Session 5: Data cleansing, functions, if-condition & for-loop

Module BUSN9690

Business Statistics with Python

EDA: Exploratory Data Analysis

- df.info() #the structure of the dataframe

```
In [32]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 6 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Name                  5 non-null      object
 1   Business_statistics   5 non-null      int64
 2   Simulation            5 non-null      int64
 3   Machine_Learning      5 non-null      int64
 4   Big_Data              5 non-null      int64
 5   Dissertation          5 non-null      int64
dtypes: int64(5), object(1)
memory usage: 368.0+ bytes
```

- df.describe() #basic statistics of the dataframe

- Question: Can you try
  - df.Machine_Learning.head()
  - df.Machine_Learning.tail()
  - df.Machine_Learning.describe()

… Not shown

```
In [31]: df.describe()
Out[31]:
       Business_statistics  Simulation  ...  Big_Data  Dissertation
count             5.000000    5.000000  ...  5.000000      5.000000
mean             68.600000   65.400000  ...  74.800000     67.800000
std               8.414274    3.847077  ...  6.220932      5.449771
min              56.000000   60.000000  ...  65.000000     62.000000
25%              65.000000   64.000000  ...  73.000000     65.000000
50%              72.000000   65.000000  ...  76.000000     65.000000
75%              72.000000   68.000000  ...  80.000000     72.000000
max              78.000000   70.000000  ...  80.000000     75.000000
```

Data cleansing

Data pre-processing

- Inconsistent data:
  - Tidy them up
- Missing data:
  - Deletion
  - Imputation
- Outliers:
  - Detection
  - Handling

A dataset

- Descriptive statistics
- education in No 3 is missing;
- age in No 6 is incorrect; and
- No 7 duplicates No 8

No	age	date	education	occupation	Income
1	39	01/01/1995	Bachelors	Adm-clerical	<=50K
2	50	02/01/1995	Bachelors	Exec-managerial	<=50K
3	38	03/01/1995		Handlers-cleaners	<=50K
4	53	04/01/1995	11th	Handlers-cleaners	<=50K
5	28	05/01/1995	Bachelors	Prof-specialty	<=50K
6	177	06/01/1995	Masters	Exec-managerial	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
8	52	08/01/1995	HS-grad	Exec-managerial	>50K
9	31	09/01/1995	Masters	Exec-managerial	>50K
10	42	10/01/1995	Bachelors	Exec-managerial	>50K
11	37	11/01/1995	Some-college	Exec-managerial	>50K
12	30	12/01/1995	Bachelors	Prof-specialty	>50K
13	23	13/01/1995	Bachelors	Adm-clerical	<=50K
14	32	14/01/1995	Assoc-acdm	Sales	<=50K
15	40	15/01/1995	Assoc-voc	Craft-repair	>50K
16	34	16/01/1995	7th-8th	Transport-moving	<=50K
17	25	17/01/1995	HS-grad	Farming-fishing	<=50K
18	32	18/01/1995	HS-grad	Machine-op-inspct	<=50K
19	38	19/01/1995	11th	Sales	<=50K
20	43	20/01/1995	Masters	Exec-managerial	>50K

The dataset can be downloaded by clicking https://moodle.kent.ac.uk/2021/mod/resource/view.php?id=338854

Data pre-processing: missing data detection

```
import pandas as pd #To import the cvs file into a dataframe

Adult_df = pd.read_csv('adult.csv') #it contains 21 records

Adult_df.info() #to find the basic information of the data

Adult_df.shape #to find both the number of variables and that of rows

Adult_df.shape[0] #obtain the number of rows

Adult_df.shape[1] #obtain the number of columns
```

```
In [14]: Adult_df.shape
Out[14]: (21, 6)
```

```
In [11]: Adult_df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   No          21 non-null     int64
 1   age         21 non-null     int64
 2   date        21 non-null     object
 3   education   20 non-null     object
 4   occupation  21 non-null     object
 5   Income      21 non-null     object
dtypes: int64(2), object(4)
memory usage: 1.1+ KB
```

No	age	date	education	occupation	Income
1	39	01/01/1995	Bachelors	Adm-clerical	<=50K
2	50	02/01/1995	Bachelors	Exec-managerial	<=50K
3	38	03/01/1995		Handlers-cleaners	<=50K
4	53	04/01/1995	11th	Handlers-cleaners	<=50K
5	28	05/01/1995	Bachelors	Prof-specialty	<=50K
6	177	06/01/1995	Masters	Exec-managerial	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
8	52	08/01/1995	HS-grad	Exec-managerial	>50K
9	31	09/01/1995	Masters	Exec-managerial	>50K
10	42	10/01/1995	Bachelors	Exec-managerial	>50K
11	37	11/01/1995	Some-college	Exec-managerial	>50K
12	30	12/01/1995	Bachelors	Prof-specialty	>50K
13	23	13/01/1995	Bachelors	Adm-clerical	<=50K
14	32	14/01/1995	Assoc-acdm	Sales	<=50K
15	40	15/01/1995	Assoc-voc	Craft-repair	>50K
16	34	16/01/1995	7th-8th	Transport-moving	<=50K
17	25	17/01/1995	HS-grad	Farming-fishing	<=50K
18	32	18/01/1995	HS-grad	Machine-op-inspct	<=50K
19	38	19/01/1995	11th	Sales	<=50K
20	43	20/01/1995	Masters	Exec-managerial	>50K

There is a missing value under *education* because there are only 20 values, while there are 21 values under the others

Remove missing values and imputation

DataFrame.dropna(axis=0, how='any', inplace=False)

- Axis=0: Drop rows which contain missing values; axis=1: Drop columns which contain missing value;
- how='any' : If any NA values are present (default), drop that row or column; how='all' : If all values are NA, drop that row or column;
- inplace: bool, default False; If True, do operation inplace and return **None**.

new_df1=Adult_df.dropna() #the original DataFrame, Adult_df, has been modified and it now contains only 20 rows (i.e., row 3 has been removed)

Adult_df1.fillna("Masters", inplace = True)#replace the missing data with "Masters"

No	age	date	education	occupation	Income
1	39	01/01/1995	Bachelors	Adm-clerical	<=50K
2	50	02/01/1995	Bachelors	Exec-managerial	<=50K
3	38	03/01/1995		Handlers-cleaners	<=50K
4	53	04/01/1995	11th	Handlers-cleaners	<=50K
5	28	05/01/1995	Bachelors	Prof-specialty	<=50K
6	177	06/01/1995	Masters	Exec-managerial	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
8	52	08/01/1995	HS-grad	Exec-managerial	>50K
9	31	09/01/1995	Masters	Exec-managerial	>50K
10	42	10/01/1995	Bachelors	Exec-managerial	>50K
11	37	11/01/1995	Some-college	Exec-managerial	>50K
12	30	12/01/1995	Bachelors	Prof-specialty	>50K
13	23	13/01/1995	Bachelors	Adm-clerical	<=50K
14	32	14/01/1995	Assoc-acdm	Sales	<=50K
15	40	15/01/1995	Assoc-voc	Craft-repair	>50K
16	34	16/01/1995	7th-8th	Transport-moving	<=50K
17	25	17/01/1995	HS-grad	Farming-fishing	<=50K
18	32	18/01/1995	HS-grad	Machine-op-inspct	<=50K
19	38	19/01/1995	11th	Sales	<=50K
20	43	20/01/1995	Masters	Exec-managerial	>50K

Fix the inconsistent value of No 6

- import pandas as pd
- Adult_df = pd.read_csv('adult.csv') #it contains 21 records
- Adult_df.loc[5, 'age'] = 35 #replace age in No.6 with 35.
  Note: Python starts from 0, No 6 therefore becomes row 5
- Note: you can also use Adult_df.iat[5,1] to access the value at the 6th row and 2nd column, where iat is a function name
  - DataFrame.iat: Access a single value for a row/column pair by integer position: Adult_df.iat[5,1]
  - DataFrame.loc: Access a group of rows and columns by label(s): Adult_df.loc[5, 'age']

No	age	date	education	occupation	Income
1	39	01/01/1995	Bachelors	Adm-clerical	<=50K
2	50	02/01/1995	Bachelors	Exec-managerial	<=50K
3	38	03/01/1995		Handlers-cleaners	<=50K
4	53	04/01/1995	11th	Handlers-cleaners	<=50K
5	28	05/01/1995	Bachelors	Prof-specialty	<=50K
6	177	06/01/1995	Masters	Exec-managerial	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
8	52	08/01/1995	HS-grad	Exec-managerial	>50K
9	31	09/01/1995	Masters	Exec-managerial	>50K
10	42	10/01/1995	Bachelors	Exec-managerial	>50K
11	37	11/01/1995	Some-college	Exec-managerial	>50K
12	30	12/01/1995	Bachelors	Prof-specialty	>50K
13	23	13/01/1995	Bachelors	Adm-clerical	<=50K
14	32	14/01/1995	Assoc-acdm	Sales	<=50K
15	40	15/01/1995	Assoc-voc	Craft-repair	>50K
16	34	16/01/1995	7th-8th	Transport-moving	<=50K
17	25	17/01/1995	HS-grad	Farming-fishing	<=50K
18	32	18/01/1995	HS-grad	Machine-op-inspct	<=50K
19	38	19/01/1995	11th	Sales	<=50K
20	43	20/01/1995	Masters	Exec-managerial	>50K

Handling duplicates

```
import pandas as pd

Adult_df = pd.read_csv('adult.csv') #it contains 21 records

Adult_df.duplicated()#to check duplicates. The result shows that row 7 is a duplicate

Adult_df.drop_duplicates(inplace = True)#this will remove the duplicate
```

No	age	date	education	occupation	Income
1	39	01/01/1995	Bachelors	Adm-clerical	<=50K
2	50	02/01/1995	Bachelors	Exec-managerial	<=50K
3	38	03/01/1995		Handlers-cleaners	<=50K
4	53	04/01/1995	11th	Handlers-cleaners	<=50K
5	28	05/01/1995	Bachelors	Prof-specialty	<=50K
6	177	06/01/1995	Masters	Exec-managerial	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
7	49	07/01/1995	9th	Other-service	<=50K
8	52	08/01/1995	HS-grad	Exec-managerial	>50K
9	31	09/01/1995	Masters	Exec-managerial	>50K
10	42	10/01/1995	Bachelors	Exec-managerial	>50K
11	37	11/01/1995	Some-college	Exec-managerial	>50K
12	30	12/01/1995	Bachelors	Prof-specialty	>50K
13	23	13/01/1995	Bachelors	Adm-clerical	<=50K
14	32	14/01/1995	Assoc-acdm	Sales	<=50K
15	40	15/01/1995	Assoc-voc	Craft-repair	>50K
16	34	16/01/1995	7th-8th	Transport-moving	<=50K
17	25	17/01/1995	HS-grad	Farming-fishing	<=50K
18	32	18/01/1995	HS-grad	Machine-op-inspct	<=50K
19	38	19/01/1995	11th	Sales	<=50K
20	43	20/01/1995	Masters	Exec-managerial	>50K

```
In [26]: Adult_df.duplicated()
Out[26]:
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7      True
8     False
9     False
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
dtype: bool
```

Question

- Download surveys-small-version.csv from moodle.kent.ac.uk, answer the following questions, respectively
  - Which variables include missing values?
  - Write a Python program to delete the rows with missing values
  - Write a Python program to remove the duplicates

Useful libraries

- Numpy
- Scipy
- Pandas
- Matplotlib
- TensorFlow
- Keras: neural network libraries
- Theano
- PyTorch
- Seaborn
- Scikit-Learn

User defined function

User-defined function

- Basic syntax #function without return value

```
def functionName(par_1,...,par_n): #par_1,...,par_n are arguments, which are optional
    statements #function body: remember the indentation
```

- Function Components: The different parts of a function are
  - def: the keyword for defining the function
  - Function Name – This is the actual name of the function.
  - Function Body – The function body contains a collection of statements defining what the function does.

- Basic syntax #function without return value

```
def functionName(par_1,...,par_n): #par_1,...,par_n are arguments, which are optional
    statements #function body
    return value1
```

- Function Components: The different parts of a function are
  - value1 is the value returned from the function;
  - The other parts are the same as explained in the second cell on this slide

Examples

- Example 1: function without arguments or return values
  - def my_function(): #define a function. This function does not have arguments
    print("Hello World") #function body. This function does not return anything
  - my_function() #call the defined function

- Example 2: function with one argument but no return values
  - def my_function(student_name): #define a function. This function has an argument
    print(student_name + " attends BUSN9690") #function body: without any return values
  - my_function("Ben") #call the defined function

- Basic rule:
  - Define a function before calling it
  - You can call a function wherever you need after its definition

Examples

- Example 3: function with arguments but no return values
  - def my_function(student1,student2): #define a function. This function has two arguments
    print(student1 + " sits next to "+student2) #function body.
  - my_function("Ben", "Ahmed") #call the defined function

- If the number of arguments is unknown, add a * before the parameter name

- Example 4: function with several unknow arguments
  - def my_function(*cities):
    print("The largest city in the UK is " + cities[2])
  - my_function("Bristol", "Canterbury", "London") #call the defined function

User-defined function with a return value

- Example 5

```
# Create a function to print squares of numbers in sequence.
>>> from math import *
>>>def expProb(nu,t):
        prob = 1 - exp(-nu*t)
        return prob        #return the value prob

# Call the function new.function supplying 6 as an argument.
>>>  expProb(0.01,10) #here, we set nu=0.01, t=10
[1] 0.09516258
```

Calling a Function with Argument Values (by position and by name)

```
# Create a function with arguments.
>>> from math import *
>>> def expProb(nu,t):
        prob = 1 – math.exp(-nu*t)
        return prob          #return the value prob

# Call the function by position of the arguments.
>>> expProb(0.01,10)
0.09516258196404048

# Call the function by names of the arguments.
>>> expProb(nu=0.01,t=10)
0.09516258196404048
```

Question

What is the difference between the following two functions?

```
# Function 1
def newFunction(a = 3, b = 6):
    import  math
    result=math.log(a) * (b**2)
    print(result)

# Function 2
import  math
def newFunction(a = 3, b = 6):
    result=math.log(a) * (b**2)
    print(result)
```

```
# Function 3
from math import *
def newFunction(a = 3, b = 6):
    result=math.log(a) * (b**2)
    print(result)
```

Calling a Function with Default Argument

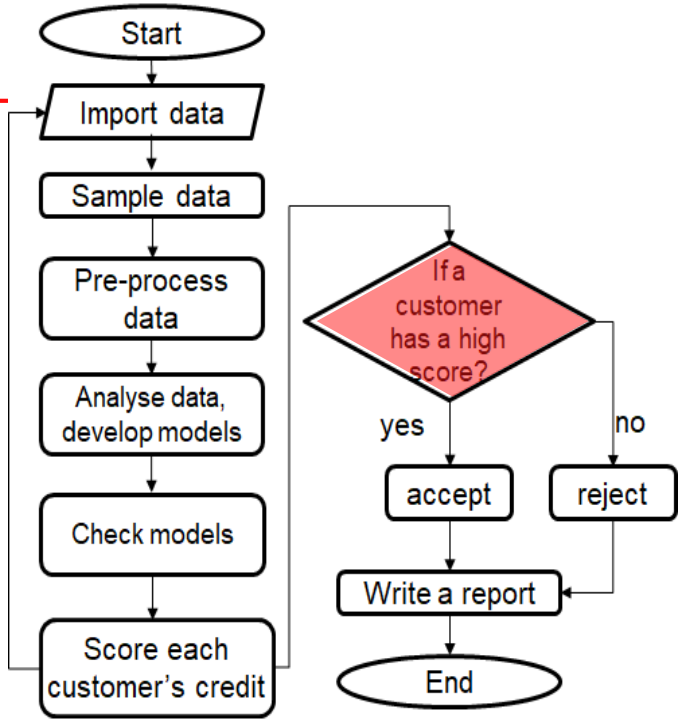
```
# Create a function with default arguments.
import  math
def newFunction(a = 3, b = 6):
    result=math.log(a) * (b**2)
    print(result)

# Call the function without giving any argument.
>>>newFunction()
[1] 39.55004
# Call the function with giving new values of the argument.
>>>newFunction(10,4)
[1] 36.84136
>>> newFunction(6) #to assign 6 to the first argument and let the 2nd uses the default
[1] 64.50334
```

```
>>> newFunction(6,6)
[1] 64.50334

# Create a function with arguments.
 def newFunction1(a, b):
     result=math.log(a) * (b**2)
     print(result)


>>> newFunction1(2,4)
[1] 11.09035
> newFunction1(2)
TypeError: newFunction() missing 1 required positional argument: 'b'
```

Return multiple values

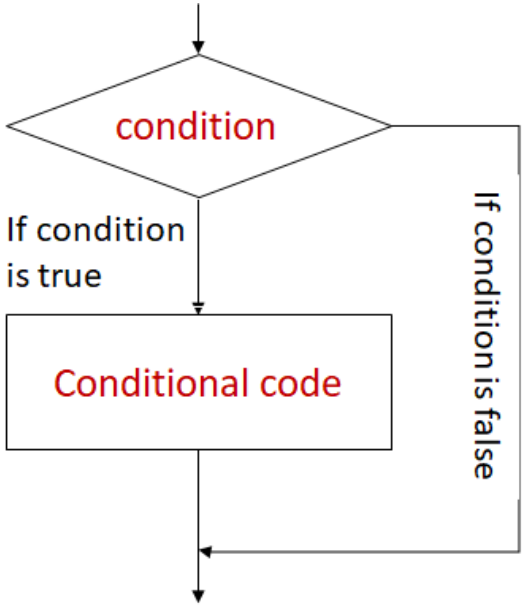
```
import math
def move(x, y, step, angle=0):
    nx = x + step * math.tan(angle)
    ny = y - step * math.sin(angle)
    return nx, ny

move(1,2,3,0)
```

Conditions and Loops



if-else

- The if-else structure allows you to execute statements depending on whether a given condition is true or false



#Syntax:

```
if  <condition>:
   statement # do something
else:
    statement## do something
```

#Syntax:

```
if <condition1>:
      statement# do something
elif <condition2>:
      statement# do something different
else:
      statement#do something different
```

- Basic rule:
  - Do not forget the **indentation** and the **colon**

Examples

- Example 1: if-clause

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

- **Note how the indentation is placed**
- Example 2: if-elif-clause

```
a = 43
b = 330
if b > a:
   print("b is greater than a")
elif a == b:
   print("a and b are equal")
```

- Example 3: if-elif-else-clause

```
a = 200
b = 33
if b > a:
   print("b is greater than a")
elif a == b:
   print("a and b are equal")
else:
   print("a is greater than b")
```

- **You may have many elif's**

Examples

- Example 4: Or
  - a = 200
    b = 33
    c = 500
    if a > b or a > c:
      print("Hi")

- Example 5: and
  - a = 200
    b = 33
    c = 500
    if a > b and c > a:
      print("Both conditions are True")

- Example 6: short-hand if-else-clause

  a = 2

  b = 330

  print("A") if a > b else print("B")

$$\text{result} = \begin{cases} A & \text{if } a > b \\ B & \text{if } a \leq b \end{cases}$$

- Example 7: short-hand if-else-clause

  a = 330

  b = 330

  print("A") if a > b else print("=") if a == b else print("B")

$$\text{result} = \begin{cases} \text{A} & \text{if } a > b \\ = & \text{if } a == b \\ B & \text{if } a < b \end{cases}$$

for-loop

- for loops take an iterator variable and assign it successive values from a sequence or vector.

#Basic syntax:

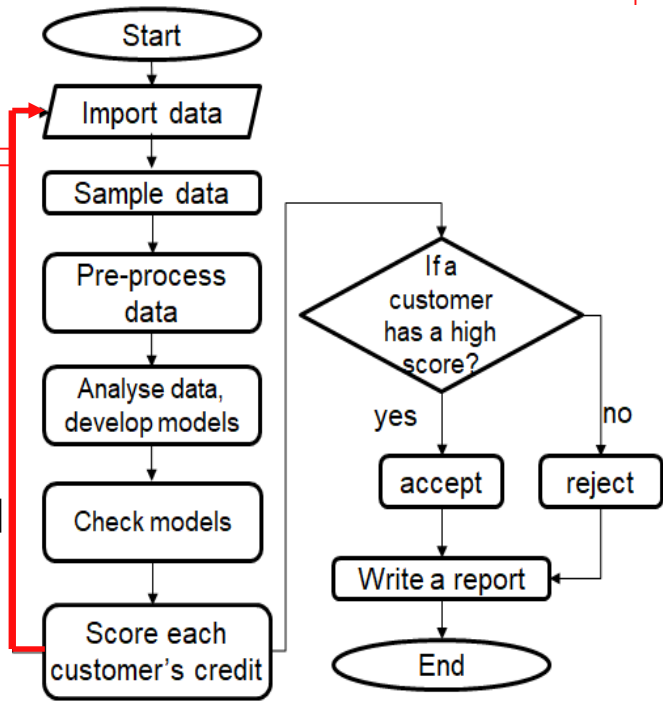
for val in sequence or vector:

    statement

- **Example 1:** Loop through a string

```
for x in "Python":
  print(x)
```

- **Example 2:** Loop through a list/set/tuple/dictionary
  - Modules= ['BusinessStats','MachineLearning','Simulation','BigData']
  - for x in Modules:
    print(x)



Examples

- Range
  - Example 3

```
for x in range(6):
  print(x)
```

  - Example 4

```
for x in range(2, 6):
  print(x)
```

  - Example 5

```
for x in range(2, 30, 3):
  print(x)
```

- Example 6

```
var=[] #to define an empty array
arr = [0 for i in range(5)] #to assign 0's to the array
```

- Example 7: calculate $s = \sum_{k=1}^{100} e^{-0.05k}$

```
from math import *
s=0 #to initialise s
for k in range(1,101): #101, instead of 100
  s=s+exp(-0.05*k) # s+=math.exp(-0.05*k)
print(s)
```

while-loop

- Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

#Basic syntax:

```
while test_expression:
  statement
```

Examples

- Example 1: The while loop requires relevant variables to be ready. In our following examples, we set i=1

```
i = 1 #to initialise the variable i
while i < 6:
  print(i)
  i += 1  #the same as i=i+1
```

- Example 2:

```
i = 1 #to initialise the variable i
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

- Example 3

```
count = 0 #to initialise the variable count
while (count < 9):
  print('The count is:', count)
  count = count + 1
  print("Good bye!")
```

```
i=1
while i < 5:
... i =i+1
... print (i)
```

- Steps
  - Step 1: i=1➔if 1<5➔i==2➔ print 2
  - Step 2: i=2 ➔if 2<5➔i==3➔ print 3
  - Step 3: i=3 ➔if 3<5➔i==4➔ print 4
  - Step 4: i=4 ➔if 4<5➔i==5➔ print 5
  - Step 5: i=5 ➔if 5<5 stop

```
i=1
while i < 5:
... print (i)
... i =i+1
```

- Steps
  - Step 1: i=1➔if 1<5➔ print 1➔ i==2
  - Step 2: i=2 ➔if 2<5➔ print 2➔ i==3
  - Step 3: i=3 ➔if 3<5➔ print 3➔i==4
  - Step 4: i=4 ➔if 4<5➔ print 4➔ i==5
  - Step 5: i=5 ➔if 5<5 stop