Lab 1: Basic Python Programming

CPE232 Data Models

WISIT SUWANNAO 67070501042

[1] Variable

1.1 Number Variable

In [1]:
```
num = 100 #integer variable
num2 = 12.5 #float variable
print(num)
print(num2)

print(num + num2)   #addition
print(num - num2)   #subtraction
print(num * num2)   #multiplication
print( num / num2)  #division
```

```
100
12.5
112.5
87.5
1250.0
8.0
```

1.2 String Variable

In [2]:
```
string = "Data Models"
print(string) #print complete string
```

```
print("Hello " + string)    #print concatenated string

# slicing string
print(string[0])            #print first character of the string
print(string[:4])           #print first to 4th character of the string
print(string[5:])           #print 6th to last character of the string
print(string[1:4])          #print 2nd to 4th character of the string
print(string * 2)           #print string 2 times
```

```
Data Models
Hello Data Models
D
Data
Models
ata
Data ModelsData Models
```

In [3]:
```
# format string
code = "CPE232"

print(f"{code}: Data Models")
```

```
CPE232: Data Models
```

1.3 Boolean Variable

In [4]:
```
#boolean variable
boolean = True
boolean2 = False

print(boolean)              #print boolean variable
print(not boolean)          #print opposite of boolean variable
print(boolean and boolean2) #print boolean and boolean2
print(boolean or boolean2)  #print boolean or boolean2
```

```
True
False
False
True
```

1.4 List Variable

In [5]:
```
#list variable
list = ["Data",20,123.23,40,50]
another_list = ["Models",60]

print(list)                 #print complete list
print(list[0])              #print first element of the list
print(list[1:3])            #print 2nd to 3rd element of the list
print(list[2:])             #print 3rd to last element of the list
print(another_list)         #print complete another_list
print(another_list * 2)     #print another_list two times
print(list + another_list)  #print concatenated list

list[0] = "CPE232"          #change first element of the list
print(list)                 #print complete list

# add element at the end
list.append(True)
print(list)

# pop element by index
list.pop(1)
print(list)
```

```
['Data', 20, 123.23, 40, 50]
Data
[20, 123.23]
[123.23, 40, 50]
['Models', 60]
['Models', 60, 'Models', 60]
['Data', 20, 123.23, 40, 50, 'Models', 60]
['CPE232', 20, 123.23, 40, 50]
['CPE232', 20, 123.23, 40, 50, True]
['CPE232', 123.23, 40, 50, True]
```

1.5 Tuple Variable

In [6]:
```
#tuple variable
tuple = ("Data",20,123.23,40,50)
another_tuple = ("Models",60)

print(tuple)                    #print complete tuple
print(tuple[0])                 #print first element of the tuple
print(tuple[1:3])               #print 2nd to 3rd element of the tuple
print(tuple[2:])                #print 3rd to last element of the tuple
print(tuple * 2)                #print tuple two times
print(tuple + another_tuple)    #print concatenated tuple
```

```
('Data', 20, 123.23, 40, 50)
Data
(20, 123.23)
(123.23, 40, 50)
('Data', 20, 123.23, 40, 50, 'Data', 20, 123.23, 40, 50)
('Data', 20, 123.23, 40, 50, 'Models', 60)
```

In [7]:
```
tuple[0] = "CPE232"          # tuple is immutable dtype, we can't change its value after declared
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-997785654.py in <cell line: 0>()
----> 1 tuple[0] = "CPE232"          # tuple is immutable dtype, we can't change its value after declared

TypeError: 'tuple' object does not support item assignment
```

1.6 Dictionary Variable

In [42]:
```
#dictionary variable
dictionary = {"name":"Alice","age":21}
another_dictionary = {}
another_dictionary["name"] = "Bob"
another_dictionary["age"] = 21

print(dictionary)                   #print complete dictionary
print(dictionary["name"])           #print value for specific key
print(dictionary.keys())            #print all the keys
print(dictionary.values())          #print all the values
```

```
print(dictionary.items())          #print all the items
print(another_dictionary)          #print complete another_dictionary
```

```
{'name': 'Alice', 'age': 21}
Alice
dict_keys(['name', 'age'])
dict_values(['Alice', 21])
dict_items([('name', 'Alice'), ('age', 21)])
{'name': 'Bob', 'age': 21}
```

1.7 Set Variable

In [43]:
```
#set variable
my_set = {"Data",20,123.23,60,"Data"}
another_set = {"Models", 60}

print(my_set)                              #print complete set (duplicates are removed)

#set operations
print(my_set.union(another_set))           # print all unique elements from both sets
print(my_set.intersection(another_set))    # print common elements from both sets

# add element
my_set.add("New Item")
my_set.add(20)                             # try adding a duplicate again
print(my_set)

# remove element
my_set.remove(123.23)
print(my_set)
```

```
{123.23, 'Data', 20, 60}
{'Data', 20, 123.23, 60, 'Models'}
{60}
{'New Item', 'Data', 20, 123.23, 60}
{'New Item', 'Data', 20, 60}
```

[2] Control Flow

2.1 IF ... ELIF ... ELSE

In [44]:
```
number = 123
number2 = 34

if number > number2:
    print("number is greater than number2")
elif number < number2:
    print("number is less than number2")
else:
    print("number is equal to number2")
```

```
number is greater than number2
```

2.2 try ... except

In [45]:
```
number1 = "number"
number2 = 2

try:
    number1 + number2
except:                             #error handling
    print("What are you doing?")
```

```
What are you doing?
```

[3] Loop

3.1 For Loop

In [46]:
```
#for loops
for num in range(0,10):
    print(num)
```

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

In [47]:
```python
#for loops with step
for num in range(5,15,2):
    print(num)
```

```
5
7
9
11
13
```

In [48]:
```python
#for loop with enumerate

list = ["Alice","Bob","Charlie","Daisy"]

for index, name in enumerate(list):
    print(f"{index}: {name}")
```

```
0: Alice
1: Bob
2: Charlie
3: Daisy
```

In [49]:
```python
#for loop with list

for name in list:
    print(name)
```

```
Alice
Bob
Charlie
Daisy
```

In [50]:
```
#continue in for loop

list = [1,23,7,"hello",True,1123,43,23,12]

for element in list:
    if type(element) != int:
        continue
    print(element)
```

```
1
23
7
1123
43
23
12
```

In [51]:
```
#break in for loop

list = [1,23,7,"hello",True,1123,43,23,12]

for element in list:
    if type(element) != int:
        break
    print(element)
```

```
1
23
7
```

3.2 While loop

In [52]:
```
#while loop

list = ["Alice","Bob","Charlie","Daisy"]
count = 0

while count < len(list):
```

```
    print(list[count])
    count += 1
```

```
Alice
Bob
Charlie
Daisy
```

In [53]:
```
#continue in while loop

list = [1,23,7,"hello",True,1123,43,23,12]
count = 0

while count < len(list):
    if type(list[count]) != int:
        count += 1
        continue
    print(list[count])
    count += 1
```

```
1
23
7
1123
43
23
12
```

In [54]:
```
#break in while loop

list = [1,23,7,"hello",True,1123,43,23,12]
count = 0

while count < len(list):
    if type(list[count]) != int:
        break
    print(list[count])
    count += 1
```

```
1
23
7
```

[4] Function

In [55]:
```python
#define function
def function_name (arg1, arg2):
    return arg1 + arg2

#calling function
function_name(1,2)
```

Out[55]: 3

In [56]:
```python
#define function with default argument
def function_with_default_arg(arg1, arg2 = 10, arg3 = 20 , arg4 = 30):
    return arg1 + arg2 + arg3 + arg4

result_1 = function_with_default_arg(1)
result_2 = function_with_default_arg(1,2,5)
result_3 = function_with_default_arg(1,2,5,10)

print(result_1)
print(result_2)
print(result_3)
```

```
61
38
18
```

In [57]:
```python
#multiple arguments
def function_with_multiple_arg(*args):
    print(args)
    print(type(args))
    sum = 0
    for num in args:
        sum += num
```

```
    return sum

function_with_multiple_arg(1,2,3,4,5)
```

```
(1, 2, 3, 4, 5)
<class 'tuple'>
```

Out[57]: 15

In [58]:
```
#lambda function
lambda_function = lambda arg1, arg2: arg1 + arg2

print(lambda_function(1,2))
```

```
3
```

[5] OOP

In [59]:
```
class Car:
    """A simple attempt to represent a car."""

    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_description_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        """Set the odometer reading to the given value."""
        self.odometer_reading = mileage
```

In [60]:
```
my_car = Car("Honda", "Civic", 2016)
my_car.read_odometer()

# update odometer
my_car.update_odometer(50)
my_car.read_odometer()
```

```
This car has 0 miles on it.
This car has 50 miles on it.
```

[6] File Handling

6.1 Text File

In [61]:
```
# write file
with open("test.txt","w") as file:
    file.write("Hello World")

# read file
with open("test.txt","r") as file:
    print(file.read())
```

```
Hello World
```

6.2 CSV File

In [62]:
```
import csv

with open("test.csv", "w", newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["Name","Surname"])
        writer.writerow(["Alice","Johnson"])
        writer.writerow(["Bob","Smith"])
```

In [63]:
```
import csv

with open("test.csv","r") as file:
```

```
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

```
['Name', 'Surname']
['Alice', 'Johnson']
['Bob', 'Smith']
```

6.3 JSON file

In [64]:
```
import json

data = {
    "username": "admin",
    "password": "admin",
    "role": "sleeping",
}

with open('test.json', 'w') as f:
    json.dump(data, f)
```

In [65]:
```
with open("test.json", "r") as f:
    data = json.load(f)

data
```

Out[65]:
```
{'username': 'admin', 'password': 'admin', 'role': 'sleeping'}
```

[7] Libraries

7.1 Numpy

import numpy library

In [66]:
```
import numpy as np
```

ndarray initialization

Construct using python list

In [67]:
```
# 1d ndarray from 1d python list
list_a1=[1,2,3.5]
arr_a1=np.array(list_a1)
arr_a1
```

Out[67]:
```
array([1. , 2. , 3.5])
```

In [68]:
```
# 2d ndarray from 2d python list (list of list)
list_a2=[[1,2],[3,4],[5,6]]
arr_a2=np.array(list_a2)
arr_a2
```

Out[68]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [69]:
```
list_a3=[ [[1,2],[2,3]], [[3,4],[4,5]] ]
arr_a3=np.array(list_a3)
arr_a3
```

Out[69]:
```
array([[[1, 2],
        [2, 3]],

       [[3, 4],
        [4, 5]]])
```

or construct using some numpy classes and functions

In [70]:
```
np.zeros(5)
```

Out[70]:
```
array([0., 0., 0., 0., 0.])
```

In [71]:
```
np.ones((3,4),dtype=float)
```

Out[71]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [72]:
```
np.eye(3, 3)
```

Out[72]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [73]:
```
np.full((4,),999)
```

Out[73]:
```
array([999, 999, 999, 999])
```

In [74]:
```
np.arange(3,10,2)
```

Out[74]:
```
array([3, 5, 7, 9])
```

In [75]:
```
np.linspace(0, 1, 5)
```

Out[75]:
```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

In [76]:
```
np.random.choice(['a','b'], 9)
```

Out[76]:
```
array(['a', 'a', 'b', 'b', 'a', 'a', 'a', 'b', 'a'], dtype='<U1')
```

In [77]:
```
np.random.randn(10)
```

Out[77]:
```
array([-0.74699693,  1.1954838 ,  0.45145429, -0.12712142,  0.87570724,
        1.09639106,  1.78018227, -0.29813011, -0.05197969, -1.53015878])
```

ndarray properties

In [78]:
```
list_a=[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
arr_a=np.array(list_a)
arr_a
```

Out[78]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [79]:
```
print("dimension:", arr_a.ndim)
print("shape:", arr_a.shape)
print("data type:", arr_a.dtype)
print("array's size (elements)", arr_a.size)
```

```
dimension: 2
shape: (3, 4)
data type: int64
array's size (elements) 12
```

Reshaping & Modification

from this original ndarray

In [80]:
```
arr_a
```

Out[80]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to convert into 3D array

In [81]:
```
arr_a.reshape((2,2,3))
```

Out[81]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

sometimes you may resize for same dimension where only known some dimension, insert -1 for unknown len

In [82]:
```
arr_a.reshape((-1,6))
```

Out[82]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
```

Would you like to try this?

In [83]:
```
arr_a.reshape((-1,5))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-3848364736.py in <cell line: 0>()
----> 1 arr_a.reshape((-1,5))

ValueError: cannot reshape array of size 12 into shape (5)
```

[Q1] From the above cell, explain in your own words why it worked or did not work.

Ans: It did not work because `arr_a` contains 12 elements (3 rows * 4 columns). When attempting to reshape it into `(-1, 5)`, it means one dimension should have a size of 5. For a reshape operation to be valid, the total number of elements must remain the same. Since 12 is not perfectly divisible by 5, NumPy cannot create an array with 5 columns using all 12 elements, leading to a `ValueError`.

Next, try to append any value(s) into existing 2darray

In [84]:
```
np.append(arr_a, 13)
```

Out[84]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13])
```

In [85]:
```
np.append(arr_a, arr_a[0])
```

Out[85]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,  1,  2,  3,  4])
```

In [86]:
```
np.append(arr_a,arr_a[0].reshape((1,-1)),axis=0)
```

Out[86]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [ 1,  2,  3,  4]])
```

In [87]:
```
np.append(arr_a,arr_a[:,0].reshape((-1,1)),axis=1)
```

Out[87]:
```
array([[ 1,  2,  3,  4,  1],
       [ 5,  6,  7,  8,  5],
       [ 9, 10, 11, 12,  9]])
```

In [88]:
```
np.concatenate([arr_a,arr_a])
```

Out[88]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [89]:
```
np.concatenate([arr_a,arr_a],axis=1)
```

Out[89]:
```
array([[ 1,  2,  3,  4,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  5,  6,  7,  8],
       [ 9, 10, 11, 12,  9, 10, 11, 12]])
```

indexing & slicing

from this original array again

In [90]:
```
arr_a
```

Out[90]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to access all element at the first row

In [91]:
```
arr_a[1]
```

Out[91]:
```
array([5, 6, 7, 8])
```

then you would like to access the second element from the first row

In [92]:
```
arr_a[1][2]
```

```
arr_a[1,2]
```

Out[92]:
```
np.int64(7)
```

Next, try to access all element start from the 2nd element in the first row

In [93]:
```
arr_a[1,1:]
```

Out[93]:
```
array([6, 7, 8])
```

In [94]:
```
arr_a[:2, 1:]
```

Out[94]:
```
array([[2, 3, 4],
       [6, 7, 8]])
```

sometimes you may specify some row number using list within indexing

In [95]:
```
arr_a[[1,2,1], 1:]
```

Out[95]:
```
array([[ 6,  7,  8],
       [10, 11, 12],
       [ 6,  7,  8]])
```

Boolean slicing

based on this original array

In [96]:
```
arr_a
```

Out[96]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

try to filter all elements which more than 5

In [97]:
```
arr_a>5
```

Out[97]:
```
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True]])
```

Next, try to filter all elements which more than 5 and less than 10

In [98]:
```
(arr_a>5)&(arr_a<10)
```

Out[98]:
```
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True, False, False, False]])
```

Run the cell below and answer a question.

In [99]:
```
arr_a[(arr_a>5)&(arr_a<10)]
```

Out[99]:
```
array([6, 7, 8, 9])
```

[Q2] From the above cell, explain in your own words how the output came about?

Ans: The output `[6, 7, 8, 9]` is a result of boolean indexing. First, `(arr_a > 5)` and `(arr_a < 10)` create two boolean arrays, checking each element against the conditions. Then, `&` performs an element-wise logical AND operation on these boolean arrays, producing a final boolean array where `True` indicates elements that are both greater than 5 and less than 10. Finally, `arr_a[...]` uses this boolean array to select only those elements from `arr_a` that correspond to `True` values, returning them as a 1D array.

Try running the cell below.

In [100…
```
arr_a[(arr_a>5) and (arr_a<10)]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-133424255.py in <cell line: 0>()
----> 1 arr_a[(arr_a>5) and (arr_a<10)]

ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()
```

[Q3] Explain in your own words why the above cell gives an error.

Ans: The error occurs because the Python keyword `and` is used for logical operations between single boolean values (scalars), not directly between NumPy arrays. When `arr_a > 5` returns a boolean array, the `and` operator attempts to evaluate the 'truthiness' of the entire array, which is ambiguous if the array contains more than one element, hence the `ValueError`. NumPy requires the element-wise logical AND operator, which is `&`.

[Q4] And what should be written instead so that the code is error-free?

Ans: To make the code error-free, you should use the element-wise bitwise AND operator `&` instead of the Python logical `and` keyword when performing logical operations on NumPy arrays:

```
arr_a[(arr_a > 5) & (arr_a < 10)]
```

Basic operations

In [102…
```
list_b=[[1,2,3,4],[1,2,3,4],[1,2,3,4]]
arr_b=np.array(list_b)
arr_b
```

Out[102…
```
array([[1, 2, 3, 4],
       [1, 2, 3, 4],
       [1, 2, 3, 4]])
```

This is some operations for only 1 array

In [103…
```
np.sqrt(arr_b)
```

Out[103…
```
array([[1.        , 1.41421356, 1.73205081, 2.        ],
       [1.        , 1.41421356, 1.73205081, 2.        ],
       [1.        , 1.41421356, 1.73205081, 2.        ]])
```

This is some operations for 2 arrays with the same shape

In [104…
```
arr_a-arr_b
```

Out[104…
```
array([[0, 0, 0, 0],
       [4, 4, 4, 4],
       [8, 8, 8, 8]])
```

In [105…
```
np.add(arr_a,arr_b)
```

Out[105…
```
array([[ 2,  4,  6,  8],
       [ 6,  8, 10, 12],
       [10, 12, 14, 16]])
```

Next, try to operate with 1 array and one numeric variable

In [106…
```
arr_a*3
```

Out[106…
```
array([[ 3,  6,  9, 12],
       [15, 18, 21, 24],
       [27, 30, 33, 36]])
```

In [107…
```
1+arr_a**2
```

Out[107…
```
array([[  2,   5,  10,  17],
       [ 26,  37,  50,  65],
       [ 82, 101, 122, 145]])
```

Broadcasting

In [108…
```
arr_c=np.array([1,2,3])
arr_d=np.array([[3],[5],[8]])
```

In [109…
```
arr_c-arr_d
```

Out[109…
```
array([[-2, -1,  0],
       [-4, -3, -2],
       [-7, -6, -5]])
```

Basic aggregations

In [110…
```
arr_a

print(arr_a)
print("sum:", arr_a.sum())
print("mean:", arr_a.mean())
```

```
print("min:", arr_a.min())
print("max:", arr_a.max())
print("standard deviation:", arr_a.std())
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
sum: 78
mean: 6.5
min: 1
max: 12
standard deviation: 3.452052529534663
```

ndarray axis

In [111…
```
# column axis
arr_a.sum(axis=0)
```

Out[111…
```
array([15, 18, 21, 24])
```

In [112…
```
# row axis
arr_a.sum(axis=1)
```

Out[112…
```
array([10, 26, 42])
```

[Q5] Summarize the value of the argument *axis*, what is the value for row-wise summation and column-wise summation, respectively?

Ans: In NumPy, the `axis` argument specifies the dimension along which an operation is performed:

- For column-wise summation (summing down each column), `axis=0` is used.
- For row-wise summation (summing across each row), `axis=1` is used.

7.2 Pandas

Series

In [113…
```
import pandas as pd
import numpy as np
```

In [114…
```
pd.Series(np.random.randn(6))
```

Out[114…

	0
0	1.546388
1	1.450549
2	0.981310
3	-1.715938
4	-0.177299
5	-0.073034

dtype: float64

In [115…
```
pd.Series(np.random.randn(6), index=['a','b','c','d','e','f'])
```

Out[115…

	0
a	-0.837794
b	1.179825
c	-1.595479
d	-0.063188
e	0.540123
f	-0.212163

dtype: float64

Constructing Dataframe

Constructing DataFrame from a dictionary

In [116…
```
d = {'col1':[1,2], 'col2': [3,4]}
```

In [117…
```
df = pd.DataFrame(data=d)
df
```

Out[117…

	col1	col2
0	1	3
1	2	4

In [118…
```
d2 = {'Name':['Joe','Nat','Harry','Sam','Monica'],
      'Age': [20,21,19,20,22]}

df2 = pd.DataFrame(data=d2)
df2
```

Out[118…

	Name	Age
0	Joe	20
1	Nat	21
2	Harry	19
3	Sam	20
4	Monica	22

Constructing DataFrame from a List

In [119…
```
marks_list = [85.10, 77.80, 91.54, 88.78, 60.55]
```

```
df3 = pd.DataFrame(marks_list, columns=['Marks'])
df3
```

Out[119…

	Marks
0	85.10
1	77.80
2	91.54
3	88.78
4	60.55

Creating DataFrame from file

In [120…

```
# Read csv file from path and store to df for create dataframe
df = pd.read_csv('nss15.csv')
df
```

Out[120…

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	33	1	9	1267
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	34	1	1	1439
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	94	1	0	3274
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	35	1	0	611
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	75	1	0	1893
…	…	…	…	…	…	…	…	…	…	…	…	…
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	76	1	1	1864
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	68	85	1	0	1931
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	79	1	0	3250
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	82	1	1	464
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	34	1	9	3273

334839 rows × 12 columns

Viewing DataFrame information

(.shape, .head, .tail, .info, select column, .unique, .describe, select row with .loc and .iloc)

Check simple information

In [121…

```
# Check dimension by .shape
df.shape
```

Out[121… (334839, 12)

In [122…

```
# Display the first 5 rows by default
df.head()
```

Out[122...

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	33	1	9	1267
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	34	1	1	1439
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	94	1	0	3274
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	35	1	0	611
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	75	1	0	1893

In [123...

```
# Display the first 3 rows
df.head(3)
```

Out[123...

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	33	1	9	1267
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	34	1	1	1439
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	94	1	0	3274

In [124...

```
# Display the last 5 rows by default
df.tail()
```

Out[124...

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	76	1	1	1864
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	68	85	1	0	1931
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	79	1	0	3250
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	82	1	1	464
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	34	1	9	3273

In [125…
```
# Overview information of dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 334839 entries, 0 to 334838
Data columns (total 12 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   caseNumber     334839 non-null  int64  
 1   treatmentDate  334839 non-null  object 
 2   statWeight     334839 non-null  float64
 3   stratum        334839 non-null  object 
 4   age            334839 non-null  int64  
 5   sex            334837 non-null  object 
 6   race           205014 non-null  object 
 7   diagnosis      334839 non-null  int64  
 8   bodyPart       334839 non-null  int64  
 9   disposition    334839 non-null  int64  
 10  location       334839 non-null  int64  
 11  product        334839 non-null  int64  
dtypes: float64(1), int64(7), object(4)
memory usage: 30.7+ MB
```

Select column, multiple column, with condition

In [126…
```
df.columns
```

Out[126…
```
Index(['caseNumber', 'treatmentDate', 'statWeight', 'stratum', 'age', 'sex',
       'race', 'diagnosis', 'bodyPart', 'disposition', 'location', 'product'],
      dtype='object')
```

In [127…
```
# column slicing
df['age']
```

Out[127…

	age
0	5
1	36
2	20
3	61
4	88
…	…
334834	7
334835	3
334836	38
334837	38
334838	5

334839 rows × 1 columns

dtype: int64

In [128…

```
df.age
```

Out[128…

	age
0	5
1	36
2	20
3	61
4	88
…	…
334834	7
334835	3
334836	38
334837	38
334838	5

334839 rows × 1 columns

dtype: int64

Viewing the unique value

In [129…
```
df.race.unique()
```

Out[129…
```
array([nan, 'White', 'Other', 'Black', 'Asian', 'American Indian'],
      dtype=object)
```

Null values

In [130…
```
df.isnull().sum()
```

Out[130…

	0
caseNumber	0
treatmentDate	0
statWeight	0
stratum	0
age	0
sex	2
race	129825
diagnosis	0
bodyPart	0
disposition	0
location	0
product	0

dtype: int64

Describe

In [131…

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

Out[131…

	age
count	334839.000000
mean	31.385451
std	26.105098
min	0.000000
25%	10.000000
50%	23.000000
75%	51.000000
max	107.000000

dtype: float64

Slicing dataframe

In [132…

```
# select multiple column
df[['treatmentDate','statWeight','age','sex']]
```

Out[132…

	treatmentDate	statWeight	age	sex
0	7/11/2015	15.7762	5	Male
1	7/6/2015	83.2157	36	Male
2	8/2/2015	74.8813	20	Female
3	6/26/2015	15.7762	61	Male
4	7/4/2015	74.8813	88	Female
...	...	...	...	...
334834	5/31/2015	15.0591	7	Male
334835	7/11/2015	5.6748	3	Female
334836	7/24/2015	15.7762	38	Male
334837	8/8/2015	97.9239	38	Female
334838	6/20/2015	49.2646	5	Female

334839 rows × 4 columns

In [133…

```
#select by condition
df[df['sex'] == 'Male']
```

Out[133…

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	33	1	9	1267
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	34	1	1	1439
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	35	1	0	611
6	150713483	6/8/2015	15.7762	V	25	Male	Black	51	33	4	9	1138
7	150704114	6/14/2015	83.2157	S	53	Male	White	57	30	1	0	5040
…	…	…	…	…	…	…	…	…	…	…	…	…
334824	150607827	5/27/2015	5.6748	C	1	Male	White	71	36	1	1	1807
334825	150600190	5/28/2015	80.8381	S	5	Male	NaN	56	94	1	0	1936
334833	150747217	7/24/2015	83.2157	S	2	Male	NaN	62	75	1	1	1301
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	76	1	1	1864
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	79	1	0	3250

182501 rows × 12 columns

In [134…

```
# select by multiple condition
df[(df['sex'] == 'Male') & (df['age'] > 80)]
```

Out[134…

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
8	150736558	7/16/2015	83.2157	S	98	Male	Black	59	76	1	1	1807
63	150418623	1/12/2015	15.0591	V	97	Male	Other	62	75	4	1	4076
97	150700375	6/28/2015	83.2157	S	85	Male	NaN	59	92	1	0	478
131	150940801	9/14/2015	15.7762	V	96	Male	NaN	62	75	1	5	1807
177	160110774	12/19/2015	85.7374	S	81	Male	White	59	82	1	1	3278
…	…	…	…	…	…	…	…	…	…	…	…	…
334616	160104368	12/30/2015	74.8813	L	86	Male	Other	71	31	4	1	4078
334677	151115099	11/4/2015	16.5650	V	83	Male	NaN	63	82	1	9	3223
334699	150633387	5/29/2015	74.8813	L	84	Male	NaN	53	83	1	0	1842
334701	150515945	4/27/2015	97.9239	M	86	Male	NaN	57	79	1	0	4074
334785	150733286	7/11/2015	15.7762	V	86	Male	White	71	87	4	1	4076

6379 rows × 12 columns

Select row with .iloc

In [135…

```
# row slicing
df.iloc[10:15]
```

Out[135...

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
10	150734952	7/4/2015	15.7762	V	20	Male	Black	59	82	1	1	1894
11	150821622	7/20/2015	83.2157	S	20	Female	White	57	36	1	9	1267
12	150713631	7/4/2015	15.7762	V	11	Male	NaN	60	88	1	0	3274
13	150666343	6/27/2015	15.7762	V	26	Female	White	62	75	1	1	1807
14	150748843	7/16/2015	37.6645	L	33	Male	Asian	53	93	1	1	4057

In [136...

```
# column slicing
df.iloc[:,[0,1,2,3,4]]
```

Out[136...

	caseNumber	treatmentDate	statWeight	stratum	age
0	150733174	7/11/2015	15.7762	V	5
1	150734723	7/6/2015	83.2157	S	36
2	150817487	8/2/2015	74.8813	L	20
3	150717776	6/26/2015	15.7762	V	61
4	150721694	7/4/2015	74.8813	L	88
...	...	...	...	...	...
334834	150739278	5/31/2015	15.0591	V	7
334835	150733393	7/11/2015	5.6748	C	3
334836	150819286	7/24/2015	15.7762	V	38
334837	150823002	8/8/2015	97.9239	M	38
334838	150723074	6/20/2015	49.2646	M	5

334839 rows × 5 columns

Select column and row with .loc

In [137…
```
# select column and low by .loc
df.loc[:6,'treatmentDate':'diagnosis']
```

Out[137…

	treatmentDate	statWeight	stratum	age	sex	race	diagnosis
0	7/11/2015	15.7762	V	5	Male	NaN	57
1	7/6/2015	83.2157	S	36	Male	White	57
2	8/2/2015	74.8813	L	20	Female	NaN	71
3	6/26/2015	15.7762	V	61	Male	NaN	71
4	7/4/2015	74.8813	L	88	Female	Other	62
5	7/2/2015	5.6748	C	1	Female	White	71
6	6/8/2015	15.7762	V	25	Male	Black	51

In [138…
```
# select row by condition
df.loc[df['age']>80, ['treatmentDate', 'age']]
```

Out[138…

	treatmentDate	age
4	7/4/2015	88
8	7/16/2015	98
39	5/3/2015	88
46	4/15/2015	91
63	1/12/2015	97
...	...	...
334701	4/27/2015	86
334784	7/7/2015	82
334785	7/11/2015	86
334815	10/28/2015	85
334819	1/13/2015	85

20422 rows × 2 columns

[Q6] What is the difference between .iloc and .loc?

Ans: `.iloc` is used for integer-location based indexing, meaning you select rows and columns by their integer positions (starting from 0). `.loc` is used for label-based indexing, allowing you to select rows and columns by their labels (names). `.loc` can also take boolean arrays for selection.

Basic aggregations

In [139…

```
# count elements in column via pandas Series method
df['sex'].value_counts()
```

Out[139…

	count
sex	
Male	182501
Female	152336

dtype: int64

In [140…
```
# count elements in column via pandas function
pd.crosstab(df['sex'], columns='N')
```

Out[140…

col_0	N
sex	
Female	152336
Male	182501

In [141…
```
summary_by_sex = df.groupby('sex').agg({
    'age': 'mean',
    'statWeight': 'sum',
    'caseNumber': 'count',
    'stratum': lambda x: x.mode()[0]
}).rename(columns={'caseNumber': 'total_cases'})

print(summary_by_sex)
```

```
              age    statWeight  total_cases stratum
sex
Female  35.585699  6.208773e+06       152336       V
Male    27.879792  6.964776e+06       182501       V
```

In [142…
```
df['age'].apply('mean')
```

Out[142… np.float64(31.38545091820248)

In [143…

```
df['race'].apply(len)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-580029316.py in <cell line: 0>()
----> 1 df['race'].apply(len)

/usr/local/lib/python3.12/dist-packages/pandas/core/series.py in apply(self, func, convert_dtype, args, by_row, **kwargs)
   4922             args=args,
   4923             kwargs=kwargs,
-> 4924         ).apply()
   4925
   4926     def _reindex_indexer(

/usr/local/lib/python3.12/dist-packages/pandas/core/apply.py in apply(self)
   1425
   1426         # self.func is Callable
-> 1427         return self.apply_standard()
   1428
   1429     def agg(self):

/usr/local/lib/python3.12/dist-packages/pandas/core/apply.py in apply_standard(self)
   1505         #  Categorical (GH51645).
   1506         action = "ignore" if isinstance(obj.dtype, CategoricalDtype) else None
-> 1507         mapped = obj._map_values(
   1508             mapper=curried, na_action=action, convert=self.convert_dtype
   1509         )

/usr/local/lib/python3.12/dist-packages/pandas/core/base.py in _map_values(self, mapper, na_action, convert)
    919             return arr.map(mapper, na_action=na_action)
    920
--> 921         return algorithms.map_array(arr, mapper, na_action=na_action, convert=convert)
    922
    923     @final

/usr/local/lib/python3.12/dist-packages/pandas/core/algorithms.py in map_array(arr, mapper, na_action, convert)
   1741     values = arr.astype(object, copy=False)
   1742     if na_action is None:
-> 1743         return lib.map_infer(values, mapper, convert=convert)
   1744     else:
   1745         return lib.map_infer_mask(
```

```
lib.pyx in pandas._libs.lib.map_infer()

TypeError: object of type 'float' has no len()
```

[Q7] In the above cell, Explain why pandas .agg()/.apply() method accepts 'len' and 'lambda' without quotation marks.

Ans: In pandas' `.agg()` and `.apply()` methods, `len` and `lambda` functions are passed as actual function objects, not as strings. `len` refers to the built-in Python function, and `lambda` defines an anonymous function. The error in `df['race'].apply(len)` occurs because the `race` column contains `NaN` values (which are floats), and the `len()` function cannot be applied to a float type.

7.3 Matplotlib

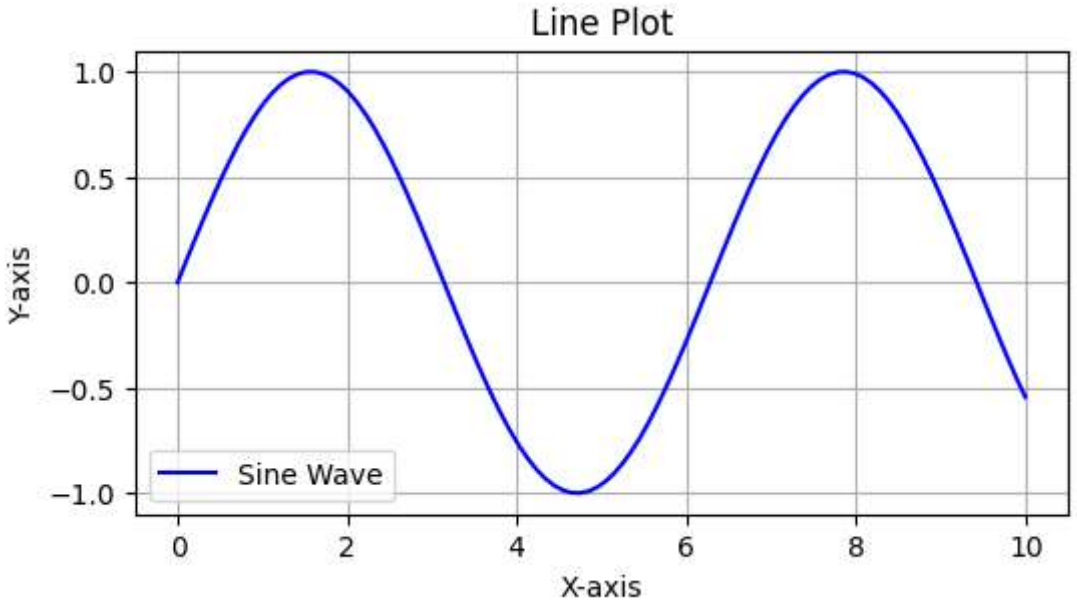
In [144…
```
import matplotlib.pyplot as plt
import numpy as np
```

In [145…
```
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Line plot

plt.figure(figsize=(6, 3))
plt.plot(x, y, label='Sine Wave', color='blue')
plt.title('Line Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.grid(True)

plt.savefig("sine.png")
plt.show()
```



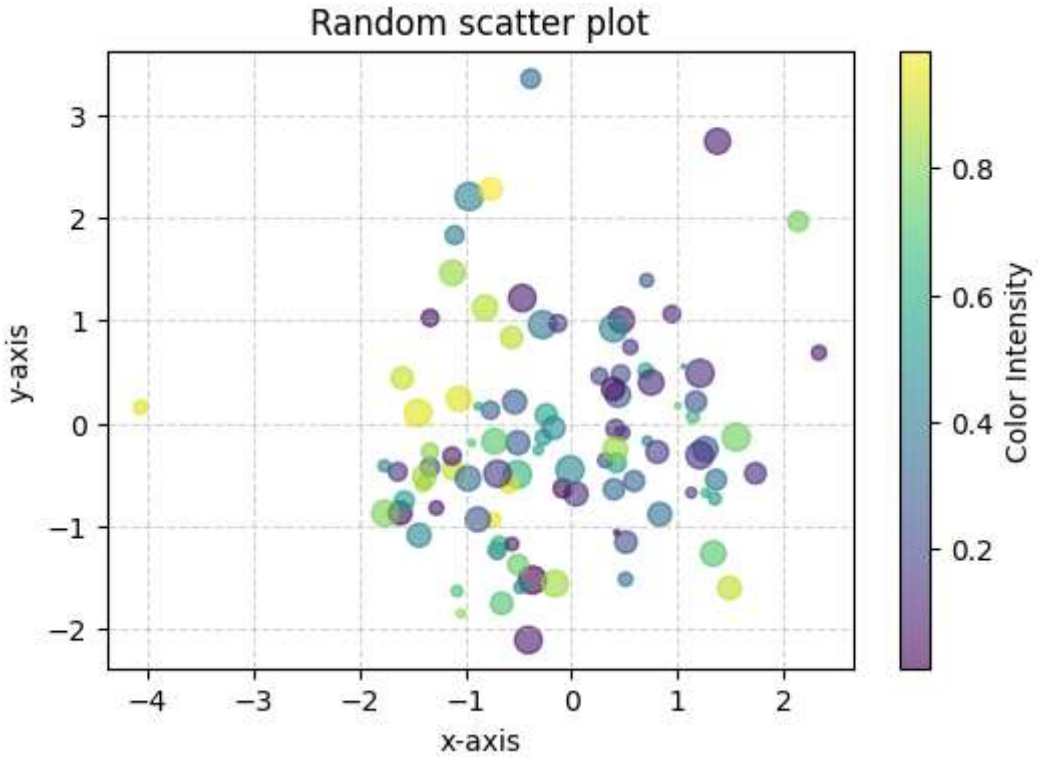
In [146…

```
x = np.random.randn(100)
y = np.random.randn(100)
colors = np.random.rand(100) # Random colors
sizes = np.random.rand(100) * 100 # Random sizes

# Scatter plot
plt.figure(figsize=(6, 4))
scatter = plt.scatter(x, y, c=colors, s=sizes, alpha=0.6, cmap='viridis')

# Add styling and labels
plt.title('Random scatter plot')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.colorbar(scatter, label='Color Intensity')
plt.grid(True, linestyle='--', alpha=0.5)

# Show the plot
plt.show()
```

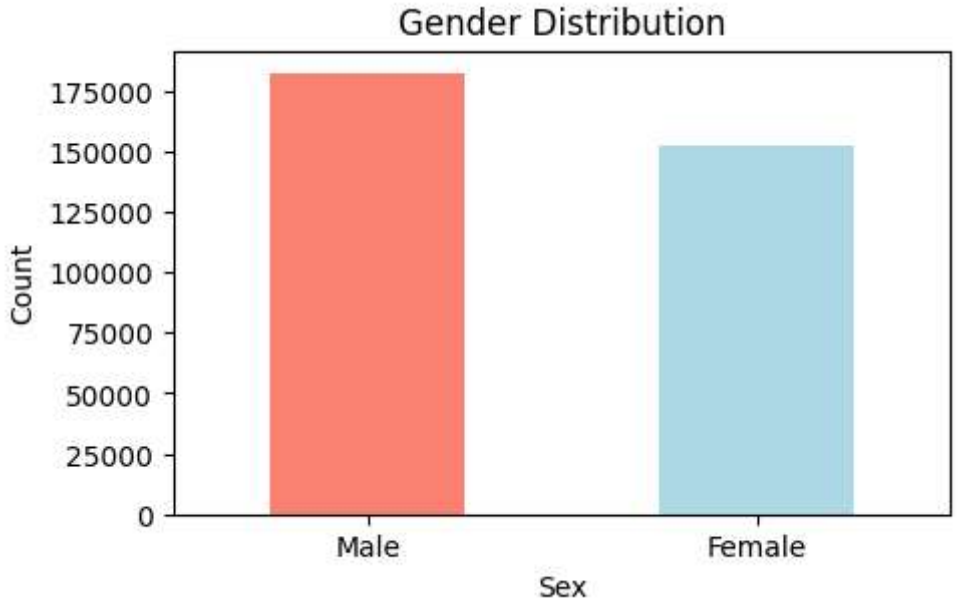


In [147…
```
plt.figure(figsize=(5, 3))

# bar plot

df['sex'].value_counts().plot(kind='bar', color=['salmon', 'lightblue'])
plt.title('Gender Distribution')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.xticks(rotation=0)
```

Out[147…
```
(array([0, 1]), [Text(0, 0, 'Male'), Text(1, 0, 'Female')])
```


Gender Distribution
175000
150000
125000
100000
75000
50000
25000
0
Count
Male
Female
Sex