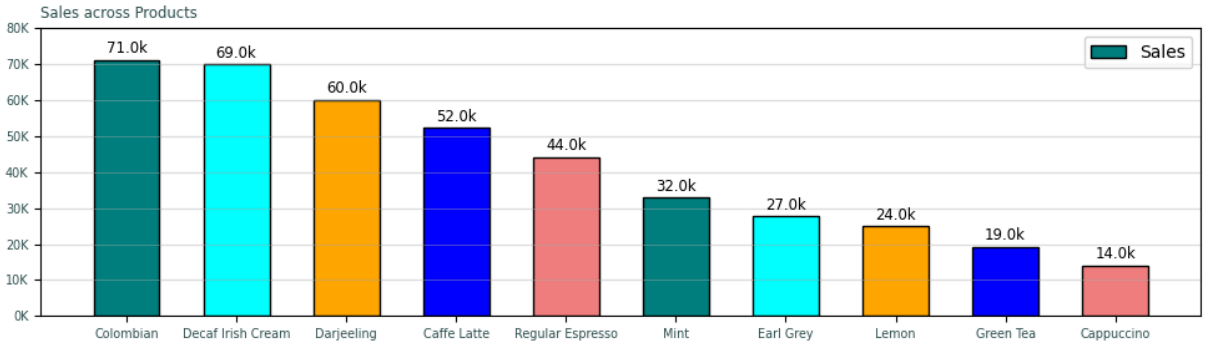
In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
products = np.array(['Caffe Latte', 'Cappuccino', 'Colombian', 'Darjeeling', 'Decaf
sales = np.array([52248.0, 14068.0, 71060.0, 60014.0, 69925.0, 27711.0, 19231.0, 24
profits = np.array([17444.0, 5041.0, 28390.0, 20459.0, 23432.0, 7691.0, -2954.0, 71
```

Bar Chart

In [6]:
```
plt.figure(figsize = (12, 3))
sort_order = np.argsort(sales)[::-1]
plt.bar(products[sort_order], sales[sort_order], color = ["teal", "cyan", "orange",

plt.title("Sales across Products", color = "darkslategrey", loc = "left", size = "s

plt.xticks(color = "darkslategrey", size = "x-small") #, rotation = 20)

y_ticks = np.arange(0, np.max(sales) + 10000, 10000)
y_labels = (y_ticks // 1000).astype(int).astype(str) + "K"
plt.yticks(y_ticks, y_labels, color = "darkslategrey", size = "x-small")

plt.grid(axis = "y", alpha = 0.4)
```

```
plt.legend()

for x, y in zip(products, sales) :
    plt.annotate(f"{round(y//1000, 1)}k", xy = (x, y+2000), ha = "center", size = "

plt.show()
```

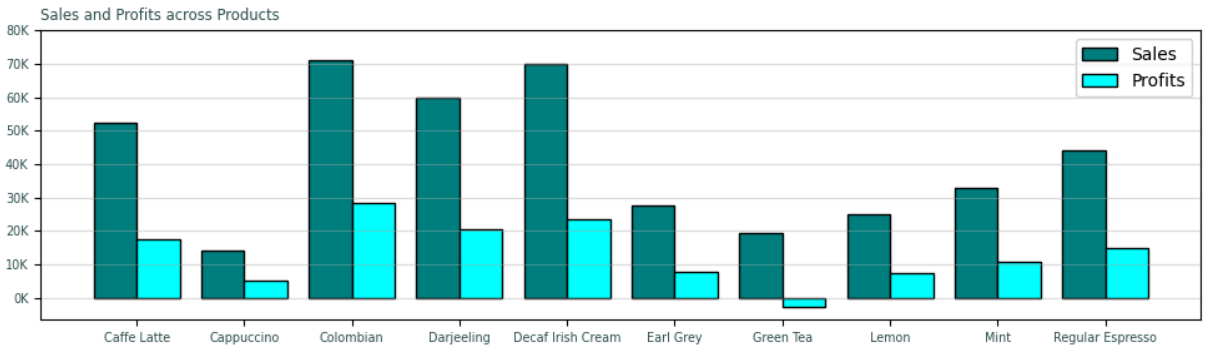


Homework -

1. Sort the bars in ASC/DESC order of Sales
2. Apply different color to each bar

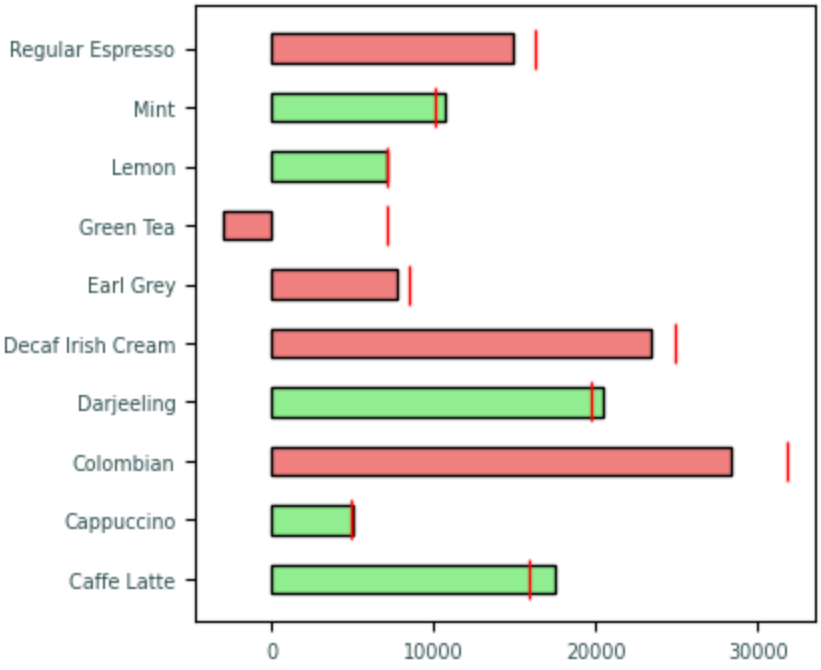
Side-by-Side Bar Chart

In [13]:
```
plt.figure(figsize = (12, 3))
plt.bar(products, sales, color = "teal", width=-0.4, edgecolor = "black", label = "
plt.bar(products, profits, color = "cyan", width=0.4, edgecolor = "black", label =

plt.title("Sales and Profits across Products", color = "darkslategrey", loc = "left

plt.xticks(color = "darkslategrey", size = "x-small") #, rotation = 20)

y_ticks = np.arange(0, np.max(sales) + 10000, 10000)
y_labels = (y_ticks // 1000).astype(int).astype(str) + "K"
plt.yticks(y_ticks, y_labels, color = "darkslategrey", size = "x-small")

plt.grid(axis = "y", alpha = 0.4)
plt.legend()

plt.show()
```

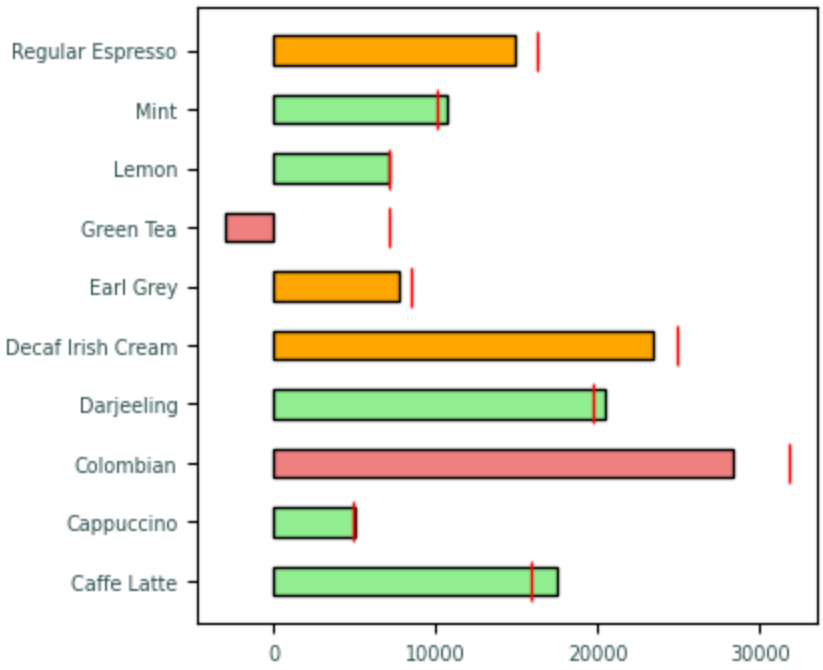


Bullet Chart

In [33]:
```
plt.figure(figsize = (4, 4))
colors_list = np.where(profits >= target_profits, "lightgreen", "lightcoral")
plt.barh(products, profits, color = colors_list, edgecolor = "black", height = 0.5)
plt.plot(target_profits, products, ls = "", marker = '|', color = "red", markersize

plt.xticks(color = "darkslategrey", size = "x-small")
plt.yticks(color = "darkslategrey", size = "x-small")

plt.show()
```



In [40]:
```
plt.figure(figsize = (4, 4))

ratio = profits/target_profits
condition_list = [ratio >= 1, ratio >= 0.9]
results_list = ["lightgreen", "orange"]

colors_list = np.select(condition_list, results_list, default="lightcoral")

plt.barh(products, profits, color = colors_list, edgecolor = "black", height = 0.5)

plt.plot(target_profits, products, ls = "", marker = '|', color = "red", markersize

plt.xticks(color = "darkslategrey", size = "x-small")
plt.yticks(color = "darkslategrey", size = "x-small")

plt.show()
```



np.where(`condition` , `True Part` , `False Part`)

In [32]:
```
np.where(profits >= target_profits, "lightgreen", "lightcoral")
```

Out[32]:
```
array(['lightgreen', 'lightgreen', 'lightcoral', 'lightgreen',
       'lightcoral', 'lightcoral', 'lightcoral', 'lightgreen',
       'lightgreen', 'lightcoral'], dtype='<U10')
```

np.select(`conditions_list` , `results_list` , `default=0`)

In [39]:
```
ratio = profits/target_profits
condition_list = [ratio >= 1, ratio >= 0.9]
results_list = ["lightgreen", "orange"]
np.select(condition_list, results_list, default="lightcoral")
```

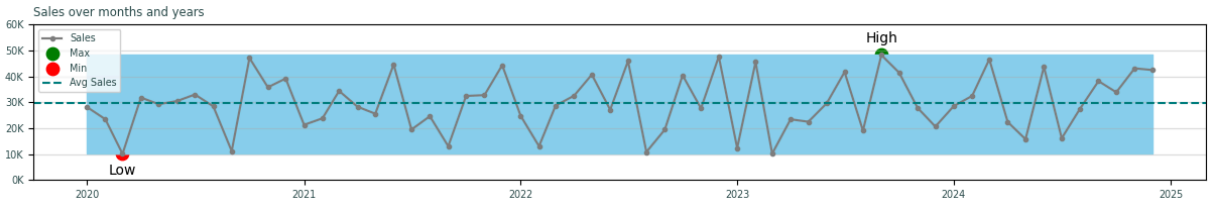
Out[39]:
```
array(['lightgreen', 'lightgreen', 'lightcoral', 'lightgreen', 'orange',
       'orange', 'lightcoral', 'lightgreen', 'lightgreen', 'orange'],
      dtype='<U10')
```

Line Chart

In [ ]:
```
date = np.arange('2020-01', '2025-01', dtype='datetime64[M]')
sales = np.random.randint(10000, 50000, 60)
```

In [97]:
```
plt.figure(figsize = (15, 2))
plt.plot(date, sales, label = "Sales", color = "grey", marker = "o", ms = 3)

max_index = sales.argmax()
min_index = sales.argmin()

# plt.scatter(date[[max_index, min_index]], sales[[max_index, min_index]], color =
plt.scatter(date[max_index], sales[max_index], color = "green", s = 80, label = "Ma
plt.scatter(date[min_index], sales[min_index], color = "red", s = 80, label = "Min"
```

```
plt.annotate(f"High", xy = (date[max_index], sales[max_index]+5000), ha = "center")
plt.annotate(f"Low", xy = (date[min_index], sales[min_index]-8000), ha = "center")

avg_sales = np.mean(sales)
plt.axhline(avg_sales, color = "teal", label = "Avg Sales", ls = "--")
plt.fill_between(date, sales[max_index], sales[min_index], color = "skyblue")

plt.title("Sales over months and years", color = "darkslategrey", loc = "left", siz

plt.xticks(color = "darkslategrey", size = "x-small") #, rotation = 20)

y_ticks = np.arange(0, np.max(sales) + 20000, 10000)
y_labels = (y_ticks // 1000).astype(int).astype(str) + "K"
plt.yticks(y_ticks, y_labels, color = "darkslategrey", size = "x-small")

plt.grid(axis = "y", alpha = 0.4)
plt.legend(labelcolor = "darkslategrey", fontsize = "x-small")

plt.show()
```

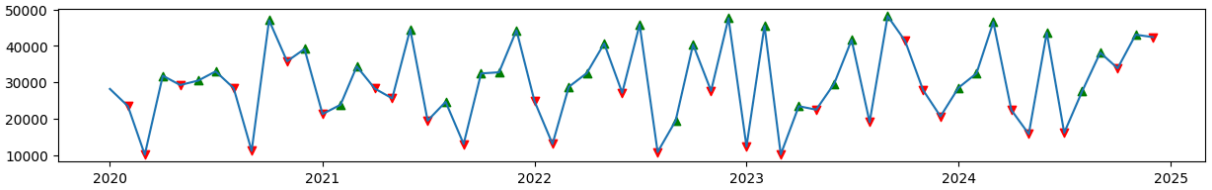


In [127…

```
negative_index = np.diff(sales) < 0
positive_index = np.diff(sales) > 0

plt.figure(figsize = (15, 2))
plt.plot(date, sales)
plt.scatter(date[1:][negative_index], sales[1:][negative_index], color = "red", mar
plt.scatter(date[1:][positive_index], sales[1:][positive_index], color = "green", m
```

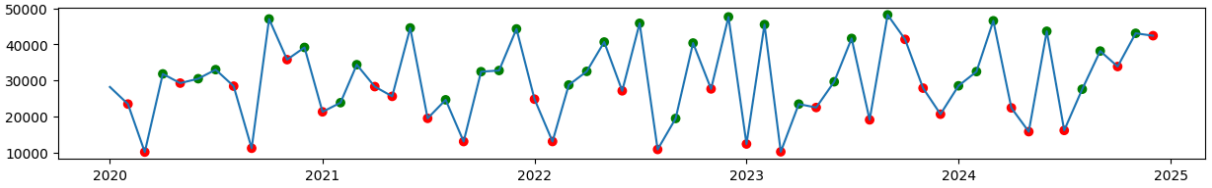
Out[127…

```
<matplotlib.collections.PathCollection at 0x1358a7c6690>
```



In [121…

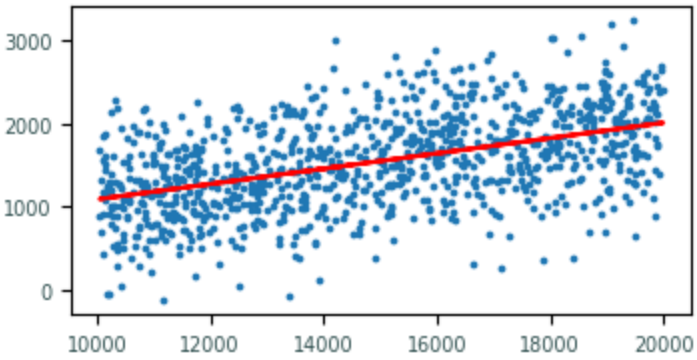
```
plt.figure(figsize = (15, 2))
plt.plot(date, sales)
plt.scatter(date[1:], sales[1:], color = np.where(np.diff(sales) < 0, "red", "green
plt.show()
```



Scatter Chart

In [146…

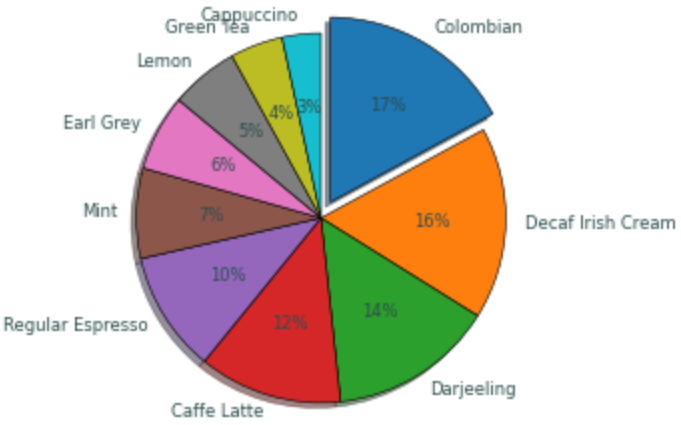
```
sales = np. array([13745, 19507, 17320, 15987, 11560, 11560, 10581, 18662, 16011, 1
profit = np.array([1463, 1283, 1922, 1904, 1436, 1696, 1475, 2096, 1566, 878, 1235,

plt.figure(figsize = (4, 2))
plt.scatter(sales, profit, s = 3)

m, c = np.polyfit(sales, profit, 1)
y = m * sales + c
plt.plot(sales, y, color = "red")
plt.xticks(color = "darkslategrey", size = "x-small")
plt.yticks(color = "darkslategrey", size = "x-small")
plt.show()
```



Pie Chart

In [168…

```
products = np.array(['Caffe Latte', 'Cappuccino', 'Colombian', 'Darjeeling', 'Decaf
sales = np.array([52248.0, 14068.0, 71060.0, 60014.0, 69925.0, 27711.0, 19231.0, 24
sort_ord = np.argsort(sales)[::-1]

plt.figure(figsize = (3, 3))
plt.pie(sales[sort_ord], labels=products[sort_ord], startangle=90, counterclock=Fal
       autopct="%1d%%", textprops={"fontsize" : 6, "color" : "darkslategrey"},
        shadow = True, wedgeprops= {"edgecolor" : "black", "linewidth" : 0.4},
        explode= np.insert(np.zeros(9), 0, 0.1)
       )
plt.show()
```



In [167…
```
np.insert(np.zeros(9), 0, 0.1)
```

Out[167…
```
array([0.1, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ])
```

DataFrame

A **DataFrame** is a **2-dimensional, labeled data structure** in Python, used via the **pandas** library.

📘 Definition:

> A **DataFrame** is like a table (similar to an Excel spreadsheet or a SQL table) that consists of rows and columns, where each column can be of a different data type (e.g., integers, floats, strings).

✅ Key Characteristics:

- **Rows and columns**: Indexed by labels (default is integers).
- **Heterogeneous data**: Each column can store different types of data.
- **Powerful operations**: Sorting, filtering, aggregation, merging, and reshaping.

📊 Common Use Cases:

- Loading and analyzing data from CSV, Excel, SQL, JSON, etc.
- Performing data cleaning and transformation.
- Statistical analysis and visualization.
- Feature engineering for machine learning.

In [170…
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Connecting to an Excel or CSV ot TXT file

Option - 1

In [171…
```
file_path = r"C:\Users\vaide\Desktop\Virtua Search\Vituare-Research\Datasets\Filena
pd.read_excel(file_path, sheet_name="Data")
```

Out[171…

	Name	Age	Gender
0	Jack	34	M
1	Jane	35	F
2	Rosie	36	F

Option - 2

Set current working directory

In [173…

```
import os
os.chdir(r"C:\Users\vaide\Desktop\Virtua Search\Vituare-Research\Datasets")
```

In [174…

```
pd.read_excel("Filename.xlsx", sheet_name="Data")
```

Out[174…

	Name	Age	Gender
0	Jack	34	M
1	Jane	35	F
2	Rosie	36	F

In [175…

```
pd.read_csv("BSE Sensex 30 Historical Data.csv")
```

Out[175...

	Date	Price	Open	High	Low	Vol.	Change %
0	11-04-2025	75,157.26	74,835.49	75,467.33	74,762.84	14.23M	1.77%
1	09-04-2025	73,847.15	74,103.83	74,103.83	73,673.06	9.15M	-0.51%
2	08-04-2025	74,227.08	74,013.73	74,859.39	73,424.92	17.06M	1.49%
3	07-04-2025	73,137.90	71,449.94	73,403.99	71,425.01	29.37M	-2.95%
4	04-04-2025	75,364.69	76,160.09	76,258.12	75,240.55	11.68M	-1.22%
5	03-04-2025	76,295.36	75,811.86	76,493.74	75,807.55	6.92M	-0.42%
6	02-04-2025	76,617.44	76,146.28	76,680.35	76,064.94	10.75M	0.78%
7	01-04-2025	76,024.51	76,882.58	77,487.05	75,912.18	10.59M	-1.80%
8	28-03-2025	77,414.92	77,690.69	77,766.70	77,185.62	16.70M	-0.25%
9	27-03-2025	77,606.43	77,087.39	77,747.46	77,082.51	12.67M	0.41%
10	26-03-2025	77,288.50	78,021.45	78,167.87	77,194.22	10.48M	-0.93%
11	25-03-2025	78,017.19	78,296.28	78,741.69	77,745.63	12.67M	0.04%
12	24-03-2025	77,984.38	77,456.27	78,107.23	77,179.35	10.10M	1.40%
13	21-03-2025	76,905.51	76,155.00	77,041.94	76,095.26	8.52M	0.73%
14	20-03-2025	76,348.06	75,917.11	76,456.25	75,684.58	9.06M	1.19%
15	19-03-2025	75,449.05	75,473.17	75,568.38	75,201.48	12.19M	0.20%
16	18-03-2025	75,301.26	74,608.66	75,385.76	74,480.15	14.52M	1.53%
17	17-03-2025	74,169.95	73,830.03	74,376.35	73,796.06	7.37M	0.46%

Ex. Read data from customer.txt file

In [225...

```
df = pd.read_csv("customers.txt", header=None)
df
```

Out[225…

	0	1	2	3	4
0	4000001	Kristina	Chung	55	Pilot
1	4000002	Paige	Chen	74	Teacher
2	4000003	Sherri	Melton	34	Firefighter
3	4000004	Gretchen	Hill	66	Computer hardware engineer
4	4000005	Karen	Puckett	74	Lawyer
...	...	...	...	...	...
9994	4009995	Rebecca	Dennis	37	Teacher
9995	4009996	Tonya	McIntosh	56	Engineering technician
9996	4009997	Ron	Grimes	36	Computer hardware engineer
9997	4009998	Tracey	Bullock	60	Computer hardware engineer
9998	4009999	Ray	Hewitt	64	Carpenter

9999 rows × 5 columns

DataFrame Attributes

In [181…
```
df.shape
```

Out[181…
```
(9999, 5)
```

In [182…
```
df.dtypes
```

Out[182…
```
0     int64
1    object
2    object
3     int64
4    object
dtype: object
```

In [184…
```
df.head(2)
```

Out[184…

	0	1	2	3	4
0	4000001	Kristina	Chung	55	Pilot
1	4000002	Paige	Chen	74	Teacher

In [185…
```
df.tail()
```

Out[185...

	0	1	2	3	4
9994	4009995	Rebecca	Dennis	37	Teacher
9995	4009996	Tonya	McIntosh	56	Engineering technician
9996	4009997	Ron	Grimes	36	Computer hardware engineer
9997	4009998	Tracey	Bullock	60	Computer hardware engineer
9998	4009999	Ray	Hewitt	64	Carpenter

In [226...

```
df.columns = ["Customer ID", "First Name", "Last Name", "Age", "Profession"]
df.head(2)
```

Out[226...

	Customer ID	First Name	Last Name	Age	Profession
0	4000001	Kristina	Chung	55	Pilot
1	4000002	Paige	Chen	74	Teacher

Ex. Add a new column - Name by concataneting FName and LName

In [228...

```
# Option 1 - add the new column to the end of the df
df["Full Name"] = df["First Name"] + " " + df["Last Name"]
df.head(2)
```

Out[228...

	Customer ID	Name	First Name	Last Name	Age	Profession	Full Name
0	4000001	Kristina Chung	Kristina	Chung	55	Pilot	Kristina Chung
1	4000002	Paige Chen	Paige	Chen	74	Teacher	Paige Chen

In [227...

```
# Option 2 - add the new column to the at index position 1 of the df
df.insert(1, "Name", df["First Name"] + " " + df["Last Name"])
df.head(2)
```

Out[227...

	Customer ID	Name	First Name	Last Name	Age	Profession
0	4000001	Kristina Chung	Kristina	Chung	55	Pilot
1	4000002	Paige Chen	Paige	Chen	74	Teacher

Ex. Remove unwanted columns from the dataframe

In [229...

```
df.drop(columns=["First Name", "Last Name", "Full Name"], inplace= True)
```

In [196...

```
df.head(2)
```

Out[196...

	Customer ID	Name	Age	Profession
0	4000001	Kristina Chung	55	Pilot
1	4000002	Paige Chen	74	Teacher

Filtering DataFrames

Ex. Extract data for all the Pilots

In [ ]:
```
df[df["Profession"] == "Pilot"]
```

Ex. Extract data for customers whose age is less than 30

In [ ]:
```
df[df["Age"] < 30]
```

Ex. Extract data for customers whose age is in between 30 to 50 yrs

In [ ]:
```
df[df["Age"].between(30, 50)]
```

In [ ]:
```
df[~ df["Age"].between(30, 50)]
```

Ex. Extract names of customers whose name starts with K

In [ ]:
```
df[df["Name"].str.startswith("K")]
```

Indexing and slicing on Dataframes

- loc
- iloc

Ex. Change profession of all the customers whose age is > 60 to Retired

In [208...
```
seniors = df[df["Age"] > 60]
```

In [209...
```
seniors["Profession"] = "Retired"
```

```
C:\Users\vaide\AppData\Local\Temp\ipykernel_16172\3776724415.py:1: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  seniors["Profession"] = "Retired"
```

In [211...
```
seniors
```

Out[211…

	Customer ID	Name	Age	Profession
1	4000002	Paige Chen	74	Retired
3	4000004	Gretchen Hill	66	Retired
4	4000005	Karen Puckett	74	Retired
7	4000008	Hazel Bender	63	Retired
13	4000014	Beth Woodard	65	Retired
...	...	...	...	...
9975	4009976	Joan Dolan	70	Retired
9980	4009981	Clarence Berry	64	Retired
9984	4009985	Rachel Corbett	66	Retired
9992	4009993	Becky Wolfe	67	Retired
9998	4009999	Ray Hewitt	64	Retired

2840 rows × 4 columns

In [212…
```
df.loc[0, "Profession"]  # Label wise
```

Out[212…
```
'Pilot'
```

In [214…
```
df.iloc[0, 3]  # index-wise
```

Out[214…
```
'Pilot'
```

In [213…
```
df.loc[0:5, ["Age", "Profession"]]
```

Out[213…

	Age	Profession
0	55	Pilot
1	74	Teacher
2	34	Firefighter
3	66	Computer hardware engineer
4	74	Lawyer
5	42	Veterinarian

In [215…
```
df.iloc[0:6, [2, 3]]
```

Out[215…

	Age	Profession
0	55	Pilot
1	74	Teacher
2	34	Firefighter
3	66	Computer hardware engineer
4	74	Lawyer
5	42	Veterinarian

In [230…

```
# to avoid the warning -
seniors = df.loc[df["Age"] > 60]
seniors.loc[:,"Profession"] = "Retired"
```

In [231…

```
seniors
```

Out[231…

	Customer ID	Name	Age	Profession
1	4000002	Paige Chen	74	Retired
3	4000004	Gretchen Hill	66	Retired
4	4000005	Karen Puckett	74	Retired
7	4000008	Hazel Bender	63	Retired
13	4000014	Beth Woodard	65	Retired
…	…	…	…	…
9975	4009976	Joan Dolan	70	Retired
9980	4009981	Clarence Berry	64	Retired
9984	4009985	Rachel Corbett	66	Retired
9992	4009993	Becky Wolfe	67	Retired
9998	4009999	Ray Hewitt	64	Retired

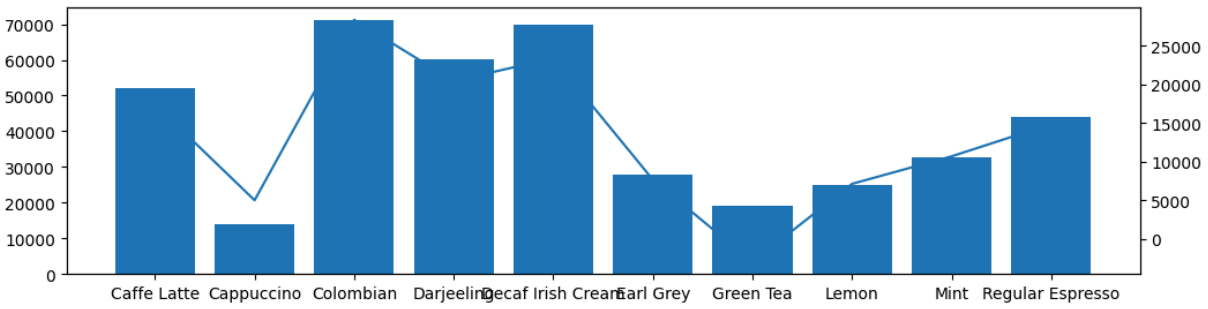
2840 rows × 4 columns

In [243…

```
fig, ax1 = plt.subplots(figsize  = (12, 3))

ax1.bar(products, sales)
ax2 = ax1.twinx()
ax2.plot(products, profits)
```

Out[243…

```
[<matplotlib.lines.Line2D at 0x13591740830>]
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: