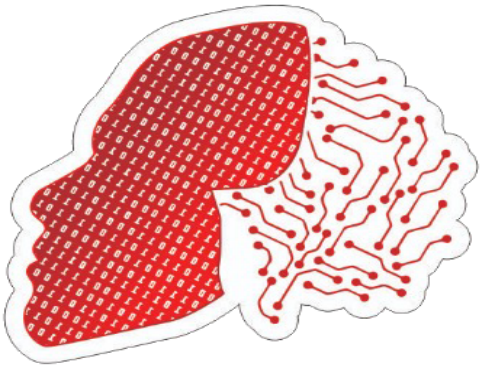


A.I.ANALYTICS

BIG DATA | DATA SCIENCE

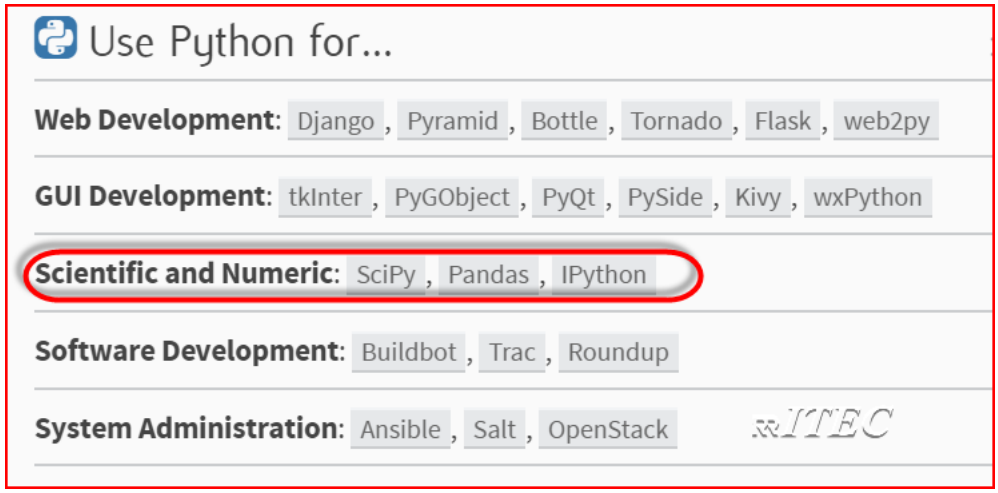
DS, ML, DL, NLP & AI With PYTHON

Classroom Notes

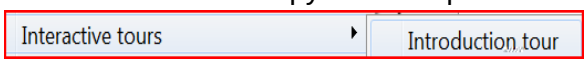
Level 01 of 08 : Basic Python

1. Introduction

1.  Guido van Rossum is a Dutch(Europe) programmer who is best known as the author of the Python programming language
2. Python is a programming language that lets you work quickly and integrate systems more effectively.
3. Use python General Purpose: build anything



4. Python is open Source! Free! Free! Free!
5. Python mainly has two version python 2.x and python 3.x. For more information refer https://www.python.org/downloads/
6. Download anaconda software https://www.anaconda.com/downloads and install it.
    a. Anaconda software = python software + around 1400 python packages of data science
7. Spyder is a powerful interactive development environment(IDE) for python programing language.
8. Observe spyder tour
    a. Start → anaconda → spyder → help →



9. Observe Jupyter IDE

a. Start → anaconda →Jupyter

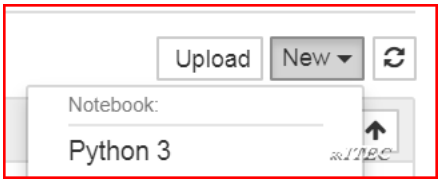
   i. It is a web based IDE,usefully to develop notebooks

   ii. Observe it ,more we will see in next lessons

2. Data Types and Variables

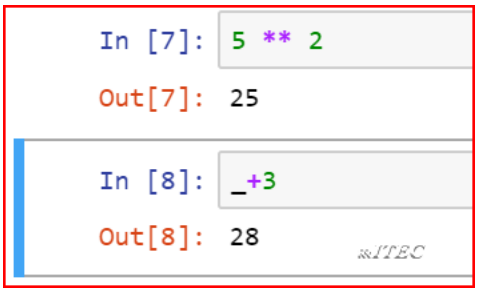
1. Exercise 1: Python as calculator

a. Open Start → All Programs → Anaconda3 → Jupyter Notebook → Right side of the browser → click on new → under notebook click on python3

b. 


```
In [1]: 2+3
Out[1]: 5

In [2]: 3-2
Out[2]: 1

In [3]: 2*3
Out[3]: 6

In [4]: 17/3
Out[4]: 5.666666666666667

In [5]: 17//3
Out[5]: 5

In [6]: 17%3
Out[6]: 2

In [7]: 5 ** 2
Out[7]: 25
```

2. We can access previous step result using underscore " _"



3. Save Jupyter notebook
4. Saved notebooks are available in the path C:\Users\Hi\
5. Let us see some Basic data types
    a. Numbers (int, float and complex)
    b. Strings
    c. Bool
    d. Lists
    e. Tuples
    f. Dictionary
6. **Exercise 2: Define some datatypes**
    a. Use type() function to know the variable data type


```
# Define some sample data types
# 01.Defining integer
savings = 1000
print(savings)
# 02. Defining float
interest = 4.5
print(interest)
# 03. Defining complex
a_complex = 1 + 2J
a_complex
```

```
# 04. Defining String
desc = "compound interest"
print(desc)
# 05. Defining Boolean
am_i_good=True
print(am_i_good)
# 06. How to know data type of a given object?
print(type(savings))
print(type(interest))
print(type(a_complex))
print(type(desc))
print(type(am_i_good))
```


7. Exercise 3: Convert one data type into other datatype

a. Convert one data type into other using functions int(),str(),float(),complex() and bool()

```
# 01. Convert int into str
a=10
type(a)
a1 = str(a)
type(a1)
# 02. Convert float into str
b=10.5
type(b)
b1 = str(b)
type(b1)
```

```
# 03.Use case: calculating simple interest

savings = 100

interest=12

total_amount = savings + 100 * interest * 2/100

# convert total amount as string and print user friendly info

print("I started with $" + str(savings) + " and now have $" + str(total_amount) + ". Awesome!")

# 04. Convert string into float

pi_string = "3.1415926"

print(type(pi_string))

# 05. Convert pi_string into float: pi_float

pi_float = float(pi_string)

print(type(pi_float))
```


8. **Exercise 4: Read characters from string**

```
# 01. Create a string

word = "python"

# 02. print y

word[1]

# 03. print h

word[3]

# 04. print slice pyth

word[0:4]

# 05. print slice starting to t

word[:3]

# 06. Fancy reading
```

```
for i in [0,3,5]:

    print(word[i])
```

9. Mutable vs Immutable Objects: A mutable object can be changed after it's created, and an immutable object can't.

10. String is immutable

```
In [11]: word='JYTHON'

In [12]: word
Out[12]: 'JYTHON'

In [13]: word[0]= 'P'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-b5a712506db9> in <module>()
----> 1 word[0]= 'P'

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

11. Refer https://docs.python.org/3/tutorial/introduction.html

12. More about numbers refer https://docs.python.org/3/tutorial/introduction.html#numbers

13. More about Strings refer https://docs.python.org/3/tutorial/introduction.html#strings

3. List

1. Python List [a, b, c]
2. The most versatile (flexible) is the list, which can be written as a list of comma-separated values (items) between square brackets.
3. Lists might contain items of different types, but usually the items all have the same type.

4. Exercise 1: Define a list with same/different data types

```
# Define  a list with same data types

rritec_emp_sal=[1000,2000,3000,4000]

print(rritec_emp_sal)

print(type(rritec_emp_sal))

# Define  a list with different data types

rritec_emp_name_sal=["ram",1000,"nancy",2000,"robert",3000,"rabson",4000]

print(rritec_emp_name_sal)

# A list may contain other list (nested list)

rritec_emp_name_sal1=[["ram",1000],["nancy",2000],["robert",3000],["rabson",4000]]

print(rritec_emp_name_sal1)

print(type(rritec_emp_name_sal1))
```


5. Exercise 2: Read list elements(Slicing and dicing)

```
# Read list values (Slicing and dicing)

print(rritec_emp_name_sal[0])

print(rritec_emp_name_sal[-1])

print(rritec_emp_name_sal[0:4])

print(rritec_emp_name_sal[:4])

print(rritec_emp_name_sal[4:])

print(rritec_emp_name_sal1[0][0])
```

```
print(rritec_emp_name_sal1[0][-1])
```

6. **Exercise 3: Manipulating lists**

   a. Modify elements ,Add elements and Delete elements of list

   b. List is mutable

```
# Changing list value
rritec_emp_name_sal[0]='Ram Reddy'
rritec_emp_name_sal[2:4]=['willis',2100]
print(rritec_emp_name_sal)
# Adding Elements
rritec_emp_name_sal=rritec_emp_name_sal + ['Helen',5000]
print(rritec_emp_name_sal)
# Deleting  Elements
del(rritec_emp_name_sal[2:4])
print(rritec_emp_name_sal)
```


7. **Exercise 4: Observe list assignments memory allocations**

```
# Need to observe 1
x = [1000,2000,3000]
y = x
z = list(x)
y[0]=1001
print(y);print(x)
# Need to observe 2
x = [1000,2000,3000]
z = list(x)
z[1] = 2001
```

```
print(z);print(x)
```

8. **Refer 1 of 2: https://docs.python.org/3/tutorial/introduction.html#lists**
9. **If time permits ,refer 2 of 2: https://docs.python.org/3/tutorial/datastructures.html**

4. Operators

1. **Operators 1 of 3: Comparison Operators** are **(<, >, ==, <=, >=, !=)**

Py22_1_Comparison_Operators

```
print("01. Comparison of booleans")
print(True == False)
print(True != False)

print("02. Comparison of integers")
print(-5 * 15 != 75)
print(2 == (1 + 1))

print("03. Comparison of strings")
print("R" != "python")
print("Python" != "python") # note P is capital

print("04. Compare a boolean with a numeric")
print(True == 1)

print("05. Compare numpy list with integer")
import numpy as np
marks = np.array([10,20,30,40,50,60,70,80,90,100])
print(marks>50)
print(marks[marks>50])

#remove comment and execute what you noticed?
#print("06. Compare python list with integer")
#print([10,20,30,40,50,60,70,80,90,100]>50)
#print("07. Compare int with string")
#print(1<"RAM")
```

2. **Operators 2 of 3: Boolean Operators** (**and, or, not**)

*Py22_2_Boolean_Operators

```
print("01. AND Boolean operator")
x=6
print(x > 5 and x < 15)
print("02. OR Boolean operator")
y=5
print(y < 7 or y > 13)
print("03. NOT Boolean operator")
print(not True)
print(not False)
print("04. working with numpy array is it working")
import numpy as np
marks = np.array([10,20,30,40,50,60,70,80,90,100])
print(marks>50 and marks<80)
```

3. **Operators 3 of 3: Numpy Boolean Operators (logical_and(), logical_or(), logical_not())**

*Py22_3_Numpy_Boolean_Operators

```
print("01. logical_and() Boolean operator")
import numpy as np
marks = np.array([10,20,30,40,50,60,70,80,90,100])
print(np.logical_and(marks>50,marks<80))

print("02. logical_or() Boolean operator")
marks = np.array([10,20,30,40,50,60,70,80,90,100])
print(np.logical_or(marks==50,marks>70))

print("03. logical_not() Boolean operator")
marks = np.array([10,20,30,40,50,60,70,80,90,100])
print(np.logical_not(marks==50))
```

4. Filtering data frame using operators

Py22_5_Filtering_Dataframe

```
print("01. Import csv file as pandas dataframe")
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
print(brics)

print("02. observe area column")
print(brics["area"]) # try brics.loc[:,"area"] or brics.iloc[:,2]

print("03. filter data area >8 ")
print(brics[brics["area"] > 8])

print("04. filter data using boolean operator")
import numpy as np
print(brics[np.logical_and(brics["area"] > 8, brics["area"] < 10)])
```

5. Control Flows

5.1 if … elif … elif … else

1. Conditional Statements (If , elif, else),observe below two examples

```
x = int(input("Please enter an integer: "))

if x < 0:

     print('Negative Number')

elif x == 0:

   print('Zero')

else :

   print('Positive Number')
```

Py22_4_Conditional_Statements

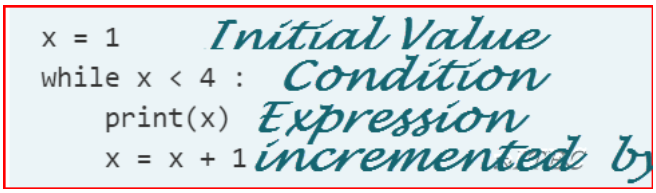
```
1 z = 5
2 if z % 2 == 0 :
3     print("z is divisible by 2")
4 elif z % 3 == 0 :
5     print("z is divisible by 3")
6 else :
7     print("z is neither divisible by 2 nor by 3")
```

2. An **if … elif … elif …** sequence is a substitute for the **switch** or **case** statements found in other languages.(example switch in R language)

5.2 While Loop:

1. The while loop is like a repeated if statement. The code is executed over and over again, as long as the condition is true.

2. **Exercise 1:** What is the output of below code?



3. **Exercise 2: write Fibonacci series up to 1000**

```
a, b = 0, 1

while b < 1000:

    print(b, end=',')

    a, b = b, a+b
```

4. **Exercise 3: Understand Break Statement**

   a. With the break statement we can stop the loop even if the while condition is true

```
# Exit the loop when i is 3

i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

5. **Exercise 4: Understand Continue Statement**

   a. With the continue statement we can stop the current iteration, and continue with the next iteration

```
# Continue to the next iteration if i is 3:

i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
```

```
print(i)
```

5.3 For Loop:

1. Python's for statement iterates over the items of any sequence (a **list** / a **string** / a **dictionary**), in the order that they appear in the sequence
2. For each value in object(list, Dictionary...etc) , execute expression

Exercise 2: Read list of items?

Py23_2_For_Loop

```
# define list of family memers height
family_heights = [1.73, 1.68, 1.71, 1.89]
#define for loop
for height in family_heights :
#write expressions
    print(height)
```

Exercise 3: Read list of items with indexes?

*Py23_3_For_Loop_with_index_and_values

```
# define list of family members height
family_heights = [1.73, 1.68, 1.71, 1.89]
#define for loop
for index,height in enumerate(family_heights) :
#write expressions
    print("index " + str(index) + ": " + str(height))
```

Exercise 4: Read list of list items?

*Py23_4_For_Loop_List_of_Lists

```
# Define list of lists
house = [["Hall area", 11.25],
         ["Kitchen area", 18.0],
         ["Living room area", 20.0],
         ["Bedroom area", 10.75],
         ["Bathroom area ", 9.50]]
# Build a for loop
for x in house :
    print("The " + str(x[0]) + " is " + str(x[1]) + " sqm")
```

Exercise 5: For loop on dictionary?

1. The items() method is useful to loop over a dictionary

*Py23_5_For_Loop_on_Dictionary

```
# Definition of dictionary
world = {"China":1.38,"India":1.34,"USA":0.32}
# Iterate over world dictionary using items method
for key,value in world.items() :
    print("The Population of  " + str(key) + " is " + str(value))
```

Exercise 6: For loop on numpy array?

1. 1D Numpy array, looping over all elements can be as simple as list looping

*Py23_6_For_Loop_on_NumPy_array_1Dimension

```
#import required modules
import numpy as np
# define list of family members height
np_family_heights = np.array([1.73, 1.68, 1.71, 1.89])
#define for loop
for height in np_family_heights :
    print(height)
```

2. A 2D array is built up of multiple 1D arrays. To explicitly iterate over all separate elements of a multi-dimensional array, you'll need the syntax: **for x in np.nditer(my_array) :**

Py23_7_For_Loop_on_NumPy_array_2_Dimensions

```
#import required modules
import numpy as np
# define list of family members height $ weight
np_family_heights = np.array([1.73, 1.68, 1.71, 1.89])
np_family_weights = np.array([70, 60, 65, 80])
#Define 2D numpy array
np_family_heights_weights = np.array([np_family_heights,np_family_weights])
# Define for loop using nditer function
for height in np.nditer(np_family_heights_weights) :
    print(height)
```

Exercise 7: For loop on pandas dataframe?

*Py23_8_For_Loop_on_pandas_dataframe

```
1  import pandas as pd
2  brics = pd.read_csv("brics.csv", index_col = 0)
3  print("******* 01. Column Names ********")
4  for val in brics :
5      print(val)
6  print("******* 02 All rows ********")
7  for lab, row in brics.iterrows() :
8      print(lab)
9      print(row)
10 print("******** 03 Selected Columns *********")
11 for lab, row in brics.iterrows() :
12     print(lab + ": " + row["capital"])
13 print("******** 04 Add Column to result *********")
14 for lab, row in brics.iterrows() :
15     brics.loc[lab, "name_length"] = len(row[ "country" ] )
16 print(brics)
17 print("******** 05 Another method to add Column to result *********")
18 brics["name_length1"] = brics["country"].apply(len)
19 print(brics)
```

Refer document: https://docs.python.org/3/tutorial/controlflow.html#

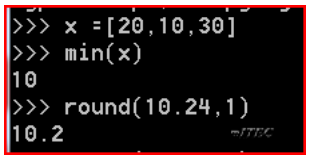
Level 02 of 08: Advanced python

6. Functions, Methods, Modules & Packages

6.1 Functions

6.1.1 System defined Functions

1. Piece of reusable code ,Solves particular task



2. **Exercise 1: understanding predefined functions (str, type, len, max…etc)**

Py5_1_Function

```
# Create variables var1 and var2
var1 = [10, 20, 30, 40]
var2 = True
# Print out type of var1
print('The datatype of var1 is :' + str(type(var1)))
# Print out length of var1
print('The length of the variable is '+ str(len(var1)))
# Convert var2 to an integer: out2
out2 = int(var2)
print('The maximum the list elements is : ' + str(max(var1)))
```

3. **Exercise 2: understanding sorted functions**

Py5_1_Function | Py5_2_Function

```
# Create lists first and second
first = [10,90,30]
second = [50,60,40]
# Paste together first and second: full
full = first + second
# Sort full in descending order: full_sorted
full_sorted = sorted(full, reverse = True)
# Print out full_sorted
print(full_sorted)
```

4. **Refer standard document of python**
   https://docs.python.org/3/tutorial/controlflow.html#defining-functions

6.1.2 User Defined Functions (UDFs)

1. Though we have lot of built-in functions sometimes, we need to write user defined functions

Exercise 1: Function without arguments

Py24_1_Function_without_arguments

```
# Define function header
def welcome():
    # Define function body
    print("welcome to Data Science with Python")
# Call the function
welcome()
```

Exercise 2: Function with one arguments

Py24_2_Function_with_one_argument

```
# Define function header
def square(value):
    # Define function body
    print(value ** 2)
# Call the function
square(10)
```

Exercise 3: Function with one arguments and return value

*Py24_3_Function_with_one_argument_with_return_value

```
# Define function header
def square(value):
    #define function body
    return value ** 2
# Call the function and assign value to variable
num = square(3)
print(num)
```

Exercise 4: Function with Docstrings

1.Docstrings describe what your function does

2.Serve as documentation for your function

3.Placed in the immediate line after the function header

4.In between triple double quotes """ or single quotes ‘’’

Py24_4_Function_with_docstrings

```
1 # Define function header
2 def square(value):
3     """Print the square of a given value"""
4     #Define function body
5     print(value ** 2)
6 # Call the function
7 square(10)
```

Exercise 5: Function with multiple arguments

Py24_5_Function_with_multiple_arguments

```
1 # Define function header
2 def square(value1,value2):
3     """Raise value1 to the power of value2"""
4     new_value = value1 ** value2
5     return new_value
6 # Call the function
7 rs = square(10,2)
8 print(rs)
```

6.1.3 Tuple

Exercise 6: working with tuple

1.Tuples needed to return multiple values by a function

2.Tuples are Like a list - can contain multiple values

3.Tuples are Immutable - can't modify values!

```
even_nums = (2,4,6,8)
even_nums[1]=9 # TypeError: 'tuple' object does not support item assignment
```

4.Tuples Constructed using parentheses ()

Py24_6_tuple

```
print("***** 01. Define tuple *****")
even_nums = (2,4,6,8)
print("***** 02. know the Data type *****")
print(type(even_nums))
print("***** 03. Map tuple items to variables *****")
a,b,c,d =even_nums
print("a = " + str(a))
print("b = " + str(b))
print("c = " + str(c))
print("d = " + str(d))
print("***** 04. Access tuple elements like you do with lists *****")
print(even_nums[0])
```

Exercise 7: Function with multiple return values

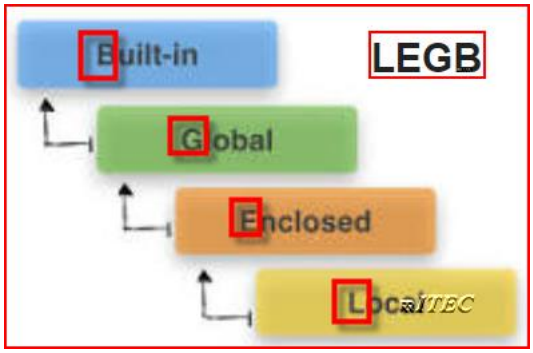
Py24_7_Function_with_multiple_return_values

```
# Define function header
def raise_both(value1,value2):
    """Raise value1 to the power of value2
    and vice versa."""
    new_value1 = value1 ** value2
    new_value2 = value2 ** value1
    new_tuple = (new_value1,new_value2)
    return new_tuple
# Call the function
rs = raise_both(10,2)
print(rs)
```

6.1.4 Scope of objects in functions

6.1.5 Scope of objects in functions / Nested Functions

1. Variable values are searched in the order LEGB



2. **Local scope** → Defined inside a function
3. **Enclosed Scope** → If a function is wrapped inside another function
4. **Global scope** → Defined in the main body of a script
5. **Built-in scope** → Names in the pre-defined built-ins module (ex: len, max, min, … etc)
6. Exercise 1: cannot access local variable globally

*Py25_1_Scope_of_functions_rule1

```
1 # Define function header
2 def square(value):
3     new_value = value ** 2  # Local Variable
4     print(new_value)
5
6 square(10)
7 print(new_value) #Error: NameError: name 'new_value' is not defined
```

7. Exercise 2: we can use same name for local and global and assign different values

Py25_2_Scope_of_functions_rule2

```
1 new_value = 10 # Global Variable
2
3 # Define function header
4 def square(value):
5     new_value = value ** 2  # Local Variable
6     print(new_value)
7
8 square(3)
9 print(new_value)
```

8. Exercise 3: searches for local variable ,if not find then searches for globally

Py25_3_Scope_of_functions_rule3

```
power = 2 # Global Variable
# Define function header
def square(value):
    new_value = value ** power
    print(new_value)

square(3)
```

9. Exercise 4: can override global variables within the function

Py25_4_Scope_of_functions_rule4

```
new_value = 10 # Global Variable
print(new_value)
# Define function header
def square(value):
    global new_value
    new_value = value ** 2
    print(new_value)

square(3)
print(new_value)
```

10. Exercise 5: Nested functions

*Py25_5_Nested_Functions1

```
def mod2plus5(x1, x2, x3):
    """Returns the remainder plus 5 of three values."""
    def inner(x):
        """Returns the remainder plus 5 of a value."""
        return x % 2 + 5
    return (inner(x1), inner(x2), inner(x3))

print(mod2plus5(2,3,4))
```

11. Exercise 6: Nested function

Py25_6_Nested_Functions2

```
def raise_val(n):
    """Return the inner function."""
    def inner(x):
        """Raise x to the power of n."""
        raised = x ** n
        return raised
    return inner
square = raise_val(2)
print(square(3))
```

12. Exercise 7: Nested Functions

*Py25_7_Nested_Functions3

```
1  def outer():
2      """Prints the value of n."""
3      n = 1
4      def inner():
5          nonlocal n
6          n = 2
7          print(n)
8      inner()
9      print(n)
10 outer()
11
```


13. Exercise 8: Builtin understanding

```
a_var = 'global variable'

"""def len(in_var):
    print('called my len() function')
    l = 0
    for i in in_var:
        l += 1
    return l"""
def a_func(in_var):
    len_in_var = len(in_var) # builtin
    print('Input variable is of length', len_in_var)
a_func('Hello, World!')
```

6.1.6 Default and flexible arguments

Exercise 1: Default arguments

*Py25_8_Default_Arguments

```
def power(number, pow=2):
    """Raise number to the power of parameter pow."""
    new_value = number ** pow
    return new_value

print(power(3))
print(power(3,3))
```

Exercise 2: Flexible arguments: *args

*Py25_9_Flexible_Arguments_starargs

```
def add_all(*args):
    """Sum all values in *args together."""
    # Initialize sum
    sum_all = 0
    # Accumulate the sum
    for num in args:
        sum_all += num
    return sum_all

print(add_all(1,2,3))
print(add_all(10,20,30,40,50))
```

Exercise 3: Flexible arguments: **kwargs

Py25_10_Flexible_Arguments_kwargs

```
def print_all(**kwargs):
    """Print out key-value pairs in **kwargs."""
    # Print out the key-value pairs
    for key, value in kwargs.items():
        print(key + ": " + value)
print_all(student1="Rabson", student2="Robert",student3 ="Ram Reddy",student4="Nancy")
```

6.2 Methods

1. Methods are Functions that belong to particular object
2. Based on object data type different methods available
3. **Exercise 1: List object methods**

```
# Define list

a_list = [10,20,30,10,30,40,50,10,60,10]

# Count how many times 10 repeated

a_list.count(10)

# add 100 to list

a_list.append(100)

a_list

# add 200 in 3rd position

a_list.insert(2,200)

a_list

# remove 20 from the list

a_list.remove(20)

a_list

# sort items

a_list.sort()

a_list

# reverse items

a_list.reverse()

a_list
```


4. Exercise 2: String object methods

```
# Define String
```

```
a_str = "Data Science"

# Convert into uppercase

a_str.upper()

# count number of times “ e ” repeated

a_str.count("e")
```

6.3 Modules

1. A module is a file containing Python definitions and statements.
2. The file name is the module name with the suffix .py appended.
3. Spyder working directory available in tool bar

4. Create a module and save into Spyder working directory as “a_module1.py”

```
# Define a number
a_num = 10
# define a string
a_str = "python"
# define a list
a_list = [10,20,30,40]
# define a float
a_float = 10.45
# define function square
def square(x):
    return x ** 2
# define function fibo
def fibo(n):
    a,b=0,1
    while b<n:
        print(b,end = " ")
        a,b=b,a+b
```

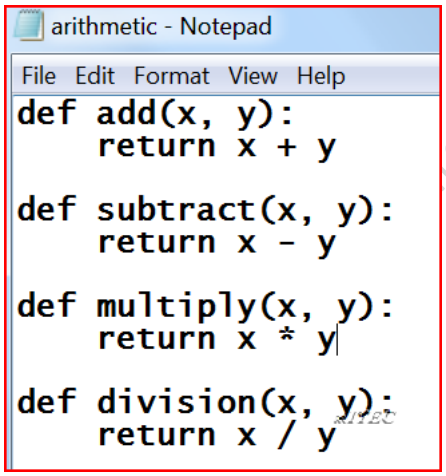

5. Import this module using IPython console **import a_module1**
6. Using the module name you can access the functions

```
# import entire module and access objects
Import a_module1
a_module1.a_num
a_module2.a_float
a_module2.fibo(1000)
# import directly object and use in work space
from a_module1 import a_num
a_num
```

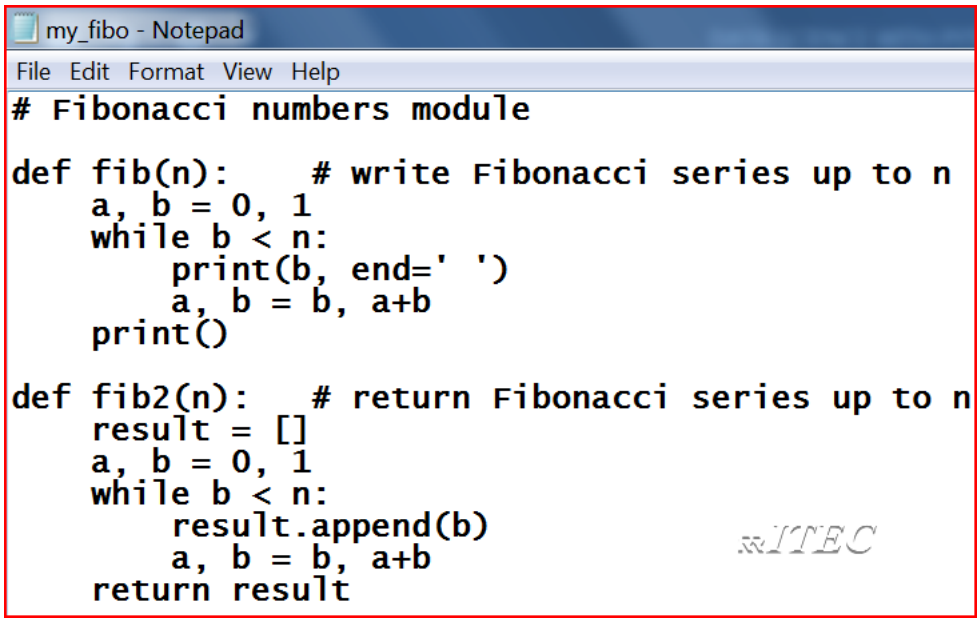
6.4 Packages

6.4.1 User Defined Packages

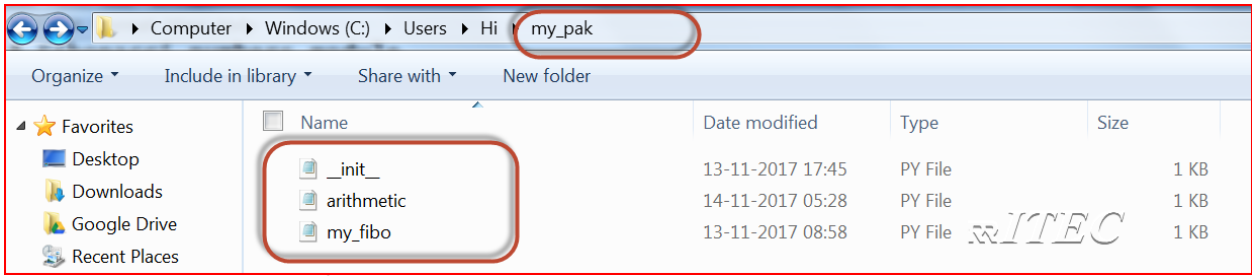
1. Directory/folder of Python files
2. Each file = one module
3. A folder contains __init__.py file then it is called as package
4. A package may have sub packages
5. For more theory see https://docs.python.org/3/tutorial/modules.html#packages
6. **Exercise 1: Create a simple package**
    1. In working directory (C:\Users\Hi) of Spyder create a folder **my_pak**
    2. Create one empty **__init__.py** file
    3. Create one module with the name of **arithmetic.py**



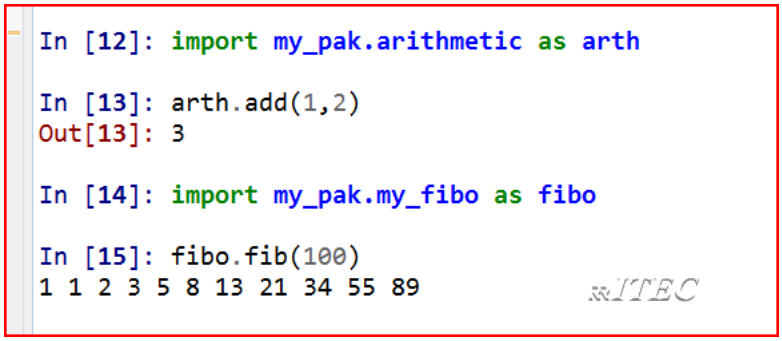
4. Create one more module with the name of **my_fibo.py**



5. The folder looks as shown



6. Import and use it

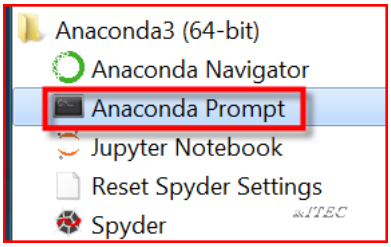


6.4.2 System defined Packages

1. Thousands of predefined packages available https://pypi.python.org/pypi

    1. **NumPy** → NumPy is the fundamental package for scientific computing with Python(example array)
    2. **Scipy** → Scientific Python distributions
    3. **Sympy** → Python library for symbolic mathematics
    4. **Matplotlib** → Python plotting package
    5. **Scikit-learn** → machine learning
    6. **Math** → Mathematical function (example pi, sin, cos ...etc)
    7. **xlrd → to work with excels**
    8. **keras → Deep Learning /neural networks**
    9. **nltk → natural language tool kit (NLP,NLU)**
    10. **rasa_nlu → NLP,NLU**
    11. **TensorFlow → programming environment**
    12. **..etc**

13. As we know anaconda software = python software + basic packages required for data science
14. To see list of packages already installed in anaconda → start → all programs → anaconda folder → right click on **Anaconda Prompt** → run as administrator



15. Type conda list

```
(C:\Anaconda3) C:\Users\Hi>conda list
```

16. Too see particular package type conda list <package name> (example see matplotlib)

```
(base) C:\Users\Hi>conda list matplotlib
# packages in environment at C:\Anaconda3:
#
# Name                    Version                   Build  Channel
matplotlib                2.1.0            py36h11b4b9c_0
```

17. To install package not available in work area
    i. Right click on anaconda prompt → run as administrator
    ii. Cond install <package name> or get command from google
    iii. Example xlrd not came with installation of anaconda
18. To install package not available in work area and in anaconda
    i. Right click on anaconda prompt → run as administrator
    ii. pip install <package name> or get command from google
    iii. Example rasa_nlu not available in anaconda
19. To see functions of a module
    i. Open spyder
    ii. Run below code

```
import math as math
for x in dir(math):
    print(x)
```

2. **Exercise 1: Import Package and calculate circumference and area of the circle**

Py7_1_Import_Ppackage

```
# Definition of radius
r = 0.42
# Import the math package
import math
# Calculate Circumference of circle
C = 2 * r * math.pi
# Calculate Area of circle
A = math.pi * r ** 2
# Build printout
print("Circumference of circle: " + str(C))
print("Area of circle: " + str(A))
```

3. **Exercise 2: Import one function from Package and calculate circumference and area of the circle**

Py7_2_Import_One_function_from_Ppackage

```
# Definition of radius
r = 0.42
# Import the pi function from math package
from math import pi
# Calculate Circumference of circle
C = 2 * r * pi
# Calculate Area of circle
A = pi * r ** 2
# Build printout
print("Circumference of circle: " + str(C))
print("Area of circle: " + str(A))
```

4. **Exercise 3: Import Package function using alias name and calculate circumference and area of the circle**

Py7_3_Import_Package_Function_Using_Alias_Names

```
# Definition of radius
r = 0.42
# Import the pi function from math package
from math import pi as my_pi
# Calculate Circumference of circle
C = 2 * r * my_pi
# Calculate Area of circle
A = my_pi * r ** 2
# Build printout
print("Circumference of circle: " + str(C))
print("Area of circle: " + str(A))
```

5. **Create project using spyder**

    1. In spyder go to project menu → new Project → provide name as spy_project → click on create

2. Right click on the project → new → package → provide name as spy_package → click on ok
3. Right click on the package → click on new → click on module → name it as spy_module
4. Rest of the process like importing is same as above

7. Dictionary

1. Why we need dictionary datatypes?

Exercise 1: Get India population using list? Is it convenient?

Py21_1_why_we_need_dictionary

```
1 #Store country names and population using list
2 pop_billions = [1.38,1.34,0.32]
3 countries = ["China", "India", "USA"]
4 #Get India element index and print it
5 India_pop = countries.index("India")
6 print(India_pop)
7 #Print India population
8 print(pop_billions[India_pop])
```

Exercise 2: Create simple dictionary?

Py21_2_Create_Simple_Dictionary

```
1 #Create dictionary
2 world = {"China":1.38,"India":1.34,"USA":0.32}
3 #print data type
4 print(type(world))
5 # Print out the keys in world
6 print(world.keys())
7 # Print out the values in world
8 print(world.values())
9 #Get values using keys: Print India population
10 print(world["India"])
```

2. Observations of dictionary:

   a. Each key is separated from its value by a colon (:), the items/ elements are separated by commas and the whole thing is enclosed in curly braces.

   b. An empty dictionary without any items is written with just two curly braces, like this: {}.

   c. Keys are unique within a dictionary while values may not be.

   d. **Syntax of dictionary:** my_dict = {

   "key1":"value1",

   "key2":"value2",

   }

   e. **Refer : https://docs.python.org/3/tutorial/datastructures.html**

Exercise 3: Insert, Update and delete dictionary items

Py21_3_Insert_Update_Delete_Dictionary_Items

```
#Create dictionary
world = {"China":1.38,"India":1.34,"USA":0.32}
#print data type
print(world)
# Add one item to Dictionary
world["Indonesia"]=0.26
world["Brazil"]=0.21
#Indonesia available in world dictionary
print('Indonesia' in world)
print(world)
# update one item of world Dictionary
world["India"]=1.39
print(world)
# delete one item of world Dictionary
del(world["Brazil"])
print(world)
```


Exercise 4: Nested dictionaries

Py21_4_Nested_Dictionaries

```
# Dictionary of dictionaries
world = {"China":{'Capital':'Beijing','Population':1.38},\
         "India":{'Capital':'New Delhi','Population':1.34},\
         "USA":{'Capital':'Washington, D.C','Population':0.32}}
# Print out the capital of India
print(world['India']['Capital'])
# Create sub-dictionary data
data = { 'capital':'rome', 'population':59.83 }
# Add data to world under key 'italy'
world['italy'] = data
# Print world
print(world)
```

Exercise 5: Create dataframe using dictionaries

Py21_5_Create DataFrame_Using_Dictionaries

```
# Pre-defined lists
country_names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']
drives_right =  [True, False, False, False, True, True, True]
cars_avg_per_1000 = [809, 731, 588, 18, 200, 70, 45]
# Import pandas as pd
import pandas as pd
# Create dictionary my_dict with three key:value pairs: my_dict
my_dict = { 'country':country_names,\
            'drives_right':drives_right,\
            'cars_per_1000':cars_avg_per_1000 }
# Build a DataFrame cars from my_dict: cars
cars = pd.DataFrame(my_dict)
# Definition of row_labels
row_labels = ['US', 'AUS', 'JAP', 'IN', 'RU', 'MOR', 'EG']
# Specify row labels of cars
cars.index = row_labels
# Print cars again
print(cars)
```

8. Lambda functions

1. Python supports the creation of anonymous functions (i.e. functions that are not bound to a name)
2. While normal functions are defined using the **def** keyword, in Python anonymous functions are defined using the **lambda** keyword. Hence, anonymous functions are also called lambda functions
3. Syntax: **lambda arguments: expression**
4. Lambda functions used with typical functional concepts like map(), filter() and reduce()

Exercise 1: Simple lambda function

```
# Regular Way

def raise_to_power(x,y):
    return x ** y

raise_to_power(2,3)

# Replication using lambda function

raise_to_power = lambda x,y: x ** y

raise_to_power(2,3)
```


Here :

1. x, y are arguments
2. x ** y is body part of the function

Exercise 2: lambda function with **map** function

 # define function

def power(i):

return i ** 2

 # Can you apply above function on python list?

a_list = [10,20,30,40]

power(a_list)

 # cam you apply above function on numpy array

```
import numpy as np

a_np_list = np.array(a_list)

power(a_np_list)

# can you avoid above defining function and converting list into numpy list using map

rs = map(lambda i: i ** 2,a_list)

rs

list(rs)
```

Exercise 3: lambda function with **filter** function

```
# Regular way

nums= [3,6,4,10,1]

def more_than_5(x):
    y =[]
    for i in x:
        if i>5:
            y.append(i)
    return y

more_than_5(nums)

# Using filter function

nums= [3,6,4,10,1]
more_than_5 = filter(lambda num: num >5, nums)
list(more_than_5)
```

Exercise 4: lambda function with **reduce** function : Find product of first 10 prime numbers

```
# define first 10 prime numbers

a_list_prime_num = [2,3,5,7,11,13,17,19,23,29]

# regular way

product = 1

for i in a_list_prime_num:
```

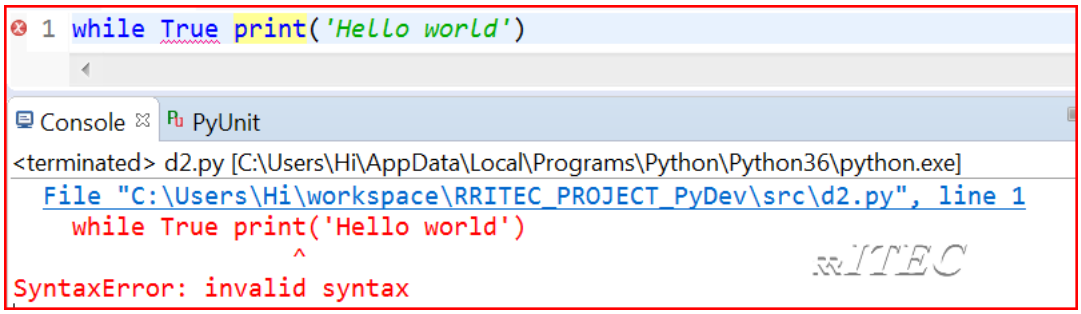
```
    product = product * i
product
# using reduce function
from functools import reduce
reduce(lambda x,y:x *y,a_list_prime_num)
```

9. Syntax Errors and Exceptions

1. There are (at least) two built in distinguishable kinds of errors: **syntax errors** and **exceptions**.

    a. Syntax Errors

        i. Syntax errors, also known as parsing errors



        ii. Try below example1 and observe syntax error

```
print(hello World)
```

        iii. Try below example 2 and observe syntax error

```
for i as "python":
    print(i)
```

    b. Exceptions

        i. Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called exceptions

        ii. Some of the exceptions are

            1. ZeroDivisionError → try it `10 * (1/0)`
            2. NameError → `4 + spam*3`
            3. TypeError → `'2' + 2`
            4. IndentationError → ` 2 + 2` (type space beginning of the line)
            5. Find more errors
               https://docs.python.org/3/library/exceptions.html#bltin-exceptions

2. User Defined Exceptions

a. Catch exceptions with try-except clause

   i. Runs the code following try

   ii. If there's an exception, run the code following except

b. **Exercise 1: user defined exception message**

*Py27_1_Errors_and_exceptions

```
1  def sqrt(x):
2      """Returns the square root of a number."""
3      try:
4          print(x ** 0.5)
5      except:
6          print('x must be an int or float')
7  print("***** 01 Execute with integer value******")
8  sqrt(9)
9  print("***** 02 Execute with string value******")
10 sqrt("ABCD")
```

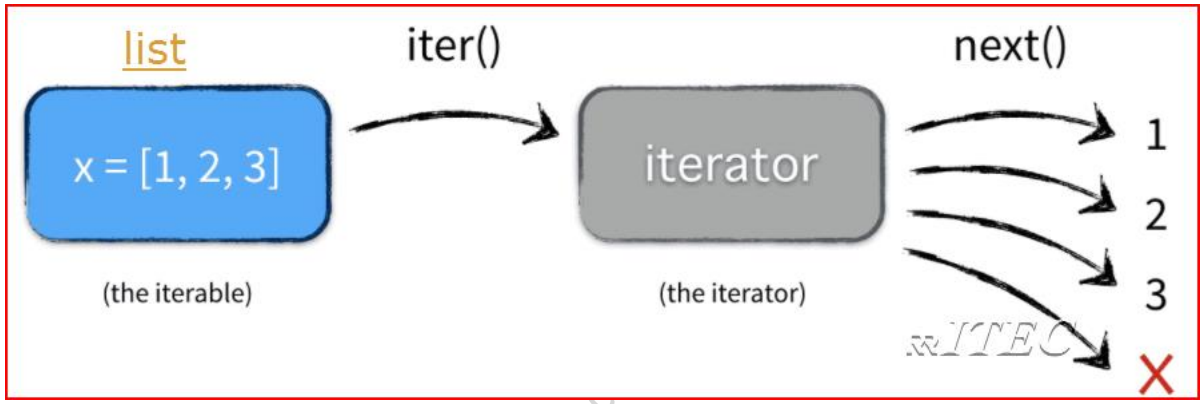
c. **Exercise 2: link user defined exception message with built-in error codes**

Py27_2_Errors_and_exceptions_raise

```
1  def sqrt(x):
2      """Returns the square root of a number."""
3      if x < 0:
4          raise ValueError('x must be non-negative')
5      else:
6          print(x ** 0.5)
7
8  print("***** 01 Execute with integer value******")
9  sqrt(9)
10 print("***** 02 Execute with negative value******")
11 sqrt(-9)
```

10. Iterables & Iterators

1. Iterable
    a. Iterable is an object with an associated **iter() method**
        i. Examples: lists, strings, tuples, dictionaries, range object and file connections.
    b. Applying iter() to an iterable creates an iterator



2. Iterator
    a. Produces next value with next()
    b. It is simple and faster than for loop
3. **Exercise 1:** list iterator

```
# Create a list of strings: names
names = ['Ram', 'Rabson', 'Robert', 'Nancy']
# Print each list item in names list using a for loop
print("****** 01 For Loop Output ******")
for person in names:
    print(person)
print("****** 02 Replicate for loop output with list_iterator Output ******")
# Create an iterator for names list: avoid_for
avoid_for = names.__iter__() # or iter(names)
```

```
print(type(avoid_for))
# Print each item from the iterator
print(next(avoid_for))
print(next(avoid_for))
print(next(avoid_for))
print(next(avoid_for))
```

4. **Exercise 2:** range iterator

```
# Loop over range(5) and print the values
print("****** 01 For Loop Output ******")
for num in range(5):
    print(num)
# Create an iterator for range(5): range_iterator
print("****** 02 Replicate for loop output with range_iterator Output ******")
range_iterator = iter(range(5))
print(type(range_iterator))
# Print the values in range_iterator
print(next(range_iterator))
print(next(range_iterator))
print(next(range_iterator))
print(next(range_iterator))
print(next(range_iterator))
```


5. **Exercise 3: Working with String iterator**

```
a_str = "python"
```

```
a_str_iter = a_str.__iter__()
next(a_str_iter)
next(a_str_iter)
next(a_str_iter)
next(a_str_iter)
next(a_str_iter)
next(a_str_iter)
```

6. **Exercise 4: Tuple iterator**

```
a_tuple = (10,20,30)
a_tuple_iter=iter(a_tuple)
next(a_tuple_iter)
next(a_tuple_iter)
next(a_tuple_iter)
next(a_tuple_iter)
```


7. **Exercise 5 Dictionary iterator**

```
a_dict = {1:"ram",2:"Nancy",3:"rabson"}
a_dict_iter = iter(a_dict)
next(a_dict_iter)
next(a_dict_iter)
next(a_dict_iter)
next(a_dict_iter)
```

8. Exercise 6: Use iterators as function arguments

```
# Create a range object iterator: range_object
range_object = range(10, 21)
print(type(range_object))
```

```
# Print the range object

print(range_object)

# use above iterator object as a parameter of list ,max functions

# Create a list of integers: range_object_list

range_object_list = list(range_object)

# Print range_object_list

print(range_object_list)

# Get the max of values: range_object_list_max

range_object_list_max = max(range_object)

# Print range_object_list_max

print(range_object_list_max)
```


9. **Exercise 7: enumerate function**

   a. Enumerate function returns an enumerate object that produces a sequence of tuples, and each of the tuples is an index-value pair.

```
# Create a list of strings: DataScience

DataScience = ['PYTHON','R','MATHS','STATS','ML','DL','NLU','NLP','AI']

print("***** 01. enumerate() function creates enumarate object *****")

print(enumerate(DataScience))

print( "***** 02. Create a list of tuples ***** ")

DataScience_list = list(enumerate(DataScience))

print(type(DataScience_list))

# Print the list of tuples

print(DataScience_list)

print("***** 03. Unpack and print the tuple pairs *****")

for index1, value1 in enumerate(DataScience):
```

```
    print(index1, value1)
print("***** 04. Unpack and print with given  start index *****")
for index2, value2 in enumerate(DataScience, start=1):
    print(index2, value2)
```

10. Exercise 8: Zip Function

    a. zip() function creates zip object

    b. Zip object is an iterator of tuples.

    c. If you want to print the values of a zip object, you can convert it into a list and then print it

```
# Create a list of tuples
std_no = (10,20,30)
std_name = ("Ram","Nancy","Rabson")
std_marks = (100,2000,300)
#create zip object
std_data = zip(std_no, std_name, std_marks)
print("***** 01. Print the list of tuples create by zip function *****")
print(type(std_data))
print(std_data)
print(list(std_data))
print("***** 02. Unpack the zip object and print the tuple values *****")
std_data1 = zip(std_no, std_name, std_marks)
for value1, value2, value3 in std_data1:
    print(value1, value2, value3)
```


11. Exercise 9: unzip function

    a. There is no unzip function for doing the reverse of what zip()

b. However we can use * and zip() to unzip

```
std_name = ("Ram","Nancy","Rabson")

std_course = ("R","PYTHON","DATASCIENCE")

# Create a zip object : z1

z1 = zip(std_course, std_name)

print("01. tuples in z1 after unpacking with *")

print(*z1)

# Re-create a zip object : z1

z1 = zip(std_course, std_name)

# 'Unzip' the tuples in z1 by unpacking with * and zip(): result1, result2

result1, result2 = zip(*z1)

print("02. Check if unpacked tuples are equivalent to original tuples")

print(result1 == std_course)

print(result2 == std_name)
```

11. List comprehensions

1. List comprehension is an simple way to define and create list in Python
2. In mathematics the square numbers of the natural numbers are for example created by { x^2 | x ∈ ℕ } or the set of complex integers { (x,y) | x ∈ ℤ , y ∈ ℤ }.
3. Guido van Rossum prefers list comprehensions and does not like lambda functions

4. List comprehension is a complete substitute for the lambda function as well as the functions map(), filter() and reduce()

Exercise 1: List Comprehension to avoid loop or lambda functions. Find square of each list item.


```
nums = [10,20,30,40,50]

# using for loop

new_nums = []

for num in nums:

    new_nums.append(num ** 2)

new_nums

# using lambda functions

a = map(lambda x: x **2,nums)

list(a)

# using list comprehension

new_nums1 = [num ** 2 for num in nums]
```

```
new_nums1
```

Exercise 2: List Comprehension to avoid nested loops

```
print("01. nested loops ")
pairs_1 = []
for num1 in range(0, 2):
    for num2 in range(6, 8):
        pairs_1.append((num1, num2))
print(pairs_1)
print("02. nested loops using list comprehension ")
pairs_2 = [(num1, num2) for num1 in range(0, 2) for num2 in range(6, 8)]
print(pairs_2)
```


Exercise 3: List Comprehension with conditions

```
# list comprehension with if statement
print([num ** 2 for num in range(10) if num % 2 == 0])
print("positive and negative numbers")
pos_neg = [{num: -num for num in range(10)}]
print(pos_neg)
print(type(pos_neg))
```

Refer: **https://docs.python.org/3/tutorial/datastructures.html**

12. Generators

1.Generator functions allow you to declare a function that behaves like an iterator, i.e. it can be used in a for loop

2.list comprehensions and generator expressions look very similar in their syntax, except for the use of parentheses () in generator expressions and brackets [] in list comprehensions

3.A list comprehension produces a list as output, a generator produces a generator object.

```
l = [n*2 for n in range(1000)] # List comprehension

g = (n*2 for n in range(1000)) # Generator expression

print(type(l))  # <type 'list'>

print(type(g))  # <type 'generator'>

import sys

print(sys.getsizeof(l))  # 9032

print(sys.getsizeof(g))  # 80

print(l[4])   # 8

print(g[4])   # TypeError: 'generator' object has no attribute '__getitem__'

for v in l:

    print(v)

for v in g:

    print(v)
```


4.Exercise 2: work with list and generator

```
# List of strings

std = ['Ram Reddy', 'Rabson', 'Robert', 'Nancy']

# List comprehension

std1 = [member for member in std if len(member) > 5]
```

```
print(type(std1))

print(std1)

# Generator expression

std2 = (member for member in std if len(member) > 5)

print(type(std2))

list(std2)
```

5.Read items of generator object using next or for loop

```
# Create generator object: result

result = (num for num in range(10))

print("***** 01 First 5 values of generator ***** ")

print(next(result))

print(next(result))

print(next(result))

print(next(result))

print(next(result))

print("***** 02 Print the rest of the values using for Loop *****")

for value in result:

    print(value)
```

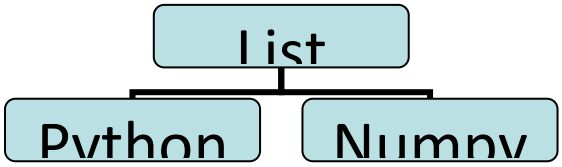

6.create a generator function with a similar mechanism as the generator expression

```
# Create a list of strings

std = ['Ram Reddy', 'Rabson', 'Robert', 'Nancy']

# Define generator function get_lengths

def get_lengths(input_list):

    """Generator function that yields the
```

```
    length of the strings in input_list."""
    # Yield the length of a string
    for person in input_list:
        yield len(person)
# Print the values generated by get_lengths()
for value in get_lengths(std):
    print(value)
```

Level 03 of 08: Python Packages for Data Science

13. NumPy package



Lists Lists

1. Data Science needs, Mathematical operations over collections of values in utmost no time. It is not possible with regular list.

2. **Exercise 1: Calculate BMI using List. Is it possible?**

```
Py8_1_Calculate_BMI_Using_List
1 #calculate BMI using list
2 #is it possible?????
3 height = [1.71, 1.58, 1.51, 1.73, 1.92]
4 weight = [65.9, 58.8, 66.8, 83.4, 68.7]
5 BMI=weight/height **2

Console  PyUnit
<terminated> Py8_1_Calculate_BMI_Using_List.py [C:\Users\Hi\AppData\Local\Programs\Python\Python36\python.exe]
Traceback (most recent call last):
  File "C:\Users\Hi\workspace\RRITEC_PROJECT_PyDev\src\Py8_1_Calculate_BMI_Using_List.py",
    BMI=weight/height **2
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

3. **Exercise 2: Calculate BMI using NumPy Array. Is it possible?**

*Py8_2_Calculate_BMI_Using_NumPy_Array

```
# Create list of heights of students
height = [1.71, 1.58, 1.51, 1.73, 1.92]
# Create list of weights of students
weight = [65.9, 58.8, 66.8, 83.4, 68.7]
# Import the NumPy package
import numpy
# Create a NumPy array
np_height = numpy.array(height)
np_weight = numpy.array(weight)
# Print out type of np_height
print(type(np_height))
print(type(np_weight))
#calculate Body Mass Index(BMI)
BMI=np_weight/np_height **2
print(BMI)
# Reading BMI
print(BMI[1]) # Reading one element of array
print(BMI[1:3]) # Reading subset of array
print(BMI[BMI>28]) #Reading Array based on Condition
```


4. NumPy main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.

5. NumPy stands for Numerical Python

6. NumPy array contains only one data type elements whereas Python array contains elements of different data types

```
import numpy as np
l = [1,2,3.9]
n = np.array([1,2,3.9])
print(l)  # out : [1, 2, 3.9]
print(n)  # out : [ 1.   2.   3.9]
```


7. NumPy is a powerful N-dimensional array object

```
import numpy as np
a = np.arange(6)                        # 1d array
b = np.arange(12).reshape(4,3)          # 2d array
c = np.arange(24).reshape(2,3,4)        # 3d array
d = np.arange(24).reshape(2,3,2,2)      # 4d array

#Observe arrays
print(a)
print(b)
print(c)
print(d)

# Observe Number of dimensions
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

8. NumPy array allows Calculations over entire elements, Easy, Fast and occupies less memory

```
import sys
import numpy as np
l = [1,2,3,4,5,6,7,8,9,10,11,12]
n = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
print(sys.getsizeof(l))  # out : 160
print(sys.getsizeof(n))  # out : 140
```

9. **Exercise 3: Working with 2D NumPy Arrays.**

*Py8_3_Working_With_2DNumPy_Array

```
1  # Import the NumPy package
2  import numpy
3  #Creating NumPy 2D Array
4  np_2d_array = numpy.array([[1.71, 1.58, 1.51, 1.73, 1.92],[65.9, 58.8, 66.8, 83.4, 68.7]])
5  print(type(np_2d_array))
6  # Dimensions of array (rows and columns)
7  print(np_2d_array.shape)
8  # Reading 2D Array elements
9  print(np_2d_array[0])  # 1 row of elements
10 print(np_2d_array[0][2]) # 1 row 3 column element
11 print(np_2d_array[:,0:2]) # All rows and 1st ,2nd columns
12 print(np_2d_array[1,:]) # All columns and 2nd row
```

10. **Exercise 4: Working with basic statistics.**

Py8_4_Working_With_Statistics

```
1 import numpy
2 x = [1, 4, 8, 10, 12]
3 print(numpy.mean(x))
4 print(numpy.median(x))
5 print(numpy.std(x))
```

11. https://www.mathsisfun.com/definitions/mean.html

12. https://www.mathsisfun.com/median.html

13. https://www.mathsisfun.com/data/standard-deviation.html

    "Standard Deviation is a "standard" way of knowing what is normal, what is extra-large and what is extra small"

14. **Exercise 5: Working with random numbers with out seed**

```
# generate 5 random integers between 10(Inclusive) and 50(Exclusive)

print(np.random.randint(10,50,5))

# run again same command and observe is it giving same result?
```

```
print(np.random.randint(10,50,5))
```

15. **Exercise 6: Working with random numbers with seed**

```
# How to reproduce same random numbers in multiple runs? Let us try
# generate 5 random integers between 10(Inclusive) and 50(Exclusive)
np.random.seed(100)
print(np.random.randint(10,50,5))
# run again same command and observe is it giving same result?
np.random.seed(100)
print(np.random.randint(10,50,5))
```

16. **Exercise 6: working with linear algebra**

```
# Create a matrix with 0(Inclusive) to 12(Exclusive) integers
a = np.arange(12).reshape(3,-1) # -1 means "whatever is needed"
print(a)
# Create a matrix with 12 random numbers
b = np.array(np.random.rand(12)).reshape(3,-1) # -1 means "whatever is needed"
print(b)
# Add these two matrices
print(a+b)
```


17. https://docs.scipy.org/doc/numpy/user/quickstart.html

14. Pandas (powerful Python data analysis toolkit)

14.1 Introduction

1. Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
2. Using pandas we can replicate any SQL query

14.2 Slicing Dataframe

1. Execute below **4** queries and observe the results

```
# 01. load csv as dataframe

import pandas as pd

df = pd.read_csv("pandas_sales.csv",index_col = 'month')

df

# 02. Read dataframe

# Method 1: by index # [columnname][rowname]

df['eggs']['May']

df['spam']

# Method 2: by column attribute and row index

df.salt[0]

df.salt[0:3] # slice

df.salt[[0,3,5]] # fancy

# Method 3: by using loc

df.loc['Jan','eggs']

df.loc['Jan':'May','eggs':'spam'] # slice

df.loc[['Jan','May'],['eggs','spam']] # fancy

# Method 4: bu using iloc

df.iloc[1,1]

df.iloc[1:3,1:3] # by slice
```

```
df.iloc[[1,3],[0,2]] # by fancy

# 03. Create dataframe from dataframe [[  ]]

df_new = df[['eggs','spam']]

df_new

# 04. Create series(1 column +  index)  from dataframe [ ]

series = df['eggs']

series
```

14.3 Filtering Dataframe

1.Execute below **11** queries and observe the results

*Py31_5_Filtering_DataFrame

```
import pandas as pd
df = pd.read_csv('pandas_sales.csv', index_col='month')
print("***** 01. observe simple dataframe *****")
print(df);    print("***** 02. Filter data *****")
print(df.salt > 60) #Creating a Boolean Series
print(df[df.salt > 60]) #Filtering with a Boolean Series
print("***** 03. Combining filters *****")
print(df[(df.salt >= 50) & (df.eggs < 200)]) # AND conditions
print(df[(df.salt > 50) | (df.eggs < 50)]) # OR condition
print("***** 04. DataFrames with zeros and NaNs *****")
df2 = df.copy()
print(df2 == df) #note that one NaN != NaN
df2['cake'] = [0, 0, 50, 60, 70, 80] # add one column to dataframe
print(df2)
print("***** 05. Select columns with all nonzeros *****")
print(df2.loc[:, df2.all()])
print("***** 06. Select columns with any nonzeros *****")
df3 = df2.copy()
df3['milk'] = [0, 0,0,0,0,0]
print(df3)
print(df3.loc[:, df3.any()])
print("***** 07. Select columns with any NaNs *****")
print(df.loc[:, df.isnull().any()])
print("***** 08. Select columns without NaNs *****")
print(df.loc[:, df.notnull().all()])
print("***** 09. Drop rows with any NaNs *****")
print(df3.dropna(how='any'))
print("***** 10. Filtering a column based on another *****")
print(df.eggs[df.salt > 55])
print("***** 11. Modifying a column based on another *****")
df.eggs[df.salt > 55] += 5
print(df)
```

14.4 Transforming Dataframe

1.Execute below **8** queries and observe the results

Py31_6_Transforming_DataFrame

```
1  import pandas as pd
2  df = pd.read_csv('pandas_sales.csv', index_col='month')
3  print("***** 01. observe simple dataframe *****")
4  print(df)
5  print("***** 02. DataFrame vectorized methods *****")
6  print(df.floordiv(12)) # Convert to dozens unit
7  print("***** 03. NumPy vectorized functions *****")
8  import numpy as np
9  print(np.floor_divide(df, 12)) # Convert to dozens unit
10 print("***** 04 - 1. Plain Python functions *****")
11 def dozens(n):
12     return n//12
13 print(df.apply(dozens))
14 print("***** 04 - 2. Plain Python functions *****")
15 print(df.apply(lambda n: n//12))
16 print("***** 05. Storing a transformation *****")
17 df['dozens_of_eggs'] = df.eggs.floordiv(12)
18 print(df)
19 print("***** 06. The DataFrame index *****")
20 print(df.index)
21 print("***** 07 - 1. Working with string values *****")
22 df.index = df.index.str.upper()
23 print(df)
24 print("***** 07 - 2. Working with string values *****")
25 df.index = df.index.map(str.lower)
26 print(df)
27 print("***** 8. Defining columns using other columns *****")
28 df['salty_eggs'] = df.salt + df.dozens_of_eggs
29 print(df)
30
```

14.5 Advanced indexing

1.In pandas Data Structures ,Key building blocks are

a. Indexes: Sequence of labels
b. Series: 1D array with Index (index + 1 column)
c. DataFrames: 2D array with Series as columns

2.Indexes

a. Immutable (Like dictionary keys)
b. Homogenous in data type (Like NumPy arrays)

3.Exercise 1: create index manually

```
import pandas as pd

prices = [10,12,13,11,9]

type(prices)


# 01. Creating a series

shares = pd.Series(prices)

shares

type(shares)


# 02. Creating an index

days = ['Mon','Tue','Wed','Thur','Fri']

shares = pd.Series(data = prices,index = days)

shares


# 03. Read indexes

shares.index
```

```
shares.index[1]

shares.index[:3]

shares.index[:-3] # from right side 3 items will be removed

shares.index[-2:]


# 04. Index names

shares.index.name

shares.index.name = 'weekday'

shares.index.name

shares


# 05.Indexs are immutable*

shares.index[2] ='Wednesday' #TypeError: Index does not support mutable operations

shares.index[:2] =['Monday','Tuesday']#TypeError: Index does not support mutable operations


# 06.Modifing all index entries

shares.index = ['Monday','Tuesday','Wednesday','Thursday','Friday']

shares
```


4.Exercise 2: Create index from file import

```
import pandas as pd

df = pd.read_csv("pandas_sales.csv",index_col= 'month')

df
```

5.Exercise 3: Hierarchical indexing

```
# 01. load data
import pandas as pd
df = pd.read_csv("pandas_sales_hierarchical_indexing.csv")
df


# 02. Create index using two columns (similar to composite key in RDBMS)
df = df.set_index(['state','month'])
df


# 03. Read index of dataframe
df.index
df.index.name
df.index.names


# 04. Sort indexes
df = df.sort_index()


# 05. Reading
# 05-01. Using index method
df.loc['CA',1]
df.loc[('CA',1),'salt']
# 05-02. Using slice
df.loc['CA']
df.loc['CA':'NY']
```

```
# 05-03. Fancy

df.loc[(['CA','TX'],1),:]

df.loc[(['CA','TX'],1),'eggs']

df.loc[('CA',[1,2]),:]
```

14.6 Stack and unstack

1.Recall pivot

```
# 01. import required modules and load data

import pandas as pd

df = pd.read_csv("pandas_users.csv")

df


# 02. weekday and city wise, Number of visitors

visitors_pivot = df.pivot(index = 'weekday',columns = 'city',values = 'visitors')

visitors_pivot


# 03. weekday and city wise, Number of visitors

signups_pivot = df.pivot(index = 'weekday',columns='city',values = 'signups')

signups_pivot


# 04.weekday and city wise, number of visitors and signups

visitors_signups_pivot = df.pivot(index = 'weekday',columns='city')

visitors_signups_pivot
```


2.Stack is useful to work with multi-level index

```
#01. import required modules and load data

import pandas as pd

df = pd.read_csv("pandas_users.csv")

df


#02. set multi level index

df = df.set_index(['city','weekday'])

df

#03. by week day

byweekday = df.unstack(level = 'weekday')

byweekday

04.  revert it

df1 = byweekday.stack(level = 'weekday')

df1
```


3.Recall melt and pivot_table from cleaning data chapter

14.7 Groupby and aggregations

1.Exercise 1: work with groupby and aggregations

```
import pandas as pd

sales = pd.DataFrame({

'weekday': ['Sun', 'Sun', 'Mon', 'Mon'],

'city': ['Austin', 'Dallas', 'Austin', 'Dallas'],

'bread': [139, 237, 326, 456],

'butter': [20, 45, 70, 98]

})

# 01. Observe Sales Data

sales

# 02. Boolean filter and count

sales.loc[sales['weekday'] == 'Sun'].count()

# 03. Groupby and count

sales.groupby('weekday').count()

# 04. Groupby and sum #####")

sales.groupby('weekday')['bread'].sum()

# 05. Groupby and sum on multiple columns

sales.groupby('weekday')[['bread','butter']].sum()

# 06. Groupby and mean: multi-level index

sales.groupby(['city','weekday']).mean()

# 07. group by and max on multiple columns

sales.groupby('city')[['bread','butter']].max()

# 08. Multiple aggregations

sales.groupby('weekday')[['bread','butter']].agg(['max','min'])

# 09. Custom aggregation
```

```
def data_range(series):
    return series.max() - series.min()
sales.groupby('weekday')[['bread', 'butter']].agg(data_range)
```

3.Refer

a. Official website http://pandas.pydata.org/

b. Read http://pandas.pydata.org/pandas-docs/stable/

c. Must read & practice http://pandas.pydata.org/pandas-docs/stable/10min.html

d. http://pandas.pydata.org/pandas-docs/stable/cookbook.html

4.Exercise 1: Py31_1_some_graphs_using_iris_data

a. From labdata use above script understand different graphs

5.Exercise 2: Py31_2_iris_statistics

a. From labdata use above script understand different statistics

6.Exercise 3: Py31_3_iris_statistics

a. From labdata use above script understand different statistics

15. Matplotlib data visualization

1. Data Visualization is a key skill for aspiring data scientists.
2. Using this library we can develop any report of BI tools (Tableau/Power BI/OBI/BO/Cognos …etc)
3. Advanced packages are seaborn/bokeh
4. Matplotlib makes it easy to create meaningful and insightful plots
5. Matplotlib is a library for making 2D plots of arrays in Python.
6. Matplotlib came from MATLAB
7. Refer for lot of information☺ . https://matplotlib.org/users/intro.html
8. See help document of particular graph (for example **help(plt.hist))**

Exercise 1: Line Chart

*Py20_1_Matplotlib_line_Graph

```
#import required packages
import matplotlib.pyplot as plt
#create two simple lists
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
#plot line graph
plt.plot(year, pop)
# Add title
plt.title("Year wise population")
# Add axis labels
plt.xlabel("Year")
plt.ylabel("Population")
# Definition of tick_val and tick_lab
tick_val = [1950, 1970, 1990, 2010]
tick_lab = ['1950year','1970year','1990year','2010year']
# Adapt the ticks on the x-axis
plt.xticks(tick_val, tick_lab)
# Adapt the ticks on the Y-axis
plt.yticks([2,3,4,5,6,7],['2B', '3B','4B', '5B','6B','7B'])
#show the graph
plt.show()
```

Exercise 2: Scatter Chart

*Py20_2_Matplotlib_Scatter_Graph

```
import matplotlib.pyplot as plt
# Import numpy as np
import numpy as np
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
# Store pop as a numpy array: np_pop
np_pop = np.array(pop)
# To see clear size changes ,increase counts of  np_pop
np_pop = np_pop * 20
plt.scatter(year, pop,s = np_pop)
plt.xlabel('year')
plt.ylabel('population]')
plt.title('year wise population')
#Name the dots
plt.text(1950, 2.519, 'India')
plt.show()
```


Exercise 3: Histogram Chart

*Py20_3_Matplotlib_Histogram_Graph

```
#import required packages
import matplotlib.pyplot as plt
# define 1 to 100 prime numbers as a list
values = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,\
          53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
#plot histogram
plt.hist(values, bins = 5)
plt.show()
#clear plot
plt.clf()
#plot histogram
plt.hist(values) # default value of bins is 10
plt.show()
plt.clf()
```


Exercise 4: Multiple plots on single axis

```
# Load data frame from csv file

import pandas as pd

df = pd.read_csv("percent-bachelors-degrees-women-usa.csv")

year = df["Year"]

physical_sciences= df["Physical Sciences"]

computer_science= df["Computer Science"]

# Import matplotlib.pyplot
```

```
import matplotlib.pyplot as plt


# Plot in blue the % of degrees awarded to women in the Physical Sciences
plt.plot(year, physical_sciences, color='blue')
# Plot in red the % of degrees awarded to women in Computer Science
plt.plot(year, computer_science, color='red')
# Display the plot
plt.show()
```

Exercise 5: Different line plots on distinct axes **using axes**

```
# Create plot axes for the first line plot
plt.axes([0.05,0.05,0.425,0.9])
# Plot in blue the % of degrees awarded to women in the Physical Sciences
plt.plot(year, physical_sciences, color='blue')
# Create plot axes for the second line plot
plt.axes([0.525,0.05,0.425,0.9])
# Plot in red the % of degrees awarded to women in Computer Science
plt.plot(year, computer_science, color='red')
# Display the plot
plt.show()
```


Exercise 6: Different line plots on distinct axes **using subplot**

```
# Create a figure with 1x2 subplot and make the left subplot active
plt.subplot(1, 2, 1)
# Plot in blue the % of degrees awarded to women in the Physical Sciences
plt.plot(year, physical_sciences, color='blue')
```

```
plt.title('Physical Sciences')
# Make the right subplot active in the current 1x2 subplot grid
plt.subplot(1, 2, 2)
# Plot in red the % of degrees awarded to women in Computer Science
plt.plot(year, computer_science, color='red')
plt.title('Computer Science')
# Use plt.tight_layout() to improve the spacing between subplots
plt.tight_layout()
plt.show()
```

Exercise 7: Different line plots on distinct axes **using subplot with 2 * 2 grid**

```
#Createtwo more series objects
education= df["Education"]
health = df["Health Professions"]
# Create a figure with 2x2 subplot layout and make the top left subplot active
plt.subplot(2, 2, 1)
# Plot in blue the % of degrees awarded to women in the Physical Sciences
plt.plot(year, physical_sciences, color='blue')
plt.title('Physical Sciences')
# Make the top right subplot active in the current 2x2 subplot grid
plt.subplot(2, 2, 2)
# Plot in red the % of degrees awarded to women in Computer Science
plt.plot(year, computer_science, color='red')
plt.title('Computer Science')
# Make the bottom left subplot active in the current 2x2 subplot grid
plt.subplot(2, 2, 3)
```

```
# Plot in green the % of degrees awarded to women in Health Professions
plt.plot(year, health, color='green')
plt.title('Health Professions')
# Make the bottom right subplot active in the current 2x2 subplot grid
plt.subplot(2, 2, 4)
# Plot in yellow the % of degrees awarded to women in the Education
plt.plot(year, education, color='yellow')
plt.title('Education')
# Improve the spacing between subplots and display them
plt.tight_layout()
plt.show()
```


Exercise 8: Using **xlim and ylim**

```
# Plot the % of degrees awarded to women in Computer Science and the Physical Sciences
plt.plot(year,computer_science, color='red')
plt.plot(year, physical_sciences, color='blue')
# Add the axis labels
plt.xlabel('Year')
plt.ylabel('Degrees awarded to women (%)')
# Set the x-axis range
plt.xlim(1990,2010)
# Set the y-axis range
plt.ylim(0,50)
# Add a title and display the plot
plt.title('Degrees awarded to women (1990-2010)\nComputer Science (red)\nPhysical Sciences (blue)')
```

```
plt.show()
```

Exercise 9: Using **axis to control xlim and ylim at a time**

```
# Plot in blue the % of degrees awarded to women in Computer Science
plt.plot(year,computer_science, color='blue')
# Plot in red the % of degrees awarded to women in the Physical Sciences
plt.plot(year, physical_sciences,color='red')
# Set the x-axis and y-axis limits
plt.axis((1990,2010,0,50))
# Show the figure
plt.show()
```

Exercise 10: Using legend

```
# Specify the label 'Computer Science'
plt.plot(year, computer_science, color='red', label='Computer Science')
# Specify the label 'Physical Sciences'
plt.plot(year, physical_sciences, color='blue', label='Physical Sciences')
# Add a legend at the lower center
plt.legend(loc='lower center')
# Add axis labels and title
plt.xlabel('Year')
plt.ylabel('Enrollment (%)')
plt.title('Undergraduate enrollment of women')
plt.show()
```


Exercise 11: Using **annotate()**

```
# Plot with legend as before
plt.plot(year, computer_science, color='red', label='Computer Science')
```

```
plt.plot(year, physical_sciences, color='blue', label='Physical Sciences')

plt.legend(loc='lower right')

# Compute the maximum enrollment of women in Computer Science: cs_max

cs_max = computer_science.max()

# Calculate the year in which there was maximum enrollment of women in Computer Science: yr_max

yr_max = year[computer_science.argmax()]

# Add a black arrow annotation

plt.annotate('Maximum', xy=(yr_max, cs_max), xytext=(yr_max+5, cs_max+5), arrowprops=dict(facecolor='black'))

# Add axis labels and title

plt.xlabel('Year')

plt.ylabel('Enrollment (%)')

plt.title('Undergraduate enrollment of women')

plt.show()
```


Exercise 12: Modifying styles

```
# Import matplotlib

import matplotlib.pyplot as plt

# Set the style to ggplot (Observe available styles: list(plt.style.available))

plt.style.use('ggplot')

# Create a figure with 2x2 subplot layout

plt.subplot(2, 2, 1)

# Plot the enrollment % of women in the Physical Sciences

plt.plot(year, physical_sciences, color='blue')

plt.title('Physical Sciences')
```

```
# Plot the enrollment % of women in Computer Science

plt.subplot(2, 2, 2)

plt.plot(year, computer_science, color='red')

plt.title('Computer Science')

# Add annotation

cs_max = computer_science.max()

yr_max = year[computer_science.argmax()]

plt.annotate('Maximum', xy=(yr_max, cs_max), xytext=(yr_max-1, cs_max-10),
arrowprops=dict(facecolor='black'))

# Plot the enrollmment % of women in Health professions

plt.subplot(2, 2, 3)

plt.plot(year, health, color='green')

plt.title('Health Professions')

# Plot the enrollment % of women in Education

plt.subplot(2, 2, 4)

plt.plot(year, education, color='yellow')

plt.title('Education')

# Improve spacing between subplots and display them

plt.tight_layout()

plt.show()
```

16. Seaborn data visualization

9. wip

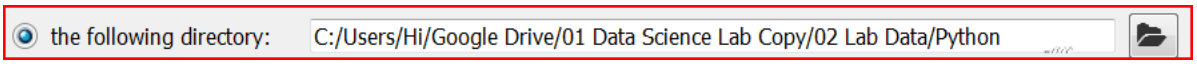
17. Bokeh data visualization

10. wip

18.

19. Import Data from Flat Files

1. Flat files
    a. Delimited (e.g.csv,tab,.. etc)
    b. Fixed (column size is fixed)
2. Change working directory using Spyder toolbar(**temporarily**). (If this toolbar not available then enable view menu bar → Show toolbars)

3. Change working directory using Spyder tools menu(**permeant**)
    a. **Tools** menu → Preferences → Current working directory
    b. 

4. **Exercise 1: About current working Directory using script**

```
# 01. import module
import os
# 02. get current working directory
wd = os.getcwd()
# 03. get list of file in working directory
os.listdir(wd)
# 04. Change working directory
os.chdir('C:\\Users\\Hi\\Google Drive\\01 Data Science Lab Copy\\02 Lab Data\\Python')
```


5. **Exercise 2: Importing plain text files (ex :News Paper)**
    a. Copy plain_text.txt file from labdata to working directory and execute below script

Py9_2_importing_plain_text_files

```
# Open a file
plain_file = open('plain_text.txt', mode='r')
# Print file content on screen
print(plain_file.read())
# Check whether file is closed
print(plain_file.closed)
# Close the file
plain_file.close()
# Check whether file is closed
print(plain_file.closed)
```

6. **Exercise 3: Importing plain text file line by line and print first three lines**

   a. readline() function is useful to read few lines from large file

   b. use **with** conjugate to read the file

Py9_3_Importing_text_files_line_by_line

```
# Read & print the first 3 lines
with open('plain_text.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())
```

7. **Exercise 4: Using numpy import MNIST file and create image to test not a robot**

 # Import package

import numpy as np

 # Load file as array: digits

digits = np.loadtxt(fname = 'mnist_digits.csv', delimiter=',')

 # Print datatype of digits

type(digits)

 # Select 21 row and reshape a row

im = digits[21, 1:]

im.shape

im_sq = np.reshape(im, (28, 28))

 # Plot reshaped data (matplotlib.pyplot already loaded as plt)

import matplotlib.pyplot as plt

```
plt.imshow(im_sq, cmap='Greys', interpolation='nearest')
plt.show()
np.load
```

8. **Exercise 5: Importing csv file as strings**

```
# Import package
import numpy as np
# import file
emp = np.loadtxt(fname = 'emp.txt',delimiter = ',',dtype = str)
type(emp)
emp
# Read first 4 rows and all columns
emp[0:4,1:]
```


9. **Exercise 6: Importing mixed data types**

```
# Import package
import numpy as np
# Load file as array: emp
emp = np.genfromtxt(fname = 'emp.txt', delimiter=',', names=True, dtype=None)
# Print datatype of emp
print(type(emp)) #<class 'numpy.ndarray'>
print(emp)  # all data
# shape
print(np.shape(emp)) # (14,)
#print sal column
```

```
print(emp['SAL'])

# Plot a scatter plot of the data

import matplotlib.pyplot as plt

plt.scatter(emp['COMM'], emp['SAL'])

plt.xlabel('Comm')

plt.ylabel('Sal')

plt.show()
```

10. Exercise 7: Importing mixed data types using pandas package

a. You were able to import flat files containing columns with different datatypes as numpy arrays.

b. We can easily import files of mixed data types as DataFrames using the pandas functions read_csv() and read_table().

Py9_7_Using Panda to import flat files_as_dataframe

```
1 # Import pandas
2 import pandas as pd
3 # Assign the filename: file
4 file = 'emp.csv'
5 # Read the file into a DataFrame: emp
6 emp= pd.read_csv(file)
7 # View the head of the DataFrame
8 print(emp.head(14))
```

11. Exercise 8: Converting dataframe to numpy array

*Py9_8_Converting data_frame_as_numpy_array

```
1 import pandas as pd
2 # Assign the filename: file
3 file = 'emp.csv'
4 # Read the first 3 rows of the file into a DataFrame: emp
5 emp = pd.read_csv(file, nrows=3, header=None)
6 # Build a numpy array from the DataFrame: emp
7 emp_array = emp.values
8 # Print the datatype of data_array to the shell
9 print(type(emp_array))
10 print(emp_array)
```

12. Exercise 9: Import file using pandas in customized format

Py9_9_Import file using pandas in customized format

```
# Import pandas
import pandas as pd
# Import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
# Assign filename: file
file = 'titanic.csv'
# Import file: data
data = pd.read_csv(file, sep=',' , comment='#', na_values=['Nothing'])
# Print the head of the DataFrame
print(data.head(3))
# Plot 'Age' variable in a histogram
pd.DataFrame.hist(data[['Age']])
plt.xlabel('Age (years)')
plt.ylabel('count')
plt.show()
```

20. Import Data from Excel Files

1. Use pandas package to import Excel spreadsheets
2. Import any given sheet of your loaded work book file as a DataFrame, by specifying either the sheet's name or its index.
3. It allows to skip rows, rename columns and select only particular columns.

Exercise 1: Read Excel work sheet as dataframe

```
import pandas as pd
xl = pd.ExcelFile("ScottData.xlsx")  # Load spreadsheet as: xl
df1 = xl.parse('EMP')  # Load a sheet into a DataFrame by name: EMP
print(df1.head()) # Print the head of the DataFrame EMP
df2 = xl.parse(1) # Load a sheet into a DataFrame by index: DEPT
print(df2.head()) # Print the head of the DataFrame df2
# Parse the first sheet and rename the columns: df3
df3 = xl.parse(0, names=['A', 'B','C','D','E','F','G','H'])
print(df3.head(14)) # Print the head of the DataFrame df3
# Parse the first column of the second sheet and rename the column: df4
df4 = xl.parse(1, parse_cols=[0], names=['A'])
print(df4.head()) # Print the head of the DataFrame df4
```


Exercise 2: Write data from dataframe to excel

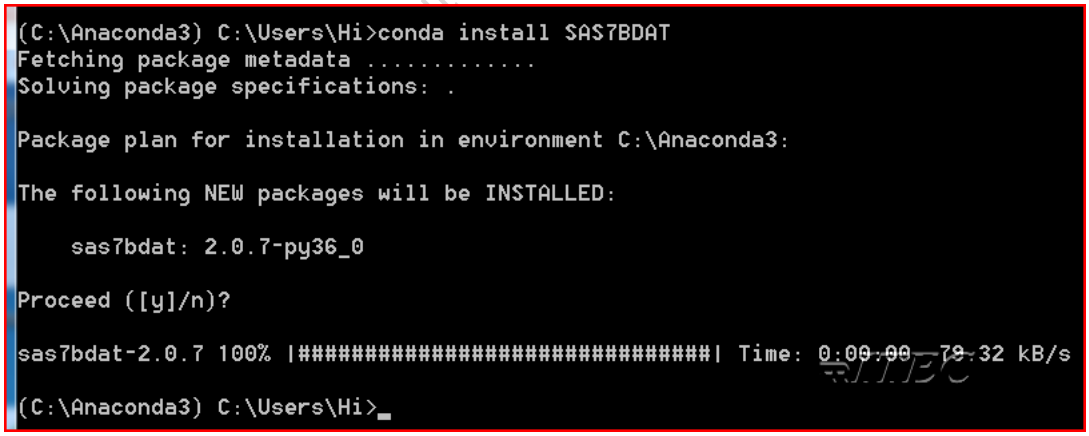
```
import pandas as pd
# Create a Pandas dataframe from the data.
df = pd.DataFrame({'Data': [10, 20, 30, 20, 15, 30, 45]})
df # verfiy data frame data
# Create a Pandas Excel writer using XlsxWriter as the engine.
writer = pd.ExcelWriter('pandas_simple.xlsx', engine='xlsxwriter')
```

```
# Convert the dataframe to an XlsxWriter Excel object.

df.to_excel(writer, sheet_name='Sheet1')

writer.save() # Close the Pandas Excel writer and output the Excel file.
```

4. Refer more http://xlsxwriter.readthedocs.io/working_with_pandas.html

21. Import SAS and STATA Files

1. SAS: Statistical Analysis System → mainly for business analytics and biostatistics
2. Stata: “Statistics” + “data” → mainly for academic social sciences research
3. SAS files are Used for:
    a. Advanced analytics
    b. Multivariate analysis
    c. Business intelligence
    d. Data management
    e. Predictive analytics
    f. Standard for computational analysis
4. Install SAS7BDAT package
    a. Go to start → all programs → anaconda → anaconda prompt
    b. Type **conda install SAS7BDAT**



5. Find some sample files here http://www.principlesofeconometrics.com/sas/

6. Exercise 1: Work with SAS file

```
import pandas as pd # Import pandas package

import matplotlib.pyplot as plt # Import matplotlib package

from sas7bdat import SAS7BDAT  # Import sas7bdat package
```

```
# Save file to a DataFrame: df_sas
with SAS7BDAT('sales.sas7bdat') as file:
    df_sas = file.to_data_frame()
print(df_sas.head()) # Print head of DataFrame
# Plot histograms of a DataFrame
pd.DataFrame.hist(df_sas[['P']])
plt.ylabel('count')
plt.show()
```

7. **Exercise 2: Work with STATA file**

   a. **Download any stata file from** **http://www.principlesofeconometrics.com/stata.htm**

```
# Import pandas
import pandas as pd
# Import matplotlib package
import matplotlib.pyplot as plt
# Load Stata file into a pandas DataFrame: df
df = pd.read_stata('disarea.dta')
# Print the head of the DataFrame df
print(df.head())
# Plot histogram of one column of the DataFrame
pd.DataFrame.hist(df[['disa10']])
plt.xlabel('Extent of disease')
plt.ylabel('Number of coutries')
plt.show()
```

22. Import HDF5 Files

1. Hierarchical Data Format version 5
2. Standard for storing large quantities of numerical data
3. Datasets can be hundreds of gigabytes or terabytes
4. HDF5 can scale to exabytes
5. For more refer https://support.hdfgroup.org/HDF5/examples/intro.html#python
6. It is open source and Official website is https://www.hdfgroup.org/
7. **Exercise 1: Work with HDF5 file**

```
import numpy as np; import h5py; import matplotlib.pyplot as plt # Import packages
data = h5py.File('LIGO_data.hdf5', 'r') # Load file: data
print(type(data))  #<class 'h5py._hl.files.File'>  # Print the datatype of the loaded file
# Print the keys of the file
for key in data.keys():
    print(key)    # meta , quality and strain
# Get the HDF5 group: group
group = data['strain']
# Check out keys of group
for key in group.keys():
    print(key)
strain = data['strain']['Strain'].value # Set variable equal to time series data: strain
num_samples = 10000 # Set number of time points to sample: num_Samples
time = np.arange(0, 1, 1/num_samples) # Set time vector
# Plot data
plt.plot(time, strain[:num_samples])
plt.xlabel('GPS Time (s)') ; plt.ylabel('strain') ; plt.show()
```

23. Import from Relational Database (Ex: SQLite)

1. Some of the relational databases are
    - **a.** MySQL
    - **b.** PostgreSQL
    - **c.** SQLite
    - **d.** Sqlserver
    - **e.** Oracle
    - **f.** Hive
    - **g.** ..etc

2. SQLite database
    a. Small. Fast. Reliable
    b. More read from https://www.sqlite.org/about.html
3. SQLAlchemy 

    a. SQLAlchemy is the Python SQL toolkit.
    b. Works with many Relational Database Management Systems
    c. More read from https://www.sqlalchemy.org/
4. **Exercise 1: list SQLite chinook database tables**

```
# Import necessary module sqlalchemy
from sqlalchemy import create_engine
# Create Python engine: engine for SQLite database
engine = create_engine('sqlite:///Chinook.sqlite')
# Save the table names to a list object : table_names
```

```
table_names = engine.table_names()
# Print the table names on console
print(table_names)
```

5. **Exercise 2: SQLite basic select query**
    a. Work flow has seven steps
        i. Import packages and functions
        ii. Create the database engine
        iii. Connect to the engine
        iv. Query the database
        v. Save query results to a DataFrame
        vi. Close the connection
        vii. Observe dataframe data


```
# Step 1 of 7: Import required packages
from sqlalchemy import create_engine ; import pandas as pd ;
# Step 2 of 7: Create engine: engine
engine = create_engine('sqlite:///Chinook.sqlite')
# Step 3 of 7: Open engine connection
con = engine.connect()
# Step 4 of 7: Perform query: result
result = con.execute("SELECT * FROM Album")
# Step 5 of 7: Save results of the query to DataFrame: df
df = pd.DataFrame(result.fetchall())
# Step 6 of 7: Close connection
con.close()
# Step 7 of 7: Print head of DataFrame df
```

```
print(df.head())
```

6. **Exercise 3: SQLite select query using context manager**

```
# Import packages
from sqlalchemy import create_engine ; import pandas as pd
# Create engine: engine
engine = create_engine('sqlite:///Chinook.sqlite')
# Open engine in context manager (to avoid open and close of the connections)
# Perform query and save results to DataFrame: df
with engine.connect() as con:
    rs = con.execute("SELECT LastName, Title FROM Employee")
    df = pd.DataFrame(rs.fetchmany(size=3))
    df.columns = rs.keys() #print column names
print(len(df)) # Print the length of the DataFrame df
print(df.head()) # Print the head of the DataFrame df
```


7. **Exercise 4: SQLite select query with where and order by clause**

```
from sqlalchemy import create_engine;  import pandas as pd # Import packages
engine = create_engine('sqlite:///Chinook.sqlite') # Create engine: engine
# Open engine in context manager ,run query and save results to DataFrame: df
with engine.connect() as con:
    rs = con.execute("SELECT * FROM Employee WHERE EmployeeId >= 6 ORDER BY
BirthDate")
    df = pd.DataFrame(rs.fetchall())
    df.columns = rs.keys()
# Print the head of the DataFrame df
```

```
print(df.head())
```

8. **Exercise 5: SQLite query using pandas Vs context manager**

```
from sqlalchemy import create_engine; import pandas as pd # Import packages

engine = create_engine('sqlite:///Chinook.sqlite') # Create engine: engine

df = pd.read_sql_query("SELECT * FROM Album", engine) # Execute query and create DataFrame: df

print(df.head()) # Print head of DataFrame

#compare pandas result with context manager result

# Open engine in context manager, run query and save results to DataFrame: df1

with engine.connect() as con:

    rs = con.execute("SELECT * FROM Album")

    df1 = pd.DataFrame(rs.fetchall())

    df1.columns = rs.keys()

print(df.equals(df1)) # Confirm that both methods yield the same result: does df = df1 ?
```


9. **Exercise 6: SQLite database joins**

```
from sqlalchemy import create_engine;import pandas as pd # Import packages

engine = create_engine('sqlite:///Chinook.sqlite') # Create engine: engine

# Open engine in context manager, run query and save results to DataFrame: df

with engine.connect() as con:

    rs = con.execute("SELECT Title, Name FROM Album INNER JOIN Artist \

    on Album.ArtistID = Artist.ArtistID")

    df = pd.DataFrame(rs.fetchall())

    df.columns = rs.keys()
```

```
print(df.head()) # Print head of DataFrame df
```

Refer more http://www.sqlitetutorial.net/sqlite-python/

24. Import web data

1. Working with web file
    a. Download web file
    b. Save to local directory
    c. Store into python dataframe
2. For more info about data refer **https://archive.ics.uci.edu/ml/datasets/wine+quality**

Exercise 1: Import Web Data

```
# Import required packages

from urllib.request import urlretrieve;import pandas as pd

# Assign url of file: url_of_file

url_of_file = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv'

# Save file locally into working directory

urlretrieve(url_of_file, 'winequality-red.csv')

# Read file into a DataFrame and print its head

df = pd.read_csv('winequality-red.csv', sep=';')

print(df.head())
```


3. Go to working directory and observe the file winequality-red.csv saved.

Exercise 2: Working with web data without saving locally

```
import matplotlib.pyplot as plt;import pandas as pd # Import packages

# Assign url of file: url

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv'

df = pd.read_csv(url, sep=';') # Read file into a DataFrame: df

print(df.head()) # Print the head of the DataFrame

# Plot first column of df
```

```
pd.DataFrame.hist(df.ix[:, 0:1])
plt.xlabel('fixed acidity (g(tartaric acid)/dm$^3$)')
plt.ylabel('count')
plt.show()
```

Exercise 3: Working with web data excel files

```
import pandas as pd # Import package
# Assign url of file: url
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00192/BreastTissue.xls'
xl = pd.read_excel(url, sheetname=None) # Read in all sheets of Excel file: xl
print(xl.keys()) # Print the sheetnames to the shell
# Print the head of the first sheet ,use sheet name: 'Data'
print(xl['Data'].head())
```

25. Import using urllib and requests packages

Exercise 1: HTTP requests/responses using urllib package to get html

```
from urllib.request import Request,urlopen # Import packages urllib
url = "http://rritec.blogspot.in/" # provide any url
request = Request(url)# packages the request: request
# Sends the request and catches the response: response
response = urlopen(request)
# Print the data type of response
print("The Data Type of response object is : " + str((type(response))))
html = response.read() # Extract the response: html
# Print the html
print(html)
# Be polite and close the response!
response.close()
```


Exercise 2: HTTP requests using requests package to get html

```
# Import package
import requests
# Specify the url: url
url = "http://rritec.blogspot.in/"
# Packages the request, send the request and catch the response: r
r = requests.get(url)
# Extract the response: text
text = r.text
# Print the html
```

```
print(text)
```

26. Read HTML with BeautifulSoup package

Exercise 1: Get HTML using BeautifulSoup package

```
# Import packages
import requests ; from bs4 import BeautifulSoup
# Specify url: url
url = 'https://www.python.org/~guido/'
# Package the request, send the request and catch the response: r
r = requests.get(url)
# Extracts the response as html: html_doc
html_doc = r.text
# Create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc)
# Prettify the BeautifulSoup object: prettify_soup
prettify_soup = soup.prettify()
# Print the response
print(prettify_soup)
```


Exercise 2: Get Titles and text using BeautifulSoup package

```
# Import packages
import requests
from bs4 import BeautifulSoup
# Specify url: url
url = 'https://www.python.org/~guido/'
```

```
# Package the request, send the request and catch the response: r
r = requests.get(url)
# Extract the response as html: html_doc
html_doc = r.text
# Create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc,"lxml")
# Get the title of Guido's webpage: guido_title
guido_title = soup.title
# Print the title of Guido's webpage to the shell
print(guido_title)
# Get Guido's text: guido_text
guido_text = soup.get_text()
# Print Guido's text to the shell
print(guido_text)
```


Exercise 3: Get hyperlinks using BeautifulSoup package

```
# Import packages
import requests ; from bs4 import BeautifulSoup
# Specify url
url = 'http://rritec.blogspot.in/'
# Package the request, send the request and catch the response: r
r = requests.get(url)
# Extracts the response as html: html_doc
html_doc = r.text
# create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc)
```

```
# Print the title of Guido's webpage
print(soup.title)
# Find all 'a' tags (which define hyperlinks): a_tags
a_tags = soup.find_all('a')
# Print the URLs to the shell
for link in a_tags:
    print(link.get('href'))
```

27. Import JSON File

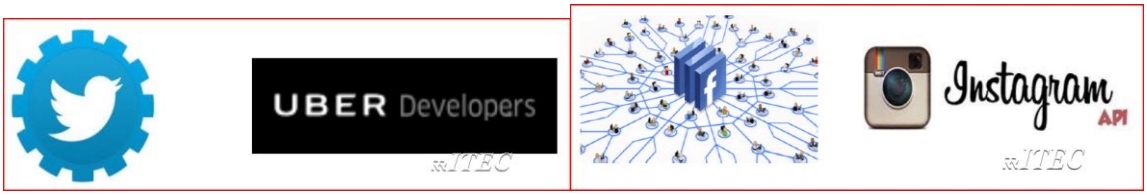
1. JavaScript Object Notation(JSON)
2. JSONs consist of key-value pairs.
3. JSONs are human-readable.
4. Real-time server-to-browser communication
5. The function json.load() will load the JSON into Python as a dictionary.


```
import json
# Load JSON: json_data
with open("employees.json") as json_file:
    json_data = json.load(json_file)
#Print each key-value pair in json_data
for k in json_data.keys():
    print(k + ': ', json_data[k])
```

6. Refer https://docs.python.org/3/library/json.html

28. Movie and Wikipedia APIs

1. An API is a set of protocols and routines for building and interacting with software applications.
2. API is an acronym and is short for Application Program interface.
3. It is common to pull data from APIs in the JSON file format.
4. An API is a bunch of code that allows two software programs to communicate with each other.
5. APIs are everywhere



Exercise 1: open movie database API

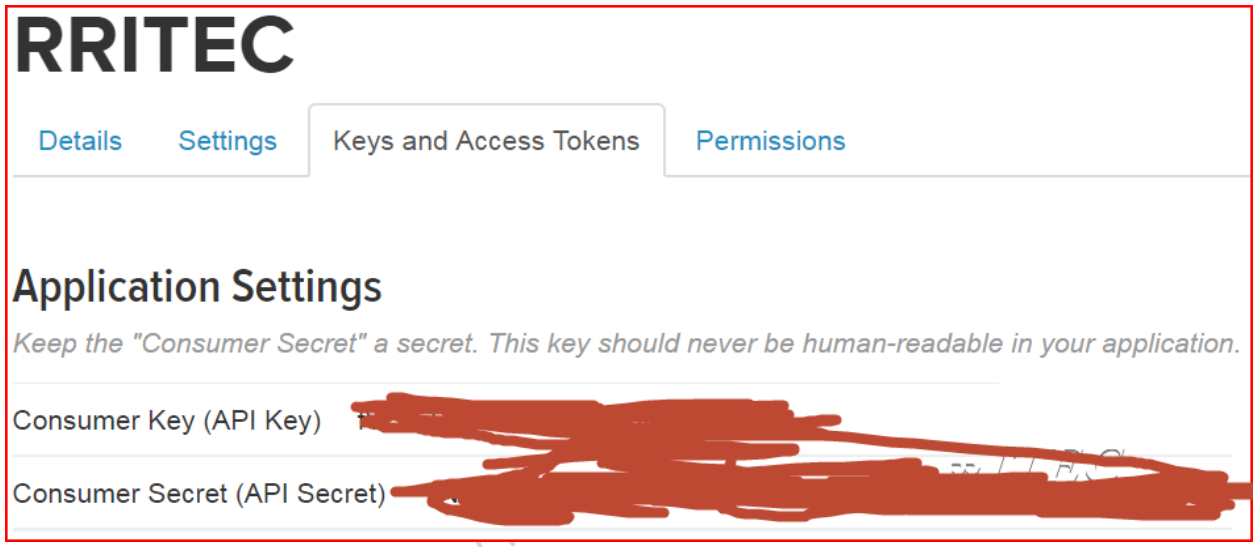
```
# Import requests package

import requests

# Assign URL to variable: url

url = 'http://www.omdbapi.com/?apikey=ff21610b&t=coco'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Print the text of the response

print(r.text)

# Decode the JSON data into a dictionary: json_data

json_data = r.json()

# Print each key-value pair in json_data

for k in json_data.keys():

    print(k + ': ', json_data[k])
```

Exercise 2: Wikipedia API

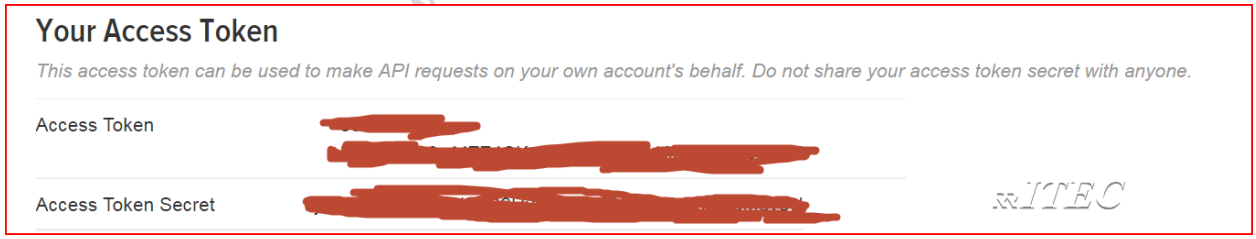
```
# Import package

import requests

# Assign URL to variable: url

url =
'https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json&exintro=&titles=pizza'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Decode the JSON data into a dictionary: json_data

json_data = r.json()

# Print the Wikipedia page extract

pizza_extract = json_data['query']['pages']['24768']['extract']

print(pizza_extract)
```

29. Twitter API

1. Login to twitter
2. In same browser go to other tab → type https://apps.twitter.com
3. Provide required information
4. Get consumer key and consumer secret



5. Get access token key and access token secret



6. **F**ollow the tutorial and do first exercise and continue rest of the things
http://tweepy.readthedocs.io/en/v3.5.0/getting_started.html

Exercise 1: Print Public tweets

Py19_1_Twitter_API_read_public_tweets

```
# Import package
import tweepy
# Store OAuth authentication credentials in relevant variables
access_token = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
access_token_secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
consumer_key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
consumer_secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
# Pass OAuth details to tweepy's OAuth handler
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
public_tweets = api.home_timeline()
for tweet in public_tweets:
    print(tweet.text)

```


https://labsblog.f-secure.com/2018/02/27/how-to-get-twitter-follower-data-using-python-and-tweepy/

30. Cleaning Data (ETL)

1. It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data (Dasu and Johnson 2003)

2. Data preparation is not just a first step, but must be repeated many times over the course of analysis as new problems come to light or new data is collected

3. The principles of tidy data provide a standard way to organize data values within a dataset. Please refer white paper https://www.jstatsoft.org/article/view/v059i10/v59i10.pdf

4. Some of the common problems are

    a. Inconsistent column names

    b. Missing data

    c. Outliers

    d. Duplicate rows

    e. Untidy

    f. Column data types

30.1 Melt() data

1.Follows Principles of tidy data

a. Columns represent separate variables

b. Rows represent individual observations

c. Observational units form tables

2.Melt function → "Unpivots" a DataFrame from wide format to long format. Observe below exercise

3.Exercise 1: wide format → long format

```
# create a dataframe

import pandas as pd

df = pd.DataFrame({'A':{0:'a',1:'b',2:'c'},

                   'B':{0:1,1:3,2:5},

                   'C':{0:2,1:4,2:6},

                   'D':{0:7,1:9,2:11},

                   'E':{0:8,1:10,2:12}})

# convert dataframe using melt

df_melt = df.melt(id_vars = ["A"],

       value_vars=["B","C","D","E"],

       var_name = "my_var",

       value_name ="my_val")

df

df_melt
```


4.Exercise 2: Columns containing values, instead of variables(melt())

```
# 01. import required module and load csv file

import pandas as pd
```

```
df = pd.read_csv("Py_cleaning_Tidydata.csv")
# 02. Observe data before melting
df
# 03. melt and print data
df_melt = pd.melt(frame = df,
                  id_vars = 'name',
                  value_vars = ['treatment a','treatment b'],
                  var_name = 'treatment',
                  value_name = 'result')
# 04. Observe data after melting
df_melt
```

30.2 Pivot (un-melting data)

1.Opposite of melting

2.In melting, we turned columns into rows

3.Pivoting: turn unique values into separate columns

4.Violates tidy data principle: rows contain observations

5.Multiple variables stored in the same column

6.Exercise 1: Pivot data

```
# 01. import required modules
import pandas as pd
# 02. load csv file as dataframe
df = pd.read_csv("py_cleaning_pivot_data.csv")
# 03. Observe the dataframe data
df
# 04. pivot the data
```

```
df_pivot = df.pivot(index = "name",columns="treatment",values = "result")

# 05. after pivot observe the dataframe

df_pivot
```

7.Exercise 2: Pivot data of duplicate data

```
# 01. import required modules

import pandas as pd

import numpy as np

# 02. load csv file as dataframe

df = pd.read_csv("py_cleaning_pivot_data1.csv")

# 03. Observe the dataframe data

df

# 04. pivot the data

df_pivot = df.pivot_table(index = "date", columns="temperature", values = "value",
aggfunc = np.mean)

# 05. after pivot observe the dataframe

df_pivot
```

30.3 Concatenating

1. Data may not always come in 1 huge file
    a. Example: 5 million row dataset may be broken into, 5 separate datasets. Because Small data sets are Easier to store and share
2. May have new data for each day
3. **Exercise 1:** Combine two DataFrames row wise (axis = 0)

```
# 01. Import required modules

import pandas as pd

# 02. load dataframes from csv files

df1 = pd.read_csv('Py_cleaning_pivot_concat1.csv')

df2 = pd.read_csv('Py_cleaning_pivot_concat2.csv')

# 03. Observe first data frame

df1

# 04. Observe second data frame

df2

# 05. concatenate two data frames row wise

concatenated = pd.concat([df1, df2])

concatenated

# 06. Observe that indexes are duplicated

concatenated.loc[0, :]

# 07. concatenat without indexes are duplicated

concatenated = pd.concat([df1, df2],ignore_index=True)

concatenated
```


4. **Exercise 2: Combine two DataFrames column wise (axis = 1)**

```
# Import pandas

import pandas as pd

# Read data from csv files

df1 = pd.read_csv('Py_cleaning_pivot_concat3.csv')

df2 = pd.read_csv('Py_cleaning_pivot_concat4.csv')

print("***** 01. Observe first data frame *****")

print(df1)

print("***** 02. Observe second data frame *****")

print(df2)

print("****** 03. Observe concatenated data frame column wise*******")

concatenated = pd.concat([df1, df2],axis=1)

print(concatenated)
```


5. Concatenating many files

a. Glob module to find all csv files in the workspace

b. You can find all .csv files with '*.csv'

   i. Ex: emp10.csv, employee.csv…etc

c. All parts with 'emp_*'.

   i. Ex: emp_*.csv (emp_03sep2017.csv, emp_04sep2017.csv, emp_05sep2017.csv)

d. Only one character is changing

   i. Ex: part?.csv (part1.csv, part2.csv, part3.csv..etc)

e. The? Wildcard represents any 1 character, and the * wildcard represents any number of characters.

f. Exercise 3: Concatenating many files

```
# import required packages
```

```
import glob
import pandas as pd
print("***** 01. Read all emp_part files from working directory *****")
csv_files = glob.glob('emp_part?.csv')
print(csv_files)
# Create an empty list: emp_list
emp_list = []
#  Iterate over csv_files
for csv in csv_files:
    #  Read csv into a DataFrame: df
    df = pd.read_csv(csv)
    # Append df to emp_list
    emp_list.append(df)
# Concatenate emp_list into a single DataFrame: emp
emp = pd.concat(emp_list)
print("***** 02. print after combining files  *****")
print(emp)
```


30.4 Merge/Joins

1.Similar to joining tables in SQL.

2.The data relation may be One-to-one, Many-to-one, one-to-many and Many-to-many

3.Exercise 1: Join **one to many** or **many to one** datasets

```
import pandas as pd
emp_df1 = pd.read_csv('emp_merge.csv')
dept_df2 = pd.read_csv('dept_merge.csv')
```

```
# Merge two DataFrames: emp_dept

emp_dept = pd.merge(left=emp_df1, right=dept_df2, on=None,left_on='DEPTNO', right_on='DEPTID')

# Print emp_dept

print(emp_dept)
```

Exercise 2: Outer join two DataFrames

```
# Create two data frames

df1 = pd.DataFrame({'lkey':['foo','bar','baz','foo'],'value':[1,2,3,4]})

df2 = pd.DataFrame({'rkey':['foo','bar','qux','bar'],'value':[5,6,7,8]})

# create outer join

pd.merge(left = df1,right = df2,how = 'outer',on = None,left_on = "lkey",right_on = "rkey")
```

30.5 Data Types Conversion

1.Converting categorical data in to 'category' dtype, can make the DataFrame smaller in memory

2.Numeric data loaded as a string (for example any – in data), then system will identify as object datatype.

3.Exercise 1: Take care above two scenarios

```
# 01. Import required module

import pandas as pd

# 02. load csv as dataframe

df = pd.read_csv('tips.csv')

print("***** 01. Observe the info of data frame and note down memory *****")

df.info()

# 03. Convert the sex column to type 'category'

df.sex = df.sex.astype('category')

# 04. Convert the smoker column to type 'category'

df.smoker = df.smoker.astype('category')

# 05. Convert 'total_bill' to a numeric dtype

df.total_bill = pd.to_numeric(df.total_bill,errors="coerce")

# 06. Convert 'tip' to a numeric dtype

df.tip = pd.to_numeric(df.tip,errors="coerce")

# 07 observe the memory difference and - replaced with NAN

print("***** 02. Observe the info of data frame and note down memory *****")

df.info()

print("***** 03. from 01 and 02, did you observed memory difference *******")

print("***** 04. did you observed - replaced with NaN *******")

df
```

30.6 Regular expression operations

1. **'re'** library is used for regular expressions.
    a. A formal way of specifying a pattern
    b. For more refer help https://docs.python.org/3/library/re.html
2. **Exercise 1: Regular expressions**

```
# 01. Import the regular expression module
import re
# 02. Example 1: match Phone Number
# 02 - 1. Define required pattern
pattern = re.compile('\d{3}-\d{3}-\d{4}')
# 02 - 2 Observe result
rs = pattern.match('123-456-7890')
bool(rs)
rs = pattern.match('1234-456-7890')
bool(rs)
# 03. Example 2: verify dollar amount format
# 03 - 1. Define required pattern
pattern = re.compile('\$\d*.\d{2}')
# 03 - 2 Observe result
rs = pattern.match('$13.25')
bool(rs)
rs = pattern.match('13.25')
bool(rs)
```


3. **Exercise 2: Create a list by reading numbers from string**

```
# 01. Import the regular expression module

import re

# 02. Find the numeric values and store it as list : rs

rs = re.findall('\d+', 'the recipe calls for 10 strawberries and 1 banana')

# 03. Print the matches

type(rs)

rs

# 04. convert list items into integers

rs = [int(x) for x in rs]

rs
```

30.7 Dropping duplicate data

1. **.drop_duplicates()** method is useful to drop duplicates from dataframe

```
# 01. Import required module

import pandas as pd

# 02 . Load csv file as dataframe

df = pd.read_csv('emp_duplicate.csv')

df.info()

# 03. Drop the duplicates: df_no_duplicates

df_no_duplicates = df.drop_duplicates()

df_no_duplicates.info()
```

30.8 Filling missing data

1. **.fillna()** method → Fill with business specified value.
2. **.dropna()** method → to drop rows

```
# 01. import required module

import pandas as pd

# 02. load csv as dataframe

df = pd.read_csv('emp_missing.csv')

df.info()

# 03.Calculate the mean of the comm column : comm_mean

COMM_mean = round(df.COMM.mean(),1)

# 04. Replace all the missing values in the comm column with the mean

df.COMM = df.COMM.fillna(COMM_mean)

# 05. Observe result

df.info()

df

# 06. drop missing values rows

df_dropped = df.dropna(axis = 0) # use axis = 1 to drop columns

df_dropped.info()
```

30.9 Testing with asserts

1. If we drop or fill NaNs, we expect **zero** missing values
2. We can write an assert statement to verify this
3. This gives us confidence that our code is running correctly
4. Verify below in python shell

    a.Assert is true ,it won't give anything

    b.Assert is false ,it through error

```
>>> assert 1 == 1
>>> assert 1 == 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
>>>
```

5. Exercise 1: Assert

```
# 01. import required module

import pandas as pd

# 02. load csv as dataframe

df = pd.read_csv('emp_missing.csv')

# 03. drop missing values rows

df_dropped = df.dropna(axis = 0) # use axis = 1 to drop columns

df_dropped.info()

# 04. Assert that there are no missing values

assert pd.notnull(df_dropped).all().all()

# 05. Assert that all values are >= 0

assert (df_dropped >= 0).all().all()

# 06. Observe data

df_dropped
```

31. Time Series Analysis

1. Just like that read article https://fivethirtyeight.com/features/how-fast-youll-abandon-your-new-years-resolutions/
2. Explore **diet** search in google using google trends https://trends.google.com/trends/
3. Exercise 1: Diet search trend

```
# Import pandas and plotting modules
import pandas as pd
import matplotlib.pyplot as plt
# Load data
df = pd.read_csv("diet.csv",skiprows = 3,names = ['week','diet'])
df
# Convert the date index to datetime
df.index = pd.to_datetime(df.week)
df.info()
del(df['week'])
# Plot 2016 data using slicing
df['2016'].plot()
plt.show()
# Plot the entire time series diet and show gridlines
df.plot(grid=True)
plt.show()
```


4. Explore **diet, gym and finance** search in google using google trends https://trends.google.com/trends/explore?date=all&q=diet,gym,finance
5. Exercise 2: Diet, gym and finance search trend

```
# Import pandas and plotting modules
```

```
import pandas as pd

import matplotlib.pyplot as plt

# Load data

df = pd.read_csv("multiTimeline.csv",skiprows = 3,names = ['month','diet','gym','finance'])

df

df.info()

# Convert the date index to datetime

df.index = pd.to_datetime(df.month)

df.info()

del(df['month'])

df.info()

# Plot the entire time series diet and show gridlines

df.plot(grid=True)

plt.show()

df.info()
```


6. Correlation

   a. Correlation values range between -1 and 1.

   b. There are two key components of a correlation value:

      i. **Magnitude** – The larger the magnitude (closer to 1 or -1), the stronger the correlation

      ii. **Sign** – If negative, there is an inverse correlation. If positive, there is a positive correlation

   c. **Positive correlation(0+ to +1)**

```
import numpy as np

np.random.seed(1)

# 1000 random integers between 0 and 50

x = np.random.randint(0, 50, 1000)
```

```
# Positive Correlation with some noise
y = x + np.random.normal(0, 10, 1000)
np.corrcoef(x, y)
```

i. What is the correlation value between x and y?

d.See same results using scatter graph

```
import matplotlib.pyplot as plt
plt.matplotlib.style.use('ggplot')
plt.scatter(x, y)
plt.show()
```

e.**Negative correlation(0- to -1)**


```
# 1000 random integers between 0 and 50
x = np.random.randint(0, 50, 1000)
# Negative Correlation with some noise
y = 100 - x + np.random.normal(0, 5, 1000)
np.corrcoef(x, y)
```

f. Scatter graph

```
plt.scatter(x, y)
plt.show()
```

g.**No/Weak correlation(zero)**

```
x = np.random.randint(0, 50, 1000)
y = np.random.randint(0, 50, 1000)
np.corrcoef(x, y)
```

h.Scatter graph

```
plt.scatter(x, y)
plt.show()
```

i. **Correlation matrix**

```
import pandas as pd
df = pd.DataFrame({'a': np.random.randint(0, 50, 1000)})
df['b'] = df['a'] + np.random.normal(0, 10, 1000) # positively correlated with 'a'
df['c'] = 100 - df['a'] + np.random.normal(0, 5, 1000) # negatively correlated with 'a'
df['d'] = np.random.randint(0, 50, 1000) # not correlated with 'a'
df.corr()
```


j. Scatter matrix

```
pd.scatter_matrix(df, figsize=(6, 6))
plt.show()
```

k.correlation matrix plot

```
plt.matshow(df.corr())
plt.xticks(range(len(df.columns)), df.columns)
plt.yticks(range(len(df.columns)), df.columns)
plt.colorbar()
plt.show()
```

7. Explore around Spurious/fake correlation

a.Can I believe blindly correlation value or we need to use bit common sense??

b.http://www.tylervigen.com/spurious-correlations

8. Exercise 2: Auto correlation

```
# Load Microsoft stock data
df = pd.read_csv("MSFT.csv",index_col=0)
df.info()
# Convert the date index to datetime
df.index = pd.to_datetime(df.index)
df.info()
# Convert the daily data to weekly data
df = df.resample(rule='W', how='last')
df
# Compute the percentage change of prices
returns = df.pct_change()
# Compute and print the autocorrelation of returns
autocorrelation = returns['Adj Close'].autocorr()
print("The autocorrelation of weekly returns is %4.2f" %(autocorrelation))
```


9. Exercise 3: Compute the ACF

```
# Import the acf module and the plot_acf module from statsmodels
from statsmodels.tsa.stattools import acf
from statsmodels.graphics.tsaplots import plot_acf
# Load Microsoft stock data
df = pd.read_csv("HRB.csv",index_col=0)
df.info()
# Convert the date index to datetime
df.index = pd.to_datetime(df.index)
df.info()
```

```
# understand data
import matplotlib.pyplot as plt
plt.plot(df.index,df.Earnings) #or
plt.scatter(df.index,df.Earnings)
# Compute the acf array of HRB
acf_array = acf(df)
print(acf_array)
# Plot the acf function
plot_acf(df, alpha=1)
plt.show()
```

10.

Level 04 of 08: Machine Learning Models

32. Machine learning

11. What is machine learning?

12. Refer http://scikit-learn.org/stable/tutorial/basic/tutorial.html

13. Two definitions of Machine Learning are offered by Arthur and Tom.

14. **Arthur Samuel** described it as: "the field of study that gives computers the ability to learn without being explicitly programmed." This is an older, informal definition.

15. **Tom Mitchell** provides a more modern definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance/accuracy measure P, if its performance at tasks in T, as measured by P, improves with experience E."

    a. Example: playing checkers.

        i. E = the experience of playing many games of checkers

        ii. T = the task of playing checkers.

        iii. P = the probability that the program will win the next game.

16. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome.

17. It is the best way to make progress towards human-level Artificial intelligence(AI)

18. Machine learning mainly divided into two types

    a. Supervised learning

        i. Classification

        ii. Regression

    b. Unsupervised learning

        i. Clustering

32.1 Supervised learning

1. Uses labelled data
2. In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.
3. Supervised learning problems mainly categorized into
    a. Regression(continuous values)
    b. Classification(Limited or categorical)
4. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. (y=a+b*x)
5. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.(yes / NO or Male /Female or bad,good,very good and excellent ,1/0 …etc)
6. Example 1:
    a. Given data about the size of houses on the real estate market, try to predict their price. Price as a function of size is a continuous output, so this is a regression problem.
    b. We could turn this example into a classification problem by instead making our output about whether the house "sells for more or less than the asking price." Here we are classifying the houses based on price into two discrete categories.
7. Example 2:
    a. (a) Regression - Given a picture of a person, we have to predict their age on the basis of the given picture
    b. (b) Classification - Given a patient with a tumor, we have to predict whether the tumor is malignant or benign.

8. Exercise 1: Explore pre-defined Datasets (example iris data)

```
from sklearn import datasets
iris = datasets.load_iris()        # 01. Load iris data set
print(type(iris))                  # 02. datatype of iris data set
print(iris.keys())                 # 03. Keys of iris data set
print(iris.data)                   # 04. Observe iris data key
print(iris.target)                 # 05. Observe iris target key
print(iris.target_names)           # 06. Observe iris target_names key
print(iris.DESCR)                  # 07. Observe iris descr
print(iris.feature_names)          # 08. Observe iris feature_names
print(type(iris.data))             # 09. Datatype of iris data key
print(type(iris.target))           # 10. datatype of iris target key
print(type(iris.target_names))     # 11. datatype of iris target_names key
print(type(iris.DESCR))            # 12. datatype of iris descr key
print(type(iris.feature_names))    # 13. datatype of iris feature_names key
print(iris.data.shape)             # 14. Nmber of rows and columns
```


9. Exercise 2: Create data frame

```
from sklearn import datasets
import pandas as pd
print(" ##### 01. Load iris data set ##### ")
iris = datasets.load_iris()
print(" ##### 02. Create Data Frame ##### ")
df = pd.DataFrame(iris.data, columns=iris.feature_names)
print(df.head(2))
```

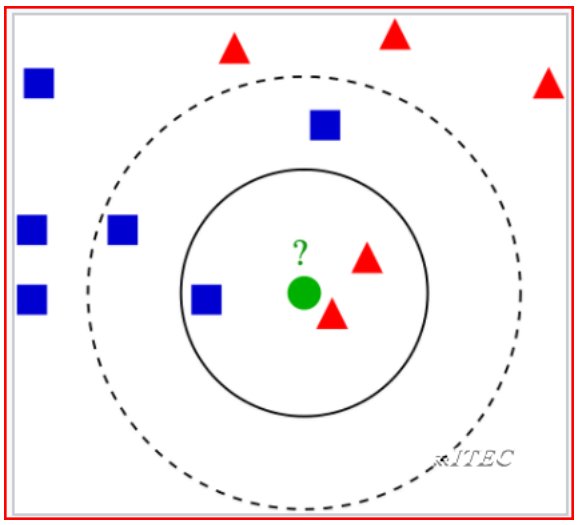
10. Exercise 3: Create a bunch with our own data (Just Knowledge purpose ,not useful in live)

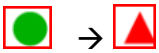
```
import numpy as np

import sklearn.datasets

examples = []

examples.append('example text 1')

examples.append('example text 2')

examples.append('example text 3')

target = np.zeros((3,), dtype=np.int64)

target[0] = 0

target[1] = 1

target[2] = 0

dataset = sklearn.datasets.base.Bunch(data=examples, target=target)
```

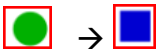
32.1.1 k-nearest neighbours algorithm

32.1.1.1. Introduction

1. http://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html
2. Read and practice examples http://scikit-learn.org/stable/modules/neighbors.html#
3. **Exercise 3: k-nearest neighbours algorithm**



a.KNeighborsClassifier(n_neighbors=2):  →

b. KNeighborsClassifier(n_neighbors=3):  →

c.Refer https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

d.It is non-parametric method

i. Refer parametric vs non parametric https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/

e.Note: how to calculate distance in case of regression problem

Euclidean: Take the square root of the sum of the squares of the differences of the coordinates.

For example, if $x = (a, b)$ and $y = (c, d)$, the Euclidean distance between x and y is $\sqrt{(a-c)^2 + (b-d)^2}$.

Manhattan: Take the sum of the absolute values of the differences of the coordinates.

For example, if $x = (a, b)$ and $y = (c, d)$, the Manhattan distance between x and y is $|a - c| + |b - d|$.

f. Note: how to calculate distance in case of classification problem

Hamming Distance

$$D_H = \sum_{i=1}^{k} |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

```
# Import KNeighborsClassifier from sklearn.neighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
knn = KNeighborsClassifier(n_neighbors=6)
f = knn.fit(iris['data'], iris['target'])
```

```
print(f)
print(iris['data'].shape)
print(iris['target'].shape)
x_new = np.array([[5.1,3.5,1.4,0.2], [5.5,2.3,4.,1.3],[6.7,3.3,5.7,2.5]], dtype=np.float)
prediction = knn.predict(x_new)
print(prediction)
print(x_new.shape)
```

32.1.1.2. Measuring model performance

g.In classification, accuracy is a commonly used metric

h.Accuracy = Fraction of correct predictions

```
# Import KNeighborsClassifier from sklearn.neighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets
iris = datasets.load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris['data'], iris['target'],\
            test_size=0.3,random_state=21, stratify=iris['target'])
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Test set predictions:\n {}".format(y_pred))
print(knn.score(X_test, y_test))
```


32.1.1.3. Hyper parameter tuning with GridSearchCV

How you know ? n_neighbors=6 ? do you have any way to find it.

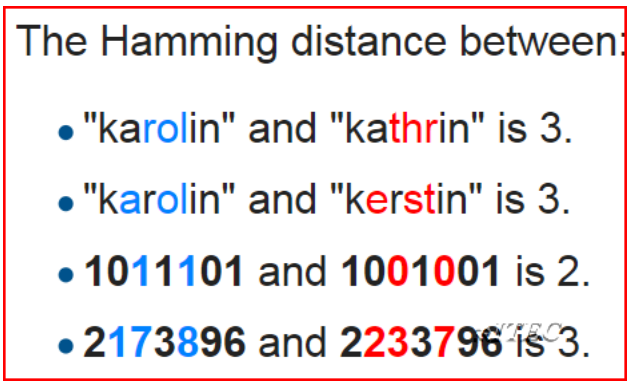
```
# Import KNeighborsClassifier from sklearn.neighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
knn = KNeighborsClassifier()
X= iris['data']
y=iris['target']
param_grid = {'n_neighbors': np.arange(1, 50)}
knn_cv = GridSearchCV(knn, param_grid, cv=5)
knn_cv.fit(X, y)
knn_cv.best_params_
knn_cv.best_score_
```


Note1: In above code change param_grid and observe result

```
param_grid = {'n_neighbors': np.arange(1, 50),
              'weights':['uniform','distance'],
'algorithm':['ball_tree', 'kd_tree', 'brute'] }
```

Note2: Explore maths related to Minkowski distance
https://en.wikipedia.org/wiki/Minkowski_distance

https://en.wikipedia.org/wiki/Hamming_distance



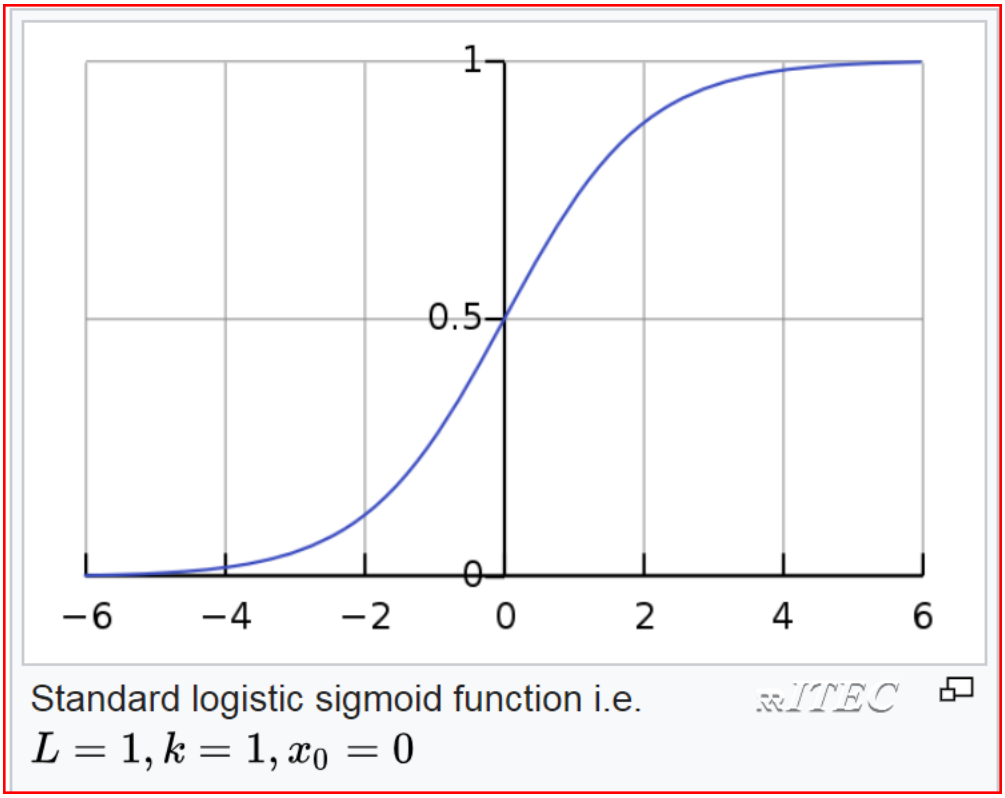
Note3: explore about RandomizedSearchCV it is similar to GridSearchCV

32.1.2 Linear Models

32.1.2.1. Logistic regression

1. Logistic regression, despite its name, is a linear model for classification rather than regression.
2. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier.
3. In this model, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.
4. A logistic function or logistic curve is a common "S" shape (sigmoid curve), with equation

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$



Standard logistic sigmoid function i.e. $L = 1, k = 1, x_0 = 0$

5. Maths behind it,just walk through it. If you are not understand do not be panic
   a. https://en.wikipedia.org/wiki/Logistic_function
   b. https://en.wikipedia.org/wiki/Logistic_regression

6. Refer http://scikit-learn.org/dev/modules/linear_model.html#logistic-regression

```
# 01. Import the necessary modules
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
# 02. load data
df = pd.read_csv(filepath_or_buffer ="pima-indians-diabetes.txt")
df.info()
#Create feature and target variable arrays
X = df[["pregnancies","glucose","diastolic","triceps","insulin","bmi","dpf","age"]]
type(X)
X.info()
#target variable
y=df["diabetes"]
type(y)
# 03. Create training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4, random_state=42)
# Create the classifier: logreg
logreg = LogisticRegression()
# Fit the classifier to the training data
logreg.fit(X_train, y_train)
# Predict the labels of the test set: y_pred
y_pred = logreg.predict(X_test)
# Compute and print the confusion matrix and classification report
print(confusion_matrix(y_test, y_pred))
```

```
print(classification_report(y_test, y_pred))
```

32.1.2.2. Understanding Classification Report

1. Precision: When it predicts yes, how often is it correct? When it predicts No, how often is it correct?
    a. Precision = TP/predicted yes = 66/98=0.67
    b. Precision = TN/predicted No = 174/210=0.828
2. True Positive Rate/Sensitivity/Recall: When it's actually yes, how often does it predict yes?
    a. TP/actual yes = 66/(36+66)=0.647
    b. TN/actual NO = 174/(174+32)=0.84
3. **F Score:** This is a weighted average of the true positive rate (recall) and precision
    a. F1-score = $$2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$
4. Support:
    a. actual No label patients count =84 + 10 = 94
    b. actual Yes label patients count =30+30 = 60

32.1.2.3. Confusion matrix and ROC Curve

5. Receiver Operating Characteristic(ROC) Curve
6. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings
7. In machine learning
    a. The true-positive rate is also known as sensitivity, recall or probability of detection
    Linear regression
    b. The false-positive rate is also known as the fall-out or probability of false alarm
8. Read https://en.wikipedia.org/wiki/Confusion_matrix
9. https://en.wikipedia.org/wiki/Receiver_operating_characteristic

```
# Import necessary modules

from sklearn.metrics import roc_curve
```

```
# Compute predicted probabilities: y_pred_prob
y_pred_prob = logreg.predict_proba(X_test)[:,1]
# Generate ROC curve values: fpr, tpr, thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
# Plot ROC curve
import matplotlib.pyplot as plt
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```

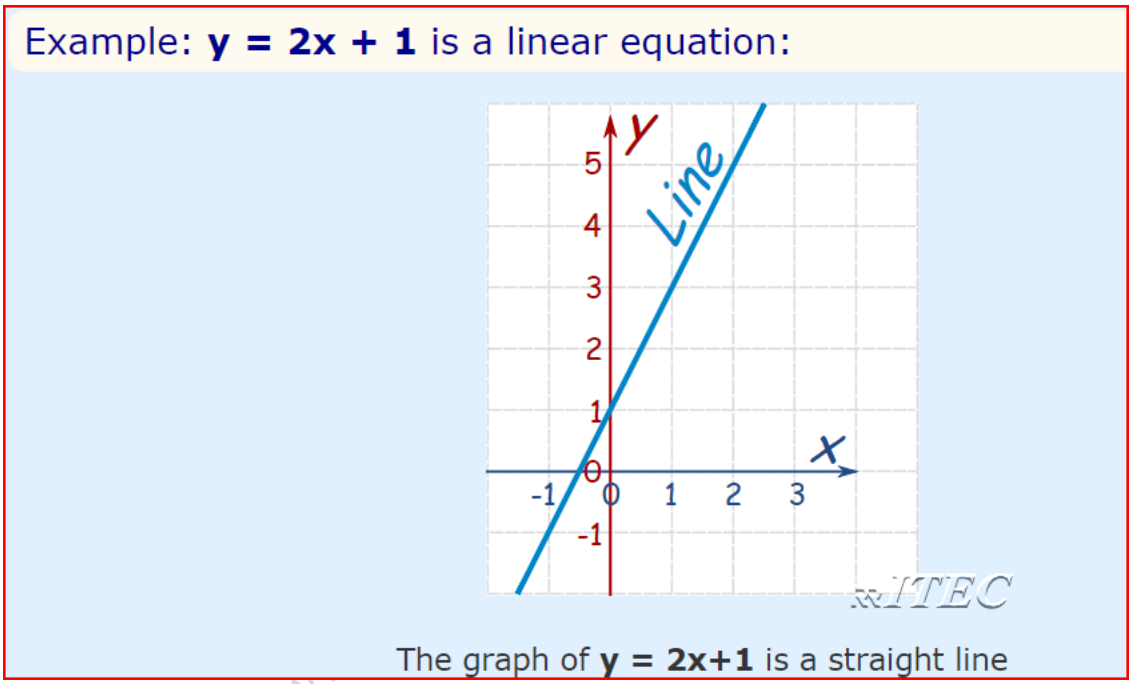
32.1.2.4. AUC computation

```
# Import necessary modules

from sklearn.model_selection import cross_val_score

from sklearn.metrics import roc_auc_score

# Compute predicted probabilities: y_pred_prob

y_pred_prob = logreg.predict_proba(X_test)[:,1]

# Compute and print AUC score

print("AUC: {}".format(roc_auc_score(y_test, y_pred_prob)))

# Compute cross-validated AUC scores: cv_auc

cv_auc = cross_val_score(logreg, X, y, cv=5, scoring='roc_auc')

# Print list of AUC scores

print("AUC scores computed using 5-fold cross-validation: {}".format(cv_auc))
```

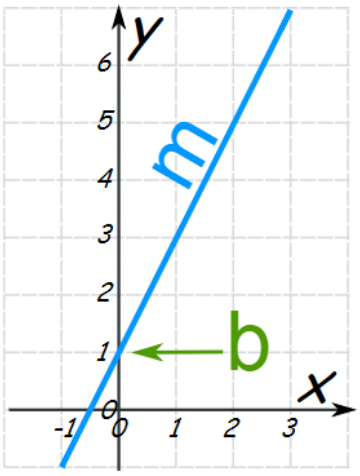
32.1.2.5. Hyperparameter tuning with GridSearchCV

```
# Import necessary modules
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import numpy as np
# Setup the hyperparameter grid
c_space = np.logspace(-5, 8, 15)
param_grid = {'C': c_space}
# Instantiate a logistic regression classifier: logreg
logreg = LogisticRegression()
# Instantiate the GridSearchCV object: logreg_cv
logreg_cv = GridSearchCV(logreg, param_grid, cv=5)
# Fit it to the data
logreg_cv.fit(X, y)
# Print the tuned parameter and score
print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
print("Best score is {}".format(logreg_cv.best_score_))
```

32.1.2.6. Linear regression

1. Maths:
    a. What is linear equation
        i. A linear equation is an equation for a straight line
        ii. 

    b. Try to draw a graph of below linear equations
        i. 5x = 6 + 3y
        ii. y/2 = 3 – x
2. Different forms of linear equations
    a. Slope-Intercept Form



$$y = mx + b$$

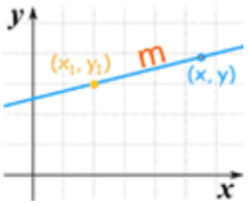
Slope (or Gradient) Y Intercept

Example: y = 2x + 1

- Slope: **m = 2**
- Intercept: **b = 1**

b.Point-Slope Form

$$y - y_1 = m(x - x_1)$$



Example: y − 3 = (¼)(x − 2)

It is in the form $\mathbf{y - y_1 = m(x - x_1)}$ where:

- $y_1 = 3$
- $m = ¼$
- $x_1 = 2$

c. General Form

$$Ax + By + C = 0$$

(A and B cannot both be 0)

Example: $3x + 2y - 4 = 0$

It is in the form **Ax + By + C = 0** where:

- A = 3
- B = 2
- C = −4

d. As a function

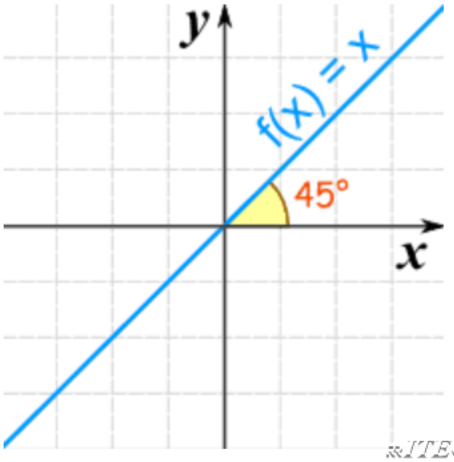
$$y = 2x - 3$$

$$f(x) = 2x - 3$$

These are the same!

e. The identity Function

i. There is a special linear function called the "Identity Function" : f(x) = x



It makes a 45° (its slope is 1)

ii. It is called "Identity" because what comes out is identical to what goes in:

1.

In	Out
0	0
5	5
−2	−2
…etc	…etc

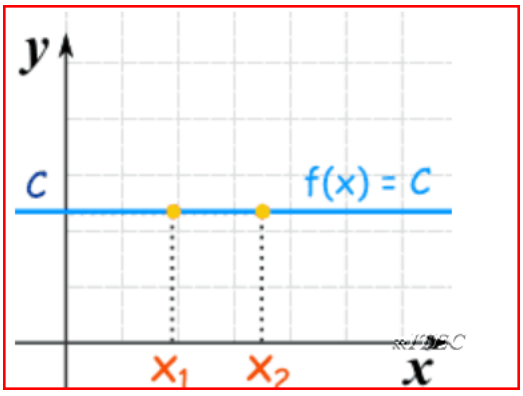
f. Constant Functions

i. Another special type of linear function is the Constant Function ... it is a horizontal line: f(x) = C (or) y = C

1. Example y = 3

ii. No matter what value of "x", f(x) is always equal to some constant value.

iii.



g.Can you identify linear equation ?

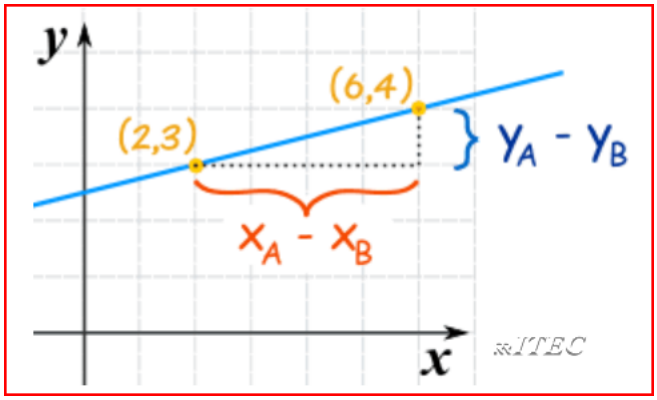
× $y^2 - 2 = 0$

× $3\sqrt{x} - y = 6$

× $x^3/2 = 16$

✓ y =3x-6

2. Can you calculate equation of a line from given two points?



a. Answer : y = x/4 + 5/2

b. Not able to understand then refer https://www.mathsisfun.com/algebra/line-equation-2points.html

3. What is regression?

   a. **Regression**: Predict a numerical outcome ("dependent variable(y)") from a set of inputs ("independent variables(x_1, x_2 ….x_n").

4. Examples: Sample questions

   a. How many units will we sell? (Regression) (0 to inf)

   b. What price will the customer pay for our product? (Regression) (0 to Inf)

5. Linear Regression

   - $y = \beta + \beta_1 x_1 + \beta_2 x_2 + ...$
   - y is linearly related to each x
   - Each x contributes additively to y

6. Refer http://www.learnbymarketing.com/tutorials/linear-regression-by-hand-in-excel/

7. Linear regression assumptions. Refer http://www.statisticssolutions.com/assumptions-of-linear-regression/

   a. Linear relationship

   b. Multivariate normality

   c. No or little multicollinearity

   d. No auto-correlation

   e. Homoscedasticity

8. Exercise 1: Linear regression with one variable

a.Step 1 of 3: Importing data and understand it.

```
# Import numpy and pandas
import numpy as np
import pandas as pd
# Read the CSV file into a DataFrame: df
df = pd.read_csv('gapminder.csv')
# Create arrays for features and target variable
y = df['life'].values
X = df['fertility'].values
# Print the dimensions of X and y before reshaping
print("Dimensions of y before reshaping: {}".format(y.shape))
print("Dimensions of X before reshaping: {}".format(X.shape))
# sklearn package needs two dimensional array, hence reshape X and y into 2D.
y = y.reshape(-1, 1)
X = X.reshape(-1, 1)
# Print the dimensions of X and y after reshaping
print("Dimensions of y after reshaping: {}".format(y.shape))
print("Dimensions of X after reshaping: {}".format(X.shape))
```


b. Step 2 of 3: Fit linear model and observe the score R^2

```
# Import LinearRegression
from sklearn.linear_model import LinearRegression
# Create the regressor: reg
reg = LinearRegression()
```

```
# Fit the model to the data
reg.fit(X, y)
# Print R^2
print("The score R^2 is: {}".format(reg.score(X, y)))
```

c.Step 3 of 3: Generate some test data and observe linear graph

```
# Create the prediction space
prediction_space = np.linspace(min(X), max(X)).reshape(-1,1)
# Compute predictions over the prediction space: y_pred
y_pred = reg.predict(prediction_space)
# Plot regression line
import matplotlib.pyplot as plt
plt.plot(prediction_space, y_pred, color='black', linewidth=3)
plt.show()
```


2. Exercise 2: Calculate **R-Squared**.

a.R^2 is a measure of how well the model fits.

b.R^2 value should be between 0 and 1

i. Near 1: model fits well

ii. Near 0: over fit / bad model. In this case better to use avg value to predict.

c.To calculate R^2 formula is $\mathbf{R^2 = 1-RSS/SS_{Tot}}$

i. Residual sum of squares (variance from model) $RSS = \sum (y - prediction)^2$

ii. Total sum of squares (variance of data) $SS_{Tot} = \sum (y - \overline{y})^2$

d.Observe help document and also understand calculateing manually

```
from sklearn.metrics import r2_score
```

```
help(r2_score)
```

3. Exercise 3: Calculate Root Mean Squared Error (RMSE)

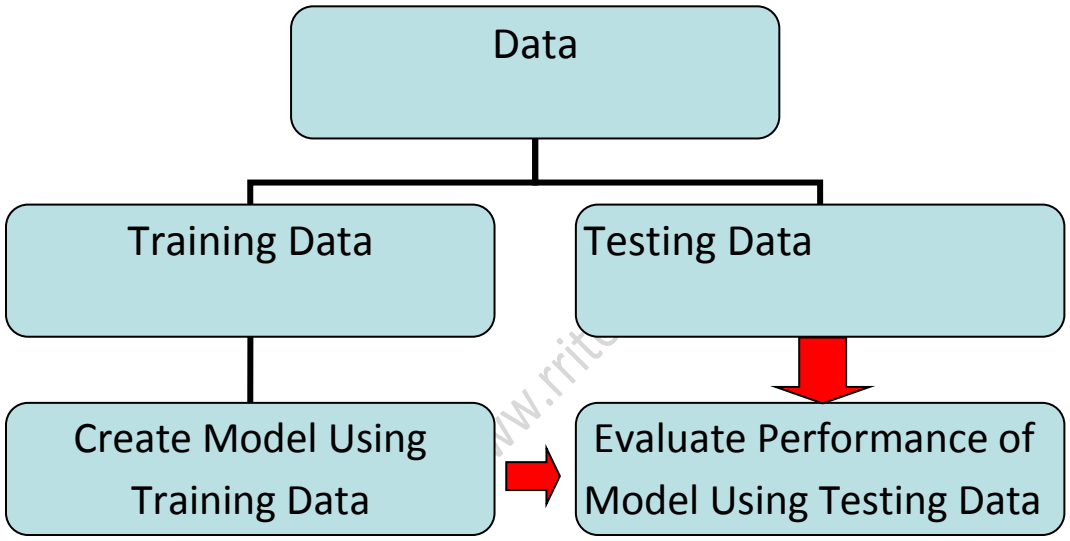
a.Formula is $RMSE = \sqrt{\overline{(pred - y)^2}}$

i. pred − y:The error or residuals

b.Make sure RMSE is smaller than standard deviation (std) (σ)

4. Exercise 4: Train/ Test Split

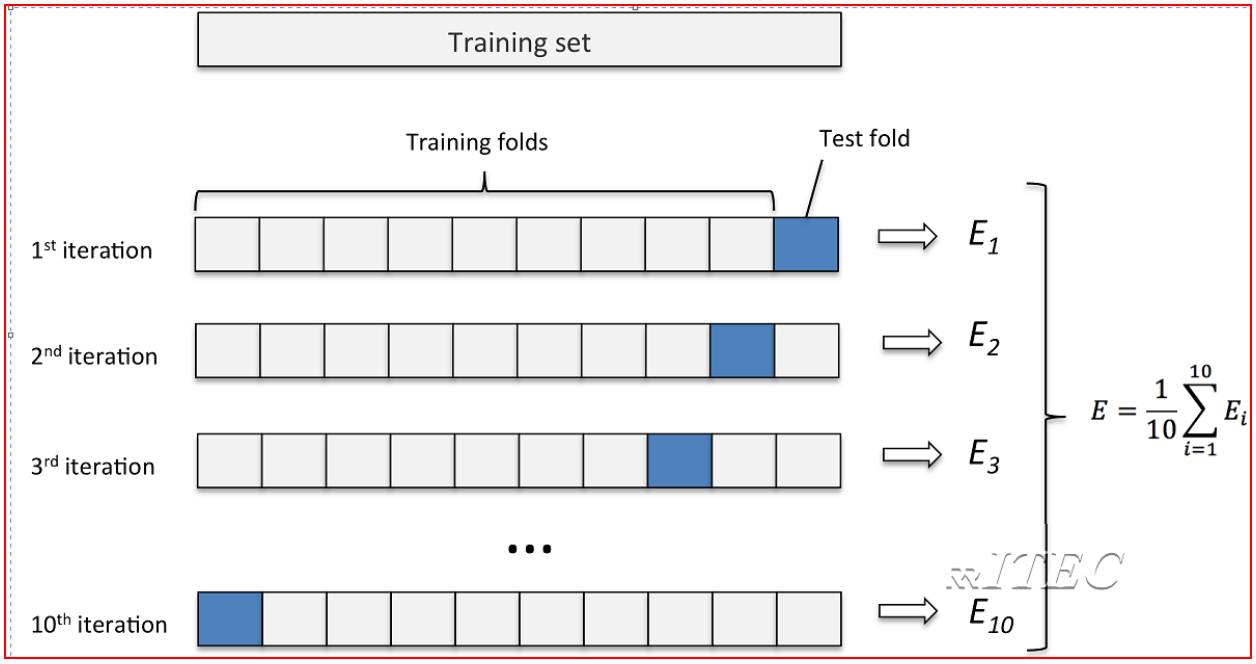
1. To improve performance use **Train/test split for regression**




```
# Step 0 of 5: Import necessary modules

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import mean_squared_error,r2_score


# Step 1 of 5: Load Data Set
 df = pd.read_csv('gapminder.csv')
# Get required variables
X = (df[df.columns[[0,1,2,3,4,5,6,8]]].values)
y = df['life'].values
# Step 2 of 5: Create training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=42)
# Step 3 of 5: Create Model
# Create the regressor: reg_all
reg_all = LinearRegression()
# Fit the regressor to the training data
reg_all.fit(X_train, y_train)
# Step 4 of 5: Predict using model
# Predict on the test data: y_pred
y_pred = reg_all.predict(X_test)
# Step 5 of 5: Test the Performance
# Compute and print R^2 # it shoud be near to one
print("R^2: {}".format(r2_score(y_test,y_pred)))
# Compute and print RMSE # it should be smaller than standard deviation
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error: {}".format(rmse))
std = np.std(y_test)
print("Standard Deviation σ : {}".format(std))
rmse <std # For  good model #it must be true
```

2. Exercise 5: Cross validation




```
# Import the necessary modules
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
# Create a linear regression object: reg
reg = LinearRegression()
# Compute 5-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(reg, X, y, cv=5)
# Print the 5-fold cross-validation scores
print(cv_scores)
# Print the average 5-fold cross-validation score
print("Average 5-Fold CV Score: {}".format(np.mean(cv_scores)))
```

3. If we increase folds accuracy will be increases .however performance will be impacted

```
In [57]: %timeit cross_val_score(reg, X, y, cv = 3)
2.34 ms ± 22.3 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

In [58]: %timeit cross_val_score(reg, X, y, cv = 10)
7.34 ms ± 82.6 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

4. Exercise 4: Linear regression using boston data(One More Example for practice)

```
from sklearn.datasets import load_boston

boston = load_boston()

print(boston.data.shape)

print(" ##### 02. datatype of boston data set ##### ")

print(type(boston))

print(" ##### 03. Keys of boston data set ##### ")

print(boston.keys())

print(" ##### 04. Observe boston data key ##### ")

print(boston.data)

X = boston.data

print(" ##### 05. Observe boston target key ##### ")

print(boston.target)

y = boston.target

print(" ##### 06. Import required modules ##### ")

from sklearn import linear_model

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error

import numpy as np

print(" ##### 07. Split data as train and test  ##### ")

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.3, random_state=42)

print(" ##### 08. Fit model using train data  ##### ")
```

```
reg_all = linear_model.LinearRegression()

reg_all.fit(X_train, y_train)

print(" ##### 09. predict on test data  ##### ")

y_pred = reg_all.predict(X_test)

print("  ###### 10. Compute and print R^2 and RMSE #####  ")

print("R^2: {}".format(reg_all.score(X_test, y_test)))

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print("Root Mean Squared Error: {}".format(rmse))
```

32.1.2.7. Ridge and lasso regression

1. Explore ridge and lasso regression methods
   https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/

32.1.3 Support Vector Machines (SVM)

32.1.3.1. Introduction

1. "Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for
    a. Classification
    b. Regression
2. However, it is mostly used in classification problems.
3. Pros:
    a. It works really well with clear margin of separation
    b. It is effective in high dimensional spaces.
    c. It is effective in cases where number of dimensions is greater than the number of samples.
    d. It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
4. Cons:
    a. It doesn't perform well, when we have large data set(>100K) because the required training time is higher
    b. It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
    c. SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python scikit-learn library.
5. Refer more theory http://scikit-learn.org/stable/modules/svm.html
6. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
7. Refer document 01 Data Science Lab Copy\03 Reference\support_vector_machines_succinctly.pdf
8. Refer help document

```
from sklearn import svm

help(svm)
```

32.1.3.2. Support Vectors Classification (SVC)

9. Exercise 1: Iris flowers classification using SVC

```
# Step 0 of 5: import required modules

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import r2_score

 # Step 1 of 5: import data

iris = load_iris()

X = iris.data[:, :2]  # we only take the first two features.

y = iris.target

# Step 2 of 5: Split Data

X_train, X_test, y_train, y_test = train_test_split(X, y,

                                    test_size=0.3,

                                    random_state=42)

# Step 3 of 5: Create model using train data

model = SVC(kernel='linear', C=1, gamma=1)

# there is various option associated with it, like changing kernel,gamma and C value.

#Will discuss more

model.fit(X_train, y_train)

# Step 4 of 5: Predict Output using test data

y_pred= model.predict(X_test)

# Step 5 of 5: find the performance /accuracy

r2_score(y_test,y_pred)
```

32.1.3.3. Tune parameters of SVM(SVC)

1. Tuning parameters effectively improves the model performance. Let's look at the list of parameters available with SVM.

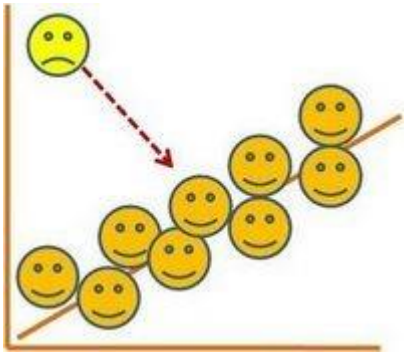
```
from sklearn.model_selection import GridSearchCV

def svc_param_selection(X, y, nfolds):

    Cs = [0.001, 0.01, 0.1, 1, 10]

    gammas = [0.001, 0.01, 0.1, 1]

    kernels = ['linear', 'poly', 'rbf']

    param_grid = {'kernel':kernels,'C': Cs, 'gamma' : gammas}

    grid_search = GridSearchCV(SVC(), param_grid, cv=nfolds)

    grid_search.fit(X, y)

    grid_search.best_params_

    return (grid_search.best_params_,grid_search.best_score_)

svc_param_selection(X_train,y_train,5)
```


32.1.3.4. Support Vectors Regression(SVR)

2. Please refer sklearn documentation

32.1.4 Pre-processing of machine learning data

32.1.4.1. Outliers

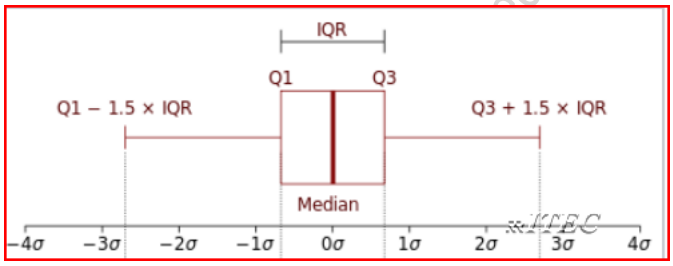


1. Exercise 1: Understanding boxplot and outliers

   a. Do you know about boxplot?

   b. Can you find outliers using boxplot?

   c. What is your understanding on below diagram?



   d. Explore https://en.wikipedia.org/wiki/Box_plot

```
import numpy as np
arr = [10, 386, 479, 627, 20, 523, 482, 483, 542, 699, 535,
       617, 577, 471, 615, 583, 441, 562, 563, 527, 453, 530,
       433, 541, 585, 704, 443, 569, 430, 637, 331, 511, 552, 496,
       484, 566, 554, 472, 335, 440, 579, 341, 545, 615, 548, 604,
       439, 556, 442, 461, 624, 611, 444, 578, 405, 487, 490, 496,
       398, 512, 422, 455, 449, 432, 607, 679, 434, 597, 639, 565,
       415, 486, 668, 414, 665, 763, 557, 304, 404, 454, 689, 610,
```

```
      483, 441, 657, 590, 492, 476, 437, 483, 529, 363, 711, 543,1000]
elements = np.array(arr)
import matplotlib.pyplot as plt
plt.boxplot(elements)
q1,q3 =np.percentile(elements,[25,75])
q1
q3
iqr = q3-q1
lower = q1-1.5*iqr
lower
upper = q3+1.5*iqr
upper
np_arr = np.array(arr)
np_arr[np.logical_or(np_arr<lower,np_arr>upper)]
```


Exercise 2: delete outliers

```
def reject_outliers(data, m=2):
    return data[abs(data - np.mean(data)) < m * np.std(data)]
without_outliers =reject_outliers(np_arr)
np.count_nonzero(without_outliers)
np.count_nonzero(np_arr)
import matplotlib.pyplot as plt
plt.boxplot(without_outliers)
```

32.1.4.2. Working with categorical features

1. Exercise 2: Exploring categorical features

```
# Import pandas

import pandas as pd

# Read 'gapminder.csv' into a DataFrame: df

df = pd.read_csv('gapminder.csv')

# Create a boxplot of life expectancy per region

import matplotlib.pyplot as plt

df.boxplot('life', 'Region', rot=60)

# Show the plot

plt.show()
```


By observing above graph answer below questions

Question 1: Which is the best place to live?

America or East Asia and Pacific

Question 2: Which is not right place to live?

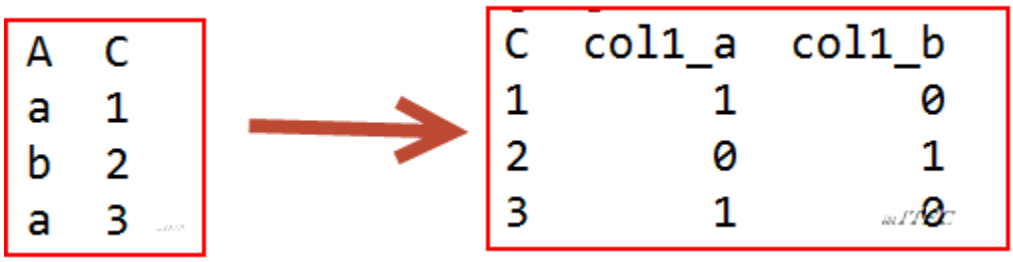
Sub Saharan Africa

2. Exercise 2: Creating dummy variables

   a. scikit-learn does not accept non-numerical features. Hence we need to convert non numeric into numeric

   b. Observe help document pd.get_dummies?

```
Df1 = pd.DataFrame({'A': ['a', 'b', 'a'],'C': [1, 2, 3]})

Df1

pd.get_dummies(Df1, prefix='col1')
```




```
# Create dummy variables: df_region

df_region = pd.get_dummies(df)

# Print the columns of df_region

print(df_region.columns)

# Drop 'Region_America' from df_region

df_region = pd.get_dummies(df, drop_first=True)

# Print the new columns of df_region

print(df_region.columns)
```


c. It is similar to one hot encoding of sklearn ML package and to_categarical of keras DL package. Explore and these to using below reading exercise

https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/

32.1.4.3. Regression with categorical features using ridge algorithm

1. Exercise 1: Regression with categorical features using ridge algorithm

```
# Import necessary modules

from sklearn.model_selection import cross_val_score

from sklearn.linear_model import Ridge

y = df_region['life'].values
```

```
type(y)

X = (df_region[df_region.columns[[0,1,2,3,4,5,6,8,9,10,11,12,13]]].values)

type(X)

# Instantiate a ridge regressor: ridge

ridge = Ridge(alpha=0.5, normalize=True)

# Perform 5-fold cross-validation: ridge_cv

ridge_cv = cross_val_score(ridge, X, y, cv=5)

# Print the cross-validated scores

print(ridge_cv)
```

32.1.4.4. Handling missing data

1. Data can have missing values for a number of reasons such as observations that were not recorded and data corruption.
2. Handling missing data is important as many machine learning algorithms do not support data with missing values.
3. We should learn
    a. How to marking invalid or corrupt values as missing in your dataset.
    b. How to remove rows with missing data from your dataset.
    c. How to impute missing values with mean values in your dataset.
4. Exercise 1: Dropping missing data


```
import pandas as pd

import numpy as np

df = pd.read_csv("house-votes-84.csv")

df.info()

# Convert '?' to NaN

df[df == '?'] = np.nan

# Print the number of NaNs

print(df.isnull().sum())

# Print shape of original DataFrame

print("Shape of Original DataFrame: {}".format(df.shape))
```

```
# Drop missing values and print shape of new DataFrame

df = df.dropna()

# Print shape of new DataFrame

print("Shape of DataFrame After Dropping All Rows with Missing Values:
{}".format(df.shape))
```

5. Refer https://machinelearningmastery.com/handle-missing-data-python/

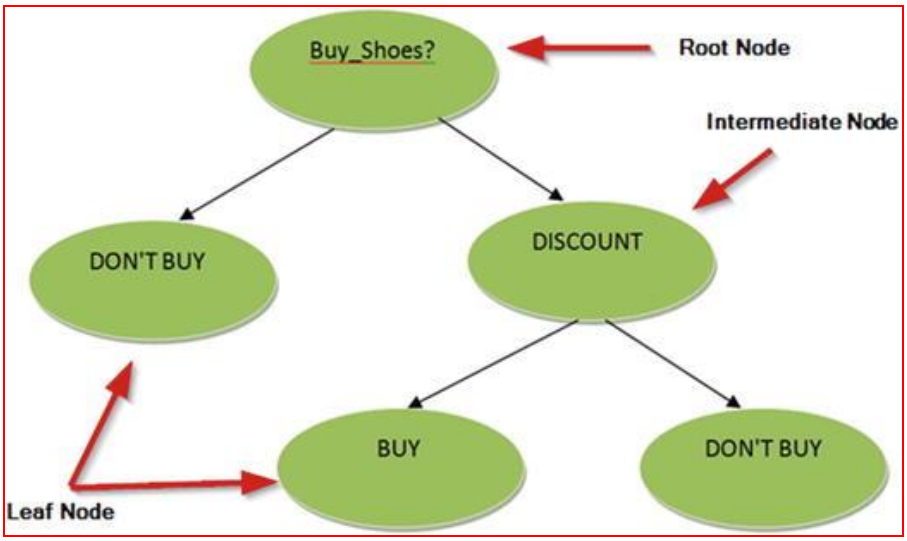
32.1.1 ML Pipeline (Putting it all together)

1. Sequentially apply a list of transforms and a final **estimator**(ML algorithm)
2. Purpose of the pipeline is to assemble several step in one order

```
# Import necessary modules
import pandas as pd
import numpy as np
from sklearn.preprocessing import Imputer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
df = pd.read_csv("house-votes-84.csv")
df=df.replace('y', 1)
df
df=df.replace('n', 0)
df
df[df == '?'] = np.nan
y = df['Class Name'].values
type(y)
X = (df[df.columns[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]]].values)
type(X)
X.shape
X
# Setup the pipeline steps: steps
steps = [('imputation', Imputer(missing_values='NaN', strategy='most_frequent', axis=0)),
        ('SVM', SVC())]
```

```
# Create the pipeline: pipeline

pipeline = Pipeline(steps)


# Create training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# Fit the pipeline to the train set

pipeline.fit(X_train, y_train)


# Predict the labels of the test set

y_pred = pipeline.predict(X_test)


# Compute metrics

print(classification_report(y_test, y_pred))
```

32.1.2 Tree Based Models

1. What is decision tree?
    a. Sequence of if-else questions about individual features
    b. Decision-Tree has a hierarchy of nodes.
2. **Node**: question or prediction.
3. Building Blocks of a Decision-Tree are three nodes
    a. Root: no parent node, question giving rise to two children nodes.
    b. Internal node: one parent node, question giving rise to two children nodes.
    c. Leaf: one parent node, no children nodes --> prediction
    d. 

32.1.2.1. Decision Tree for Classification

4. **Exercise 1**: predict whether a tumor is malignant(cancer) or benign(non Cancer) based on two features the mean radius of the tumor (radius_mean) and its mean number of concave points (concave points_mean)

```
# Step 0 of 5: Import required Modules

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.metrics import accuracy_score

# Step 1 of 5: load data

df = pd.read_csv("Wisconsin_Breast_Cancer_Dataset.csv")

df.info()

X = df[["radius_mean","concave points_mean"]]

y=df["diagnosis"]

y = y.replace('M',1)

y = y.replace('B',0)

SEED = 1 # for reproducing

# Step 2 of 5: Create training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=SEED,
stratify=y)

X_train.shape # (455, 2)

y_train.shape # (455,)

X_test.shape # (114, 2)

y_test.shape # (114,)

# Step 3 of 5: Create DecisionTreeClassifier Model with a maximum depth of 6

dt = DecisionTreeClassifier(max_depth=6, random_state=SEED)

# Fit dt to the training set

dt.fit(X_train, y_train)

# Step 4 of 5: Predict test set labels using model

y_pred = dt.predict(X_test)

# Step 5 of 5: Test the Performance

# Compute test set accuracy

acc = accuracy_score(y_test, y_pred)
```

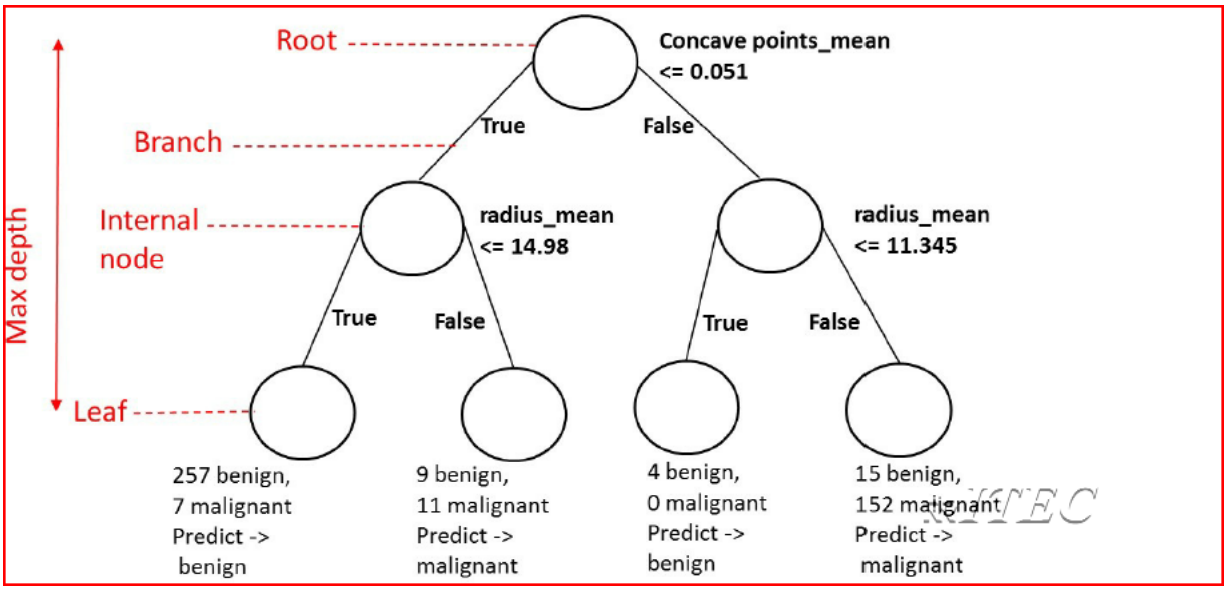
```
print("Test set accuracy: {:.2f}".format(acc))
```

Note: Not bad! Using only two features, your tree was able to achieve an accuracy of 89% ☺

32.1.2.2. LogisticRegression Vs Decision Tree Classification

```
# Import LogisticRegression from sklearn.linear_model

from sklearn.linear_model import  LogisticRegression

# Instatiate logreg

logreg = LogisticRegression(random_state=1)

# Fit logreg to the training set

logreg.fit(X_train, y_train)

# predict

y_pred1 = logreg.predict(X_test)

acc1 = accuracy_score(y_test, y_pred1)

acc1
```

32.1.2.3. Information Gain (IG)



1. At each node, split the data based on information gain value or gini value

32.1.2.3.1. Entropy and Information Gain

Entropy

Entropy $H(S)$ is a measure of the amount of uncertainty in the (data) set S (i.e. entropy characterizes the (data) set S).

$$H(S) = \sum_{c \in C} -p(c) \log_2 p(c)$$

Where,

- S – The current (data) set for which entropy is being calculated (changes every iteration of the ID3 algorithm)
- C – Set of classes in S C={ yes, no }
- $p(c)$ – The proportion of the number of elements in class c to the number of elements in set S

When $H(S) = 0$, the set S is perfectly classified (i.e. all elements in S are of the same class).

In ID3, entropy is calculated for each remaining attribute. The attribute with the **smallest** entropy is used to split the set S on this iteration. The higher the entropy, the higher the potential to improve the classification here.

Information gain

Information gain $IG(A)$ is the measure of the difference in entropy from before to after the set S is split on an attribute A. In other words, how much uncertainty in S was reduced after splitting set S on attribute A.

$$IG(A, S) = H(S) - \sum_{t \in T} p(t) H(t)$$

Where,

- $H(S)$ – Entropy of set S
- T – The subsets created from splitting set S by attribute A such that $S = \bigcup_{t \in T} t$
- $p(t)$ – The proportion of the number of elements in t to the number of elements in set S
- $H(t)$ – Entropy of subset t

In ID3, information gain can be calculated (instead of entropy) for each remaining attribute. The attribute with the **largest** information gain is used to split the set S on this iteration.

Must read:

https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1

Refer:

https://medium.com/udacity/shannon-entropy-information-gain-and-picking-balls-from-buckets-5810d35d54b4

http://dni-institute.in/blogs/gini-index-work-out-example/

32.1.2.3.2. Gini Index

1. Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified. It means an attribute with lower gini index should be preferred

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

2. Exercise :To identify breast cancer by using gini and entropy

```
# Step 0 of 5: Import required Modules

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

# Step 1 of 5: load data

df = pd.read_csv("Wisconsin_Breast_Cancer_Dataset.csv")

df.info()

X = df.iloc[:,2:32]

type(X)

X.info()

y=df["diagnosis"]

y = y.replace('M',1)

y = y.replace('B',0)

SEED = 1 # for reproducing

# Step 2 of 5: Create training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y,

                                    test_size = 0.2,
```

```
                                    random_state=SEED,
                                    stratify=y)
X_train.shape # (455, 2)
y_train.shape # (455,)
X_test.shape # (114, 2)
y_test.shape # (114,)
# Step 3 of 5: Create Model using criterion as entropy
# Create dt_entropy model, set 'entropy' as the information criterion
dt_entropy = DecisionTreeClassifier(max_depth=8,
                                    criterion='entropy',
                                    random_state=SEED)
# Fit dt_entropy to the training set
dt_entropy.fit(X_train, y_train)
# Use dt_entropy to predict test set labels
y_pred = dt_entropy.predict(X_test)
# Evaluate accuracy_entropy
accuracy_entropy = accuracy_score(y_test, y_pred)
# Step 4 of 5: Create Model using criterion as gini
# Instantiate dt_gini, set 'gini' as the information criterion
dt_gini= DecisionTreeClassifier(max_depth=8,
                                    criterion='gini',
                                    random_state=SEED)
# Fit dt_entropy to the training set
dt_gini.fit(X_train, y_train)
# Use dt_entropy to predict test set labels
```

```
y_pred_gini = dt_gini.predict(X_test)

# Evaluate accuracy_entropy

accuracy_gini = accuracy_score(y_test, y_pred_gini)

# Step 5 of 5: compare entropy and gini accuracy

# Print accuracy_entropy

print('Accuracy achieved by using entropy: ', accuracy_entropy)

# Print accuracy_gini

print('Accuracy achieved by using the gini index: ', accuracy_gini)
```

Note: Notice how the two models achieve exactly the same accuracy. Most of the time, the gini index and entropy lead to the same results. The gini index is slightly faster to compute and is the default criterion used in the DecisionTreeClassifier model of scikit-learn.

3.

32.1.3 Decision Tree For Regression

1. Decision trees can also be applied to regression problems, using the DecisionTreeRegressor

```
# Step 0 of 5: Import required Modules

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean_squared_error,r2_score

# Step 1 of 5: load data

df = pd.read_csv("auto-mpg.csv")

df.info()

df = pd.get_dummies(df)

df.info()

X = df.iloc[:,1:]

type(X)

X.info()

y=df["mpg"]

SEED = 3 # for reproducing

# Step 2 of 5: Create training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,  random_state=SEED)

X_train.shape # (313, 8)

y_train.shape # (313,)

X_test.shape # (79, 8)

y_test.shape # (79,)

# Step 3 of 5: Create DecisionTreeRegressor Model with a maximum depth of 8
```

```
dt = DecisionTreeRegressor(max_depth=8,
                           min_samples_leaf=0.13,
                           random_state=SEED)
# Fit dt to the training set
dt.fit(X_train, y_train)
# Step 4 of 5: Predict test set using model
# Compute y_pred
y_pred = dt.predict(X_test)
# Step 5 of 5: Test the Performance
# Compute mse_dt
rmse_dt = np.sqrt(mean_squared_error(y_test, y_pred))
# Compute standard deviation
sd = np.std(y_test)
# Print rmse_dt and sd
print("Test set RMSE of dt: {:.2f}".format(rmse_dt))
print("Test set RMSE of dt: {:.2f}".format(sd))
# Note: RMSE shoud be smaller than standard deviation
# find r2 it should be near to 1
r2_score(y_test,y_pred)
```

32.1.4 Linear regression vs regression tree

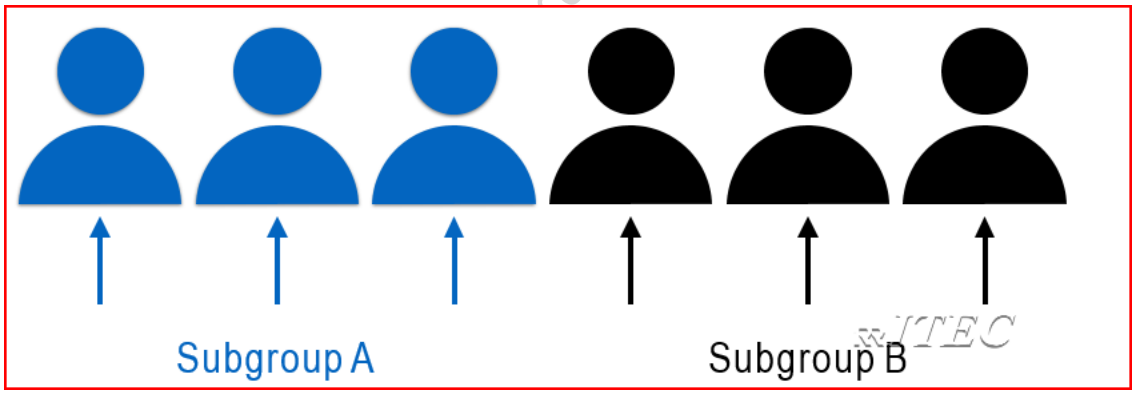
```
from sklearn.linear_model import LinearRegression

# Step 1: Create linear Model

lr = LinearRegression()

# Fit lr to the training data set

lr.fit(X_train, y_train)

# Step 2: Predict test set using model

y_pred_lr = lr.predict(X_test)

# Step 3: Compute rmse_lr

rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))

rmse_lr

# Note : compare rmse_dt and rmse_lr, what you noticed
```

32.2 Unsupervised Learning

1. Unsupervised learning: Uses unlabelled data
2. Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.
3. We can derive this structure by clustering the data based on relationships among the variables in the data.
4. Simple words : Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data
5. **Clustering Examples:**

   a. Example 1: Take a collection of 1,000,000 different genes, and find a way to automatically group these genes into groups that are somehow similar or related by different variables, such as lifespan, location, roles, and so on.

   b. Example 2: Finding homogenous subgroups within larger group based on the fact People have features such as income, education attainment and gender.



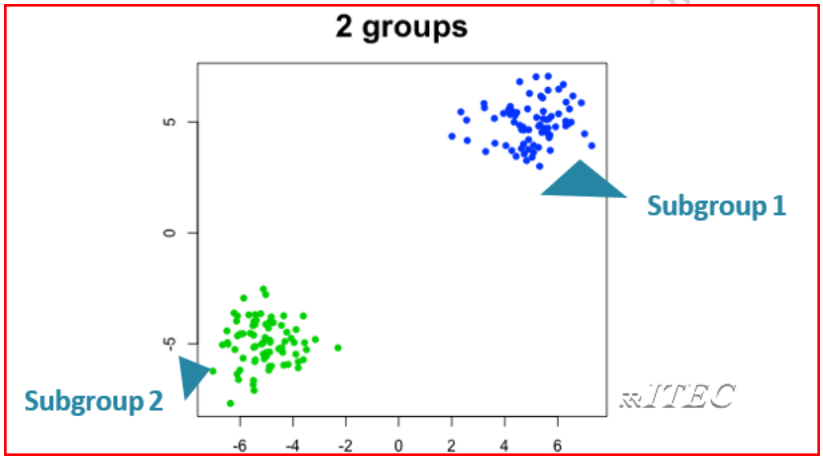
   c. Example 3: Market Segmentation

   d. Example 4: Movies Segmentation

e.Non-clustering: The "Cocktail Party Algorithm", allows you to find structure in a chaotic/disorder environment. (i.e. identifying individual voices and music from a mesh of sounds at a cocktail party).

32.2.1 k-means clustering

1. **k-means clustering** algorithm ,Breaks observations into pre-defined number of clusters



2. In k means clustering, we have to specify the number of clusters we want the data to be grouped into. The algorithm randomly assigns each observation to a cluster, and finds the centroid of each cluster. Then, the algorithm iterates through two steps:

   a.Reassign data points to the cluster whose centroid is closest.

   b.Calculate new centroid of each cluster.

3. These two steps are repeated till the within cluster variation cannot be reduced any further.

4. The within cluster variation is calculated as the sum of the euclidean distance between the data points and their respective cluster centroids.

```
# Step 1 of 5: import  required modules
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# Step 2 of 5: load iris data
iris = datasets.load_iris()
samples = iris.data
# Step 3 of 5: k-means clustering with scikit-learn
model = KMeans(n_clusters=3)
model.fit(samples)
labels = model.predict(samples)
print(labels)
# Step 4 of 5: Cluster labels for new samples
new_samples = [[ 5.7, 4.4, 1.5, 0.4],[ 6.5, 3., 5.5, 1.8],[ 5.8, 2.7, 5.1, 1.9]]
new_labels = model.predict(new_samples)
print(new_labels)
# Step 5 of 5: Scatter plot of sepal length vs petal length
xs = samples[:,0]
ys = samples[:,2]
plt.scatter(xs, ys, c=labels)
plt.show()
```


5. Crosstab of labels and species

```
def species_label(theta):
    if theta==0:
        return iris.target_names[0]
```

```
    if theta==1:
        return iris.target_names[1]
    if theta==2:
        return iris.target_names[2]
#create dataframe
import pandas as pd
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = [species_label(theta) for theta in iris.target]
#create crosstab
pd.crosstab(df['species'],labels)
```

6. Clustering quality:

1. Inertia measures clustering quality
2. Measures how spread out the clusters are (lower is beter)
3. Distance from each sample to centroid of its cluster
4. After fit(), available as attribute inertia_
5. k-means atempts to minimize the inertia when choosing clusters

```
In [13]: print(model.inertia_)
78.9408414261
```


7. How many clusters to choose?

8. Choose an "elbow" in the inertia plot
9. Where inertia begins to decrease more slowly

```
ks = range(1,11)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)
    # Fit model to samples
```

```
    model.fit(samples)
    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)
# Plot ks vs inertias
plt.plot(ks, inertias, '-o')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```

10. Exercise 1: Do same job using red wine data set ,you will learn

a.Standardscaler

b. Make_pipeline

https://datahexa.com/kmeans-clustering-with-wine-dataset/

Level 05 of 08: Deep Learning

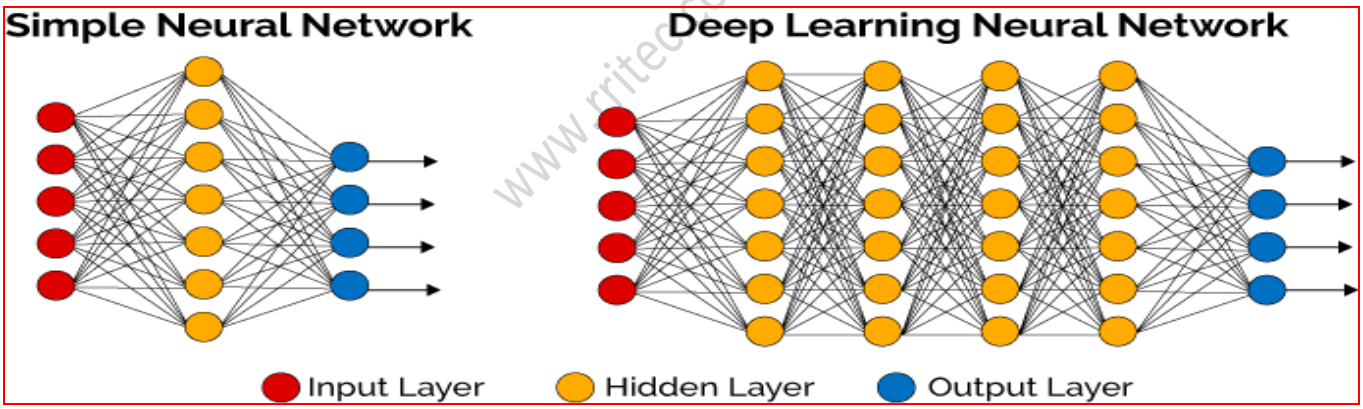
33. Deep learning

33.1 Introduction

1. Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example

2. Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks.

3. The term “deep” usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as required.



4. Deep learning models are trained by using large sets of labelled data and neural network architectures that learn features directly from the data without the need for manual feature extraction.

Examples of Deep Learning at Work

1. Deep learning applications are used in industries from automated driving to medical devices.

2. **Automated Driving**: Automotive researchers are using deep learning to automatically detect objects such as stop signs and traffic lights. In addition, deep learning is used to detect pedestrians, which helps decrease accidents.

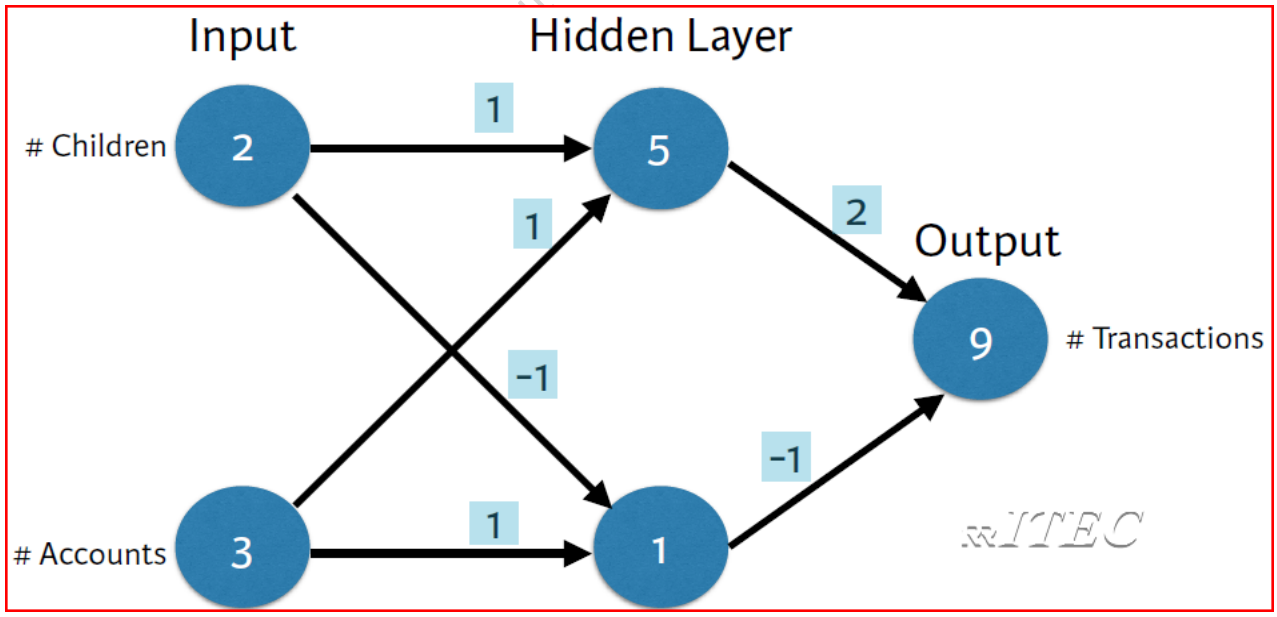
3. **Aerospace and Defence:** Deep learning is used to identify objects from satellites that locate areas of interest, and identify safe or unsafe zones for troops.
4. **Medical Research:** Cancer researchers are using deep learning to automatically detect cancer cells. Teams at UCLA built an advanced microscope that yields a high-dimensional data set used to train a deep learning application to accurately identify cancer cells.
5. **Industrial Automation:** Deep learning is helping to improve worker safety around heavy machinery by automatically detecting when people or objects are within an unsafe distance of machines.
6. **Electronics:** Deep learning is being used in automated hearing and speech translation. For example, home assistance devices that respond to your voice and know your preferences are powered by deep learning applications.

33.2 Forward propagation

1. Bank transactions example
2. Make predictions based on:

    a.Number of children

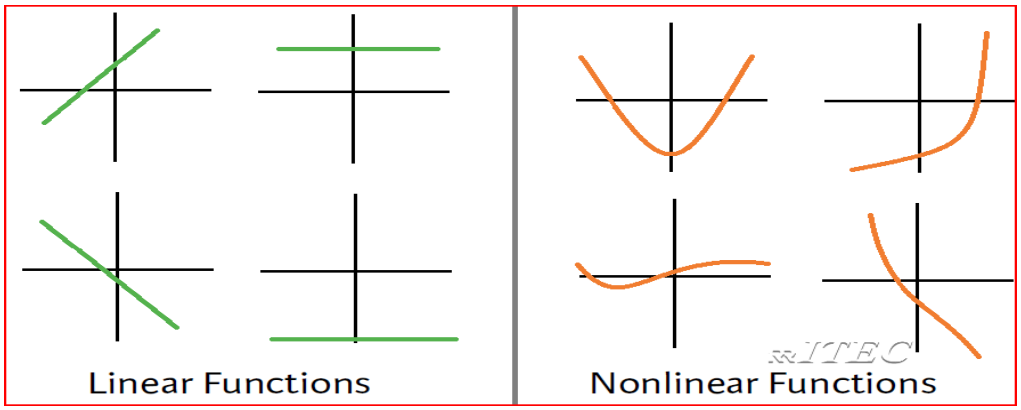
    b.Number of existing accounts



```
import numpy as np
input_data = np.array([2, 3])
weights = {'node_0': np.array([1, 1]),
'node_1': np.array([-1, 1]),
'output': np.array([2, -1])}
node_0_value = (input_data * weights['node_0']).sum()
node_1_value = (input_data * weights['node_1']).sum()
hidden_layer_values = np.array([node_0_value, node_1_value])
print(hidden_layer_values) #[5, 1]
output = (hidden_layer_values * weights['output']).sum()
print(output)
```

33.3 Activation functions

1. These are two types
    a. Linear
    b. Non linear



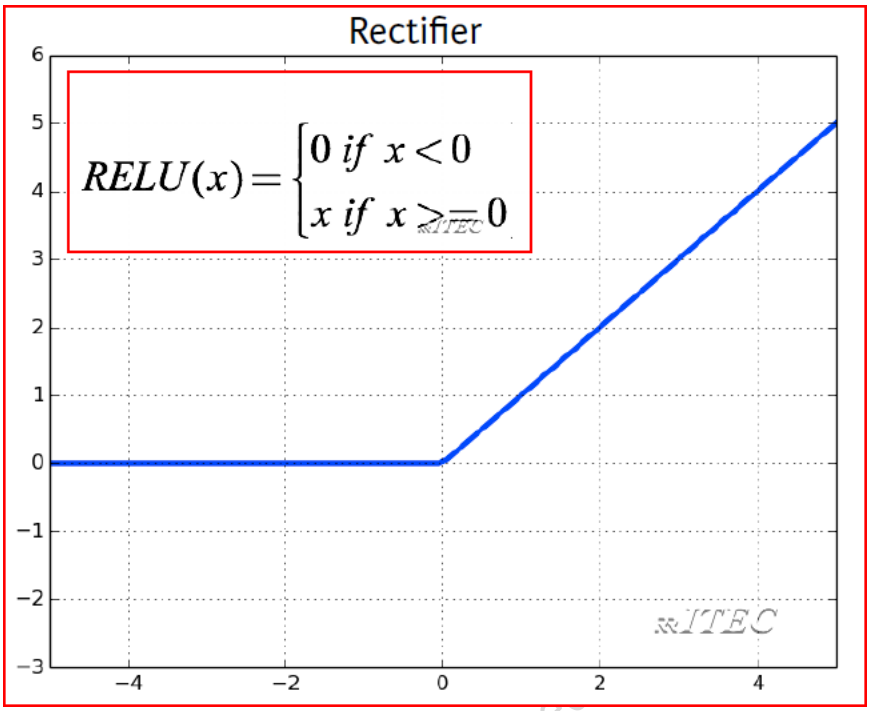
2. Applied to hidden node inputs to produce node output

```
import numpy as np
input_data = np.array([2, 3])
```

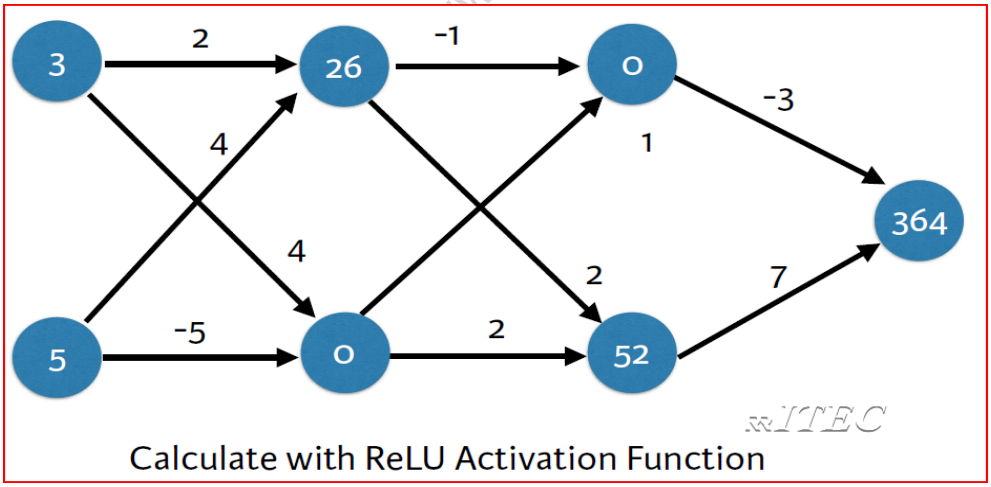
```
weights = {'node_0': np.array([1, 1]), 'node_1': np.array([-1, 1]),'output': np.array([2, -1])}

node_0_input = (input_data * weights['node_0']).sum()

node_0_output = np.tanh(node_0_input)

node_1_input = (input_data * weights['node_1']).sum()

node_1_output = np.tanh(node_1_input)

hidden_layer_output = np.array([node_0_output, node_1_output])

output = (hidden_layer_output * weights['output']).sum()

print(output)
```

33.4 Deeper networks

1. ReLU (Rectified Linear Activation).



2. Multiple hidded layers




```
import numpy as np

input_data = np.array([3, 5])

weights = {'node_0': np.array([2, 4]),

           'node_1': np.array([4, -5]),
```

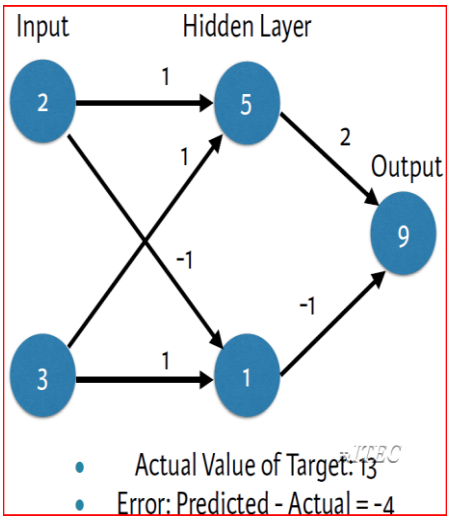
```
        'node_2': np.array([-1, 1]),
        'node_3': np.array([2, 2]),
        'output': np.array([-3, 7])}

node_0_output = (input_data * weights['node_0']).sum()
node_0_output_relu = np.maximum(node_0_output,0)
node_1_output = (input_data * weights['node_1']).sum()
node_1_output_relu = np.maximum(node_1_output,0)
hidden_layer1_output = np.array([node_0_output_relu, node_1_output_relu])
hidden_layer1_output_relu = np.maximum(hidden_layer1_output,0)

node_2_output = (hidden_layer1_output_relu * weights['node_2']).sum()
node_2_output_relu = np.maximum(node_2_output,0)
node_3_output = (hidden_layer1_output_relu * weights['node_3']).sum()
node_3_output_relu = np.maximum(node_3_output,0)
hidden_layer2_output = np.array([node_2_output_relu, node_3_output_relu])
hidden_layer2_output_relu = np.maximum(hidden_layer2_output,0)

output = (hidden_layer2_output_relu * weights['output']).sum()
output_relu = np.maximum(output,0)
print(output_relu)
```

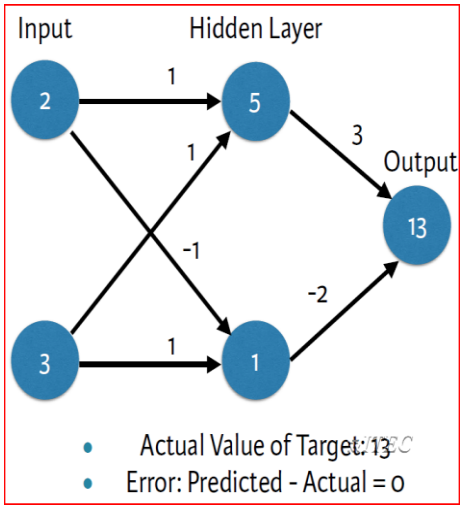

3. Find more ways in implementing relu function

   https://stackoverflow.com/questions/32109319/how-to-implement-the-relu-function-in-numpy

33.5 Need for optimization

1. Understanding how weights change model accuracy?





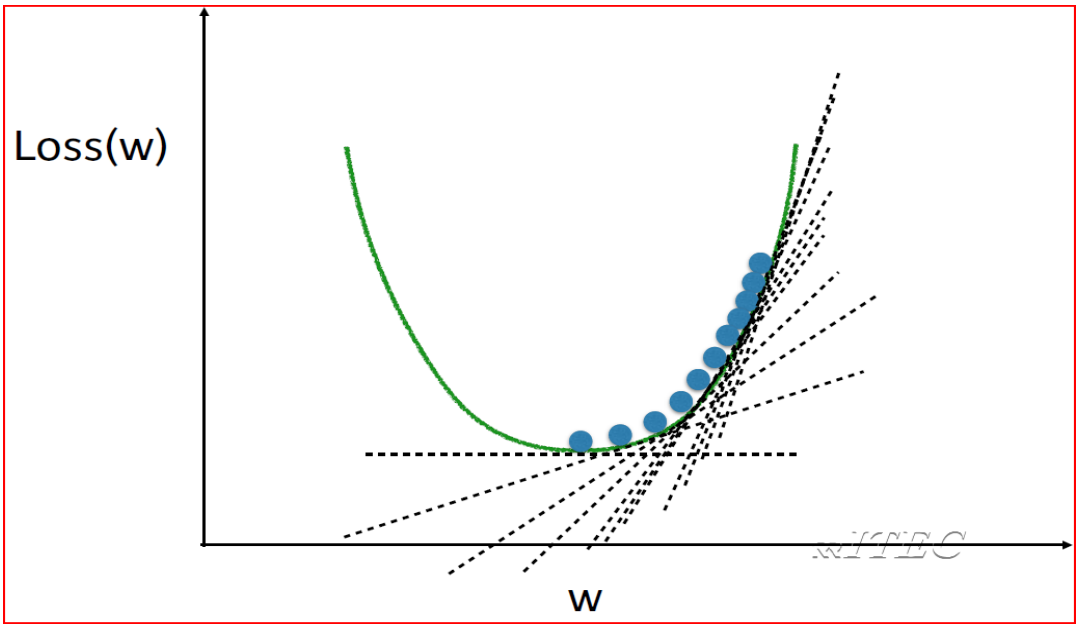
2. Hence in ordered to get good quality/optimization, choosing right weights play main role.
3. **Exercise 1: Coding how weight changes affect accuracy**


```
#Step 1: Create relu function
def relu(my_input):
    return(max(0, my_input))
#Step 2: Create predict with network function
def predict_with_network(input_data_point, weights):
    node_0_input = (input_data_point * weights['node_0']).sum()
    node_0_output = relu(node_0_input)
    node_1_input = (input_data_point * weights['node_1']).sum()
    node_1_output = relu(node_1_input)
    hidden_layer_values = np.array([node_0_output, node_1_output])
    input_to_final_layer = (hidden_layer_values * weights['output']).sum()
    model_output = relu(input_to_final_layer)
    return(model_output)
```

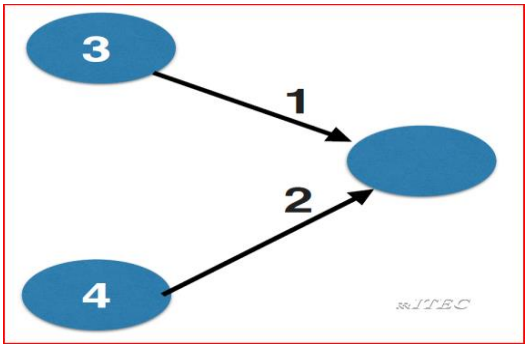
```
#Step 3: Use above functions to predict
# The data point you will make a prediction for
input_data = np.array([2, 3])
# Sample weights
weights_0 = {'node_0': [1, 1],
             'node_1': [-1, 1],
             'output': [2, -1]
# The actual target value, used to calculate the error
target_actual = 3
# Make prediction using original weights
model_output_0 = predict_with_network(input_data, weights_0)
# Calculate error: error_0
error_0 = model_output_0 - target_actual
# Create weights that cause the network to make perfect prediction (3): weights_1
weights_1 = {'node_0': [1, 1],
             'node_1': [-1, 1],
             'output': [3, -2]
            }
# Make prediction using new weights: model_output_1
model_output_1 = predict_with_network(input_data, weights_1)
# Calculate error: error_1
error_1 = model_output_1 - target_actual
print(error_0);print(error_1)
```

33.6 Gradient descent

1. How many



2. If the slope is positive:
    a. Going opposite the slope means moving to lower numbers
    b. Subtract the slope from the current value
    c. Too big a step might lead us astray
3. Solution: learning rate
    a. Update each weight by subtracting
    b. learning rate * slope
4. Exercise 1: Calculating Slope and Improving model weights



```
# Step 1 of 2: Calculate slope/gradient
import numpy as np
# Define weights
weights = np.array([1, 2])
input_data = np.array([3, 4])
target = 6
learning_rate = 0.01
# Calculate the predictions: preds
preds = (weights * input_data).sum()
# Calculate the error: error
error = preds - target
print(error)
# Calculate the slope: slope/gradient
gradient = 2 * input_data * error
gradient
# Step 2 of 2: Improving Model weights
weights_updated = weights - learning_rate * gradient
preds_updated = (weights_updated * input_data).sum()
error_updated = preds_updated - target
print(error_updated)
```


5. Exercise 2: Calculating Slope , Improving model weights and mse

```
# Step 1 of 3: Calculating slopes
import numpy as np
weights = np.array([0,2,1])
```

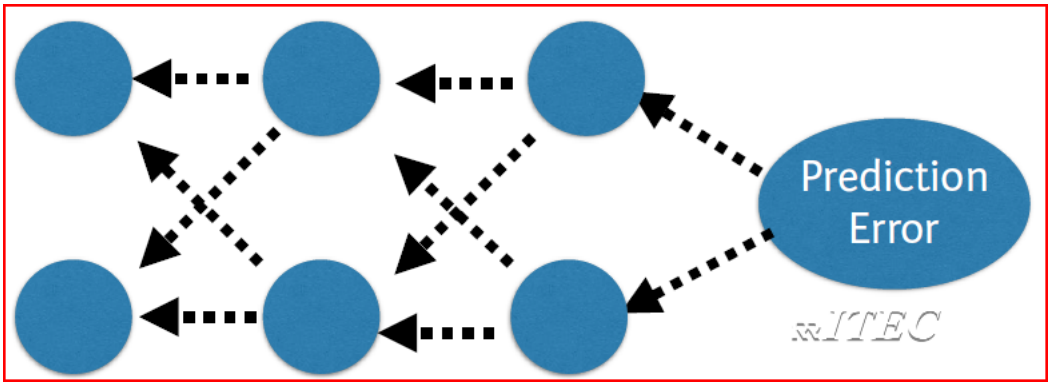
```
input_data = np.array([1,2,3])

target = 0

# Calculate the predictions: preds

preds = (weights * input_data).sum()

# Calculate the error: error

error = preds - target

# Calculate the slope: slope

slope = 2 * input_data * error

# Print the slope

print(slope)

#######################################################

# Step 2 of 3: Improving model weights

# Set the learning rate: learning_rate

learning_rate = 0.01

# Update the weights: weights_updated

weights_updated = weights - learning_rate * slope

# Get updated predictions: preds_updated

preds_updated = (weights_updated * input_data).sum()

# Calculate updated error: error_updated

error_updated = preds_updated - target

# Print the original error

print(error)

# Print the updated error

print(error_updated)

#######################################################

# Step 3 of 3:Making multiple updates to weights
```

```
def get_error(input_data, target, weights):
    preds = (weights * input_data).sum()
    error = preds - target
    return(error)
def get_slope(input_data, target, weights):
    error = get_error(input_data, target, weights)
    slope = 2 * input_data * error
    return(slope)
def get_mse(input_data, target, weights):
    errors = get_error(input_data, target, weights)
    mse = np.mean(errors**2)
    return(mse)
n_updates = 20
mse_hist = []
# Iterate over the number of updates
for i in range(n_updates):
    # Calculate the slope: slope
    slope = get_slope(input_data, target, weights)
    # Update the weights: weights
    weights = weights - 0.01 * slope
    # Calculate mse with new weights: mse
    mse = get_mse(input_data, target, weights)
    # Append the mse to mse_hist
    mse_hist.append(mse)
# Plot the mse history
import matplotlib.pyplot as plt
```

```
plt.plot(mse_hist)

plt.xlabel('Iterations')

plt.ylabel('Mean Squared Error')

plt.show() # Notice that, the mean squared error decreases as the number of iterations go up.
```

33.7 Backpropagation

1. Update weights using error and iterate till we meet actual target data



2. Try to understand the process, however you will generally use a library that implements forward and backward propagations.

3. It is working based on chain rule of calculus

Differentiation Formulas:

1. $\frac{d}{dx}(x) = 1$
2. $\frac{d}{dx}(ax) = a$
3. $\frac{d}{dx}(x^n) = nx^{n-1}$
4. $\frac{d}{dx}(\cos x) = -\sin x$
5. $\frac{d}{dx}(\sin x) = \cos x$
6. $\frac{d}{dx}(\tan x) = \sec^2 x$
7. $\frac{d}{dx}(\cot x) = -\csc^2 x$
8. $\frac{d}{dx}(\sec x) = \sec x \tan x$
9. $\frac{d}{dx}(\csc x) = -\csc x(\cot x)$
10. $\frac{d}{dx}(\ln x) = \frac{1}{x}$
11. $\frac{d}{dx}(e^x) = e^x$

Integration Formulas:

1. $\int 1\,dx = x + C$
2. $\int a\,dx = ax + C$
3. $\int x^n dx = \frac{x^{n+1}}{n+1} + C,\ n \neq -1$
4. $\int \sin x\,dx = -\cos x + C$
5. $\int \cos x\,dx = \sin x + C$
6. $\int \sec^2 x\,dx = \tan x + C$
7. $\int \csc^2 x\,dx = -\cot x + C$
8. $\int \sec x(\tan x)\,dx = \sec x + C$
9. $\int \csc x(\cot x)\,dx = -\csc x + C$
10. $\int \frac{1}{x} dx = \ln|x| + C$
11. $\int e^x dx = e^x + C$

4. If a variable z depends on the variable y, y variables depends on the variable x.We can say z depends on x . we can write chain rule as for below

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}.$$

5. Find the derivative of sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] \\
&= \frac{d}{dx}\left(1+\mathrm{e}^{-x}\right)^{-1} \\
&= -(1+e^{-x})^{-2}(-e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{(1+e^{-x})-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\
&= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\
&= \sigma(x) \cdot (1-\sigma(x))
\end{aligned}$$

6. Please refer it is very clear here https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

7. The relationship between forward and backward propagation

   a. If you have gone through 4 iterations of calculating slopes (using backward propagation) and then updated weights, how many times must you have done forward propagation?

      i. 0

      ii. 1

      iii. 4

      iv. 8

b. If your predictions were all exactly right, and your errors were all exactly 0, the slope of the loss function with respect to your predictions would also be 0. In that circumstance, which of the following statements would be correct?

   i. **The updates to all weights in the network would also be 0.**

   ii. The updates to all weights in the network would be dependent on the activation functions.

   iii. The updates to all weights in the network would be proportional to values from the input data.

33.8 Creating keras Regression Model

1. Keras Model building has mainly 4 steps
    1. Specify Architecture
        i. Define input nodes/columns
        ii. Define Hidden layers
        iii. Define hidden nodes
        iv. Define activation functions
        v. Define output
    2. Compile
        i. Define optimizer
        ii. Define Loss function
    3. Fit
        i. Applying backpropagation
        ii. Updating weights
    4. Predict
2. **Step 1 of 4: Specify the architecture**
    a. Predict **workers wages** based on characteristics like their industry, education, level of experience… etc


```
# Step 1: load data
import pandas as pd
df = pd.read_csv("hourly_wages.csv")
df
# Get required variables as numpy array
predictors = (df[df.columns[[1,2,3,4,5,6,7,8,9]]].values)
target = (df[df.columns[0]].values)
#Step 2: Specifying a model
```

```
# Import necessary modules

import keras

from keras.layers import Dense

from keras.models import Sequential

# Save the number of columns in predictors: n_cols

n_cols = predictors.shape[1]

# Set up the model: model

model = Sequential()

# Add the first layer

model.add(Dense(50, activation='relu', input_shape=(n_cols,)))

# Add the second layer

model.add(Dense(32, activation='relu'))

# Add the output layer

model.add(Dense(1))
```


3. Step 2 of 4: Compiling the model
    a. To compile the model, you need to specify the optimizer and loss function
        i. Specify the optimizer
            1. Many options and mathematically complex
            2. “Adam” is usually a good choice
        ii. Loss function
            1. “mean_squared_error” common for regression
    b. Read more on optimizers https://keras.io/optimizers/#adam
    c. See the original paper of adam https://arxiv.org/abs/1412.6980v8

```
# Compile the model
```

```
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])

# Verify that model contains information from compiling

print("Loss function: " + model.loss)
```

4. **Step 3 of 4: Fitting the model**

    a. What is fitting a model

        i. Applying backpropagation and gradient descent with your data to update the weights

        ii. Scaling data before fitting can ease optimization

```
# Fit the model

model.fit(predictors, target)
```

5. **Step 4 of 4: predict**

```
model.predict(predictors)
```

33.9 Creating keras Classification Models

1. 'categorical_crossentropy' loss function
2. Similar to log loss: Lower is better
3. Add metrics = ['accuracy'] to compile step for easy-to-understand diagnostics
4. Output layer has separate node for each possible outcome, and uses 'softmax' activation

Process:

1. Modelling with a new dataset "Titanic" for a classification problem
2. You will use predictors such as age, fare and where each passenger embarked from to predict who will survive.

```
# understand data

import pandas as pd

from keras.utils import to_categorical

df = pd.read_csv("titanic_all_numeric_train.csv")

predictors = df.drop(['survived'], axis=1).as_matrix()

target = to_categorical(df.survived)

df = pd.read_csv("titanic_all_numeric_test.csv")

test_data = df.drop(['survived'], axis=1).as_matrix()

# Import necessary modules

import keras

from keras.layers import Dense

from keras.models import Sequential

# Set up the model

model = Sequential()

# Save the number of columns in predictors: n_cols

n_cols = predictors.shape[1]
```

```
# Add the first layer
model.add(Dense(32, activation='relu', input_shape=(n_cols,)))
# Add the output layer
model.add(Dense(2, activation='softmax'))
# Compile the model
model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Fit the model
model.fit(predictors, target)
```

33.10 Using models

1. Save
2. Reload
3. Make predictions

```
from keras.models import load_model
model.save('model_file.h5')
my_model = load_model('model_file.h5')
# Calculate predictions: predictions
predictions = model.predict(test_data)
# Calculate predicted probability of survival: predicted_prob_true
predicted_prob_true = predictions[:,1]
# print predicted_prob_true
print(predicted_prob_true)
```

33.11 Understanding Model Optimization

1. Why optimization is hard in deep learning
    a. Simultaneously optimizing 1000s of parameters with complex relationships
    b. Updates may not improve model meaningfully
    c. Updates too small (if learning rate is low) or too large (if learning rate is high)
2. **Scenario:** Try to optimize a model at a very low learning rate, a very high learning rate, and a "just right" learning rate. We need to look at the results after running this exercise, remembering that a **low value for the loss function is good**
    a. Exercise 1: Let us optimize using Stochastic Gradient Descent

```
# Step 1 of 3: create a tuple
input_shape=(10,)
type(input_shape)
input_shape
# Step 2 of 3: Create model as a function to loop from starting
def get_new_model(input_shape = input_shape):
    model = Sequential()
    model.add(Dense(100, activation='relu', input_shape = input_shape))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    return(model)
# Step 3 of 3: Changing optimization parameters
# Import the SGD optimizer
from keras.optimizers import SGD
# Create list of learning rates: lr_to_test
lr_to_test = [.000001, 0.01, 1]
# Loop over learning rates
for lr in lr_to_test:
```

```
    print('\n\nTesting model with learning rate: %f\n'%lr )

    # Build new model to test, unaffected by previous models

    model = get_new_model()

    # Create SGD optimizer with specified learning rate: my_optimizer

    my_optimizer = SGD(lr=lr)

    # Compile the model

    model.compile(optimizer=my_optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

    # Fit the model

    model.fit(predictors, target)
```

3. Which of the following could prevent a model from showing an improved loss in its first few epochs?

    a. Learning rate too low.

    b. Learning rate too high.

    c. Poor choice of activation function.

    d. All of the above.

33.12 Model Validation

1. Validation in deep learning
    a. Commonly used validation is **split** rather than **cross validation**
    b. Deep learning widely used on large datasets
    c. Single validation score is based on large amount of data, and is reliable
    d. Repeated training from cross-validation would take long time
2. **Exercise 1: Evaluating model accuracy on validation dataset**

```
# Save the number of columns in predictors: n_cols

n_cols = predictors.shape[1]

input_shape = (n_cols,)

# Specify the model

model = Sequential()

model.add(Dense(100, activation='relu', input_shape = input_shape))

model.add(Dense(100, activation='relu'))

model.add(Dense(2, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Fit the model

hist = model.fit(predictors, target, validation_split=0.3)
```

3. **Exercise 2: Early stopping, Optimizing the optimization**

```
# Import EarlyStopping

from keras.callbacks import EarlyStopping

# Save the number of columns in predictors: n_cols

n_cols = predictors.shape[1]

input_shape = (n_cols,)
```

```
# Specify the model

model = Sequential()

model.add(Dense(100, activation='relu', input_shape = input_shape))

model.add(Dense(100, activation='relu'))

model.add(Dense(2, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Define early_stopping_monitor

early_stopping_monitor = EarlyStopping(patience=2)

# Fit the model

model.fit(predictors, target, epochs=30, validation_split=0.3,
callbacks=[early_stopping_monitor])
```


4. Exercise 3: Experimenting with wider networks

```
# Define early_stopping_monitor

early_stopping_monitor = EarlyStopping(patience=2)

# Create the new model: model_1

model_1 = Sequential()

# Add the first and second layers

model_1.add(Dense(10, activation='relu', input_shape=input_shape))

model_1.add(Dense(10, activation='relu'))

# Add the output layer

model_1.add(Dense(2, activation='softmax'))

# Compile model_1

model_1.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
# Create the new model: model_2

model_2 = Sequential()

# Add the first and second layers

model_2.add(Dense(100, activation='relu', input_shape=input_shape))

model_2.add(Dense(100, activation='relu'))

# Add the output layer

model_2.add(Dense(2, activation='softmax'))

# Compile model_2

model_2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Fit model_1

model_1_training = model_1.fit(predictors, target, epochs=15, validation_split=0.2,
callbacks=[early_stopping_monitor], verbose=False)

# Fit model_2

model_2_training = model_2.fit(predictors, target, epochs=15, validation_split=0.2,
callbacks=[early_stopping_monitor], verbose=False)

# Create the plot

import matplotlib.pyplot as plt

plt.plot(model_1_training.history['val_loss'], 'r', model_2_training.history['val_loss'], 'b')

plt.xlabel('Epochs')

plt.ylabel('Validation score')

plt.show()
```


5. Note: Model_2 blue line in the graph has less loss ,so it is good

6. **Exercise 4: Adding layers to a network(Deeper Network)**

   a. In above exercise 3, you've seen how to experiment with wider networks. In this exercise, you'll try a deeper network (more hidden layers).

```
The input shape to use in the first hidden layer
```

```
input_shape = (n_cols,)

# Create the new model: model_1

model_1 = Sequential()

# Add one hidden layer

model_1.add(Dense(50, activation='relu', input_shape=input_shape))

# Add the output layer

model_1.add(Dense(2, activation='softmax'))

# Compile model_1

model_1.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Create the new model: model_2

model_2 = Sequential()

# Add the first, second, and third hidden layers

model_2.add(Dense(50, activation='relu', input_shape=input_shape))

model_2.add(Dense(50, activation='relu'))

model_2.add(Dense(50, activation='relu'))

# Add the output layer

model_2.add(Dense(2, activation='softmax'))

# Compile model_2

model_2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Fit model 1

model_1_training = model_1.fit(predictors, target, epochs=20, validation_split=0.4,
callbacks=[early_stopping_monitor], verbose=False)

# Fit model 2

model_2_training = model_2.fit(predictors, target, epochs=20, validation_split=0.4,
callbacks=[early_stopping_monitor], verbose=False)
```

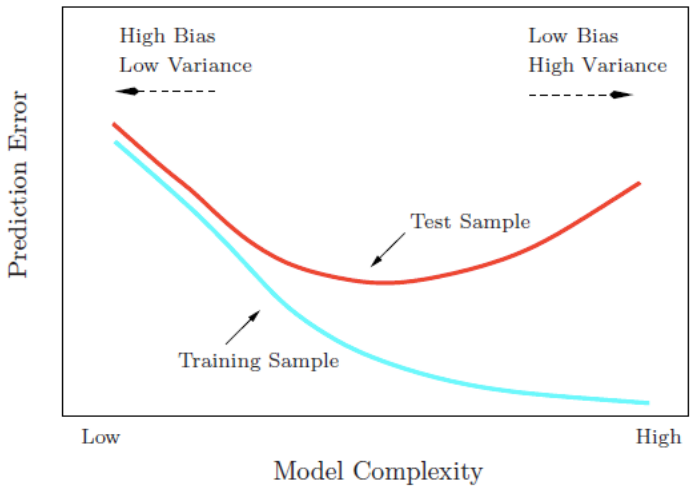
```
# Create the plot
plt.plot(model_1_training.history['val_loss'], 'r', model_2_training.history['val_loss'], 'b')
plt.xlabel('Epochs')
plt.ylabel('Validation score')
plt.show()
```

33.13 Model Capacity

1. How much wider and deeper networks are good?
    a. Workflow for optimizing model capacity
        i. Start with a small network
        ii. Gradually increase capacity
        iii. Keep increasing capacity until validation score is no longer Improving

Hidden Layers	Nodes Per Layer	Mean Squared Error	Next Step
1	100	5.4	Increase Capacity
1	250	4.8	Increase Capacity
2	250	4.4	Increase Capacity
3	250	4.5	Decrease Capacity
3	200	4.3	Done

2. If we are not checking as shown above ,there is a chance to overfit the model



Level 06 of 08: Project on Deep Learning

34. Project using keras and tensorflow

Business Statement: MNIST ` Digit Recognition in Keras.

Step 1 of 5: Setting up the Environment

```
# imports for array-handling and plotting
import numpy as np
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
# for testing on CPU
os.environ['CUDA_VISIBLE_DEVICES'] = ''
# keras imports for the dataset and building our neural network
from keras.datasets import mnist
from keras.models import Sequential, load_model
from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils
```

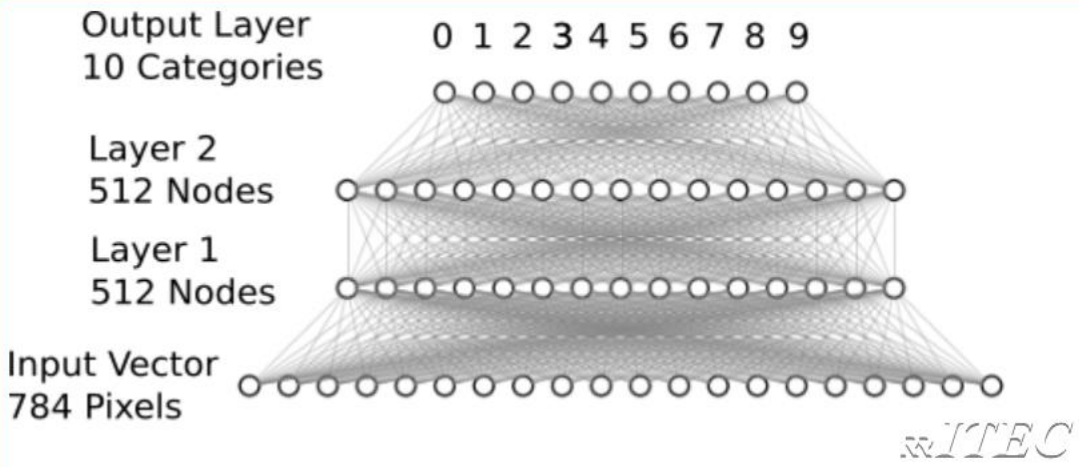

Step2 of 5: Understanding and Preparing the Dataset

```
# Load the dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# Observe some images
fig = plt.figure()
for i in range(9):
  plt.subplot(3,3,i+1)
```

```
  plt.tight_layout()

  plt.imshow(X_train[i], cmap='gray', interpolation='none')

  plt.title("Class {}".format(y_train[i]))

  plt.xticks([])

  plt.yticks([])

fig

# In order to train our neural network to classify images we first have to unroll the # height ×width pixel format into one big vector - the input vector. So its length

# must be 28 * 28 = 784. But let's graph the distribution of our pixel values.

fig = plt.figure()

plt.subplot(2,1,1)

plt.imshow(X_train[0], cmap='gray', interpolation='none')

plt.title("Class {}".format(y_train[0]))

plt.xticks([])

plt.yticks([])

plt.subplot(2,1,2)

plt.hist(X_train[0].reshape(784))

plt.title("Pixel Value Distribution")

fig

#Note that the pixel values range from 0 to 255: the background majority close to 0, and those close to #255 representing the digit.


# Normalizing the input data helps to speed up the training. Also, it reduces the chance of getting stuck # in local optima, since we're using stochastic gradient descent to find the optimal weights for the #network.

#Let's reshape our inputs to a single vector and normalize the pixel values to lie between 0 and 1.

# let's print the shape before we reshape and normalize
```

```
print("X_train shape", X_train.shape)

print("y_train shape", y_train.shape)

print("X_test shape", X_test.shape)

print("y_test shape", y_test.shape)

# building the input vector from the 28x28 pixels

X_train = X_train.reshape(60000, 784)

X_test = X_test.reshape(10000, 784)

X_train = X_train.astype('float32')

X_test = X_test.astype('float32')

# normalizing the data to help with the training

X_train /= 255

X_test /= 255

# print the final input shape ready for training

print("Train matrix shape", X_train.shape)

print("Test matrix shape", X_test.shape)

#Let us see the number of expected outcomes

print(np.unique(y_train, return_counts=True))

#Let's encode our categories - digits from 0 to 9 - using one-hot encoding. The result is a vector
with a #length equal to the number of categories. The vector is all zeroes except in the position
for the #respective category. Thus a '5' will be represented by [0,0,0,0,0,1,0,0,0,0]

# one-hot encoding using keras' numpy-related utilities

n_classes = 10

print("Shape before one-hot encoding: ", y_train.shape)

Y_train = np_utils.to_categorical(y_train, n_classes)

Y_test = np_utils.to_categorical(y_test, n_classes)

print("Shape after one-hot encoding: ", Y_train.shape)
```


Step 3 of 5: Building the Network



Our densely-connected network with two hidden layers.

1. Our pixel vector serves as the input. Then, two hidden 512-node layers, with enough model complexity for recognizing digits. For the multi-class classification we add another densely-connected (or fully-connected) layer for the 10 different output classes. For this network architecture we can use the Keras Sequential Model. We can stack layers using the .add() method.

2. When adding the first layer in the Sequential Model we need to specify the input shape so Keras can create the appropriate matrices. For all remaining layers the shape is inferred automatically.

3. In order to introduce nonlinearities into the network and elevate it beyond the capabilities of a simple perceptron we also add activation functions to the hidden layers. The differentiation for the training via backpropagation is happening behind the scenes without having to implement the details.

4. We also add dropout as a way to prevent overfitting. Here we randomly keep some network weights fixed when we would normally update them so that the network doesn't rely too much on very few nodes.

5. The last layer consists of connections for our 10 classes and the softmax activation which is standard for multi-class targets.

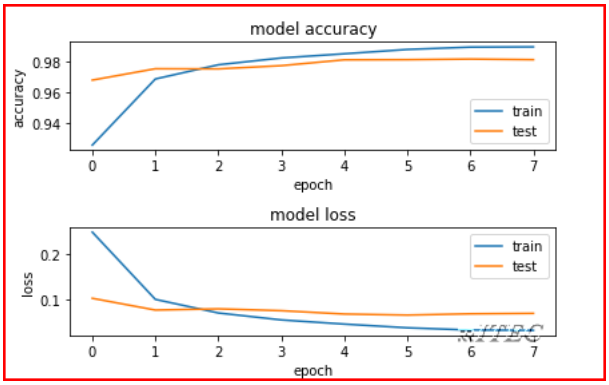
```
# building a linear stack of layers with the sequential model

model = Sequential()

model.add(Dense(512, input_shape=(784,)))

model.add(Activation('relu'))

model.add(Dropout(0.2))
```

```
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))


model.add(Dense(10))
model.add(Activation('softmax'))
```

Step 4 of 5: Compiling and Training the Model

```
# compiling the sequential model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
#We can start the training the model
# training the model and saving metrics in history
history = model.fit(X_train, Y_train,
        batch_size=128, epochs=8,
        verbose=2,
        validation_data=(X_test, Y_test))
# saving the model
save_dir = "C:/Users/Hi/Google Drive/01 Data Science Lab Copy/02 Lab Data/Python/Deep Learning/"
model_name = 'keras_mnist.h5'
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)
# plotting the metrics
fig = plt.figure()
plt.subplot(2,1,1)
```

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.subplot(2,1,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.tight_layout()
fig
```

Note: This learning curve looks quite good! We see that the loss on the training set is decreasing rapidly for the first two epochs. This shows the network is learning to classify the digits pretty fast. For the test set the loss does not decrease as fast but stays roughly within the same range as the training loss. This means our model generalizes well to unseen data.

Step 5 of 5: Evaluate the Model Performance

```
mnist_model = load_model("C:/Users/Hi/Google Drive/01 Data Science Lab Copy/02 Lab Data/Python/Deep Learning/keras_mnist.h5")

loss_and_metrics = mnist_model.evaluate(X_test, Y_test, verbose=2)


print("Test Loss", loss_and_metrics[0])

print("Test Accuracy", loss_and_metrics[1])


# load the model and create predictions on the test set

mnist_model = load_model("C:/Users/Hi/Google Drive/01 Data Science Lab Copy/02 Lab Data/Python/Deep Learning/keras_mnist.h5")

predicted_classes = mnist_model.predict_classes(X_test)


# see which we predicted correctly and which not

correct_indices = np.nonzero(predicted_classes == y_test)[0]

incorrect_indices = np.nonzero(predicted_classes != y_test)[0]

print()

print(len(correct_indices)," classified correctly")

print(len(incorrect_indices)," classified incorrectly")



# plot 9 correct predictions

for i, correct in enumerate(correct_indices[:9]):

    plt.subplot(6,3,i+1)

    plt.imshow(X_test[correct].reshape(28,28), cmap='gray', interpolation='none')

    plt.title("Predicted {}, Class {}".format(predicted_classes[correct], y_test[correct]))

    plt.xticks([])
```

```
    plt.yticks([])


# plot 9 incorrect predictions
for i, incorrect in enumerate(incorrect_indices[:9]):
    plt.subplot(6,3,i+10)
    plt.imshow(X_test[incorrect].reshape(28,28), cmap='gray', interpolation='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[incorrect], y_test[incorrect]))
    plt.xticks([])
    plt.yticks([])
```

35. Convolutional Neural Networks(CNN)

1. How self-drive car identify road signs?

2. Image is a data of pixels
3. Exercise 1: read image as a array and print array as a image

```
# Import matplotlib

import matplotlib.pyplot as plt

# Load the image

data = plt.imread('C:\\Users\\Hi\\Google Drive\\01 Data Science Lab Copy\\02 Lab Data\\Python\\bricks.png')

# Display the image
```

```
plt.imshow(data)

plt.show()
```

4. Exercise 2:Changing images data, hence image

```
# Set the red channel in this part of the image to 1

data[:40, :40, 0] = 1

# Set the green channel in this part of the image to 0

data[:40, :40, 1] = 0

# Set the blue channel in this part of the image to 0

data[:40, :40, 2] = 0

# Visualize the result

plt.imshow(data)

plt.show()
```


5. Exercise 3: One hot encoding understanding

```
labels= ['shoe', 'shirt', 'shoe', 'shirt', 'dress', 'dress', 'dress']

# The number of image categories

n_categories = 3

# The unique values of categories in the data

categories = np.array(["shirt", "dress", "shoe"])

# Initialize one_hot_encoding_labels  as all zeros

one_hot_encoding_labels = np.zeros((len(labels), n_categories))

# Loop over the labels

for ii in range(len(labels)):
```

```
    # Find the location of this label in the categories variable
    jj = np.where(categories == labels[ii])
    # Set the corresponding zero to one
    one_hot_encoding_labels[ii, jj] = 1
one_hot_encoding_labels
```

6.

Business Statement: MNIST ` Digit Recognition in Keras.

Level 07 of 08: NLU / NLP / Text Analytics/ Text Mining

36. Natural Language Understanding/Processing (NLU/P)

36.1 Introduction

1. NLP is a way for computers to analyse, understand, and derive meaning from human language in a smart and useful way.
2. By utilizing NLP, developers can organize and structure knowledge to perform tasks such as automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation.
3. **Summarize blocks of text** using Summarizer to extract the most important and central ideas while ignoring irrelevant information.
4. **Create a chat bot** using Parsey McParseface, a language parsing deep learning model made by Google that uses Point-of-Speech tagging.
5. **Automatically generate keyword tags** from content using AutoTag, which leverages LDA, a technique that discovers topics contained within a body of text.
6. **Identify the type of entity extracted**, such as it being a person, place, or organization using Named Entity Recognition.
7. **Use Sentiment Analysis** to identify the sentiment of a string of text, from very negative to neutral to very positive.
8. **Reduce words to their root,** or stem, using PorterStemmer, or break up text into tokens using Tokenizer

36.2 Regular Expressions

1. Regular Expressions are useful to identify Strings with a special syntax
2. Allow us to match patterns in other strings
3. Applications of regular expressions:
    a. Find all web links in a document
    b. Parse email addresses, remove/replace unwanted characters
4. Common regex patterns are

pattern	matches	example
\w+	word	'Magic'
\d	digit	9
\s	space	' '
.*	wildcard	'username74'
+ or *	greedy match	'aaaaaa'
\S	**not** space	'no_spaces'
[a-z]	lowercase group	'abcdefg'

5. In below exercise, the 'r' in front tells Python the expression is a raw string. In a raw string, escape sequences are not parsed. For example, '\n' is a single newline character. But, r'\n' would be two characters: a backslash and an 'n'.
6. **Exercise 1: Practicing regular expressions: re.split() and re.findall().**

```
import re

my_string = "Let's write RegEx!  Won't that be fun?  I sure think so.  Can you find 4 sentences?  Or perhaps, all 19 words?"

# Write a pattern to match sentence endings: sentence_endings

sentence_endings = r"[.?!]"

# Split my_string on sentence endings and print the result
```

```
print(re.split(sentence_endings, my_string))

# Find all capitalized words in my_string and print the result

capitalized_words = r"[A-Z]\w+"

print(re.findall(capitalized_words, my_string))

# Split my_string on spaces and print the result

spaces = r"\s+"

print(re.split(spaces, my_string))

# Find all digits in my_string and print the result

digits = r"\d+"

print(re.findall(digits, my_string))
```

7. Explore re.match vs re.search

```
import re

re.match("b","abcdef") # No Match

re.search("d","abcdef") # Match
```


8. For more refer help https://docs.python.org/3/library/re.html

36.3 Tokenization

1. Dividing a string or document into tokens (smaller chunks)
2. First step in preparing a text for NLP
3. Many different theories and rules available in creating tokens ,You can create your own rules using regular expressions
4. Some examples:
    a. Breaking out words or sentences
    b. Separating punctuation
    c. Separating all hashtags in a tweet
5. Tokenization useful
    a. Easier to map part of speech(POS)
    b. Matching common words
    c. Removing unwanted tokens
6. NLTK(natural language toolkit) library is useful for tokenization
7. Some of the main NLTK tokenization's are
    a. word_tokenize: to split into words
    b. sent_tokenize: tokenize a document into sentences
    c. regexp_tokenize: tokenize a string or document based on a regular expression pattern
    d. TweetTokenizer: special class just for tweet tokenization, allowing you to separate hashtags, mentions and lots of exclamation points!!!
    e. Let us see simple example of word_tokenize

```
In [67]: from nltk.tokenize import word_tokenize

In [68]: word_tokenize("Welcome to RRITEC!")
Out[68]: ['Welcome', 'to', 'RRITEC', '!']
```

8. **Exercise 1: Sentence and Word tokenization with NLTK**

```
# Import necessary modules
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
scene_one = "I like data science. I like RRITEC Training"
# Split scene_one into sentences: sentences
sentences = sent_tokenize(scene_one)
# Use word_tokenize to tokenize the second sentence: tokenized_sent
tokenized_sent = word_tokenize(sentences[1])
# Make a set of unique tokens in the entire scene: unique_tokens
unique_tokens = set(word_tokenize(scene_one))
# Print the unique tokens result
print(unique_tokens)
```


36.4 Advanced tokenization with regex

1. Regex groups using or "|"
2. OR is represented using |
3. You can define a group using ()
4. You can define explicit character ranges using []
5. Regex ranges [] and groups ()

pattern	matches	example
[A-Za-z]+	upper and lowercase English alphabet	'ABCDEFghijk'
[0-9]	numbers from 0 to 9	9
[A-Za-z\-\.]+	upper and lowercase English alphabet, - and .	'My-Website.com'
(a-z)	a, - and z	'a-z'
(\s+|,)	spaces or a comma	', '

6. Example 1 : Capturing digits and alphabets

```
# only digits
tokenize_digits_and_words= ('\d+')
re.findall(tokenize_digits_and_words,"he has 8 dogs and 11 cats")

# only alphabets
tokenize_digits_and_words= ('[a-z]+')
re.findall(tokenize_digits_and_words,"he has 8 dogs and 11 cats")

# Both digits and alphabets
tokenize_digits_and_words= ('[a-z]+|\d+')
re.findall(tokenize_digits_and_words,"he has 8 dogs and 11 cats")
```

7. Example 2:

```
my_str = 'match lowercase spaces nums like 12, but no commas'
re.match('[a-z0-9 ]+', my_str)
```


8. Example 3:

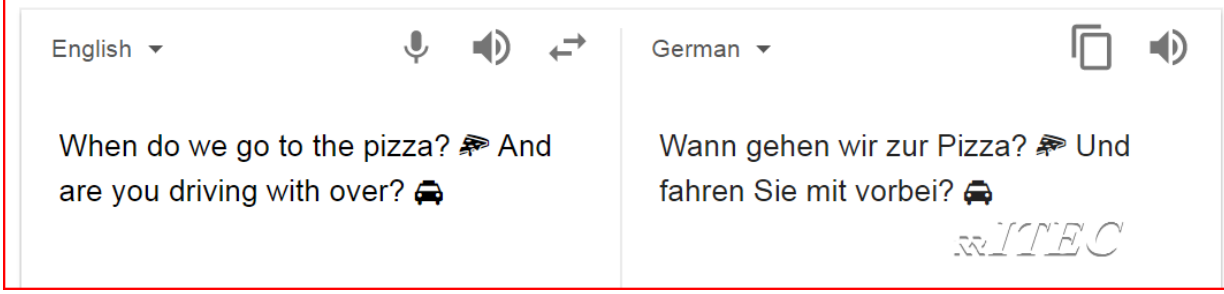
```
from nltk import regexp_tokenize
my_string = "SOLDIER #1: Found them? In Mercea? The coconut's tropical!"
pattern1 = r"(\w+|#\d|\?|!)"
regexp_tokenize(my_string,pattern=pattern1)
```

9. Example 4: Regex with NLTK tweet tokenization

```
# Import the necessary modules
from nltk.tokenize import regexp_tokenize
from nltk.tokenize import TweetTokenizer
```

```
tweets = ['This is the best #nlp exercise ive found online! #python',
          '#NLP is super fun! <3 #learning',
          'Thanks @RRITEC :) #nlp #python']
# Define a regex pattern to find hashtags: pattern1
pattern1 = r"#\w+"
# Use the pattern on the first tweet in the tweets list
regexp_tokenize(tweets[0], pattern1)
# Write a pattern that matches both @ and hashtags
pattern2 = r"([#|@]\w+)"
# Use the pattern on the last tweet in the tweets list
regexp_tokenize(tweets[-1], pattern2)
# Use the TweetTokenizer to tokenize all tweets into one list
tknzr = TweetTokenizer()
all_tokens = [tknzr.tokenize(t) for t in tweets]
print(all_tokens)
```


10. Exercise 5: Non-ascii tokenization



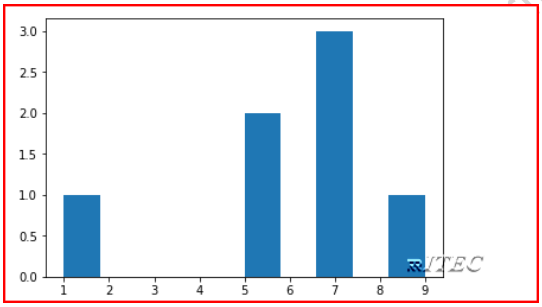

```
# Create a string
german_text="Wann gehen wir zur Pizza? 🍕 Und fahren Sie mit vorbei? 🚘 "
# Tokenize and print all words in german_text
all_words = word_tokenize(german_text)
print(all_words)
```

```
# Tokenize and print only capital words

capital_words = r"[A-ZÜ]\w+"

print(regexp_tokenize(german_text, capital_words))

# Tokenize and print only emoji

emoji = "['\U0001F300-\U0001F5FF'|'\U0001F600-\U0001F64F'|'\U0001F680-\U0001F6FF'|'\u2600-\u26FF\u2700-\u27BF']"

print(regexp_tokenize(german_text, emoji))
```

36.5 Charting word length with nltk

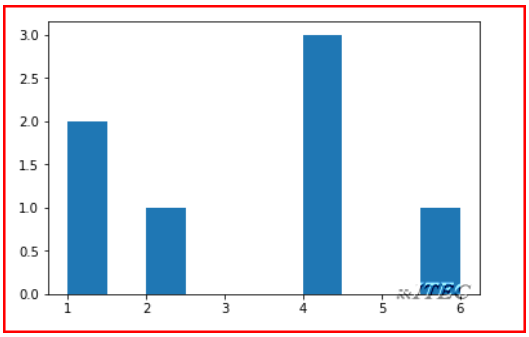
1. Exercise 1: Plotting a histogram with matplotlib

```
from matplotlib import pyplot as plt

plt.hist([1, 5, 5, 7, 7, 7, 9])
```



2. Exercise 2: Combining NLP data extraction with plotting

```
from nltk.tokenize import word_tokenize

words = word_tokenize("This is a pretty cool tool!")

word_lengths = [len(w) for w in words]

plt.hist(word_lengths)

plt.show()
```



36.6 Word counts with bag of words

1. Bag-of-words is a basic method for finding topics in a text
2. Need to first create tokens using tokenization, and then count up all the tokens
3. The more frequent a word, the more important it might be
4. It can be a great way to determine the significant words in a text
5. Exercise 1: Building a Counter with bag-of-words

```
from nltk.tokenize import word_tokenize
from collections import Counter
counter = Counter(word_tokenize(
"""The cat is in the box. The cat likes the box.
The box is over the cat."""))
counter
counter.most_common(5)
```

6. Exercise 2: Building a Counter with bag-of-words and identify the topic

```
article = open('nlp_topic_identification.txt', 'r').read()
# Import Counter
from collections import Counter
# Tokenize the article: tokens
tokens = word_tokenize(article)
# Convert the tokens into lowercase: lower_tokens
```

```
lower_tokens = [t.lower() for t in tokens]
# Create a Counter with the lowercase tokens: bow_simple
bow_simple = Counter(lower_tokens)
# Print the 10 most common tokens
print(bow_simple.most_common(15))
```

36.7 Text pre-processing

1. Any NLP program need below minimum text pre-process
    a. Tokenization to create a bag of words
    b. Lowercasing words
    c. Lemmatization/Stemming: Shorten words to their root stems
    d. Removing stop words, punctuation, or unwanted tokens
2. Exercise 1: Text pre-processing # removing stopwords


```
from nltk.corpus import stopwords # type stopwords
# observe stopwords C:\Users\Hi\AppData\Roaming\nltk_data\corpora\stopwords
text = """The cat is in the box. The cat likes the box.The box is over the cat."""
tokens = [w for w in word_tokenize(text.lower()) if w.isalpha()]
no_stops = [t for t in tokens if t not in stopwords.words('english')]
Counter(no_stops).most_common(2)
```

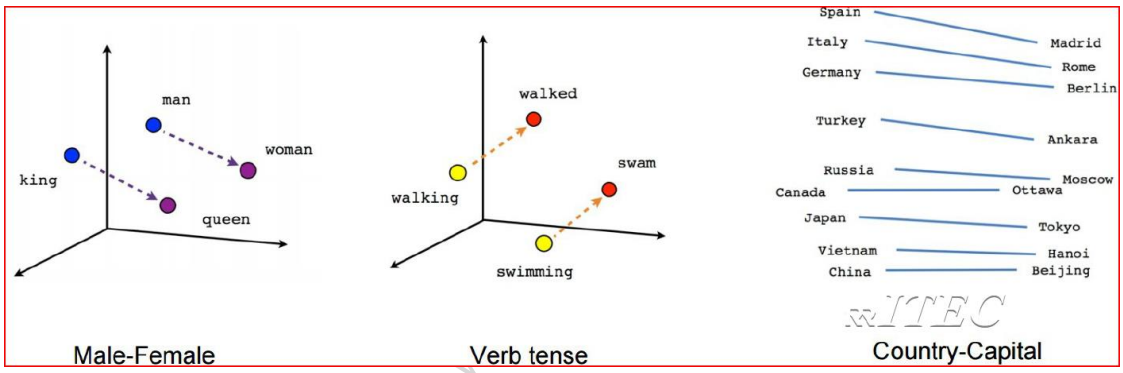
3. Exercise 2: Text pre-processing

```
# Import WordNetLemmatizer
from nltk.stem import WordNetLemmatizer
# Retain alphabetic words: alpha_only
alpha_only = [t for t in lower_tokens if t.isalpha()]
# Remove all stop words: no_stops
```

```
no_stops = [t for t in alpha_only if t not in stopwords.words('english')]

# Instantiate the WordNetLemmatizer

wordnet_lemmatizer = WordNetLemmatizer()

# Lemmatize all tokens into a new list: lemmatized

lemmatized = [wordnet_lemmatizer.lemmatize(t) for t in no_stops]

# Create the bag-of-words: bow

bow = Counter(lemmatized)

# Print the 10 most common tokens

print(bow.most_common(10))
```

36.8 Gensim

1. Gensim(topic modelling for humans): generate similar
2. Genism is useful for
    a. building document or word vectors
    b. Performing topic identification and document comparison
3. Word vector
    a. Word vectors are multi-dimensional mathematical representations of words created using deep learning methods. They give us insight into relationships between words in a corpus.
4. Example of word vector is



5. Time permits walk through https://radimrehurek.com/gensim/ http://tlfvincent.github.io/2015/10/23/presidential-speech-topics/
6. Exercise 1: Creating a gensim dictionary and corpus

```
from gensim.corpora.dictionary import Dictionary

from nltk.tokenize import word_tokenize

my_documents = ['The movie was about a spaceship and aliens.',

'I really liked the movie!',

'Awesome action scenes, but boring characters.',

'The movie was awful! I hate alien films.',

'Space is cool! I liked the movie.',
```

```
'More space films, please!',]

tokenized_docs = [word_tokenize(doc.lower()) for doc in my_documents]

dictionary = Dictionary(tokenized_docs)

dictionary.token2id

corpus = [dictionary.doc2bow(doc) for doc in tokenized_docs]

corpus
```

7. gensim models can be easily saved, updated, and reused
8. Our dictionary can also be updated
9. This more advanced and feature rich bag-of-words can be used in future exercises

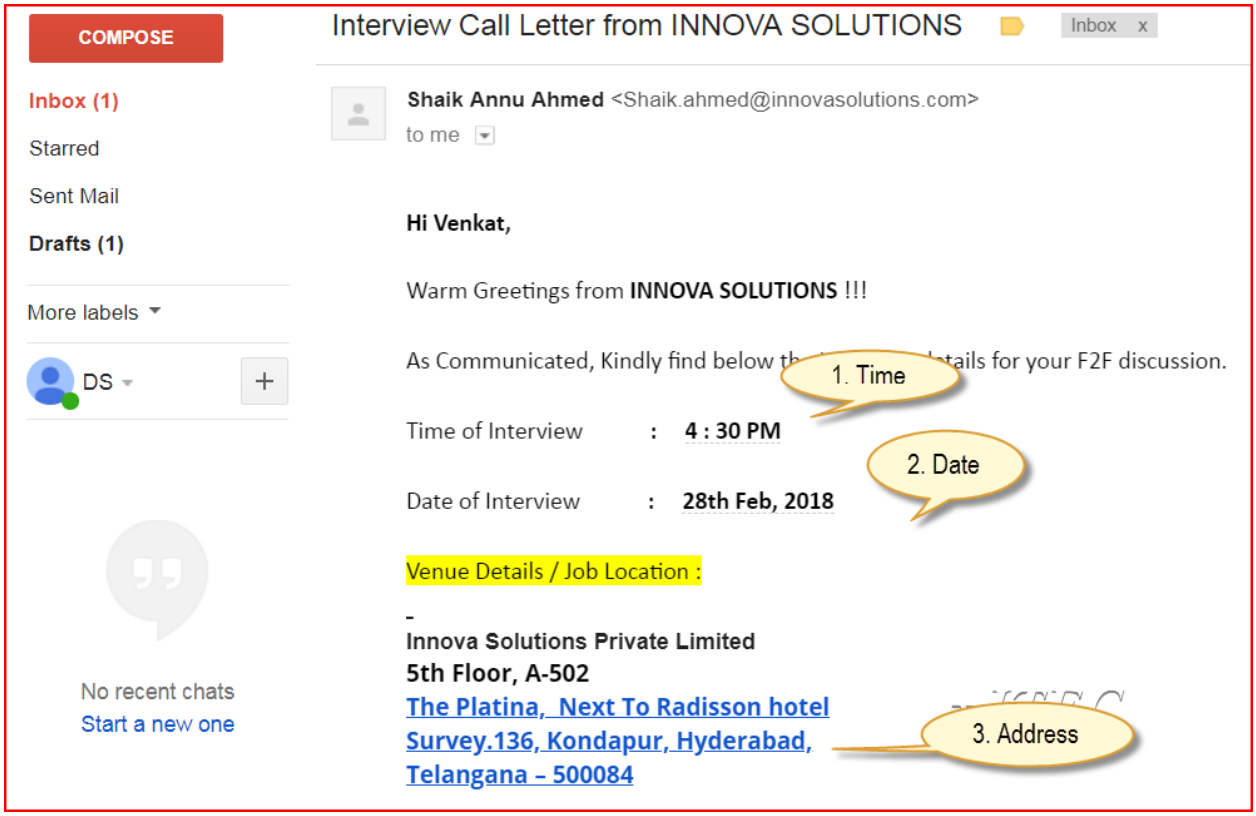
36.9 Tf-idf with gensim

1. Term frequency - inverse document frequency
2. Allows you to determine the most important words in each document
3. Each corpus may have shared words beyond just stopwords
4. These words should be down-weighted in importance
5. Example from astronomy: "Sky"
6. Ensures most common words don't show up as key words
7. Keeps document specific frequent words weighted high
8. Tf-idf formula $$w_{i,j} = tf_{i,j} * \log(\frac{N}{df_i})$$
    a. $W_{i,j}$ = tf-idf weight for token i in document j
    b. $Tf_{i,j}$ = number of occurrences of token i in document j
    c. df_i = number of documents that contain token i
    d. N = total number of documents
9. Exercise 1: Tf-idf with gensim

```
from gensim.models.tfidfmodel import TfidfModel

tfidf = TfidfModel(corpus)

tfidf[corpus[1]]
```

36.10 Named Entity Recognition

1. NLP task to identify important named entities in the text
    a. People, places, organizations
    b. Dates, states, works of art
    c. … etc
2. Mainly Who? What? When? Where?
3. Best example of NER is mail. Can you identify named entity?



4. Exercise 1: Using nltk for Named Entity Recognition

```
import nltk
sentence = '''In New York, I like to ride the Metro to visit MOMA
and some restaurants rated well by Ruth Reichl.'''
tokenized_sent = nltk.word_tokenize(sentence)
# Tag each tokenized sentence into parts of speech
```

```
tagged_sent = nltk.pos_tag(tokenized_sent)
tagged_sent[:3]
print(nltk.ne_chunk(tagged_sent))
```

5. Exercise 2: NER with NLTK using scraped news article

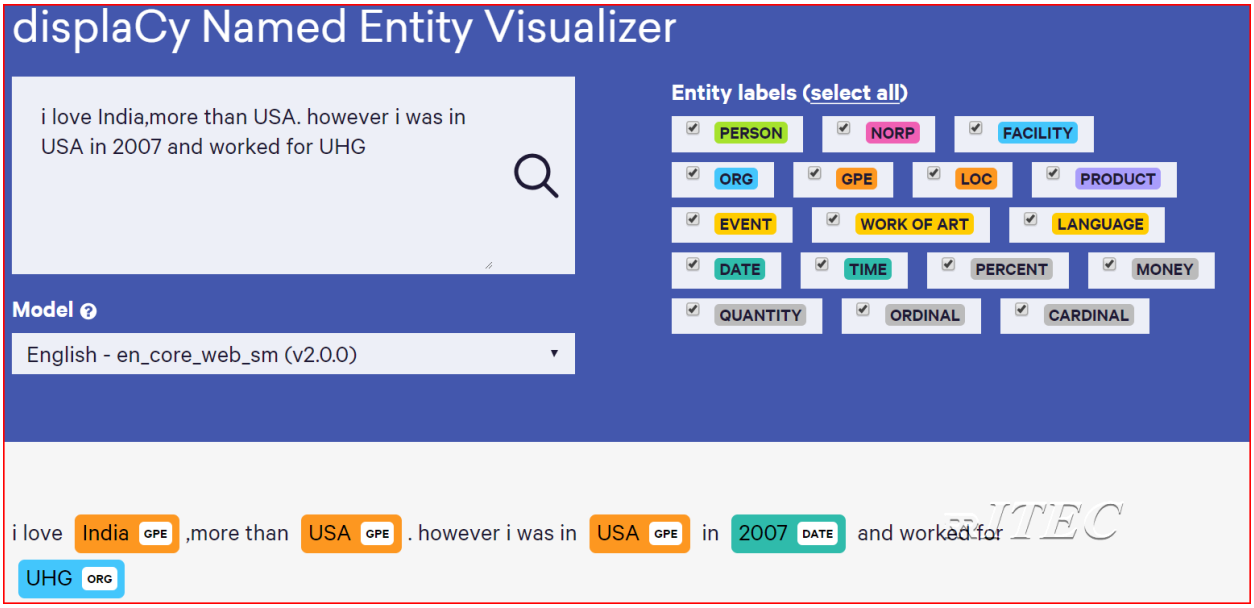
```
# Load a scraped news article
article = open('nlp_ner.txt', 'r').read()
type(article)
article
# Tokenize the article into sentences: sentences
sentences = nltk.sent_tokenize(article)
# Tokenize each sentence into words: token_sentences
token_sentences = [nltk.word_tokenize(sent) for sent in sentences]
# Tag each tokenized sentence into parts of speech: pos_sentences
pos_sentences = [nltk.pos_tag(sent) for sent in token_sentences]
# Create the named entity chunks: chunked_sentences
chunked_sentences = nltk.ne_chunk_sents(pos_sentences, binary=True)
# Test for stems of the tree with 'NE' tags
for sent in chunked_sentences:
    for chunk in sent:
        if hasattr(chunk, "label") and chunk.label() == "NE":
            print(chunk)
```

36.11 Introduction to SpaCy

1. SpaCy is a NLP library similar to gensim, with different implementations
2. Focus on creating NLP pipelines to generate models and corpora
3. Open-source, with extra libraries and tools . example: Displacy entity recognition visualizer
4. Easy pipeline creation
5. Different entity types compared to nltk
6. Easily find entities in Tweets and chat messages
7. Quickly growing package!
8. Verify

    a. https://demos.explosion.ai/displacy/

    b. https://demos.explosion.ai/displacy-ent/



9. Exercise 1: SpaCy for NER

```
import spacy

# First time users may need to execute in anaconda cmd prompt #python -m spacy download en

nlp = spacy.load('en')
```

```
# nlp.entity

doc = nlp("""Ram Reddy is a mentor,teaching Machine Learning and staying in India""")

doc.ents

print(doc.ents[0], doc.ents[0].label_)

print(doc.ents[1], doc.ents[1].label_)

print(doc.ents[2], doc.ents[2].label_)
```

10. Exercise 2: Comparing NLTK with spaCy NER

```
# Import spacy

import spacy

# Instantiate the English model: nlp

nlp = spacy.load('en', tagger=False, parser=False, matcher=False)

# Create a new document: doc

doc = nlp(article)

# Print all of the found entities and their labels

for ent in doc.ents:

    print(ent.label_, ent.text)
```


11. Which are the extra categories that spacy uses compared to nltk in its named-entity recognition?

    a.NORP, CARDINAL, MONEY, WORK OF ART, LANGUAGE, EVENT

36.12 Multilingual NER with polyglot

1. Polyglot is a NLP library which uses word vectors
2. Why polyglot? (gensim and spacy not enough?)
    a. It supports multilingual NER
    b. Vectors for many different languages
    c. More than 130!

3. Exercise 1: Spanish NER with polyglot

WIP

36.13 Building a "fake news" classifier

1. Which of the following are possible features for a text classification problem ?

   a. Number of words in a document.

   b. Specific named entities.

   c. Language.

   d. All of the above.

2. **Step 1 of 7: CountVectorizer for text classification**

   a. **CountVectorizer**: Convert a collection of text documents to a matrix of token counts

   b. Refer http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

```
# Load Fake news csv file, available in lab data folder

import pandas as pd

df = pd.read_csv("fake_or_real_news.csv")

df.info()

# Import the necessary modules

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import train_test_split

# Print the head of df

print(df.head())

# Create a series to store the labels: y

y = df.label

# Create training and test sets

X_train, X_test, y_train, y_test = train_test_split(df['text'], y, test_size=0.33,
random_state=53)

# Initialize a CountVectorizer object: count_vectorizer

count_vectorizer = CountVectorizer(stop_words='english')
```

```
count_vectorizer

# Transform the training data using only the 'text' column values: count_train

count_train = count_vectorizer.fit_transform(X_train)

# Transform the test data using only the 'text' column values: count_test

count_test = count_vectorizer.transform(X_test)

# Print the first 10 features of the count_vectorizer

print(count_vectorizer.get_feature_names()[:10])
```

3. **Step 2 of 7: TfidfVectorizer for text classification**

   a. What is the difference between **CountVectorizer** and **TfidfVectorizer** https://stackoverflow.com/questions/22489264/is-a-**countvectorizer**-the-same-as-**tfidfvectorizer**-with-use-idf-false

```
# Import TfidfVectorizer

from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize a TfidfVectorizer object: tfidf_vectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)

# Transform the training data: tfidf_train

tfidf_train = tfidf_vectorizer.fit_transform(X_train)

# Transform the test data: tfidf_test

tfidf_test = tfidf_vectorizer.transform(X_test)

# Print the first 10 features

print(tfidf_vectorizer.get_feature_names()[:10])

# Print the first 5 vectors of the tfidf training data

print(tfidf_train.A[:5])
```


4. **Step 3 of 7: Inspecting the vectors created using above two methods**

   a. To get a better idea of how the vectors work, you'll investigate them by converting them into pandas DataFrames

```
# Create the CountVectorizer DataFrame: count_df
count_df = pd.DataFrame(count_train.A, columns=count_vectorizer.get_feature_names())
# Create the TfidfVectorizer DataFrame: tfidf_df
tfidf_df = pd.DataFrame(tfidf_train.A, columns=tfidf_vectorizer.get_feature_names())
# Print the head of count_df
print(count_df.head())
# Print the head of tfidf_df
print(tfidf_df.head())
# Calculate the difference in columns: difference
difference = set(count_df.columns) - set(tfidf_df.columns)
print(difference)
```


Note: Which of the below is the most reasonable model to use when training a new supervised model using text vector data?

a. Random Forests

b. Naive Bayes

c. Linear Regression

d. Deep Learning

5. **Step 4 of 7: Training and testing the "fake news" model with CountVectorizer**

   a. Train and test a **Naive Bayes model** using the **CountVectorizer data.**

```
# Import the necessary modules
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
# Instantiate a Multinomial Naive Bayes classifier: nb_classifier
nb_classifier = MultinomialNB()
# Fit the classifier to the training data
```

```
nb_classifier.fit(count_train, y_train)

# Create the predicted tags: pred

pred = nb_classifier.predict(count_test)

# Calculate the accuracy score: score

score = metrics.accuracy_score(y_test, pred)

print(score)

# Calculate the confusion matrix: cm

cm = metrics.confusion_matrix(y_test, pred, labels=['FAKE', 'REAL'])

print(cm)
```

6. **Step 5 of 7: Training and testing the "fake news" model with TfidfVectorizer**

   a. In above step we evaluated the model using the **CountVectorizer**, you'll do the same using the **TfidfVectorizer** with a **Naive Bayes model**

```
# Create a Multinomial Naive Bayes classifier: nb_classifier

nb_classifier = MultinomialNB()

# Fit the classifier to the training data

nb_classifier.fit(tfidf_train, y_train)

# Create the predicted tags: pred

pred = nb_classifier.predict(tfidf_test)

# Calculate the accuracy score: score

score = metrics.accuracy_score(y_test, pred)

print(score)

# Calculate the confusion matrix: cm

cm = metrics.confusion_matrix(y_test, pred, labels=['FAKE', 'REAL'])

print(cm)
```


7. **Improving the model, what are the possible next steps you could take to improve the model?**

   a. Tweaking alpha levels.

b. Trying a new classification model.

c. Training on a larger dataset.

d. Improving text pre-processing.

e. All of the above

8. **Step 6 of 7: Improving your model**

   a. Test a few different alpha levels using the Tfidf vectors to determine if there is a better performing combination.

```
# Create the list of alphas: alphas
import numpy as np
alphas = np.arange(0, 1, .1)
# Define train_and_predict()
def train_and_predict(alpha):
    # Instantiate the classifier: nb_classifier
    nb_classifier = MultinomialNB(alpha=alpha)
    # Fit to the training data
    nb_classifier.fit(tfidf_train, y_train)
    # Predict the labels: pred
    pred = nb_classifier.predict(tfidf_test)
    # Compute accuracy: score
    score = metrics.accuracy_score(y_test, pred)
    return score
# Iterate over the alphas and print the corresponding score
for alpha in alphas:
    print('Alpha: ', alpha)
    print('Score: ', train_and_predict(alpha))
    print()
```

9. Step 7 of 7: Inspecting your model

a. You can map the important vector weights back to actual words using some simple inspection techniques.

```
# Get the class labels: class_labels

class_labels = nb_classifier.classes_

# Extract the features: feature_names

feature_names = tfidf_vectorizer.get_feature_names()

# Zip the feature names together with the coefficient array and sort by weights: feat_with_weights

feat_with_weights = sorted(zip(nb_classifier.coef_[0], feature_names))

# Print the first class label and the top 20 feat_with_weights entries

print(class_labels[0], feat_with_weights[:20])

# Print the second class label and the bottom 20 feat_with_weights entries

print(class_labels[1], feat_with_weights[-20:])
```

36.14 Dialog Flow

1.

2. https://dialogflow.com/

3.

36.15 RASA NLU

Dialog Flow

Level 08 of 08: Projects on NLU/NLP

37. Introduction

1. Conversational software is not a new idea https://en.wikipedia.org/wiki/ELIZA

38. EchoBot

```
# Define a function that responds to a user's message: respond
def respond(message):
    # Concatenate the user's message to the end of a standard bot respone
    bot_message = "I can hear you! You said: " + message
    # Return the result
    return bot_message
# or
def respond1(message):
    return "I can hear you! you said: {}".format(message)
# Call the above functions
respond("hello")
respond1("hello")
```


2. **Exercise 2: Call EchoBot Function**

```
bot_template = "BOT : {0}"
user_template = "USER : {0}"
# Define a function that sends a message to the bot: send_message
def send_message(message):
```

```
    # Print user_template including the user_message
    print(user_template.format(message))
    # Get the bot's response to the message
    response = respond(message)
    # Wait 2 secs and Print the bot template including the bot's response.
    import time
    time.sleep(2)
    print(bot_template.format(response))
# Send a message to the bot
send_message("hello")
```

39. ChitChat Bot

1. **Exercise 3:Create a bot which can answer simple questions such as "What's your name?" and "What's today's weather?"**

    a. You'll use a dictionary with these questions as keys, and the correct responses as values.

    b. This means the bot will only respond correctly if the message matches exactly, which is a big limitation. In later exercises you will create much more robust solutions.

```
# Define variables
name = "Greg"
weather = "cloudy"
# Define a dictionary with the predefined responses
responses = {
  "what's your name?": "my name is {0}".format(name),
  "what's today's weather?": "the weather is {0}".format(weather),
  "default": "default message"
}
```

```
# Return the matching response if there is one, default otherwise
def respond(message):
    # Check if the message is in the responses
    if message in responses:
        # Return the matching message
        bot_message = responses[message]
    else:
        # Return the "default" message
        bot_message = responses["default"]
    return bot_message
# Call the Functions
respond("what's your name?")
respond("what's today's weather?")
respond("what's today's weather?_wrong")
```


2. **Exercise 4: Adding Variety of answers**

```
# Import the random module
import random
name = "Greg"
weather = "cloudy"
# Define a dictionary containing a list of responses for each message
responses = {
  "what's your name?": [
     "my name is {0}".format(name),
```

```
    "they call me {0}".format(name),
    "I go by {0}".format(name)
  ],
 "what's today's weather?": [
    "the weather is {0}".format(weather),
    "it's {0} today".format(weather)
  ],
 "default": ["default message"]
}
# Use random.choice() to choose a matching response
def respond(message):
  # Check if the message is in the responses
  if message in responses:
    # Return a random matching response
    bot_message = random.choice(responses[message])
  else:
    # Return a random "default" response
    bot_message = random.choice(responses["default"])
  return bot_message
# Call the Function
respond("what's your name?") # run few times and observe outputs
```


40. Text Munging with regular expressions

1. **Exercise 1: Pattern Matching**

```
import re

pattern = "do you remember "

pattern

message = "do you remember when you ate strawberries in the garden"

match = re.search(pattern,message)

if match:

   print("String Matches")
```

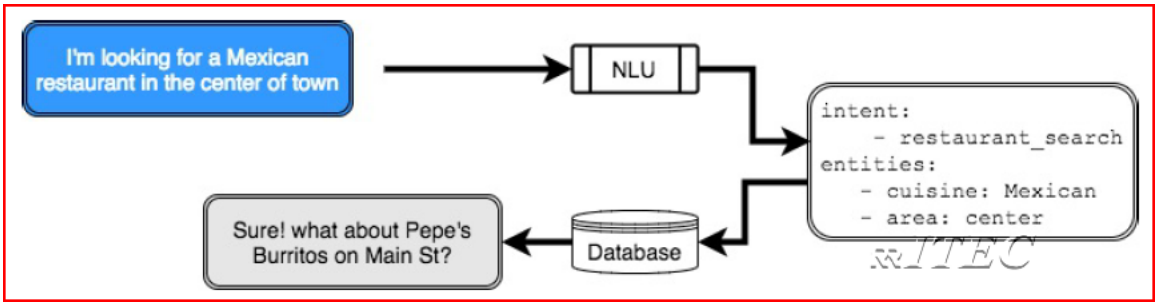
2. Extracting Key Phrases

```
import re

pattern = "if (.*)"

message = "what would happen if bots took over the world"

match = re.search(pattern, message)

match.group(0)

match.group(1)
```


3. Grammatical Transformation

```
import re

def swap_pronouns(phrase):

   if 'I' in phrase:

      phrase = re.sub('I', 'you', phrase)

      if 'my' in phrase:

         return re.sub('my', 'your', phrase)

      else:

         return phrase

swap_pronouns("I walk my dog")
```

41. Understanding intents and entities



1. **Intents**
    a. A restaurant_search can be expressed many different ways:
        i. I'm hungry
        ii. Show me good pizza spots
        iii. I want to take my boyfriend out for dinner
            1. Can also be request_booking
2. **Entities**
    a. Book a table for **Aug 24th** at a **Westin** restaurant in **Hyderabad** city
        i. NER = Named Entity Recognition
3. Exercise 1: Regular expressions to recognize intents
    a. Regular expressions advantages
        i. Simpler than machine learning approaches
        ii. Highly computationally efficient
    b. Drawback:
        i. Debugging regular expressions can become difficult
    c. '|' is equivalent to OR
    d. \b matches the beginning or end of a word

```
re.search(r"(hello|hey|hi)", "hey there!")

re.search(r"(hello|hey|hi)", "which one?")

re.search(r"\b(hello|hey|hi)\b", "hey there!")

re.search(r"\b(hello|hey|hi)\b", "which one?")
```

e. Exercise 2: Using regex for entity recognition

```
pattern = re.compile('[A-Z]{1}[a-z]*')

message = """

Mary is a friend of mine,

she studied At Oxford and

now works at Google"""

pattern.findall(message)
```


f. Exercise 3: Intent classification with regex

```
bot_template = "BOT : {0}"

user_template = "USER : {0}"

keywords ={'goodbye': ['bye', 'farewell'],

          'greet': ['hello', 'hi', 'hey'],

          'thankyou': ['thank', 'thx']}

responses = {'goodbye': 'goodbye for now',

            'greet': 'Hello you! :)',

            'default': 'default message',

            'thankyou': 'you are very welcome'}

def send_message(message):

    print(user_template.format(message))

    response = respond(message)
```

```
    print(bot_template.format(response))
# Define a dictionary of patterns
patterns = {}
# Iterate over the keywords dictionary
for intent, keys in keywords.items():
    # Create regular expressions and compile them into pattern objects
    patterns[intent] = re.compile('|'.join(keys))
# Print the patterns
print(patterns)
# Define a function to find the intent of a message
def match_intent(message):
    matched_intent = None
    for intent, pattern in patterns.items():
        # Check if the pattern occurs in the message
        if pattern.search(message):
            matched_intent = intent
    return matched_intent
# Define a respond function
def respond(message):
    # Call the match_intent function
    intent = match_intent(message)
    # Fall back to the default response
    key = "default"
    if intent in responses:
        key = intent
```

```
    return responses[key]

# Send messages

send_message("hello!")

send_message("bye byeee")

send_message("thanks very much!")
```

g. Exercise 4: Entity extraction with regex

```
# Define find_name()

def find_name(message):

    name = None

    # Create a pattern for checking if the keywords occur

    name_keyword = re.compile('name|call')

    # Create a pattern for finding capitalized words

    name_pattern = re.compile('[A-Z]{1}[a-z]*')

    if name_keyword.search(message):

        # Get the matching words in the string

        name_words = name_pattern.findall(message)

        if len(name_words) > 0:

            # Return the name if the keywords are present

            name = ' '.join(name_words)

    return name


# Define respond()

def respond(message):

    # Find the name
```

```
    name = find_name(message)
    if name is None:
        return "Hi there!"
    else:
        return "Hello, {0}!".format(name)


# Send messages
send_message("my name is David Copperfield")
send_message("call me Ishmael")
send_message("People call me Cassandra")
```

42. Word vectors

1. Word vectors are part of machine learning
2. Programs which can get better at a task by being exposed to more data
3. Identifying which intent a user message belongs to
4. Word vector representation
    a. Example : “can you help me please”

Units	examples	vectors
characters	"c", "a", "n", ...	v_c, v_a, v_n, ...
words	"can", "you", ...	v_{can}, v_{you}, ...
sentences	"can you help..."	v_{can you help ...}

5. Word vectors
    a. Word vectors try to represent meaning of words
    b. Words which appear in similar context have similar vectors

Context	Candidates
let's meet at the ___ tomorrow	office, gym, park, beach, party
I love going to the ___ to play with the dogs	beach, park

    c. Word vectors are computationally intensive
    d. Training word vectors requires a lot of data
    e. High quality word vectors are available for anyone to use
    f. GloVe algorithm
        i. Cousin of word2vec
    g. spaCy
6. Exercise 1: Word vectors in spaCy

```
import spacy
```

```
nlp = spacy.load('en') # if any FileNotFoundError , run from anconda prompt "python -m spacy download en"

nlp.vocab.vectors_length

doc = nlp('hello can you help me?')

doc

for token in doc:

   print("{} : {}".format(token, token.vector[:3]))
```

7. Similarity
    a. Direction of vectors matters
    b. "Distance" between words = angle between the vectors
    c. Cosine similarity
        i. 1: If vectors point in the same direction
        ii. 0: If they are perpendicular
        iii. -1: If they point in opposite directions
    d. Exercise 1: Similarity


```
import spacy

nlp = spacy.load('en')

doc = nlp("cat")

doc.similarity(nlp("can"))

doc.similarity(nlp("dog"))

doc.similarity(nlp("cat"))
```

8. Exercise 3: word vectors with spaCy using ATIS dataset
    a. Create a 2D array X with as many rows as there are sentences in the dataset, where each row is a vector describing that sentence.

```
with open('nlp_atis_words.txt', "r") as word_list:
    sentences = word_list.read().split(',')
with open('nlp_atis_labels.txt', "r") as word_list:
    labels = word_list.read().split(',')


# Load the spacy model: nlp
nlp = spacy.load('en')


# Calculate the length of sentences
n_sentences = len(sentences)


# Calculate the dimensionality of nlp
embedding_dim = nlp.vocab.vectors_length


# Initialize the array with zeros: X
import numpy as np
X = np.zeros((n_sentences, embedding_dim))


# Iterate over the sentences
for idx, sentence in enumerate(sentences):
    # Pass each each sentence to the nlp object to create a document
    doc = nlp(sentence)
    # Save the document's .vector attribute to the corresponding row in X
    X[idx, :] = doc.vector
```

43. Intents and classification

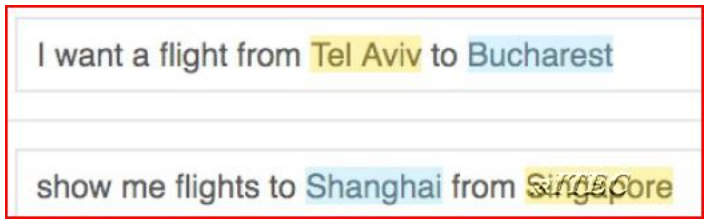
1. Supervised Learning
    a. A classifier predicts the intent label for a given sentence
    b. 'Fit' classifier by tuning it on training data
    c. Evaluate performance on test data
    d. Accuracy: the fraction of labels we predict correctly
2. Need to write algorithms using sklearn svm …etc . we will do later

44. Entity Extraction

1. play Jailhouse Rock by Elvis
2. Keywords don't work for entities you haven't seen before
3. Context:
    a. Use contextual clues:
        i. Spelling
        ii. Capitalization
        iii. Words occurring before & after
4. Exercise 1: pre-built NER(Named Entity Recognition)

```
import spacy

nlp = spacy.load('en')

doc = nlp("my friend Mary has worked at Google since 2009")

for ent in doc.ents:

   print(ent.text, ent.label_)
```


5. Pattern Recognition




```
import re

pattern_1 = re.compile('.* from (.*) to (.*)')

pattern_2 = re.compile('.* to (.*) from (.*)')
```

6. Exercise 2: Using spaCy's entity recognizer

```
nlp = spacy.load('en')

# Define included entities

include_entities = ['DATE', 'ORG', 'PERSON']



# Define extract_entities()

def extract_entities(message):

    # Create a dict to hold the entities

    ents = dict.fromkeys(include_entities)

    # Create a spacy document

    doc = nlp(message)

    for ent in doc.ents:

        if ent.label_ in include_entities:

            # Save interesting entities

            ents[ent.label_] = ent.text

    return ents



print(extract_entities('friends called Mary who have worked at Google since 2010'))

print(extract_entities('people who graduated from MIT in 1999'))
```


7. Exercise 3: Assigning roles using spaCy's parser

```
colors = ['black', 'red', 'blue']

items = ['shoes', 'handback', 'jacket', 'jeans']

def entity_type(word):

    _type = None

    if word.text in colors:

        _type = "color"
```

```
    elif word.text in items:
        _type = "item"
    return _type
# Create the document
doc = nlp("let's see that jacket in red and some blue jeans")
# Iterate over parents in parse tree until an item entity is found
def find_parent_item(word):
    # Iterate over the word's ancestors
    for parent in word.ancestors:
        # Check for an "item" entity
        if entity_type(parent) == "item":
            return parent.text
    return None
# For all color entities, find their parent item
def assign_colors(doc):
    # Iterate over the document
    for word in doc:
        # Check for "color" entities
        if entity_type(word) == "color":
            # Find the parent
            item =  find_parent_item(word)
            print("item: {0} has color : {1}".format(item, word))
# Assign the colors
assign_colors(doc)
```

45. Robust NLU with Rasa

1. Rasa NLU Installations
    a. Open anaconda prompt
    b. conda pip
    c. activate base
    d. pip install rasa_nlu
2. RASA NLU
    a. Library for intent recognition & entity extraction
    b. Based on spaCy, scikit-learn, & other libraries
    c. Built in support for chatbot specific tasks
3. Explore below sites
    a. Google → https://dialogflow.com/
    b. Microsoft → https://www.luis.ai/
    c. Facebook → https://wit.ai/
        i. If you use any of above three ,then also explore https://nlu.rasa.com/migrating.html#section-migration
    d. Rasa nlu trainer → https://rasahq.github.io/rasa-nlu-trainer/

    Note: If you need parse json use http://json.parser.online.fr/
4. Exercise 1:Rasa NLU
    a. Please note that iam using rasa nlu version 0.13.1.If you are using any other versions, some of the functions may not work.
    b. In this exercise you'll use Rasa NLU to create an interpreter, which parses incoming user messages and returns a set of entities.
    c. Your job is to train an interpreter using the MITIE entity recognition model in rasa NLU

```
# Import necessary modules

from rasa_nlu.training_data import load_data
```

```
from rasa_nlu.config import RasaNLUModelConfig

from rasa_nlu.model import Trainer

# Create args dictionary

args = {"pipeline": "spacy_sklearn"}

# Create a configuration and trainer

config = RasaNLUModelConfig(configuration_values=args)

trainer = Trainer(config)

# Load the training data

training_data = load_data("./training_data.json")

# Create an interpreter by training the model

interpreter = trainer.train(training_data)

# Try it out

print(interpreter.parse("I'm looking for a Mexican restaurant in the North of town"))
```

46. Building a virtual assistant

1. In this chapter, you're going to build a personal assistant to help you plan a trip. It will be able to respond to questions like **"are there any cheap hotels in the north of town?"** by looking inside a hotels database for matching results

46.1 Access data from sqlite with parameters

1. Common chatbot use cases:
    a. Scheduling a meeting
    b. Booking a flight
    c. Searching for a restaurant
2. Require information about the outside world
3. Need to interact with databases or APIs
4. Exercise 1: SQL statements in Python


```
# Step 1 of 7: Import required packages

from sqlalchemy import create_engine ; import pandas as pd ;

# Step 2 of 7: Create engine: engine

engine = create_engine('sqlite:///hotels.db')

# Step 3 of 7: Open engine connection

con = engine.connect()

# Step 4 of 7: Perform query: result

result = con.execute("SELECT * FROM hotels")

# Step 5 of 7: Save results of the query to DataFrame: df

df = pd.DataFrame(result.fetchall(),columns = result.keys())

# Step 6 of 7: Close connection

con.close()
```

```
# Step 7 of 7: Print head of DataFrame df

print(df.head())

# or

# Step 1 of 5: Import module sqlite3

import sqlite3

# Step 2 of 5: Open connection to DB

conn = sqlite3.connect('hotels.db')

# Step 3 of 5: Create a cursor

con = conn.cursor()

# Step 4 of 5: Execute the query

con.execute("SELECT * FROM hotels WHERE area='south' and price='hi'")

# Step 5 of 5: Print the results

con.fetchall()
```


5. Exercise 2: SQL statements in Python parameter passing

```
# Step 1 of 6: Import module sqlite3

import sqlite3

# Step 2 of 6: Open connection to DB

conn = sqlite3.connect('hotels.db')

# Step 3 of 6: Create a cursor

c = conn.cursor()

# Step 4 of 6: Define area and price parameters

par_area, par_price = "south", "hi"

t = (par_area, par_price)

# Step 5 of 6: Execute the query

c.execute('SELECT * FROM hotels WHERE area=? AND price=?', t)
```

```
# Step 6 of 6: Print the results

print(c.fetchall())
```

46.2 Exploring a DB with natural language

1. Example messages
    a. "Show me a great hotel"
    b. "I'm looking for a cheap hotel in the south of town"
    c. "Anywhere so long as it's central"
2. Exercise 1: Creating queries from parameters
    a. Now you're going to implement a more powerful function for querying the hotels database. The goal is to take arguments that can later be specified by other parts of your code.
    b. Specifically, your job here is to define a find_hotels() function which takes a single argument - a dictionary of column names and values - and returns a list of matching hotels from the database.


```
# Define find_hotels()

def find_hotels(params):

    # Create the base query

    query = 'SELECT * FROM hotels'

    # Add filter clauses for each of the parameters

    if len(params) > 0:

        filters = ["{}=?".format(k) for k in params]

        print(filters)

        query += " WHERE " + " and ".join(filters)

        print(query)

    # Create the tuple of values

    t = tuple(params.values())

    print(t)

        # Open connection to DB
```

```
conn = sqlite3.connect('hotels.db')

# Create a cursor

c = conn.cursor()

# Execute the query

c.execute(query, t)

# Return the results

return c.fetchall()
```

3. Exercise 2: Using your custom function to find hotels

    a. Here, you're going to put your find_hotels() function into action! Recall that it accepts a single argument, params, which is a dictionary of column names and values.

```
# Create the dictionary of column names and values

params = {"area": "south", "price":"lo"}

# Find the hotels that match the parameters

print(find_hotels(params))
```


4. Exercise 3: Creating SQL from natural language

    a. Now you'll write a respond() function which can handle messages like "I want an expensive hotel in the south of town" and respond appropriately according to the number of matching results in a database. This is important functionality for any database-backed chatbot.

    b. Your find_hotels() function from the previous exercises has already been defined for you, along with a rasa NLU interpreter object which can handle hotel queries and a list of responses, which you can explore in the Shell.

```
responses = ["I'm sorry :( I couldn't find anything like that",

            '{} is a great hotel!',

'{} or {} would work!',
```

```
            '{} is one option, but I know others too :)']

# Define respond()

def respond(message):

    # Extract the entities

    entities = interpreter.parse(message)["entities"]

    # Initialize an empty params dictionary

    params = {}

    # Fill the dictionary with entities

    for ent in entities:

        params[ent["entity"]] = str(ent["value"])


    # Find hotels that match the dictionary

    results = find_hotels(params)

    # Get the names of the hotels and index of the response

    names = [r[0] for r in results]

    n = min(len(results),3)

    # Select the nth element of the responses array

    return responses[n].format(*names)


# Observe database data

df

# Query Database Data based on area

print(respond("I want an expensive hotel in the west of town"))
```

46.3 Incremental slot filling and negation

1. Incremental filters

2. Exercise 1: Refining your search

   i. Now you'll write a bot that allows users to add filters incrementally, in case they don't specify all of their preferences in one message.

   ii. To do this, initialize an empty dictionary params outside of your respond() function (unlike inside the function, like in the previous exercise). Your respond() function will take in this dictionary as an argument.

```
# Define a respond function, taking the message and existing params as input

def respond(message, params):

    # Extract the entities

    entities = interpreter.parse(message)["entities"]

    # Fill the dictionary with entities

    for ent in entities:

        params[ent["entity"]] = str(ent["value"])

    # Find the hotels
```

```
    results = find_hotels(params)
    names = [r[0] for r in results]
    n = min(len(results), 3)
    # Return the appropriate response
    return responses[n].format(*names), params
# Initialize params dictionary
params = {}
# Pass the messages to the bot
for message in ["in the north of town","I want an expensive hotel"]:
    print("USER: {}".format(message))
    response, params = respond(message, params)
    print("BOT: {}".format(response))
```

1. Negated Entities

   a. assume that "not" or "n't" just before an entity means user wants to exclude this

   b. normal entities in green, negated entities in purple

2. Exercise 1: Catching negations

```
import spacy
nlp = spacy.load('en')
doc = nlp('not sushi, maybe pizza?')
```

```
indices = [1, 4]

ents, negated_ents = [], []

start = 0

for i in indices:

   phrase = "{}".format(doc[start:i])

   print(phrase)

   if "not" in phrase or "n't" in phrase:

      negated_ents.append(doc[i])

   else:

      ents.append(doc[i])

   start = i

negated_ents
```


3. Exercise 2: Basic negation

   a. Quite often you'll find your users telling you what they don't want - and that's important to understand! In general, negation is a difficult problem in NLP. Here we'll take a very simple approach that works for many cases.

   b. A list of tests called tests has been defined for you. Explore it in the Shell - you'll find that each test is a tuple consisting of:

      i. A string containing a message with entities

      ii. A dictionary containing the entities as keys, and a Boolean saying whether they are negated as the key

   c. Your job is to define a function called negated_ents() which looks for negated entities in a message.

```
tests = [("no I don't want to be in the south", {'south': False}),

      ('no it should be in the south', {'south': True}),

      ('no in the south not the north', {'north': False, 'south': True}),
```

```
        ('not north', {'north': False})]

# Define negated_ents()

def negated_ents(phrase, ent_vals):

    ents = [e for e in ent_vals if e in phrase]

    ends = sorted([phrase.index(e)+len(e) for e in ents])

    start = 0

    chunks = []

    for end in ends:

        chunks.append(phrase[start:end])

        start = end

    result = {}

    for chunk in chunks:

        for ent in ents:

            if ent in chunk:

                if "not" in chunk or "n't" in chunk:

                    result[ent] = False

                else:

                    result[ent] = True

    return result
```


4. Filtering with excluded slots

    a. Now you're going to put together some of the ideas from previous exercises, and allow users to tell your bot about what they do and what they don't want, split across multiple messages.

    b. The negated_ents() function has already been defined for you. Additionally, a slightly tweaked version of the find_hotels() function, which accepts a neg_params dictionary in addition to a params dictionary, has been defined.

```
def find_hotels(params, neg_params):
    query = 'SELECT * FROM hotels'
    if len(params) > 0:
        filters = ["{}=?".format(k) for k in params] + ["{}!=?".format(k) for k in neg_params]
        query += " WHERE " + " and ".join(filters)
    t = tuple(params.values())

    # open connection to DB
    conn = sqlite3.connect('hotels.db')
    # create a cursor
    c = conn.cursor()
    c.execute(query, t)
    return c.fetchall()

def negated_ents(phrase, ent_vals):
    ents = [e for e in ent_vals if e in phrase]
    ends = sorted([phrase.index(e)+len(e) for e in ents])
    start = 0
    chunks = []
    for end in ends:
        chunks.append(phrase[start:end])
        start = end
    result = {}
    for chunk in chunks:
        for ent in ents:
```

```
            if ent in chunk:
                if "not" in chunk or "n't" in chunk:
                    result[ent] = False
                else:
                    result[ent] = True
    return result
# Define the respond function
def respond(message, params, neg_params):
    # Extract the entities
    entities = interpreter.parse(message)["entities"]
    ent_vals = [e["value"] for e in entities]
    # Look for negated entities
    negated = negated_ents(message, ent_vals)
    for ent in entities:
        if ent["value"] in negated and negated[ent["value"]]:
            neg_params[ent["entity"]] = str(ent["value"])
        else:
            params[ent["entity"]] = str(ent["value"])
    # Find the hotels
    results = find_hotels(params, neg_params)
    names = [r[0] for r in results]
    n = min(len(results),3)
    # Return the correct response
    return responses[n].format(*names), params, neg_params
```

```
# Initialize params and neg_params

params = {}

neg_params = {}



# Pass the messages to the bot

for message in ["but not in the north of town","I want a cheap hotel"]:

    print("USER: {}".format(message))

    response, params, neg_params = respond(message, params, neg_params)

    print("BOT: {}".format(response))
```

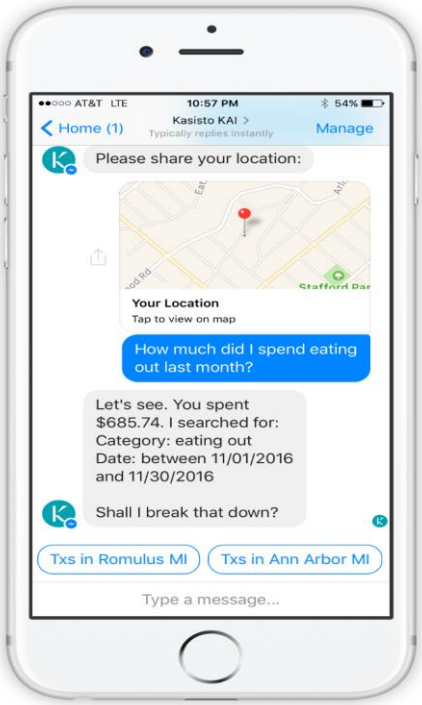
c.

d.

47. Dialogue

1. Everything you've built so far has statelessly mapped intents to actions & responses. It's amazing how far you can get with that! But to build more sophisticated bots you will always want to add some statefulness. That's what you'll do here, as you build a chatbot that helps users order coffee. Have fun!

47.1 Stateful bots



1. Form filling

    a. You'll often want your bot to guide users through a series of steps, such as when they're placing an order.

    b. In this exercise, you'll begin building a bot that lets users order coffee. They can choose between two types: Colombian, and Kenyan. If the user provides unexpected input, your bot will handle this differently depending on where they are in the flow.

    c. Your job here is to identify the appropriate state and next state based on the intents and response messages provided. For example, if the intent is "order", then the state changes from INIT to CHOOSE_COFFEE.

    d. A function send_message(policy, state, message) takes the policy, the current state and message as arguments, and returns the new state as a result.

```
def send_message(policy, state, message):
    print("USER : {}".format(message))
    new_state, response = respond(policy, state, message)
    print("BOT : {}".format(response))
    return new_state
def respond(policy, state, message):
    (new_state, response) = policy[(state, interpret(message))]
    return new_state, response
def interpret(message):
    msg = message.lower()
    if 'order' in msg:
        return 'order'
    if 'kenyan' in msg or 'columbian' in msg:
        return 'specify_coffee'
    return 'none'


# Define the INIT state
INIT = 0


# Define the CHOOSE_COFFEE state
CHOOSE_COFFEE = 1


# Define the ORDERED state
ORDERED = 2
```

```
# Define the policy rules

policy = {

   (INIT, "order"): (CHOOSE_COFFEE, "ok, Columbian or Kenyan?"),

   (INIT, "none"): (INIT, "I'm sorry - I'm not sure how to help you"),

   (CHOOSE_COFFEE, "specify_coffee"): (ORDERED, "perfect, the beans are on their way!"),

   (CHOOSE_COFFEE, "none"): (CHOOSE_COFFEE, "I'm sorry - would you like Colombian or
Kenyan?"),

}

# Create the list of messages

messages = [

   "I'd like to become a professional dancer",

   "well then I'd like to order some coffee",

   "my favourite animal is a zebra",

   "kenyan"

]


# Call send_message() for each message

state = INIT

for message in messages:

   state = send_message(policy, state, message
```


2. Exercise 2: Asking contextual questions

   a. Sometimes your users need some help! They will have questions and expect the bot to help them.

   b. In this exercise, you'll allow users to ask the coffee bot to explain the steps to them. Like before, the answer they get will depend on where they are in the flow.

```
# Define send_messages()
```

```
def send_messages(messages):
    state = INIT
    for msg in messages:
        state = send_message(state, msg)
def send_message(state, message):
    print("USER : {}".format(message))
    new_state, response = respond(state, message)
    print("BOT : {}".format(response))
    return new_state
def respond(state, message):
    (new_state, response) = policy_rules[(state, interpret(message))]
    return new_state, response
def interpret(message):
    msg = message.lower()
    if 'order' in msg:
        return 'order'
    if 'kenyan' in msg or 'columbian' in msg:
        return 'specify_coffee'
    if 'what' in msg:
        return 'ask_explanation'
    return 'none'
 # Define the states
INIT=0
CHOOSE_COFFEE=1
ORDERED=2
```

```
# Define the policy rules dictionary

policy_rules = {

   (INIT, "ask_explanation"): (INIT, "I'm a bot to help you order coffee beans"),

   (INIT, "order"): (CHOOSE_COFFEE, "ok, Columbian or Kenyan?"),

   (CHOOSE_COFFEE, "specify_coffee"): (ORDERED, "perfect, the beans are on their
way!"),

   (CHOOSE_COFFEE, "ask_explanation"): (CHOOSE_COFFEE, "We have two kinds of
coffee beans - the Kenyan ones make a slightly sweeter coffee, and cost $6. The
Brazilian beans make a nutty coffee and cost $5.")

}

# Send the messages

send_messages([

   "what can you do for me?",

   "well then I'd like to order some coffee",

   "what do you mean by that?",

   "kenyan"

])
```


3. Exercise 3: Dealing with rejection

   a. What happens if you make a suggestion to your user, and they don't like it? Your bot will look really silly if it makes the same suggestion again right away.

   b. Here, you're going to modify your respond() function so that it accepts and returns 4 arguments:

   c. The user message as an argument, and the bot response as the first return value.

   d. A dictionary params including the entities the user has specified.

   e. A suggestions list. When passed to respond(), this should contain the suggestions made in the previous bot message. When returned by respond(), it should contain the current suggestions.

   f. An excluded list, which contains all of the results your user has already explicitly rejected.

g. Your function should add the previous suggestions to the excluded list whenever it receives a "deny" intent. It should also filter out excluded suggestions from the response.

h. Need to write logic

47.2 Asking questions & queuing answers

1.

48. Heading

2. simple line

49. Heading