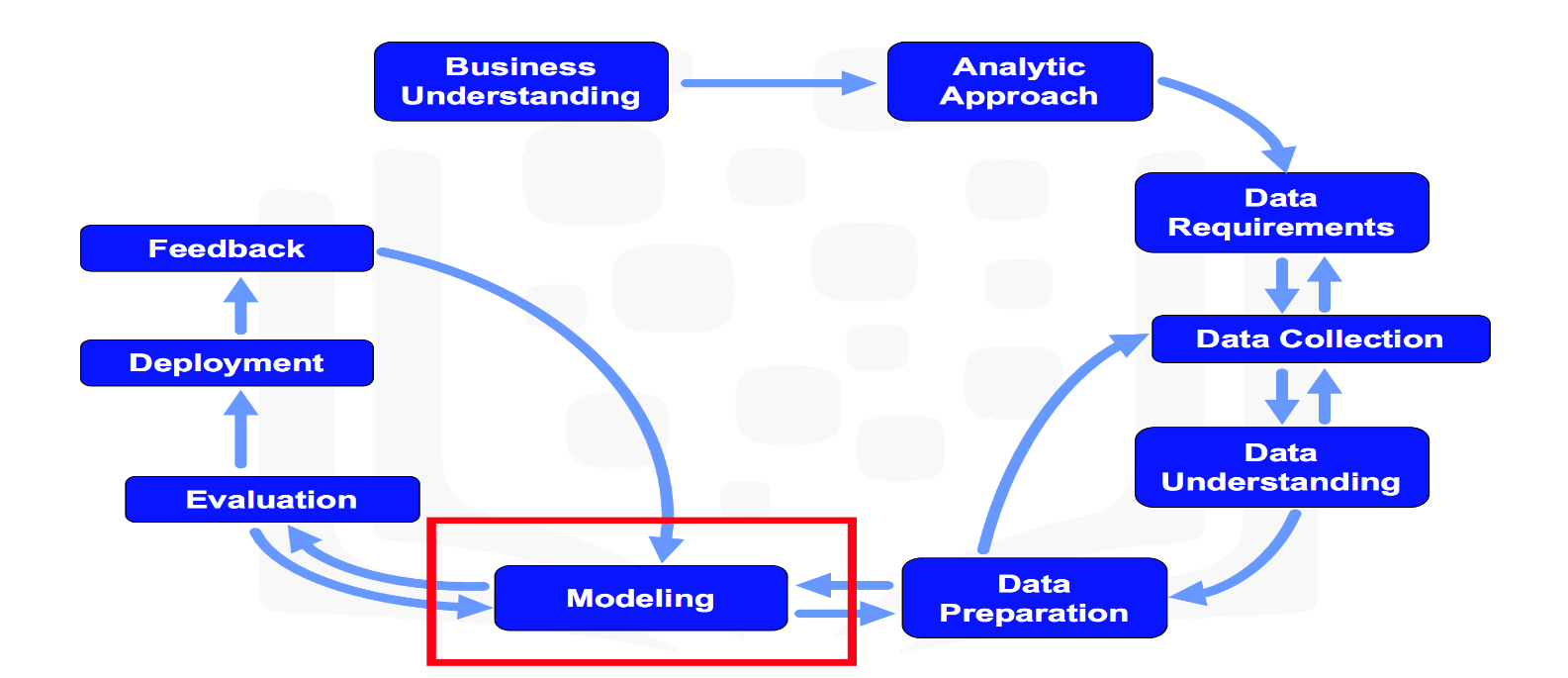
**Where to start**

* **Do not know then you can’t assess**
* **start capturing, cost, from where revenue comes etc. and analyse apply algorithms, data analytics**
* **Archive the data, do not overwrite, documents**
* **Having Team having strength**
* **Hiring Data Scientist**
* **Firms DNA, Queries, Sens of humour, Storytelling, technical skills, problem solving, where to utilise them**
* **Statistical method, R, Hadoop**

**Medical, Google, Augmented Reality**

* **Consumer information and study**
* **Abuse of data**
* **False positive, nothing is perfect**
* **Predictive Analytics, don’t believe 100% on model**
* **Design Experiment**

****

**Module 1: From Problem to Approach and from Requirements to Collection**

* Includes Business Understanding, Analytic Approach, Data Requirements, Data Collection

|  |  |
| --- | --- |
| ***Lab1: From Problem to Approach***   * The need to understand and prioritize the business goal. * The way stakeholder support influences a project. * The importance of selecting the right model. * When to use a predictive, descriptive, or classification model. | ***Lab2: From Requirement to Collection***   * The significance of defining the data requirements for your model. * Why the content, format, and representation of your data matter. * The importance of identifying the correct sources of data for your project. * How to handle unavailable and redundant data. * To anticipate the needs of future stages in the process. |

**Module 2: From Understanding to Preparation and from Modelling to Evaluation**

* Includes Data Understanding, Data Preparation, Modelling, Evaluation

|  |  |
| --- | --- |
| ***Lab1: From Understanding to Preparation***   * The importance of descriptive statistics. * How to manage missing, invalid, or misleading data. * The need to clean data and sometime transform. * The consequences of bad data for the model. * Data understanding is iterative; you learn more about your data the more you study it. | ***Lab2: From Modelling to Evaluation***   * The difference between descriptive and predictive models. * The role of training sets and test sets. * The importance of asking if the question has been answered. * Why diagnostic measures tools are needed. * The purpose of statistical significance tests. * That modelling and evaluation are iterative processes. |

**Module 3: From Deployment to Feedback**

* Includes Model Deployment & Feedback

|  |  |
| --- | --- |
| ***Lab1:*** Model Deployment   * The difference between descriptive and predictive models. The role of training sets and test sets. * The importance of asking if the question has been answered. * Why diagnostic measures tools are needed. * The purpose of statistical significance tests. & That modelling and evaluation are iterative processes. | Feedback   * The importance of stakeholder input. * To consider the scale of deployment. * The importance of incorporating feedback to refine the model. * The refined model must be redeployed. * This process should be repeated as often as necessary. |

**Data Science Methodology Assignment**

I would like to choose “Emails" as a topic to apply the data science methodology. Given most of the e-commerce and IT companies, now has IT enabled services for data and analytics offerings. also, there is inhouse requirement for every organisation where they have to deal with customer\client emails and feedbacks to either designing product or service or solve problems based on data driven solutions.

Scenario: Let's assume that this is for the organisation offering E-service for travel across country. E-commerce organisation has customer connect centre and many of the request, queries, and complains comes through emails. Organisation conducts the surveys regularly, usually asks to fill the feedback forms, which has free text option other than ratings as well. that means has data in various forms structured as well as unstructured.  
  
Business owners’ requirement: Given the free text in emails and survey forms would It be possible to identify the areas of improvement in any given process which needs immediate attention for day to day operations.   
  
Can we automatically Identify that what are the driving factors for satisfaction or dissatisfaction of customers for particular time?

1. Analytic Approach – Given the questions of deriving insights between service provider and customer there will be relational variables which is either dependant or in-dependent and sometime driven by specific events such as promotions, season, response time and skills of executive handling the customer.
2. Would suggest and discuss the descriptive model approach with stake holders, in order design standard automatically generated reports showcasing Key performing Index (KPI) and progress for day to day operation.
3. Given the availability of customer survey data with classification and rating scale, we will design a report using statistical analysis for analysing customer behaviours and deriving customer satisfaction driving factor.

At this the stage I would discuss the key terminology, Analysis and content of reports and deliverables

1. Data Requirements

Given we have already had the transactional data for customer as for every booking made by customer along with email communication and feedback given per case, I would further focus on analysing the data requirement for building dashboard and gathering information where to find the requisite information.

1. Data Collection

At the stage, will be consolidating data and looking in case we have the data that we need for modelling and designing automated reports, also we can review the survey forms available in order to collect some specific information about customer

1. Data Understanding and Preparation

At this stage, will be working with data cleaning, organising and bring in the consistent format and classifying it by processes, service offering or channels such as voice, chat, email. check for data quality for ambiguous, duplicate, redundant, missing, unrealistic information.

1. Modelling and Evaluation

Plot data using statistical, visualising tools to understand the different variables. Sample the data from each representative process, Chanel etc. and publish the reports to key process owners and stake holders to seek feedback and evaluation of reports and insight derived out of such reports. Based on the feedback and initial refinement, process and methodology derived can be automated in order to gain quick insights.

**Python**:

* *Module 1 - Python Basics*
* Your first program
* Types

|  |  |
| --- | --- |
| int | 10 |
| float | 10.111 |
| str | “Hello” |
| bool | True or False |

* Check data type by 🡪 type(10)
* Convert data type by assigning data type as 🡪 float(10) gives 10.0, int(10.1) gives 1 so on
* Expressions and Variables
* operands (values in expression) and operator
* Note // rounds value to integer
* Define variable by “=” and assign value by “:”
* You can redefine same variable as in x=4 then x=x/2 will give x=2.0
* String Operations

|  |  |
| --- | --- |
| print('Hello, Python!') | Print single quotes |
| import sys  print(sys.version)  # Check the Python Version  [Tip:] sys is a built-in module that contains many system-specific parameters and functions, including the Python version in use. Before using it, we must explicitly import it. | |
| print('Hello, Python!') # This line prints a string  # print('Hi') # Practice on writing comments | |
| Errors gives if it is in the module name or syntax error | |
| known as slice, you can access String character by variable name and index no, you can use -number as well for index [-1]  Name = ‘hello’  Name[2] gives -> e,  Name[-1] gives -> o,  Name[2:3] will give -> l  Name1 = Name1 + “, Python!” +’\_’ + Name[::2] +’\_’+ Name[1:4:2] +’\_’+ Name[0:4] +’\_’+ Name.lower() | |
| Name[0:5:2] will give every second up to 5 letter  Name[::2] will give every second char -known as stride | Known as stride, Name is a variable here with str value |
| Name1 = Name.upper() - > new variable with upper text  Name.upper() -> will just print in upper | Upper and lower case to text |
| Name + “, Python!” | You can merge string just by + |
| 3\*Name will give same text 3 times as -> ‘HelloHelloHello’ | You can multiple string known as tuple by |
| String are immutable you can create new variable but you can’t edit you can just apply method only  .upper  .lower  .find(‘a’) this will return index and -1 In case not found  .replce(a,b) replace a with b | You can use escape sequence such as \n for new line and \t for tab in string, you need to put \\ in case you need \ with in your string as char while printing. Or you can use r such as print(r”Hello\chk”) |

* *Module 2 - Python Data Structures*
* Lists and Tuples

|  |  |
| --- | --- |
| String store character, Lists and tuple stores elements | “”,[“”,””] , (“”, “”, “”) |
| # Create tuple  # Sort the tuple - We can sort the values in a tuple and save it to a new tuple:  # Create a nest tuple  # A tuple can contain another tuple as well as other more complex data types. This process is called 'nesting'. Consider the following tuple with several elements:  # Find the first index of "disco": | - tuple1 = ("disco",10,1.2) , Ratings = (0, 9, 6, 5, 10)  - RatingsSorted = sorted(Ratings)  - NestedT = (1, 2, ("pop", "rock"),(3,4),("disco",(1,2)))  - Print("Element 4 of Tuple: ", NestedT[4])  - NestedT[2][1][2]  - tuple1.index("disco") |
| # Create a list  # Print the elements on each index  For list methods are  L.extend[] adds individual elements separately (sql add columns)  L.append[] adds block or another whole list to list i.e. two columns added but only one element (sql merge table)  L[indexno] = ‘’ (sql update or set)  Del(L[indexno]) (sql del column)  L.split() to split text by space  L.split(‘,’) split by given symbol in ‘’ | L = ["Michael Jackson", 10.1, 1982]  print('the same element using negative and positive indexing:\n Postive:',L[0],'\n Negative:' , L[-3])  L = [ "Michael Jackson", 10.2]  L.extend(['pop', 10])  Results - ['Michael Jackson', 10.2, 'pop', 10]  L.append(['a','b'])  Results - ['Michael Jackson', 10.2, 'pop', 10, ['a', 'b']]  L[0] = ‘Michael'  Results - ['Michael', 10.2, 'pop', 10, ['a', 'b']]  Del(L[0])  Results - [ 10.2, 'pop', 10, ['a', 'b']] |
| # Copy (copy by reference) the list A  A = ["hard rock", 10, 1.2]  B = A | Initially, the value of the first element in **B** is set as hard rock. If we change the first element in **A** to **banana**, we get an unexpected side effect. As **A** and **B** are referencing the same list, if we change list **A**, then list **B** also changes. If we check the first element of **B** we get banana instead of hard rock:  # Clone (clone by value) the list A  Do copy by list by using B = A[:] instead to make a copy of list |

* Sets

|  |
| --- |
| A **set is a unique collection of objects in Python**. You can denote a set with a curly bracket {}. **Python will automatically remove duplicate items:**  set1 = {"pop", "rock", "soul", "hard rock", "rock", "R&B", "rock", "disco"}  set1  album\_list = [ "Michael Jackson", "Thriller", 1982, "00:42:19", "Pop, Rock, R&B", 46.0, 65, "30-Nov-82", None]  album\_set = set(album\_list)  print(album\_list)  print (album\_set)  ['Michael Jackson', 'Thriller', 1982, '00:42:19', 'Pop, Rock, R&B', 46.0, 65, '30-Nov-82', None]  {65, None, '00:42:19', 46.0, 'Pop, Rock, R&B', '30-Nov-82', 'Thriller', 'Michael Jackson', 1982}  # Convert list to set  music\_genres = set(["pop", "pop", "rock", "folk rock", "hard rock", "soul", "progressive rock", "R&B", "disco"])  music\_genres  We can add, remove an element to a set using the add(), remove() method  A = set(["Thriller", "Back in Black", "AC/DC"]) # Sample set  A.add("NSYNC") results {'AC/DC', 'Back in Black', 'NSYNC', 'Thriller'} **# Add the element to set**  A.remove("NSYNC") results {'AC/DC', 'Back in Black', 'Thriller'} **# Remove the element from set**  "AC/DC" in A **# Verify if the element is in the set**  **Remember that with sets you can check the difference between sets, as well as the symmetric difference, intersection, and union:**  **Consider the following two sets:**  **# Sample Sets**  album\_set1 = set(["Thriller", 'AC/DC', 'Back in Black'])  album\_set2 = set([ "AC/DC", "Back in Black", "The Dark Side of the Moon"])  album\_set1, album\_set2 **# Print two sets**  intersection  album\_set1.difference(album\_set2) **# Find the difference in set1 but not set2**  album\_set2.difference(album\_set1) **# The elements in**album\_set2**but not in**album\_set1**is given**  album\_set1.intersection(album\_set2) **# Use intersection method to find the intersection of sets**  intersection = album\_set1 & album\_set2 **# Find the intersections using &**  album\_set1.union(album\_set2) **# Find the union of two sets**  set(album\_set1).issuperset(album\_set2) **# Check if superset**  set(album\_set2).issubset(album\_set1) **# Check if subset**  set({"Back in Black", "AC/DC"}).issubset(album\_set1) **# Check if subset**  album\_set1.issuperset({"Back in Black", "AC/DC"}) **# Check if superset** |

* Dictionaries

|  |
| --- |
| **What are Dictionaries?** A dictionary consists of keys and values. It is helpful to compare a dictionary to a list. Instead of the numerical indexes such as a list, dictionaries have keys. These keys are the keys that are used to access values within a dictionary.  Each key is separated from its value by a colon ":". Commas separate the items, and the whole dictionary is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this "{}".  # Create the dictionary # each  Dict = {"key1": 1, "key2": "2", "key3": [3, 3, 3], "key4": (4, 4, 4), ('key5'): 5, (0, 1): 6}  Dict  # Access to the value by the key  Dict["key1"]  # Access to the value by the key  Dict[(0, 1)]  # Create a sample dictionary  release\_year\_dict = {"Thriller": "1982", "Back in Black": "1980", \  "The Dark Side of the Moon": "1973", "The Bodyguard": "1992", \  "Bat Out of Hell": "1977", "Their Greatest Hits (1971-1975)": "1976", \  "Saturday Night Fever": "1977", "Rumours": "1977"}  release\_year\_dict  In summary, like a list, a dictionary holds a sequence of elements. Each element is represented by a key and its corresponding value. Dictionaries are created with two curly braces containing keys and values separated by a colon. For every key, there can only be one single value, however, multiple keys can hold the same value. Keys can only be strings, numbers, or tuples, but values can be any data type.  It is helpful to visualize the dictionary as a table, as in the following image. The first column represents the keys, the second column represents the values.  # Get value by keys  release\_year\_dict['Thriller']  # Get all the keys in dictionary  release\_year\_dict.keys()  # Get all the values in dictionary  release\_year\_dict.values()  # Append value with key into dictionary  release\_year\_dict['Graduation'] = '2007'  # Delete entries by key  del(release\_year\_dict['Thriller'])  del(release\_year\_dict['Graduation'])  # Verify the key is in the dictionary  'The Bodyguard' in release\_year\_dict  album\_sales\_dict = {"The Bodyguard":50, "Back in Black":50, "Thriller":65}  album\_sales\_dict["Thriller"]  total\_sale = sum(album\_sales\_dict.values()) |
|  |

* *Module 3 - Python Programming Fundamentals*
* Conditions and Branching

|  |
| --- |
| **Condition Statements :**  Comparison Operators, Comparison operations compare some value or operand and, based on a condition, they produce a Boolean. When comparing two values you can use these operators:   * equal: **==** * not equal: **!=** * greater than: **>** * less than: **<** * greater than or equal to: **>=** * less than or equal to: **<=**   a == 6 gives false as operators  **# If statement example**  age = 19  #age = 18  #expression that can be true or false  if age > 18:  #within an indent, we have the expression that is run if the condition is true  print("you can enter" )  #The statements after the if statement will run regardless if the condition is true or false  print("move on")  ---------  **# Else statement example**  age = 18  # age = 19  if age > 18:  print("you can enter" )  else:  print("go see Meat Loaf" )  print("move on")  ---------  **# Elif statment example**  age = 18  if age > 18:  print("you can enter" )  elif age == 18:  print("go see Pink Floyd")  else:  print("go see Meat Loaf" )    print("move on")  **# Condition statement example**  album\_year = 1983  album\_year = 1970  if album\_year > 1980:  print("Album year is greater than 1980")  print('do something..')  Notice that the code in the above **indented** block will only be executed if the results are **True**.  As before, we can add an else block to the if block. The code in the else block will only be executed if the result is **False**.  **Syntax:**  if (condition): # do something else: # do something else  If the condition in the if statement is **False**, the statement after the else block will execute. This is demonstrated in the figure:  Sometimes you want to check more than one condition at once. For example, you might want to check if one condition and another condition is **True**. Logical operators allow you to combine or modify conditions.   * and * or * not   The and statement is only **True** when both conditions are true. The or statement is true if one condition is **True**. The not statement outputs the opposite truth value.  **And Statement**  album\_year = 1980  if(album\_year > 1979) and (album\_year < 1990):  print ("Album year was in between 1980 and 1989")  print("")  print("Do Stuff..")  **OR Statement**  album\_year = 1990  if(album\_year < 1980) or (album\_year > 1989):  print ("Album was not made in the 1980's")  else:  print("The Album was made in the 1980's ")  **IF NOT Statement**  if not (album\_year == '1984'):  print ("Album year is not 1984") |

* Loops

|  |
| --- |
| Range  Sometimes, you might want to repeat a given operation many times. Repeated executions like this are performed by **loops**. We will look at two types of loops, for loops and while loops.  Before we discuss loops lets discuss the range object. It is helpful to think of the range object as an ordered list. For now, let's look at the simplest case. If we would like to generate a sequence that contains three elements ordered from 0 to 2 we simply use the following command:  range(3)  # For loop example  dates = [1982,1980,1973]  N = len(dates)  for i in range(N):  print(dates[i])  # Example of for loop  for i in range(0, 8):  print(i)  **# Use for loop to change the elements in list**  squares = ['red', 'yellow', 'green', 'purple', 'blue']  for i in range(0, 5):  print("Before square ", i, 'is', squares[i])  squares[i] = 'weight'  print("After square ", i, 'is', squares[i])  **# Loop through the list and iterate on both index and element value**  squares=['red', 'yellow', 'green', 'purple', 'blue']  for i, square in enumerate(squares):  print(i, square)  **# While Loop Example**  dates = [1982, 1980, 1973, 2000]  i = 0; year = 0; while(year != 1973):  year = dates[i]  i = i + 1  print(year)  print("It took ", i ,"repetitions to get out of loop.")  A while loop iterates merely until the condition in the argument is not met, also do not forget to indent on statements  squares = ['orange', 'orange', 'purple', 'blue ', 'orange']  new\_squares = []  i = 0  while(squares[i] == 'orange'):  new\_squares.append(squares[i])  i = i + 1  print (new\_squares)  -->  squares = ['orange', 'orange', 'purple', 'blue ', 'orange']  new\_squares = []  i = 0  sqcolor = squares[i]  while sqcolor == 'orange':  new\_squares.extend(sqcolor)  print (new\_squares[i])  i = i+1 |

* Functions

|  |
| --- |
| Functions  A function is a reusable block of code which performs operations specified in the function. They let you break down tasks and allow you to reuse your code in different programs.  There are two types of functions:  **Pre-defined functions**  **User defined functions**  What is a Function?  You can define functions to provide the required functionality. Here are simple rules to define a function in Python:  Functions blocks begin def followed by the function name and parentheses ().  There are input parameters or arguments that should be placed within these parentheses.  You can also define parameters inside these parentheses.  There is a body within every function that starts with a colon (:) and is indented.  You can also place documentation before the body  The statement return exits a function, optionally passing back a value  An example of a function that adds on to the parameter a prints and returns the output as b:  **# First function example: Add 1 to a and store as b**  def add(a):  b = a + 1  print(a, "if you add one", b)  return(b)    help(add) # help on function  add(1) # calling function with parameter  **# Define a function for multiple two numbers**  def Mult(a, b):  c = a \* b  return(c)  Mult(2, 3)  Mult(10.0, 3.14)  **# We can even replicate a string by multiplying with an integer: Use mult() multiply two different type values together**  Mult(2, "Michael Jackson ")  # Function Definition  def square(a):  # Local variable b  b = 1  c = a \* a + b  print(a, "if you square + 1", c)  return(c)  **# Make a Function for the calculation**  def Equation(a,b):  c = a + b + 2 \* a \* b - 1  if(c < 0):  c = 0  else:  c = 5  return(c)  **# Function example**  def type\_of\_album(artist, album, year\_released):    print(artist, album, year\_released)  if year\_released > 1980:  return "Modern"  else:  return "Oldie"    x = type\_of\_album("Michael Jackson", "Thriller", 1980)  print(x)  **# Print the list using for loop**  def PrintList(the\_list):  for element in the\_list:  print(element)  **# Implement the printlist function**  PrintList(['1', 1, 'the man', "abc"])  **Setting default argument values in your custom functions**  You can set a default value for arguments in your function. For example, in the isGoodRating() function, what if we wanted to create a threshold for what we consider to be a good rating? Perhaps by default, we should have a default rating of 4  *# Example for setting param with default value*  ​  **def** isGoodRating(rating**=**4):  **if**(rating **<** 7):  print("this album sucks it's rating is",rating)    **else**:  print("this album is good its rating is",rating)  *# Test the value with default value and with input*  ​  isGoodRating()  isGoodRating(10)  ​  **# Example of global variable**  artist = "Michael Jackson"  def printer1(artist):  internal\_var = artist  print(artist, "is an artist")    printer1(artist)  [ ]:  If we print internal\_var we get an error.  **We got a Name Error: name 'internal\_var' is not defined. Why?**  It's because all the variables we create in the function is a **local variable**, meaning that the variable assignment does not persist outside the function.  But there is a way to create **global variables** from within a function as follows:  artist = "Michael Jackson"  def printer(artist):  global internal\_var  internal\_var= "Whitney Houston"  print(artist,"is an artist")  printer(artist)  printer(internal\_var)  # Example of global variable and local variable with the same name  myFavouriteBand = "AC/DC"  def getBandRating(bandname):  myFavouriteBand = "Deep Purple"  if bandname == myFavouriteBand:  return 10.0  else:  return 0.0  print("AC/DC's rating is:",getBandRating("AC/DC"))  print("Deep Purple's rating is: ",getBandRating("Deep Purple"))  print("My favourite band is:",myFavouriteBand)  AC/DC's rating is: 0.0  Deep Purple's rating is: 10.0  My favourite band is: AC/DC |

* Objects and Classes

|  |
| --- |
| Introduction to Classes and Objects  Creating a Class  The first part of creating a class is giving it a name: In this notebook, we will create two classes, Circle and Rectangle. We need to determine all the data that make up that class, and we call that an attribute. Think about this step as creating a blue print that we will use to create objects. In figure 1 we see two classes, circle and rectangle. Each has their attributes, they are variables. The class circle has the attribute radius and color, while the rectangle has the attribute height and width. Let’s use the visual examples of these shapes before we get to the code, as this will help you get accustomed to the vocabulary.    Instances of a Class: Objects and Attributes  An instance of an object is the realisation of a class, and in Figure 2 we see three instances of the class circle. We give each object a name: red circle, yellow circle and green circle. Each object has different attributes, so let's focus on the attribute of colour for each object.    Methods  Methods give you a way to change or interact with the object; they are functions that interact with objects. For example, let’s say we would like to increase the radius by a specified amount of a circle. We can create a method called **add\_radius(r)** that increases the radius by **r**. This is shown in figure 3, where after applying the method to the "orange circle object", the radius of the object increases accordingly. The “dot” notation means to apply the method to the object, which is essentially applying a function to the information in the object.    **Creating a Class**  **Now we are going to create a class circle, but first, we are going to import a library to draw the objects:**  **# Import the library**  import matplotlib.pyplot as plt  %matplotlib inline  The first step in creating your own class is to use the class keyword, then the name of the class as shown in Figure 4. In this course the class parent will always be object:    *Figure 4: Three instances of the class circle or three objects of type circle.*  The next step is a special method called a constructor \_\_init\_\_, which is used to initialize the object. The input are data attributes. The term self contains all the attributes in the set. For example the self.color gives the value of the attribute color and self.radius will give you the radius of the object. We also have the method add\_radius() with the parameter r, the method adds the value of r to the attribute radius. To access the radius we use the syntax self.radius. The labeled syntax is summarized in Figure 5:    The actual object is shown below. We include the method drawCircle to display the image of a circle. We set the default radius to 3 and the default colour to blue:  **# Create a class Circle**  class Circle(object):    # Constructor  def \_\_init\_\_(self, radius=3, color='blue'):  self.radius = radius  self.color = color    # Method  def add\_radius(self, r):  self.radius = self.radius + r  return(self.radius)    # Method  def drawCircle(self):  plt.gca().add\_patch(plt.Circle((0, 0), radius=self.radius, fc=self.color))  plt.axis('scaled')  plt.show()  Creating an instance of a class Circle  Let’s create the object RedCircle of type Circle to do the following:  **# Create an object RedCircle**  RedCircle = Circle(10, 'red')  We can use the dir command to get a list of the object's methods. Many of them are default Python methods.  dir(RedCircle)  # Print the object attribute radius  RedCircle.radius # Print the object attribute radius  RedCircle.color # Print the object attribute color  RedCircle.radius = 1 # Set the object attribute radius  RedCircle.drawCircle() # Call the method drawCircle  **# Use method to change the object attribute radius**  print('Radius of object:',RedCircle.radius)  RedCircle.add\_radius(2)  print('Radius of object of after applying the method add\_radius(2):',RedCircle.radius)  RedCircle.add\_radius(5)  print('Radius of object of after applying the method add\_radius(5):',RedCircle.radius)  # Create a new Rectangle class for creating a rectangle object  class Rectangle(object):    # Constructor  def \_\_init\_\_(self, width=2, height=3, color='r'):  self.height = height  self.width = width  self.color = color    # Method  def drawRectangle(self):  plt.gca().add\_patch(plt.Rectangle((0, 0), self.width, self.height ,fc=self.color))  plt.axis('scaled')  plt.show() |

* *Module 4 - Working with Data in Python*
* **Reading & Writing files with open**

|  |
| --- |
| **# Read the Example1.txt**  example1 = "/resources/data/Example1.txt"  file1 = open(example1, "r")  file1.name # Print the path of file  file1.mode # Print the mode of file, either 'r' or 'w'  **# Read the file #** We can read the file and assign it to a variable : The \**n** means that there is a new line.  FileContent = file1.read()  FileContent  print(FileContent) # Print the file with '\n' as a new line  type(FileContent) # Type of file content  file1.close() # Close file after finish  A Better Way to Open a File  Using the with statement is better practice, it automatically closes the file even if the code encounters an exception. The code will run everything in the indent block then close the file object.  **# Open file using with**  with open(example1, "r") as file1:  FileContent = file1.read()  print(FileContent)    **# Read first four characters**  with open(example1, "r") as file1:  print(file1.read(4))  **# Read certain amount of characters**  with open(example1, "r") as file1:  print(file1.read(4))  print(file1.read(4))  print(file1.read(7))  print(file1.read(15))  **# Read one line**  with open(example1, "r") as file1:  print("first line: " + file1.readline())  **# Iterate through the lines**  with open(example1,"r") as file1:  i = 0;  for line in file1:  print("Iteration", str(i), ": ", line)  i = i + 1;  **# Read all lines and save as a list,** note difference of “s” i.e readline() vs readlines()  with open(example1, "r") as file1:  FileasList = file1.readlines()  **# Write line to file**  with open('/resources/data/Example2.txt', 'w') as writefile:  writefile.write("This is line A")  **# Read file**  with open('/resources/data/Example2.txt', 'r') as testwritefile:  print(testwritefile.read())  **# Write lines to file**  with open('/resources/data/Example2.txt', 'w') as writefile:  writefile.write("This is line A\n")  writefile.write("This is line B\n")  **The method .write() works similar to the method .readline() except instead of reading a new line it writes a new line**  **# Check whether write to file**  with open('/resources/data/Example2.txt', 'r') as testwritefile:  print(testwritefile.read())  **By setting the mode argument to append a you can append a new line as follows:**  **# Write a new line to text file**  with open('/resources/data/Example2.txt', 'a') as testwritefile:  testwritefile.write("This is line C\n")  **# Sample list of text, # Write the strings in the list to text file**  Lines = ["This is line A\n", "This is line B\n", "This is line C\n"]  with open('Example2.txt', 'w') as writefile:  for line in Lines:  print(line)  writefile.write(line)  **# Copy file to another**  with open('Example2.txt','r') as readfile:  with open('Example3.txt','w') as writefile:  for line in readfile:  writefile.write(line) |

* Loading data with Pandas.0

|  |
| --- |
| Introduction of Pandas  **# Dependency needed to install file**  !pip install xlrd  **# Import required library**  import pandas as pd  After the import command, we now have access to a large number of pre-built classes and functions. This assumes the library is installed; in our lab environment all the necessary libraries are installed. One way pandas allows you to work with data is a dataframe. Let's go through the process to go from a comma separated values (.csv) file to a dataframe. This variable csv\_path stores the path of the .csv, that is used as an argument to the read\_csv function. The result is stored in the object df, this is a common short form used for a variable referring to a Pandas dataframe.  **# Read data from CSV file**  csv\_path = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/PY0101EN/Chapter%204/Datasets/TopSellingAlbums.csv'  df = pd.read\_csv(csv\_path)  **# Print first five rows of the dataframe**  df.head()  **# Read data from Excel File and print the first five rows,** We use the path of the excel file and the function read excel. The result is a data frame as before:  xlsx\_path = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/PY0101EN/Chapter%204/Datasets/TopSellingAlbums.xlsx'  df = pd.read\_excel(xlsx\_path)  df.head()  **# Access to the column Length**  x = df[['Length']]  x  **# Get the column as a series**  x = df['Length']  x  **# Get the column as a dataframe**  x = type(df[['Artist']])  x # pandas.core.frame.DataFrame  **# Access to multiple columns**  y = df[['Artist','Length','Genre']]  y  **# Access the value by row and column no.**  df.iloc[0, 0] # Access the value on the first row and the first column  df.iloc[0,2] # Access the value on the first row and the third column  **# Access the column using the name, not use only loc when using column or raw names vs iloc with no.**  df.loc[0, 'Artist']  df.loc[0, 'Released']  df.loc[1, 'Released']  **# Slicing the dataframe,** You can perform slicing using both the index and the name of the column:  df.iloc[0:2, 0:3] # Slicing the dataframe using number  df.loc[0:2, 'Artist':'Released'] # Slicing the dataframe using name  df\_teams[df\_teams['nickname']=='Warriors'] # select |

* Working with and Saving data with Pandas

|  |
| --- |
|  |

* Numpy

|  |
| --- |
| **1D Numpy in Python**  **Welcome!** This notebook will teach you about using Numpy in the Python Programming Language. By the end of this lab, you'll know what Numpy is and the Numpy operations.  **Preparation:**  **# Import the libraries**  import time ;import sys; import numpy as np  import matplotlib.pyplot as plt  %matplotlib inline  **# Plotting functions**  def Plotvec1(u, z, v):    ax = plt.axes()  ax.arrow(0, 0, \*u, head\_width=0.05, color='r', head\_length=0.1)  plt.text(\*(u + 0.1), 'u')    ax.arrow(0, 0, \*v, head\_width=0.05, color='b', head\_length=0.1)  plt.text(\*(v + 0.1), 'v')  ax.arrow(0, 0, \*z, head\_width=0.05, head\_length=0.1)  plt.text(\*(z + 0.1), 'z')  plt.ylim(-2, 2)  plt.xlim(-2, 2)  def Plotvec2(a,b):  ax = plt.axes()  ax.arrow(0, 0, \*a, head\_width=0.05, color ='r', head\_length=0.1)  plt.text(\*(a + 0.1), 'a')  ax.arrow(0, 0, \*b, head\_width=0.05, color ='b', head\_length=0.1)  plt.text(\*(b + 0.1), 'b')  plt.ylim(-2, 2)  plt.xlim(-2, 2)  **# Create a python list**  a = ["0", 1, "two", "3", 4]  **# We can access the data via an index # Print each element**  print("a[0]:", a[0])  print("a[1]:", a[1])  **What is Numpy?**  A numpy array is similar to a list. It's usually fixed in size and each element is of the same type. We can cast a list to a numpy array by first importing numpy:  **# Create a numpy arrays**  import numpy as np  a = np.array([0, 1, 2, 3, 4])  b = np.array([3.1, 11.02, 6.2, 213.2, 5.2])  c = np.array([20, 1, 2, 3, 4])  Each element is of the same type, as with lists, we can access each element via a square bracket:  print("a[2]:", a[2])  print("b[3]:", b[3])  **# Check the type of the numpy array, it will show as numpy.ndarray**  type(a)  **# Check the type of the values stored in array, it will show as dtype('int64') for array “a” and dtype('float64') for array “b”**  a.dtype  b.dtype  **# Assign values, We can change the value of the array, consider the array c above**  # We can change the first element of the array to 100 and 5th element of the array to 0 as follows:  c[0] = 100  c[4] = 0 # updated array will look like : array([100, 1, 2, 3, 0])  c[1:4] # Slicing the numpy array  c[3:5] = 300, 400 # Set the fourth element and fifth element to 300 and 400, array([100, 1, 2, 300, 400])  **Similarly, we can use a list to select a specific index. The list ' select ' contains several values: such as**  **# Create the list with index value to select**  selecteg = [0, 2, 3]  d = c[selecteg]  d # will show array as array([100, 2, 300])  c[selecteg] = 100000 # Assign the specified elements to new value array([100000, 1, 100000, 100000, 400])  **Other** **Attributes:**  a.size # Get the size of numpy array  a.ndim # Get the number of dimensions  a.shape # Get the shape/size  a.mean() # Get the mean  a.std() # Get the standard deviation  b.max() # Get the biggest value  b.min() # Get the smallest value  **Numpy Array Operations:**  u = np.array([1, 0])  v = np.array([0, 1])  z = u + v # We can add the two arrays and assign it to z: array([1, 1]), The operation is equivalent to vector addition:  **# Plot numpy arrays**  Plotvec1(u, z, v)    **Array Multiplication**  y = np.array([1, 2]) # Create a numpy array  z = 2 \* y # Numpy Array Multiplication array([2, 4])  **Product of Two Numpy Arrays**  u = np.array([1, 2])  v = np.array([3, 2])  z = u \* v # Calculate the production of two numpy arrays array([3, 4])  **The dot product of the two numpy arrays u and v is given by**  np.dot(u, v) : see above value of array so dot product is = (1\*3)+(2\*2) = 7 np.dot[[1,3],[2,2]]  **# Add the constant to array**  u+1 # Adding the constant 1 to each element in the u array: array([2, 3])  **Mathematical Functions,** We can access the value of pie in numpy as follows:  np.pi # The value of pie  x = np.array([0, np.pi/2 , np.pi]) # Create the numpy array in radians  **We can apply the function sin to the array x and assign the values to the array y; this applies the sine function to each element in the array:**  y = np.sin(x) # Calculate the sin of each elements of array x : array([ 0.84147098, -0.84147098, 0.84147098, -0.84147098])  **Linspace**  A useful function for plotting mathematical functions is "linespace". Linespace returns evenly spaced numbers over a specified interval. We specify the starting point of the sequence and the ending point of the sequence. The parameter "num" indicates the Number of samples to generate, in this case 5:  np.linspace(-2, 2, num=5) # Makeup a numpy array within [-2, 2] and 5 elements array([-2., -1., 0., 1., 2.])  np.linspace(-2, 2, num=9) # Makeup a numpy array within [-2, 2] and 9 elements array([-2. , -1.5, -1. , -0.5, 0. , 0.5, 1. , 1.5, 2. ])    # Makeup a numpy array within [0, 2π] and 100 elements  x = np.linspace(0, 2\*np.pi, num=100)  # Calculate the sine of x list  y = np.sin(x)  # Plot the result  plt.plot(x, y)  **Create a 2D Numpy Array**  # Create a list, it has nested list that’s why its 2d [ [] ]  a = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]  **# Convert list to Numpy Array**  # Note Every element is the same type  A = np.array(a)  A.ndim # Get the number of dimensions # 2  A.size # Get the size of numpy array # 9  A.shape # Get the shape/size # (3,3)  **# Access the element**  A[1, 2] or A[1][2] # 23  A[0][0] # 11  **# Access the elements**  A[0][0:2] # array([11, 12])  A[0:2, 2] # array([13, 23])  **# Get the transposed matrix**  # Create a matrix C  C = np.array([[1,1],[2,2],[3,3]])  C.T  # array([[1, 2, 3],[1, 2, 3]])  [1,2]  [3,1]  [2,2]  [2,2] |
| Pandas is an API (as libraray in R pacakges)  import pandas as pd  import matplotlib.pyplot as plt  You create a dictionary, this is just data.  dict\_={'a':[11,21,31],'b':[12,22,32]}  When you create a Pandas object with the Dataframe constructor in API lingo, this is an "instance". The data in the dictionary is passed along to the pandas API. You then use the dataframe to communicate with the API.  df=pd.DataFrame(dict\_)  df.head()   |  |  |  | | --- | --- | --- | |  | a | b | | 0 | 11 | 12 | | 1 | 21 | 22 | | 2 | 31 | 32 |   type(df) # pandas.core.frame.DataFrame  df.mean() # both element of list i.e. a and b will have separate mean as output as a 21.0, b 22.0 and dtype: float64  **REST APIs**  Installed a package for API  !pip install nba\_api  Rest API’s function by sending a request, the request is communicated via HTTP message. The HTTP message usually contains a JSON file. This contains instructions for what operation we would like the service or resource to perform. In a similar manner, API returns a response, via an HTTP message, this response is usually contained within a JSON.  In this lab, we will use the NBA API to determine how well the Golden State Warriors performed against the Toronto Raptors. We will use the API do the determined number of points the Golden State Warriors won or lost by for each game. So if the value is three, the Golden State Warriors won by three points. Similarly, it the Golden State Warriors lost by two points the result will be negative two. The API is relatively will handle a lot of the details such a Endpoints and Authentication  In the nba api to make a request for a specific team, it's quite simple, we don't require a JSON all we require is an id. This information is stored locally in the API we import the module teams  from nba\_api.stats.static import teams  import matplotlib.pyplot as plt    # here teams is a module, <module 'nba\_api.stats.static.teams' from '/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/nba\_api/stats/static/teams.py'>  The method get\_teams() returns a list of dictionaries the dictionary key id has a unique identifier for each team as a value  nba\_teams = teams.get\_teams()  nba\_teams[0:1]  [{'id': 1610612737,  'full\_name': 'Atlanta Hawks',  'abbreviation': 'ATL',  'nickname': 'Hawks',  'city': 'Atlanta',  'state': 'Atlanta',  'year\_founded': 1949}]  To make things easier, we can convert the dictionary to a table. First, we use the function one dict, to create a dictionary. We use the common keys for each team as the keys, the value is a list; each element of the list corresponds to the values for each team. We then convert the dictionary to a dataframe, each row contains the information for a different team.  dict\_nba\_team=one\_dict(nba\_teams)  **Following function is used for this exercise:**  def one\_dict(list\_dict):  keys=list\_dict[0].keys()  out\_dict={key:[] for key in keys}  for dict\_ in list\_dict:  for key, value in dict\_.items():  out\_dict[key].append(value)  return out\_dict  for dict\_ in nba\_teams:  for key, value in dict\_.items():  my\_dict[key].append(value)  print(key,value);  df\_teams=pd.DataFrame(dict\_nba\_team)  df\_teams.head()  **Will use the team's nickname to find the unique id, we can see the row**  **that contains the warriors by using the column nickname as follows:**  df\_warriors = df\_teams[df\_teams['nickname']=='Warriors']  df\_warriors  **we can use the following line of code to access the first column of the**  **dataframe:**  id\_warriors=df\_warriors[['id']].values[0][0]  #we now have an integer that can be used to request the Warriors information  id\_warriors  **The function "League Game Finder " will make an API call, its in the module stats.endpoints**  from nba\_api.stats.endpoints import leaguegamefinder  The parameter team\_id\_nullable is the unique ID for the warriors. Under the hood, the NBA API is making a HTTP request.  The information requested is provided and is transmitted via an HTTP response this is assigned to the object gamefinder  # Since https://stats.nba.com does lot allow api calls from Cloud IPs and Skills Network Labs uses a Cloud IP.  # The following code is comment out, you can run it on jupyter labs on your own computer.  gamefinder = leaguegamefinder.LeagueGameFinder(team\_id\_nullable=id\_warriors)  we can see the json file by running the following line of code.  gamefinder.get\_json()  The game finder object has a method get\_data\_frames(), that returns a dataframe. If we view the dataframe, we can see it contains information about all the games the Warriors played. The PLUS\_MINUS column contains information on the score, if the value is negative the Warriors lost by that many points, if the value is positive, the warriors one by that amount of points. The column MATCHUP had the team the Warriors were playing, GSW stands for golden state and TOR means Toronto Raptors; vs signifies it was a home game and the @ symbol means an away game.  # games = gamefinder.get\_data\_frames()[0]  # games.head()  you can download the dataframe from the API call for Golden State and run the rest like a video.  ! wget https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/PY0101EN/Chapter%205/Labs/Golden\_State.pkl  file\_name = "Golden\_State.pkl"  games = pd.read\_pickle(file\_name)  games.head()  We can create two dataframes, one for the games that the Warriors faced the raptors at home and the second for away games.  games\_home=games [games ['MATCHUP']=='GSW vs. TOR']  games\_away=games [games ['MATCHUP']=='GSW @ TOR']  We can calculate the mean for the column PLUS\_MINUS for the dataframes games\_home and games\_away:  games\_home.mean()['PLUS\_MINUS']  games\_away.mean()['PLUS\_MINUS']  We can plot out the PLUS MINUS column for for the dataframes games\_home and games\_away. We see the warriors played better at home.  fig, ax = plt.subplots()  games\_away.plot(x='GAME\_DATE',y='PLUS\_MINUS', ax=ax)  games\_home.plot(x='GAME\_DATE',y='PLUS\_MINUS', ax=ax)  ax.legend(["away", "home"])  plt.show()  **Excersice 2 for Api – Text to Speech**  #you will need the following library  !pip install ibm\_watson wget  First we import SpeechToTextV1 from ibm\_watson. For more information on the API, please click on this [link](https://cloud.ibm.com/apidocs/speech-to-text?code=python)  from ibm\_watson import SpeechToTextV1  import json  from ibm\_cloud\_sdk\_core.authenticators import IAMAuthenticator  The service endpoint is based on the location of the service instance, we store the information in the variable URL. To find out which URL to use, view the service credentials.  # url\_s2t = <https://stream.watsonplatform.net/speech-to-text/api>  url\_s2t = <https://api.us-south.speech-to-text.watson.cloud.ibm.com/instances/d348d178-30f0-4760-85b1-086f92c75314>  You require an API key, and you can obtain the key on the Dashboard .  iam\_apikey\_s2t = "ACH2VqejQAlJfojB6iiUgYAfHJqBFmEDnJHVHjXEgdJ1"  You create a Speech To Text Adapter object the parameters are the endpoint and API key  authenticator = IAMAuthenticator(iam\_apikey\_s2t)  s2t = SpeechToTextV1(authenticator=authenticator)  s2t.set\_service\_url(url\_s2t)  s2t  Lets download the audio file that we will use to convert into text.  !wget -O PolynomialRegressionandPipelines.mp3 https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/PY0101EN/labs/PolynomialRegressionandPipelines.mp3  We have the path of the wav file we would like to convert to text  filename='PolynomialRegressionandPipelines.mp3'  We create the file object wav with the wav file using open ; we set the mode to "rb" , this is similar to read mode, but it ensures the file is in binary mode. We use the method recognize to return the recognized text. The parameter audio is the file object wav, the parameter content type is the format of the audio file.  with open(filename, mode="rb") as wav:  response = s2t.recognize(audio=wav, content\_type='audio/mp3')  The attribute result contains a dictionary that includes the translation:  response.result # dictionary  from pandas.io.json import json\_normalize  json\_normalize(response.result['results'],"alternatives") # it’s a function to simplify dictionary to table chk help(json\_normalize)  response # <ibm\_cloud\_sdk\_core.detailed\_response.DetailedResponse at 0x7fde7cb8db70> from IBM API  We can obtain the recognized text and assign it to the variable recognized\_text:  recognized\_text=response.result['results'][0]["alternatives"][0]["transcript"]  **Language Translator**  First we import LanguageTranslatorV3 from ibm\_watson. For more information on the API click here  from ibm\_watson import LanguageTranslatorV3  The service endpoint is based on the location of the service instance, we store the information in the variable URL. To find out which URL to use, view the service credentials.  #url\_lt='https://gateway.watsonplatform.net/language-translator/api'  url\_lt='https://api.us-south.language-translator.watson.cloud.ibm.com/instances/c7517c08-ae6a-46c6-8504-51d7a13b4630'  You require an API key, and you can obtain the key on the Dashboard.  apikey\_lt=''  apikey\_lt='XsTCJU58GgK4b5AP7t1wew1GNoGgrnOTeFyLlZrk-ZcL'  API requests require a version parameter that takes a date in the format version=YYYY-MM-DD. This lab describes the current version of Language Translator, 2018-05-01  version\_lt='2018-05-01'  we create a Language Translator object language\_translator:  authenticator = IAMAuthenticator(apikey\_lt)  language\_translator = LanguageTranslatorV3(version=version\_lt,authenticator=authenticator)  language\_translator.set\_service\_url(url\_lt)  language\_translator  We can get a Lists the languages that the service can identify. The method Returns the language code. For example English (en) to Spanis (es) and name of each language.  from pandas.io.json import json\_normalize  ​  json\_normalize(language\_translator.list\_identifiable\_languages().get\_result(), "languages")  We can use the method translate this will translate the text. The parameter text is the text. Model\_id is the type of model we would like to use use we use list the the langwich . In this case, we set it to 'en-es' or English to Spanish. We get a Detailed Response object translation\_response  translation\_response = language\_translator.translate(\  text=recognized\_text, model\_id='en-es')  translation\_response  The result is a dictionary.  translation=translation\_response.get\_result()  translation  We can obtain the actual translation as a string as follows:  spanish\_translation =translation['translations'][0]['translation']  spanish\_translation  We can translate back to English  translation\_new = language\_translator.translate(text=spanish\_translation ,model\_id='es-en').get\_result()  We can obtain the actual translation as a string as follows:  translation\_eng=translation\_new['translations'][0]['translation']  translation\_eng  We can convert it to french as well:  French\_translation=language\_translator.translate(  text=translation\_eng , model\_id='en-fr').get\_result()  French\_translation['translations'][0]['translation'] |

* *Module 5 - Final Project*

|  |
| --- |
| **Final Assignment**  **Analyzing US Economic Data and Building a Dashboard**  Description  Extracting essential data from a dataset and displaying it is a necessary part of data science; therefore, individuals can make correct decisions based on the data. In this assignment, you will extract some essential economic indicators from some data, you will then display these economic indicators in a Dashboard. You can then share the dashboard via an URL.  Gross domestic product (GDP) is a measure of the market value of all the final goods and services produced in a period. GDP is an indicator of how well the economy is doing. A drop in GDP indicates the economy is producing less; similarly, an increase in GDP suggests the economy is performing better. In this lab, you will examine how changes in GDP impact the unemployment rate. You will take screen shots of every step; you will share the notebook and the URL pointing to the dashboard.  **Define a Function that Makes a Dashboard**  Question 1: Create a data frame that contains the GDP data and display it  Question 2: Create a data frame that contains the unemployment data and display it  Question 3: Display a data frame where unemployment was greater than 8.5%  Question 4: Use the function make dashboard to make a dashboard  (Optional not marked) Save the dashboard on IBM cloud and display it  **Define Function that Makes a Dashboard**  We will import the following libraries.  import pandas as pd  from bokeh.plotting import figure, output\_file, show,output\_notebook  output\_notebook()  BokehJS 1.0.4 successfully loaded.  In this section, we define the function make\_dashboard. You don't have to know how the function works; you should only care about the inputs. The function will produce a dashboard as well as an html file. You can then use this html file to share your dashboard. If you do not know what an html file is don't worry everything you need to know will be provided in the lab.  def make\_dashboard(x, gdp\_change, unemployment, title, file\_name):  output\_file(file\_name)  p = figure(title=title, x\_axis\_label='year', y\_axis\_label='%')  p.line(x.squeeze(), gdp\_change.squeeze(), color="firebrick", line\_width=4, legend="% GDP change")  p.line(x.squeeze(), unemployment.squeeze(), line\_width=4, legend="% unemployed")  show(p)  The dictionary links contain the CSV files with all the data. The value for the key GDP is the file that contains the GDP data. The value for the key unemployment contains the unemployment data.  links={'GDP':'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/PY0101EN/projects/coursera\_project/clean\_gdp.csv',\  'unemployment':'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/PY0101EN/projects/coursera\_project/clean\_unemployment.csv'} |

**Markdown** is a simple way to format text that looks great on any device. It doesn’t do anything fancy like change the font size, color, or type — just the essentials, using keyboard symbols you already know.

[**TRY OUR 10 MINUTE MARKDOWN TUTORIAL**](https://commonmark.org/help/tutorial/)

| **Type** | **Or** | **… to Get** |
| --- | --- | --- |
| \*Italic\* | \_Italic\_ | Italic |
| \*\*Bold\*\* | \_\_Bold\_\_ | **Bold** |
| # Heading 1 | Heading 1 ========= | **Heading 1** |
| ## Heading 2 | Heading 2 --------- | **Heading 2** |
| [Link](http://a.com) | [Link][1] ⋮ [1]: http://b.org | [Link](https://commonmark.org/) |
| ![Image](http://url/a.png) | ![Image][1] ⋮ [1]: http://url/b.jpg | Markdown |
| > Blockquote |  | Blockquote |
| \* List \* List \* List | - List - List - List | * List * List * List |
| 1. One 2. Two 3. Three | 1) One 2) Two 3) Three | 1. One 2. Two 3. Three |
| Horizontal Rule  --- | Horizontal Rule  \*\*\* | Horizontal Rule |
| `Inline code` with backticks |  | Inline code with backticks |
| ``` # code block print '3 backticks or' print 'indent 4 spaces' ``` | ····# code block ····print '3 backticks or' ····print 'indent 4 spaces' | # code block print '3 backticks or' print 'indent 4 spaces' |

https://cloud.ibm.com/conapi/download/audit/resource/crn%3Av1%3Abluemix%3Apublic%3Acloud-object-storage%3Aglobal%3Aa%2Fb52705dbe62b445080953c74bb6a6f8c%3A4d2df738-a402-40c9-a9b6-5b26ba5d6d93%3A%3A/json/audit\_cloud-object-storage\_4d2df738-a402-40c9-a9b6-5b26ba5d6d93.json