**PROFESSIONAL INDUSTRIAL INTERNSHIP /**

**INTERNSHIP TRAINING**

**ON**

**“PYTHON PROGRAMMING”**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AT**

**WEBFREAK SOLUTIONS**

**( Plot No. - F-178, Industrial Area, Phase - 8B, Mohali, Punjab 160055 )**

**AN INDUSTRY INTERNSHIP REPORT SUBMITTED**

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS**

**FOR THE AWARD OF DEGREE OF**

**BACHELOR OF ENGINEERING**

**In**

**COMPUTER SCIENCE AND ENGINEERING**

**SUBMITTED BY:**

**Iftisam Tariq**

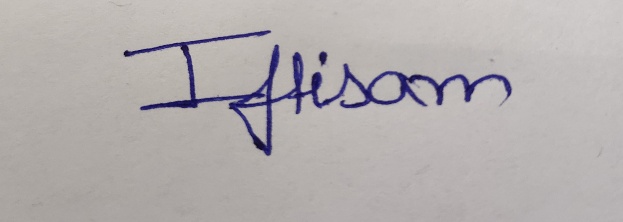
**2021a1l017**

****

**CANDIDATES’ DECLARATION**

I, **Iftisam Tariq**, bearing roll number **2021a1l017**, hereby declare that the work is being presented in the Industry Internship Report entitled, **“Python Programming”** in partial fulfillment of requirement for the award of degree of B.E. (CSE) and submitted in the **CSE Department**, Model Institute of Engineering and Technology (Autonomous),

Jammu is an authentic record of my own work carried by me at **“WEBFREAK SOLUTIONS,** Plot No. - F-178, Industrial Area, Phase - 8B, Mohali, Punjab 160055**”** under the supervision and mentorship of **Mr. Pawneshwar Gupta, Training Co-Ordinator** at WEBFREAK SOLUTIONS and **Mrs. Vishalika, Assistant Professor** (C.S.E department) at **Model Institute of Engineering and Technology** respectively. The matter presented in this report has not been submitted in this or any other University/ Institute for the award of a B.E. Degree.

****

**Student’s Signature Dated**: 22/09/2022

**Name:** Iftisam Tariq

**Roll Number:** 2021a1l017

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**INTERNSHIP CERTIFICATE**





**Department of Computer Science and Engineering**

**Model Institute of Engineering and Technology (Autonomous)**

**Kot Bhalwal, Jammu, India**

*(NAAC “A” Grade Accredited)*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Ref. No.: 2020A1R080 Dated: 22/09/2022**

**To Whom It May Concern**

This is to certified that this Industry Internship Report entitled “ **Data Pipeline - End to end workflow** ” is the bonafide work of **“ PARMEET KOUR**, **2020A1R080**, **of 5th Semester, Department of Computer Science and Engineering, Model Institute of Engineering and Technology (Autonomous), Jammu ”**, who carried out the Industry Internship at “ **WEBFREAK SOLUTIONS**, C-133, Plot No. - F-178, Industrial Area, Phase - 8B, Mohali, Punjab 160055 ” work under my mentorship during 18th,July 2022 to 26th,August 2022.

**Mrs. Vishalika Dr. Ashok Kumar**

**Assistant Professor Dean Academics**

**CSE, MIET HOD, CSE**

**MIET, Jammu**

*This is to certify that the above statement is correct to the best of my knowledge.*

**ACKNOWLEDGEMENT**

This Summer internship opportunity was a great chance for learning and professional development. I am grateful for having a chance to meet so many wonderful people and professionals who led me though this internship period.

It is my pleasant duty to pay my heartfelt gratitude to **MR. VARUN SHARMA**, CMO at WEBFREAK SOLUTIONS who have guided me through the course of this Internship.

I must record my deep sense of gratitude to **PROF. DR. ANKUR GUPTA** (Director, MIET) and **PROF. DR. ASHOK KUMAR** (Dean Academics & HOD CSE, MIET) for their guidance, constant inspiration and encouragement, and for their keen involvement throughout the course of present work.

Gratitude and thanks although mean a very small thing to convey my thanks to my parents who have always given me a parental source of love, motivation and strength right from the journey of my life.

Bearing in mind previous I am using this opportunity to express my deepest gratitude and special thanks to the teachers who in spite of being extraordinarily busy with their duties, took time out to hear, guide and keep me on the correct path and allowing me to carry out my project at their esteemed organization and extending during the training.

I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives. Hope to continue cooperation with all of you in the future.

I express my sincere gratitude to **WEBFREAK SOLUTIONS** and **MODEL INSTITUTE OF ENGINEERING AND TECHNOLOGY (Autonomous)**, Jammu for giving me the opportunity.

**IFTISAM TARIQ**

**2021A1L017**

**ABSTRACT \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

This report contains a detailed overview of my internship learning and project implementation through my learning during my internship training at Webfreak Solutions. During my Internship I have learned a lot about how is the work environment in the industry is and how it actually works, what are the parameters, how to work on an actual project, how to work in a flow of team work. I have known about the work flow of python developer’s roles and responsibilities. I have learned to work in a corporate space which not only enriched me professionally but also helped me grow personally as well. The career path I would be selecting for myself is quite influenced from my internship as I have had a great opportunity to practically see how the python development sector is working and evolving in the entire Globe. I have tried my level best to make it meaningful by reflecting what I learned during my training at the Company and how I implemented the project through my learning. Also, I have summarized every part of learning through different chapters in the report and in the last chapter project work is described. And also, through this industrial training learning technology like python programming was first criteria but also gained experience which can be used in the corporate sector for my growth and grabbing different opportunities.

The objective of a practical training is to learn something about industries practically and to be familiar with the working style of a technical worker to adjust simply according to the industrial environment.

**CONTENTS \_**

|  |  |  |
| --- | --- | --- |
| **1.** | **Python Introduction:** | **35 - 40** |
| 1.1 | Features |  |
| 1.2 | Application |  |

|  |  |  |
| --- | --- | --- |
| **2.** | **Python Functions:** | **41 - 47** |
| 2.1 | Create Class |  |
| 2.2 | Create Object |  |
| 2.3 | The init () Function |  |
| 2.4 | The str () Function |  |
| 2.5 | Object Methods |  |
| 2.6 | The Self Parameter |  |
| 2.7 | Modify Object Properties |  |
| 2.8 | Delete Object Properties |  |
| 2.9 | Delete Objects |  |
| 2.10 | The Pass Statement |  |
| 2.11 | Python Inheritance |  |

|  |  |  |
| --- | --- | --- |
| 2.12 | Create a Parent Class |  |
| 2.13 | Create a Child Class |  |
| 2.14 | Add the \_\_init\_\_() Function |  |
| 2.15 | Use the super() Function |  |
| 2.16 | Add Properties |  |
| 2.17 | Add Methods |  |
| **3.** | **Python Scope :** | **48 - 49** |
| 3.1 | Local Scope |  |
| 3.2 | Global Scope |  |
| 3.3 | Global Keyword |  |
| **4.** | **Python Modules:** | **50 - 51** |
| 4.1 | What is a Module? |  |
| 4.2 | Create a Module |  |
| 4.3 | Use a Module |  |
| 4.4 | Variable in Module |  |
| 4.5 | Naming a Module |  |
| 4.6 | Re-naming a Module |  |
| 4.7 | Built-in Modules |  |
| 4.8 | Using the dir () Function |  |
| 4,9 | Import From Module |  |
| **5.** | **Python Try Except:** |  |
| 5.1 | Exception Handling | **52 - 55** |
| 5.2 | Many Exceptions |  |
| 5.3 | Else |  |
| 5.4 | Finally |  |
| 5.5 | Raise an exception |  |
| **6.** | **Numpy:** | **56 - 59** |
| 6.1 | Introduction |  |

|  |  |  |
| --- | --- | --- |
| 6.2 | Shape and Reshape |  |
| 6.3 | Accessing Elements |  |
| 6.4 | Maths and Statistics |  |
| 6.5 | Advantages |  |
| 7. | **Data Exploration:** | **60 - 62** |
| 7.1 | Introduction and its concepts |  |
| 7.2 | Load Data & Identify Variables |  |
| 7.3 | Variables Analyst |  |
| **8.** | **Project - Data Pipeline:** | **63 - 76** |
| 8.1 | What id Data Pipeline? |  |
| 8.2 | Components of Data Pipeline |  |
| 8.3 | Benefits of Data Pipelining |  |
| 8.4 | Project - Introduction |  |
| 8.5 | Workflow |  |
| 8.6 | Procedure |  |
| 14.7 | Outputs |  |
| 14.8 | Conclusion |  |
| **.** | ***References*:** | **77** |

**Chapter 1**

**PYTHON INTRODUCTION**

**What is Python?**

[Python](https://www.w3schools.com/python/python_intro.asp) is a popular programming language. It was created by Guido van Rossum,

and released in 1991.

It is used for

* Web Development (server-side)
* Software Development
* Mathematics
* System Scripting

**What can Python do?**

* Python can be used on a server to create web applications
* Python can be used alongside software to create workflows
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-oriented way or a functional way.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

Python was designed for readability, and has some similarities to the English language with influence from mathematics. Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses. Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

**Python Application**

Dynamic Typing, Built-In [Data Structures](https://www.edureka.co/blog/data-structures-in-python/), [Powerful Libraries](https://www.edureka.co/blog/python-libraries/), [Frameworks](https://www.edureka.co/blog/python-frameworks/), Community Support are just some of the reasons which make Python an attractive language for rapidly developing any sort of application

**Web and Internet Development**

Python offers many choices for web development:

* Frame works such as [Django](http://www.djangoproject.com/) and [Pyramid](http://www.pylonsproject.org/).
* Micro-frameworks such as [Flask](http://flask.pocoo.org/) and [Bottle](http://bottlepy.org/).
* Advanced content management systems such as [Plone](http://www.plone.org/) and [django CMS](https://www.django-cms.org/).

Python's standard library supports many Internet protocols:

* [HTML and XML](http://docs.python.org/library/markup)
* [JSON](http://docs.python.org/library/json.html)
* [E-mail processing](http://docs.python.org/library/email).
* Support for [FTP](http://docs.python.org/library/ftplib.html), [IMAP](http://docs.python.org/2/library/imaplib.html), and other [Internet protocols](http://docs.python.org/library/internet).
* Easy-to-use [socket interface](http://docs.python.org/howto/sockets.html).
* And the Package Index has yet more libraries:
* [Requests](https://pypi.org/project/requests/), a powerful HTTP client library.
* [Beautiful Soup](http://www.crummy.com/software/BeautifulSoup/), an HTML parser that can handle all sorts of oddball HTML.
* [Feedparser](https://pypi.org/project/feedparser/) for parsing RSS/Atom feeds.
* [Paramiko](https://pypi.org/project/paramiko/), implementing the SSH2 protocol.
* [Twisted Python](http://twistedmatrix.com/), a framework for asynchronous network programming.

**Scientific and Numeric**

Python is widely used in [scientific and numeric](http://wiki.python.org/moin/NumericAndScientific) computing:

* [SciPy](http://scipy.org/) is a collection of packages for mathematics, science, and engineering.
* [Pandas](http://pandas.pydata.org/) is a data analysis and modeling library.
* [IPython](http://ipython.org/) is a powerful interactive shell that features easy editing and recording of a work session, and supports visualizations and parallel computing.
* The [Software Carpentry Course](http://software-carpentry.org/) teaches basic skills for scientific computing, running bootcamps and providing open-access teaching materials.

**Education**

* Python is a superb language for teaching programming, both at the introductory level and in more advanced courses.
* Books such as [How to Think Like a Computer Scientist](http://www.openbookproject.net/thinkcs/python/english2e/), [Python Programming: An Introduction to Computer Science](http://mcsp.wartburg.edu/zelle/python/), and [Practical Programming](http://pragprog.com/book/gwpy2/practical-programming).
* The [Education Special Interest Group](https://www.python.org/community/sigs/current/edu-sig) is a good place to discuss teaching issues.

## **Features of Python**

As a programming language, the features of Python brought to the table are many. Some of the most significant features of Python are:

### Features of Python - Edureka

**Easy to Code**

Python is a very developer-friendly language which means that anyone and everyone can learn to code it in a couple of hours or days. As compared to other object-oriented programming languages like Java, C, C++, and C#, Python is one of the easiest to learn.

**Open Source and Free**

Python is an open-source programming language which means that anyone can create and contribute to its development. Python has an online forum where thousands of coders gather daily to improve this language further. Along with this Python is free to download and use in any operating system, be it Windows, Mac or Linux.

**Support for GUI**

GUI or Graphical User Interface is one of the key aspects of any programming language because it has the ability to add flair to code and make the results more visual. Python has support for a wide array of GUIs which can easily be imported to the interpreter, thus making this one of the most favorite languages for developers.

**Object-Oriented Approach**

One of the key aspects of Python is its object-oriented approach. This basically means that Python recognizes the concept of class and object encapsulation thus allowing programs to be efficient in the long run.

**High-Level Language**

Python has been designed to be a high-level programming language, which means that when you code in Python you don’t need to be aware of the coding structure, architecture as well as memory management

**Chapter 2**

**Python Functions**

**2.1 Creating a Function -**

In Python a function is defined using the def keyword.

**Example:**

def my\_ function():

print("Hello from a function")

**2.2 Calling a Function -**

To call a function, use the function name followed by parenthesis:

**Example:**

def my\_function():

print("Hello from a function")

my\_function()

**2.3 Arguments -**

Information can be passed into functions as arguments. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

**Example:**

def my\_function(**fname**):

print(fname + " Refsnes")

my\_function(**"Emil"**)

my\_function(**"Tobias"**)

my\_function(**"Linus"**)

Arguments are often shortened to *args* in Python documentations.

**2.4 Parameters or Arguments?**

The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.

From a function's perspective: -

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it is called.

**2.5 Number of Arguments -**

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more and not less.

**Example:**

This function expects 2 arguments, and gets 2 arguments.

def my\_function(fname, lname):

print(fname + " " + lname)

my\_function("Emil", "Refsnes")

If you try to call the function with 1 or 3 arguments, you will get an error.

**Example:**

This function expects 2 arguments, but gets only 1.

def my\_function(fname, lname):

print(fname + " " + lname)

my\_function("Emil")

**2.6 Arbitrary Arguments, \*args -**

If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.

This way the function will receive a *tuple* of arguments and can access the items accordingly:

**Example:**

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

def my\_function(\*kids):

print("The youngest child is " + kids[2])

my\_function("Emil", "Tobias", "Linus")

*Arbitrary Arguments* are often shortened to *\*args* in Python documentations.

**2.7 Keyword Arguments -**

You can also send arguments with the *key* = *value* syntax. This way the order of the arguments does not matter.

**Example:**

def my\_function(child3, child2, child1):

print("The youngest child is " + child3)

my\_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")

The phrase *Keyword Arguments* are often shortened to *kwargs* in Python documentations.

**2.8 Arbitrary Keyword Arguments, \*\*kwargs -**

If you do not know how many keyword arguments that will be passed into your function, add two asterisks: \*\* before the parameter name in the function definition.

This way the function will receive a *dictionary* of arguments, and can access the items accordingly:

**Example:**

If the number of keyword arguments is unknown, add a double \*\* before the parameter name.

def my\_function(\*\*kid):

print("His last name is " + kid["lname"])

my\_function(fname = "Tobias", lname = "Refsnes")

*Arbitrary Keyword Arguments* are often shortened to *\*\*kwargs* in Python documentations.

**2.9 Default Parameter Value -**

The following example shows how to use a default parameter value. If we call the function without argument, it uses the default value.

**Example:**

def my\_function(**country = "Norway"**):

print("I am from " + country)

my\_function("Sweden")

my\_function("India")

my\_function()

my\_function("Brazil")

**2.10 Passing a List as an Argument -**

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g., if you send a List as an argument, it will still be a List when it reaches the function:

**Example:**

def my\_function(food):

for x in food:

print(x)

fruits = ["apple", "banana", "cherry"]

my\_function(fruits)

**2.11 Return Values -**

To let a function return a value, use the return statement:

**Example:**

def my\_function(x):

**return 5 \* x**

print(my\_function(3))

print(my\_function(5))

print(my\_function(9))

**2.12 The pass Statement -**

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

**Example:**

def myfunction():

pass

**2.13 Recursion -**

Python also accepts function recursion, which means a defined function can call itself. Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result. The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power.

However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

In this example, tri\_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e., when it is 0).

To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

**Example:**

def tri\_recursion(k):

if(k > 0):

result = k + tri\_recursion(k - 1)

print(result)

else:

result = 0

return result

print("\n\nRecursion Example Results")

**2.14 Python Lambda -**

A lambda function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression.

**Syntax:**

lambda *arguments*: *expression*

The expression is executed and the result is returned:

**Example:**

Add 10 to argument a, and return the result.

x = lambda a: a + 10

print(x(5))

Lambda functions can take any number of arguments.

**Example:**

Multiply argument a with argument b and return the result.

x = lambda a, b: a \* b

print(x(5, 6))

**Example:**

Summarize argument a, b, and c and return the result.

x = lambda a, b, c: a + b + c

print(x(5, 6, 2))

**2.15 Why Use Lambda Functions?**

The power of lambda is better shown when you use them as an anonymous function inside another function. Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number.

def myfunc(n):

return lambda a: a \* n

Use that function definition to make a function that always doubles the number you send in:

**Example:**

def myfunc(n):

return lambda a: a \* n

mydoubler = myfunc(2)

print(mydoubler(11))

Or, use the same function definition to make a function that always *triples* the number you send in.

**Example:**

def myfunc(n):

return lambda a: a \* n

mytripler = myfunc(3)

print(mytripler(11))

Or, use the same function definition to make both functions, in the same program.

**Example:**

def myfunc(n):

return lambda a: a \* n

mydoubler = myfunc(2)

mytripler = myfunc(3)

print(mydoubler(11))

print(mytripler(11))

**Chapter 3**

**OOPs**

Python is an object-oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects.

**3.1 Create a Class -**

To create a class, use the keyword class.

**Example:**

Create a class named MyClass, with a property named x.

class MyClass:

x = 5

3**.2 Create Object -**

Now we can use the class named MyClass to create objects.

**Example:**

Create an object named p1, and print the value of x.

p1 = MyClass()

print(p1.x)

**3.3 The \_\_init\_\_() Function -**

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in \_\_init\_\_() function.

All classes have a function called \_\_init\_\_(), which is always executed when the class is being initiated.

Use the \_\_init\_\_() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

**Example:**

Create a class named Person, use the \_\_init\_\_() function to assign values for name and age:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

p1 = Person("John", 36)

print(p1.name)

print(p1.age)

**Note:** The \_\_init\_\_() function is called automatically every time the class is being used to create a new object.

**3.4 The \_\_str\_\_() Function -**

The \_\_str\_\_() function controls what should be returned when the class object is represented as a string. If the \_\_str\_\_() function is not set, the string representation of the object is returned.

**Example:**

The string representation of an object WITHOUT the \_\_str\_\_() function.

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

p1 = Person("John", 36)

print(p1)

**Example:**

The string representation of an object WITH the \_\_str\_\_() function:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def \_\_str\_\_(self):

return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)

**3.5 Object Methods -**

Objects can also contain methods. Methods in objects are functions that belong to the object. Let us create a method in the Person class:

**Example:**

Insert a function that prints a greeting, and execute it on the p1 object.

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def myfunc(self):

print("Hello my name is " + self.name)

p1 = Person("John", 36)

p1.myfunc()

**Note:** The self-parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

**3.6 The Self-Parameter -**

The self-parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named self, you can call it whatever you like, but it has to be the first parameter of any function in the class:

**Example:**

Use the words *mysillyobject* and *abc* instead of *self*:

class Person:

def \_\_init\_\_(mysillyobject, name, age):

mysillyobject.name = name

mysillyobject.age = age

def myfunc(abc):

print("Hello my name is " + abc.name)

p1 = Person("John", 36)

p1.myfunc()

**3.7 Modify Object Properties -**

You can modify properties on objects like this:

Example:

Set the age of p1 to 40.

p1.age = 40

**3.8 Delete Object Properties -**

You can delete properties on objects by using the del keyword:

**Example:**

Delete the age property from the p1 object.

del p1.age

**3.9 Delete Object -**

You can delete objects by using the del keyword.

**Example:**

Delete the p1 object.

del p1

**3.10 The pass Statement -**

Class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.

**Example:**

class Person:

pass

**3.11 Python Inheritance -**

Inheritance allows us to define a class that inherits all the methods and properties from another class. Parent class is the class being inherited from, also called base class. Child class is the class that inherits from another class, also called derived class.

**3.12 Create a Parent Class -**

Any class can be a parent class, so the syntax is the same as creating any other class.

**Example:**

Create a class named Person, with firstname and lastname properties, and a printname method.

class Person:

def \_\_init\_\_(self, fname, lname):

self.firstname = fname

self.lastname = lname

def printname(self):

print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method.

x = Person("John", "Doe")

x.printname()

**3.13 Create a Child Class -**

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class.

**Example:**

Create a class named Student, which will inherit the properties and methods from the Person class.

class Student(Person):

pass

**Note:** Use the pass keyword when you do not want to add any other properties or methods to the class.

Now the Student class has the same properties and methods as the Person class.

**Example:**

Use the Student class to create an object, and then execute the printname method.

x = Student("Mike", "Olsen")

x.printname()

**3.14 Add the \_\_init\_\_() Function -**

So far we have created a child class that inherits the properties and methods from its parent.

We want to add the \_\_init\_\_() function to the child class (instead of the pass keyword).

**Note:** The \_\_init\_\_() function is called automatically every time the class is being used to create a new object.

**Example:**

Add the \_\_init\_\_() function to the Student class.

class Student(Person):

def \_\_init\_\_(self, fname, lname):

#add properties etc.

When you add the \_\_init\_\_() function, the child class will no longer inherit the parent's \_\_init\_\_() function.

**Note:** The child's \_\_init\_\_() function **overrides** the inheritance of the parent's \_\_init\_\_() function.

To keep the inheritance of the parent's \_\_init\_\_() function, add a call to the parent's \_\_init\_\_() function:

**Example:**

class Student(Person):

def \_\_init\_\_(self, fname, lname):

Person.\_\_init\_\_(self, fname, lname)

Now we have successfully added the \_\_init\_\_() function, and kept the inheritance of the parent class, and we are ready to add functionality in the \_\_init\_\_() function.

**3.15 Use the super() Function -**

Python also has a super() function that will make the child class inherit all the methods and properties from its parent.

**Example:**

class Student(Person):

def \_\_init\_\_(self, fname, lname):

super().\_\_init\_\_(fname, lname)

By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

**3.16 Add Properties -**

**Example:**

Add a property called graduationyear to the Student class.

class Student(Person):

def \_\_init\_\_(self, fname, lname):

super().\_\_init\_\_(fname, lname)

self.graduationyear = 2019

In the example below, the year 2019 should be a variable, and passed into the Student class when creating student objects.

To do so, add another parameter in the \_\_init\_\_() function.

**Example:**

Add a year parameter, and pass the correct year when creating objects.

class Student(Person):

def \_\_init\_\_(self, fname, lname, year):

super().\_\_init\_\_(fname, lname)

self.graduationyear = year

x = Student("Mike", "Olsen", 2019)

**3.17 Add Methods -**

**Example:**

Add a method called welcome to the Student class.

class Student(Person):

def \_\_init\_\_(self, fname, lname, year):

super().\_\_init\_\_(fname, lname)

self.graduationyear = year

def welcome(self):

print("Welcome", self.firstname, self.lastname, "to the class of",

self.graduationyear)

**Chapter 4**

**‘Python Scope’**

A variable is only available from inside the region it is created. This is called **scope**.

**4.1 Local Scope -**

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

**Example:**

A variable created inside a function is available inside that function.

def myfunc():

x = 300

print(x)

myfunc()

Function Inside Function:-

As explained in the example above, the variable x is not available outside the function, but it is available for any function inside the function:

**Example**:

The local variable can be accessed from a function within the function.

def myfunc():

x = 300

def myinnerfunc():

print(x)

myinnerfunc()

myfunc()

**4.2 Global Scope -**

A variable created in the main body of the Python code is a global variable and belongs to the global scope. Global variables are available from within any scope, global and local.

**Example:**

A variable created outside of a function is global and can be used by anyone.

x = 300

def myfunc():

print(x)

myfunc()

print(x)

Naming Variables:-

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function).

**Example:**

The function will print the local x, and then the code will print the global x.

x = 300

def myfunc():

x = 200

print(x)

myfunc()

print(x)

**4.3 Global Keyword -**

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword. The global keyword makes the variable global.

**Example:**

If you use the global keyword, the variable belongs to the global scope.

def myfunc():

global x

x = 300

myfunc()

print(x)

Also, use the global keyword if you want to make a change to a global variable inside a function.

**Example:**

To change the value of a global variable inside a function, refer to the variable by using the global keyword.

x = 300

def myfunc():

global x

x = 200

myfunc()

print(x)

**Chapter - 4**

**‘Python Modules’**

**4.1 What is a Module?**

Consider a module to be the same as a code library. A file containing a set of functions you want to include in your application.

**4.2 Create a Module -**

To create a module just save the code you want in a file with the file extension .py .

**Example:**

Save this code in a file named mymodule.py

def greeting(name):

print("Hello, " + name)

**4.3 Use a Module -**

Now we can use the module we just created, by using the import statement.

**Example:**

Import the module named mymodule, and call the greeting function. import mymodule

mymodule.greeting("Jonathan")

**Note:** When using a function from a module, use the syntax: *module\_name.function\_name*.

**4.4 Variables in Module -**

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc.).

**Example:**

Save this code in the file mymodule.py

person1 = {

"name": "John",

"age": 36,

"country": "Norway"

}

**Example:**

Import the module named mymodule, and access the person1 dictionary.

import mymodule

a = mymodule.person1["age"]

print(a)

**4.5 Naming a Module -**

You can name the module file whatever you like, but it must have the file extension .py

**4.6 Re-naming a Module -**

You can create an alias when you import a module, by using the as keyword:

**Example:**

Create an alias for mymodule called mx.

import mymodule as mx

a = mx.person1["age"]

print(a)

**4.7 Built-in Modules -**

There are several built-in modules in Python, which you can import whenever you like.

**Example:**

Import and use the platform module.

import platform

x = platform.system()

print(x)

**4.8 Using the dir() Function -**

There is a built-in function to list all the function names (or variable names) in a module. The dir() function:-

**Example:**

List all the defined names belonging to the platform module.

import platform

x = dir(platform)

print(x)

**Note:** The dir() function can be used on *all* modules, also the ones you create yourself.

**4.9 Import From Module -**

You can choose to import only parts from a module, by using the from keyword.

**Example:**

The module named mymodule has one function and one dictionary.

def greeting(name):

print("Hello, " + name)

person1 = {

"name": "John",

"age": 36,

"country": "Norway"

}

**Example:**

Import only the person1 dictionary from the module.

from mymodule import person1

print (person1["age"])

**Chapter - 5**

**Python Try**

The try block lets you test a block of code for errors. The except block lets you handle the error. The else block lets you execute code when there is no error. The finally block lets you execute code, regardless of the result of the try- and except blocks.

**5.1 Exception Handling -**

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

**Example:**

The try block will generate an exception, because x is not defined.

try:

print(x)

except:

print("An exception occurred")

Since the try block raises an error, the except block will be executed. Without the try block, the program will crash and raise an error:

**Example:**

This statement will raise an error, because x is not defined.

print(x)

**5.2 Many Exceptions -**

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error.

**Example:**

Print one message if the try block raises a NameError and another for other errors:

try:

print(x)

except NameError:

print("Variable x is not defined")

except:

print("Something else went wrong")

**5.3 Else -**

You can use the else keyword to define a block of code to be executed if no errors were raised.

**Example:**

In this example, the try block does not generate any error.

try:

print("Hello")

except:

print("Something went wrong")

else:

print("Nothing went wrong")

**5.4 Finally -**

The finally block, if specified, will be executed regardless if the try block raises an error or not.

**Example:**

try:

print(x)

except:

print("Something went wrong")

finally:

print("The 'try except' is finished")

This can be useful to close objects and clean up resources.

**Example:**

Try to open and write to a file that is not writable.

try:

f = open("demofile.txt")

try:

f.write("Lorum Ipsum")

except:

print("Something went wrong when writing to the file")

finally:

f.close()

except:

print("Something went wrong when opening the file")

The program can continue, without leaving the file object open.

**5.5 Raise an exception -**

As a Python developer you can choose to throw an exception if a condition occurs. To throw (or raise) an exception, use the raise keyword.

**Example:**

Raise an error and stop the program if x is lower than 0.

x = -1

if x < 0:

raise Exception("Sorry, no numbers below zero")

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

**Example:**

Raise a TypeError if x is not an integer:

x = "hello"

if not type(x) is int:

raise TypeError("Only integers are allowed")

**Chapter-6**

**NUMPY**

**6.1 Introduction -**

The data in Numpy arrays is of homogeneous type, meaning all the data in an array is of the same type, while lists are just pointers to objects, even though all the data is of the same type. As a consequence, the Numpy arrays use much less memory than regular lists. Also, most of the Numpy operations is implemented in the C language, meaning, the cost of Python loops and dynamic checking of the data type is avoided. This, yields a significant increase in processing speed when comparing Numpy to a Python list.

More often than one we encounter large datasets with tens of thousands of rows of data, just think of hourly air temperature measurements for a county or region since the measurements beginning in this region. If your weather service is measuring hourly air temperature for last 50 years, that’s more that 400 000 rows of data, just for one station.

**How to install Numpy?**

Well, if using Anaconda, Numpy is preinstalled in the base environment. However, more often than not, it’s good practice to create new environments for new projects. To install Numpy, we run an Anaconda prompt and type:-

*conda install numpy*

Or

*conda install -c anaconda numpy*

If pip is being used, Numpy can be installed by typing:-

*pip install numpy*

**How to import Numpy?**

When importing certain libraries, including Numpy, we follow a convention, basically this means we use well established abbreviations for libraries. In the case of Numpy we use “**np**”.

*import numpy as np*

The goal is that our code is reproducible, and every Python programmer in the World, knows what the following line does.

*a = np.array([3,4])*

If you have imported Numpy, and used the above command, you have successfully created your first **Numpy array**. Let’s see what happens if we print it out. Print gives us something that looks like a list, but it’s not.

When we check the type we see that’s a “**numpy.ndarray**” (*n dimensional array*).

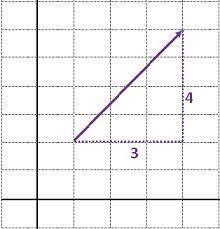
**Fig.3.1 :** .

## 

## **Vectors?**

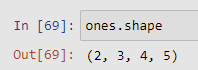
## In the example we saw how we can create an 1-dimensional array. If you remember vectors from Math, well a 1-dimensional Numpy array is basically a vector. Since we gave two numbers, 3 and 4, this vector lies in the 2-dimensional space (geometric plane). It’s same as in math when you had a vector:-

## v= 3i + 4j

**Fig.3.2** : 

<https://en.wikiversity.org/>

**6.2 Shape and Reshape -**

The shape attribute comes in handy. Since we have 4 dimension, we get a tuple of 4 numbers. **Fig.3.3 :** 

To count the number of elements in the whole array we use the size attribute.

**Fig.3.4 :** 

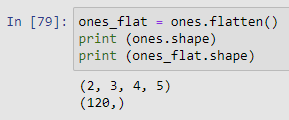
In order to change the shape of an array, we use the .reshape() method. Care has to be taken though, the newly reshaped array has to be of same size as the old one.

**Fig.3.5 :** 

The original zeros array had the shape of (2, 3), and we can reshape it into (3,2), (6, 1) or (1,6), since it has a size of 6 elements. I shall mention, that in case of reshaping it to (6, 1) or (1, 6) we change the number of dimensions, from a 2-D array, to a 1-D array, but as long we take care of the array size, we are on the safe side.

A handy “shortcut” to a 1-D array are the flatten() and ravel() methods. The difference is that flatten creates a 1-D copy of the original array, while ravel creates a reference to the original array. So, using ravel() has the consequence that changing for example some of the data in the newly created array while also change the data in the original array.

The usage depends on the specific task, most of the time I’ve used the flatten() method.

**Fig.3.6 :** 

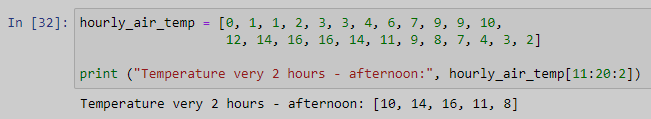
Last but not least, let’s not forget the transpose() method. This method simply swaps the rows and columns of an array.

**.3 Accessing elements -**

Until now, we saw how to create, find proportions and reshape or flatten an array. Let’s turn our focus now on data extraction from an array using indexing and slicing. To slice an array means to access it’s elements by providing the desired elements index.

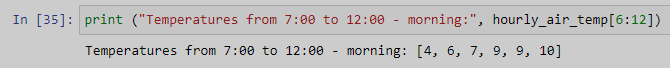
The default syntax of slicing involves the array name and square brackets, like for Python lists, as follows:-

*array\_name[start\_index : end\_index : step\_size]*

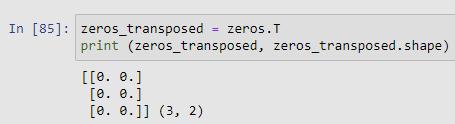
**Fig.3.7 :** **

In our array the temperature measurements start from 1:00. To clarify the things, we printed our hourly temperatures at 12:00, 14:00, 16:00, 18:00 and 20:00.

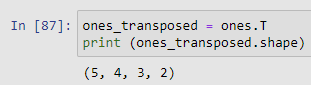
If we don’t define the step size, every element in the specified range get returned. For example if we need the hourly temperatures from 7:00 to 12:00.

**Fig.3.8 :** 

As usual in Lists, so in Numpy also, the start\_index is including, while the end\_index is not including. Also, the first index in an array is always zero. So to access the temperature at 7:00, we input 6-th index. And since we need the measurements until 12 (which is at index no. 11), we provided 12th index (it’s excluding).

**Fig.3.9 :** 

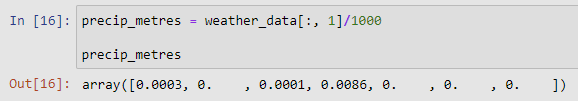
In this case, the result is same as before with reshape. In case of a multidimensional array, all dimensions get swapped, let’s see.

**Fig.3.10 :** 

**6.2 Maths and Statistics -**

It’s the simplicity and speed advantage when dealing with a great amount of numerical data. Let’s first take a look to mathematical operations. An example with division:-

Our weather data array contains precipitation data in millimetres, lets convert those to metres. To convert millimetres to metres, we need to divide the value by 1000.

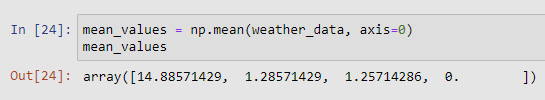
**Fig.3.11 :** 

Again we use slicing, to select the second column of the array, and divide the values by 1000. For practice, try to replace precipitation values from the array weather\_f with the ones converted to metres. (you can do the change on the weather\_f array directly)

As for the statistics example, I shall use the most common case, we need to calculate the average temperature, precipitation, wind speed and snow depth values for the week. The Numpy function to calculate the average values is called np.mean(). The basic syntax of np. mean() function is as follows:

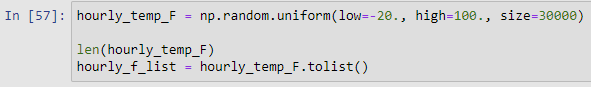
*np. mean(a, axis=None, dtype=None, out=None, keepdims=<no value>, \*, where=<no value>)*

For our case, the important part is the axis. Since our goal is to calculate the mean values for each column, we need to set the axis parameter to 0. Setting the axis to 1, would yield the result row-wise.

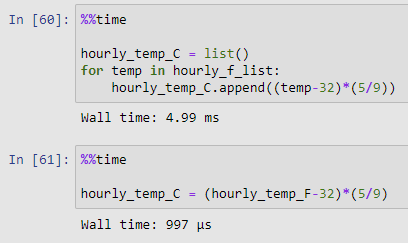
**Fig.3.12 :** 

**6.3 Advantages of Numpy -**

Numpy also has some speed advantages, this probably got you tempted. We will create an random array of floats, let’s say its hourly air temperature is measured at some location in th US. The length of the array is 30 000.

**Fig.3.13 :** 

We want to convert those numbers to Celsius degrees. Let’s measure the time needed using a for loop and Python list, and then using Numpy.

**Fig.3.14 :** 

So, the time taken using an loop over a Python list took around 5 ms, while using an Numpy array the same operation took less than 1 ms. So Numpy is in this specific case (task) around 5 times faster.

Also, please consider that this test is not completely applicable, it really depends on the speed of your computer (mostly CPU) and the chosen task. Since this is not the main topic of the article, I’ll leave it to you to check other articles that are covering the speed benefits of Numpy, and hopefully, try it out yourself, on your specific task with your data.

**Chapter - 7**

**‘Data Exploration’**

**-7.1 Introduction**

Data exploration is a key aspect of data analysis and model building. Without spending significant time on understanding the data and its patterns one cannot expect to build efficient predictive models. Data exploration takes major chunk of time in a data science project comprising of data cleaning and preprocessing.

The key steps involved in data exploration are:-

* Load data
* Identify variables
* Variable analysis
* Handling missing values
* Handling outliers
* Feature engineering

**7.2 Load data and Identify variables** -

Variable can be of different types such as character, numeric, categorical, and continuous.

Python codes to understand the types of variables in your dataset.

**#Import required libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

**#Load the data**

titan=pd.read\_csv("../input/titan.csv")

**#get an overview of the data**

titan.head()

titan.tail()

titan.sample(10)

**#identify variable type**

titan.dtypes

titan.info()

titan.describe()

**7.3 Variable Analysis -**

Variable analysis can be done in three ways, univariate analysis, bivariate analysis, and multivariate analysis.

**Univariate analysis** is used to highlight missing and outlier values. Here each variable is analyzed on its own for range and distribution. **Univariate analysis differs for categorical and continuous variables.** For categorical variables, you can use frequency table to understand distribution of each category. For continuous variables, you have to understand the central tendency and spread of the variable. It can be measured using mean, median, mode, etc. It can be visualized using box plot or histogram.

**#Understand various summary statistics of the data**

include =['object', 'float', 'int']

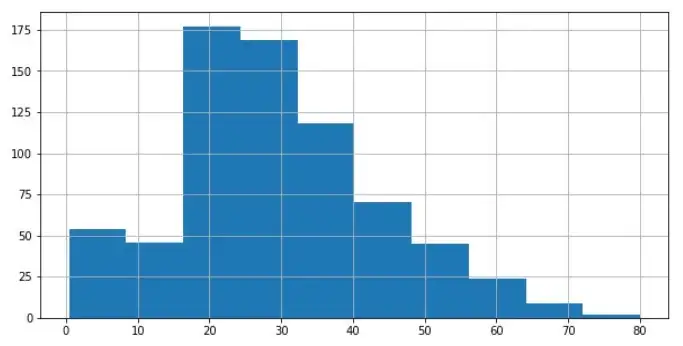
titan.describe(include=include)

titan.describe()

**#Get count of values in a categorical variable**

titan.survived.value\_counts()

titan.age.hist(figsize=(10,5))

**Fig 4.1 :** 

**Bivariate Analysis** is used to find the relationship between two variables. Analysis can be performed for combination of categorical and continuous variables. **Scatter plot is suitable for analyzing two continuous variables. Bar charts helps to understand relation between two categorical variables.** Certain statistical tests are also used to effectively understand bivariate relationship.

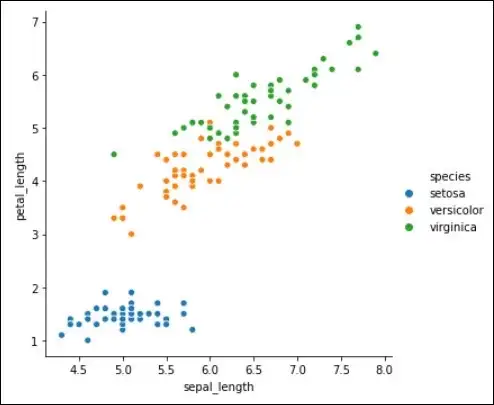
**iris = sns.load\_dataset("iris")**

sns.relplot(x = 'sepal\_length', y = 'petal\_length', hue='species',data = iris)

relplot = sns.catplot(x="pclass", hue="who", col="survived",data="titan", kind="count",

height=4, aspect=.7);

relplot()

**Fig.4.2 :**  

**Chapter – 8 ‘Project: Data Pipeline’**

**8.1 What is Data Pipeline?**

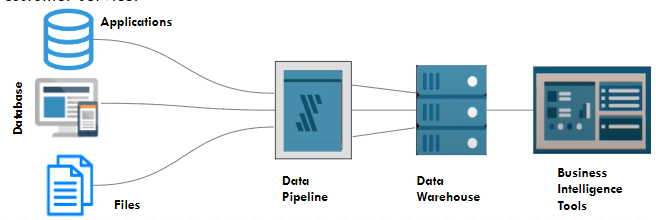
A data pipeline is a method in which raw data is ingested from various data sources and then ported to data store, like a data lake or data warehouse, for analysis. Before data flows into a data repository, it usually undergoes some data processing. This is inclusive of data transformations, such as filtering, masking, and aggregations, which ensure appropriate data integration and standardization. This type of data repository has a defined schema which requires alignment—i.e. matching data columns and types—to update existing data with new data.

As the name suggests, data pipelines act as the “piping” for data science projects or business intelligence dashboards. Data can be sourced through a wide variety of places—APIs, SQL and [NoSQL databases](https://www.ibm.com/cloud/learn/nosql-databases), files, et cetera, but unfortunately, that data usually isn’t ready for immediate use. The type of data processing that a data pipeline requires is usually determined through a mix of exploratory data analysis and defined business requirements. Once the data has been appropriately filtered, merged, and summarized, it can then be stored and surfaced for use. Well-organized data pipelines provide the foundation for a range of data projects; this can include exploratory data analyses, data visualizations, and machine learning tasks.

Data is growing at a rapid rate and will continue to grow. The data pipelines are widely used in ingesting data that is used for transforming all the raw data efficiently to optimize the data continuously generated daily.

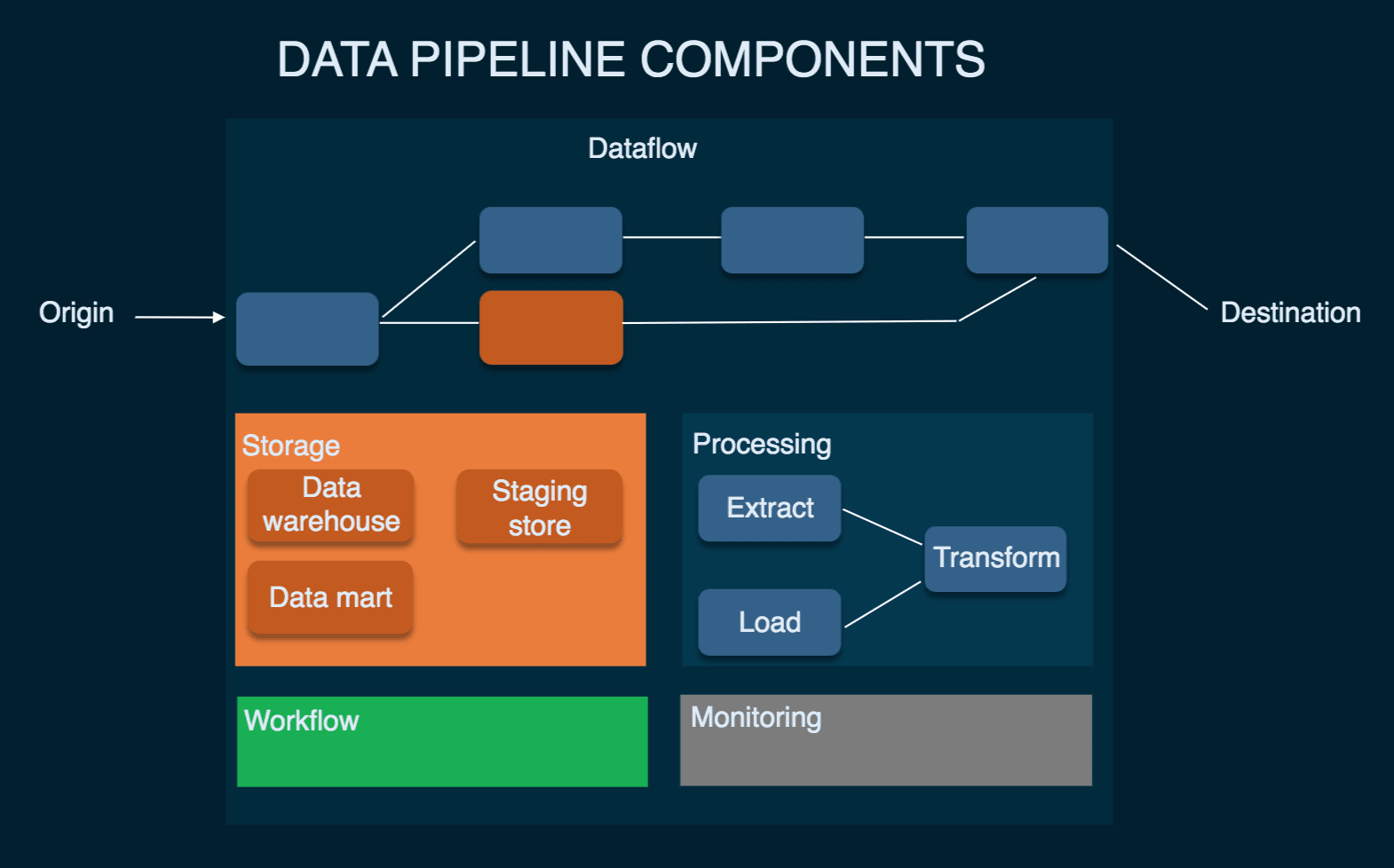
This transformed data can be used for Data Analytics, Machine Learning, and applications. Some of the use cases of what is Data Pipeline are listed below:

* Delivering the Sales and Marketing data to CRM platforms to enhance customer service.
* Streaming the data from sensors to the applications for monitoring the performance and status.
* Unifying the data together so that it can speed up the development of new products.

**Fig.5.1 :** 

## 

## **14.2 Components of a Data Pipeline -**

**Fig.5.2** : 

The components of a Pipeline are as follows :-

> **Origin:** Origin is the point of entry for data from all data sources in the pipeline. Most pipelines have transactional processing applications, application APIs, IoT device sensors, etc., or storage systems such as Data Warehouses, Data Lakes, etc. as their origin.

> **Destination:** This is the final point to which data is transferred. The final destination depends on the use case. The destination is a [Data Warehouse](https://hevodata.com/learn/data-warehouse/), Data Lake, or Data Analysis and Business Intelligence tool for most use cases.

> **Dataflow:** This refers to the movement of data from origin to destination, along with the transformations that are performed on it. One of the most widely used approaches to data flow is called ETL (Extract, Transform, Load). The three phases in ETL are as follows:

> **Load:** Loading can be defined as the process of storing the transformed data in the destination of choice, normally a Data Warehouse such as [Amazon Redshift](https://hevodata.com/integrations/destination/redshift), [Google BigQuery](https://hevodata.com/integrations/destination/bigquery), [Snowflake](https://hevodata.com/integrations/destination/snowflake), etc.

> **Storage:** Storage refers to all systems that are leveraged to preserve data at different stages as it progresses through the pipeline.

> **Processing:** Processing includes all activities and steps for ingesting data from sources, storing it, transforming, and loading it into the destination. While data processing is associated with the data flow, the focus in this step is on the implementation of the data flow.

> **Workflow:** Workflow defines a sequence of processes along with their dependency on each other in the Pipeline.

> **Monitoring:** The goal of monitoring is to ensure that the Pipeline and all its stages are working correctly and performing the required operations.

> **Technology:** These are the infrastructure and tools behind Data Flow, Processing, Storage, Workflow, and Monitoring. Some of the tools and technologies that can help build efficient Pipelines are as follows:-

i) ETL tools

ii) Data Warehouses

iii) Data Lakes

iv) Batch Workflow Schedulers

v) Streaming Data Processing Tools

vi) Programming Languages: These are used to define pipeline processes as code. Python and Java are widely used to create Pipelines.

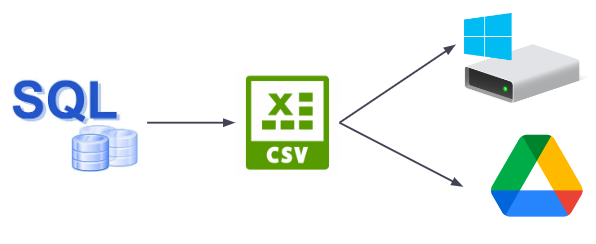
**8.3 Benefits of a Data Pipeline -**

When companies don’t know what is Data Pipeline, they used to manage their data in an unstructured and unreliable way. But as they came to know about What is Data Pipeline and how it helps companies save time and keep their data organized always. A few benefits of Pipeline are listed below:-

* Data Quality
* Incremental Build
* Replicable Patterns

**8.4 Project Introduction -**

* **Picking up the data on daily basis and the need to store it on different layers**. So, we have created **python script** that uses the **pipelining** principle to load data from **MySQL** and save it on the **local drive** as **CSV** file and same as to **Google drive.**

**Fig.5.3 -**

The main libraries that we have used in our project are :-

1. **Shutil -**

Syntax : import shutil

(Used for transferring files)

2. **Time -**

Syntax : import time

(To work with all time related functions)

3. **Os -**

Syntax : import os

(Provides functions for interacting with the operating system)

4**. Traceback -**

Syntax : import traceback

(Provides key to resolve unhandled exceptions that occur during the execution of a python program)

5. **Logging -**

Syntax : import logging

(Used to track when the program is running)

6. **PyMySQL -**

Syntax : import pymysql

(Provides an interface for connecting to MySQL database server from python)

7. **CSV -**

Syntax : import csv

(Used for parsing tabular like data structure such as data in excel format & these files are saved in csv extension)

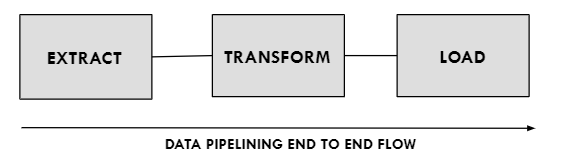
8. **google-auth -**

Syntax : import GoogleAuth

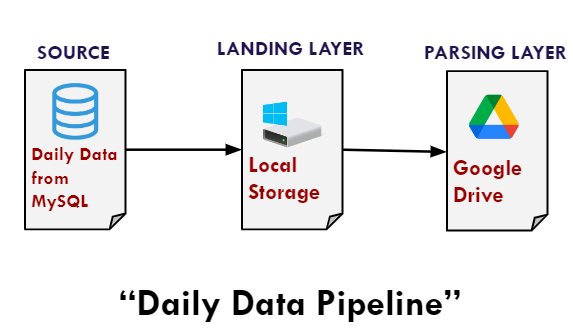
(Provides the ability to authenticate Google APIs using various methods)

Few features are :-

* Complete pipeline solution for ETL ( Extract, Transform & Load ) process.
* Real time problem statement.
* Allows you to handle and analyze vast quantities of data in much less time

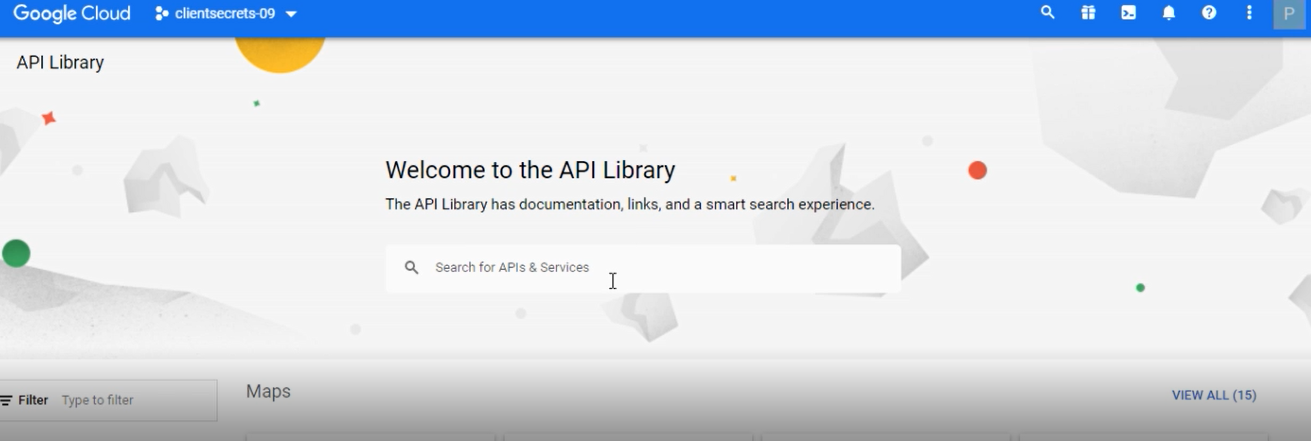
**Fig.5.4 :** 

**8.5 Workflow of Data pipeline project -**

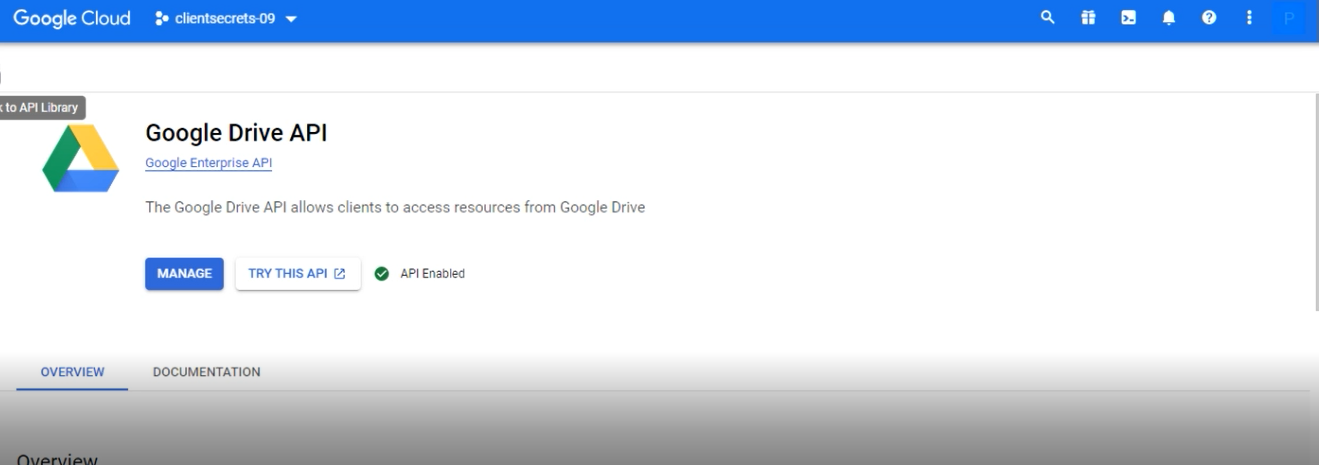
**Fig.5.5 : **

**8.6 Procedure -**

We have to work on the google cloud console so that we can create credentials for our project.

**Fig.5.6 :** 

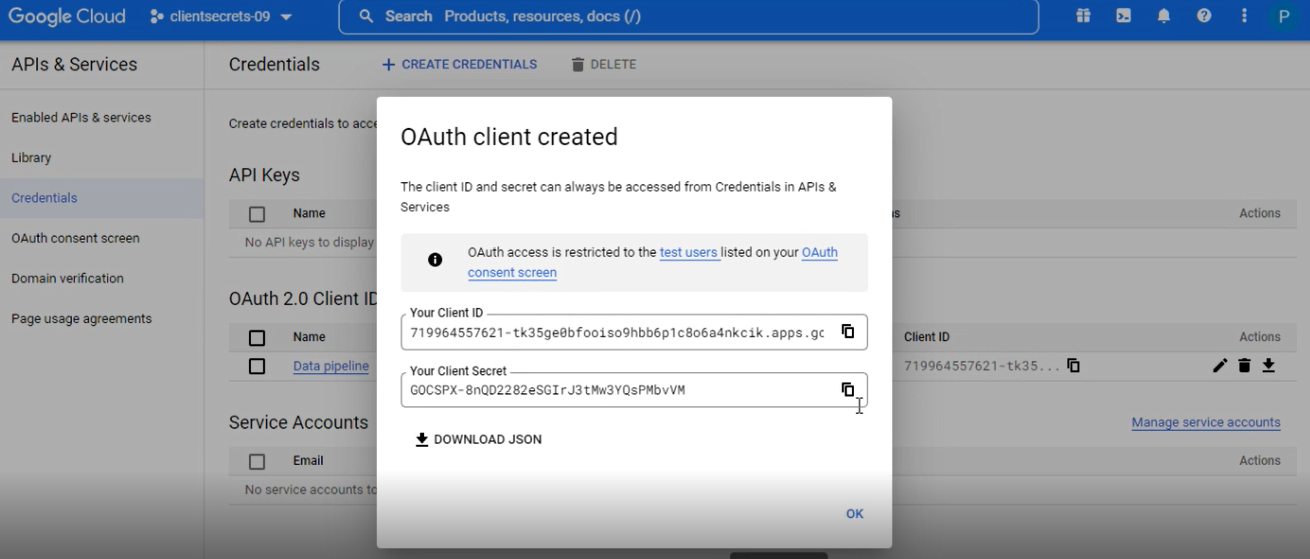
Then, we will download Google Drive API for Download and upload files to Drive.

**Fig.5.7 :** 

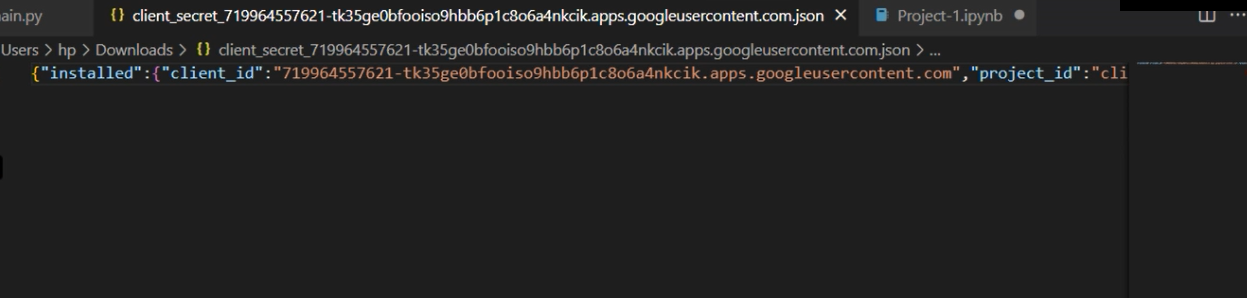
We use this for:-

* Search for files and folders.
* Create complex search queries that return any of the file metadata fields in the Files resource.
* Allow users to share files, folders, and drives to cooperate on content.
* Combine with the Picker API to search all files in Google Drive, and return the file name, URL, last modified date, and user.
* Create shortcuts, and external links to data kept outside of Drive, in a different data store or cloud storage system.
* Form a dedicated folder to store your application’s data to prevent the app from accessing all the user’s content.
* Integrate with the Drive UI, which can interact with Drive files.

After that, We will create OAuth client ID.

**Fig.5.7 :** 

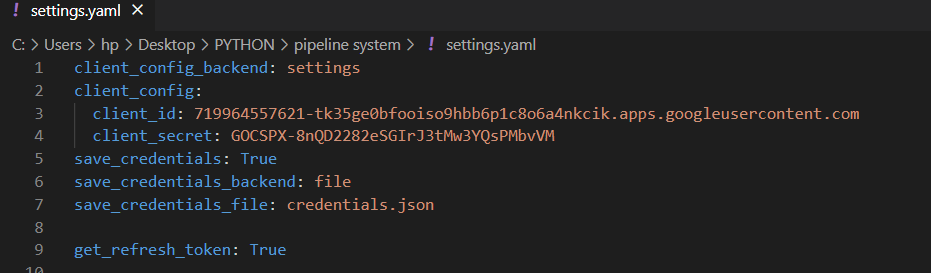
After that, we will get a JSON file.

**Fig.5.8 :** 

**JSON** stands for **J**ava**S**cript **O**bject **N**otation. It is a format for structuring data. This format is used by different web applications to communicate with each other. Advantages of JSON are:-

* It stores all the data in an array so that data transfer makes it easier. That’s why it is the best for sharing data of any size even audio, video, etc.
* Its syntax is very small, easy, and light-weighted that’s the reason it executes and responds in a faster way.
* It has a wide range of browser support compatibility with the operating systems. It doesn’t require much effort to make it all browser compatible.On the server-side parsing is the most important part that developers want. If the parsing will be fast on the server side then the user can get a fast response, so in this case, JSON server-side parsing is the strong point compared to others.

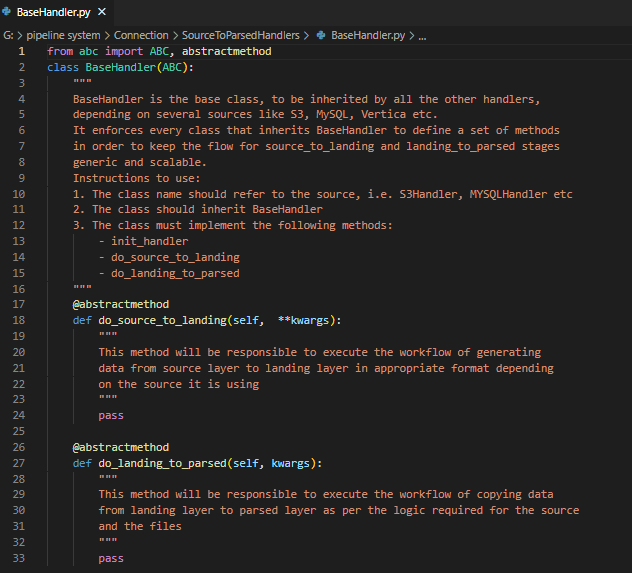
We are also using Yaml file.

**Fig.5.9 :** 

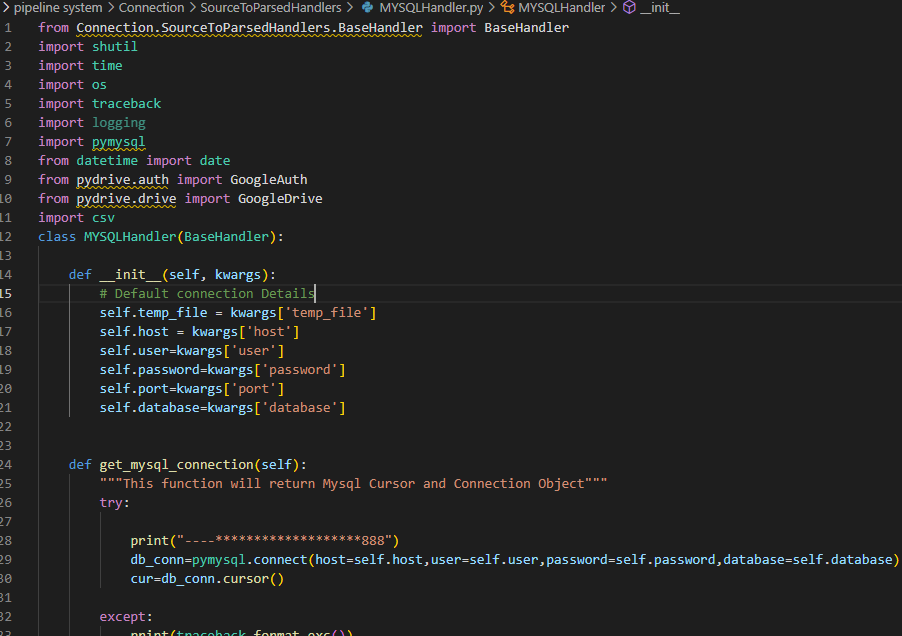
YAML works in concurrence with any programming language, it is often used **to write configuration files**. YAML files are easy to work with in a text editor, portable between programming languages, and expressive and extensible. They support the Unicode character set, and YAML files perform many of the same functions as XML or JSON.

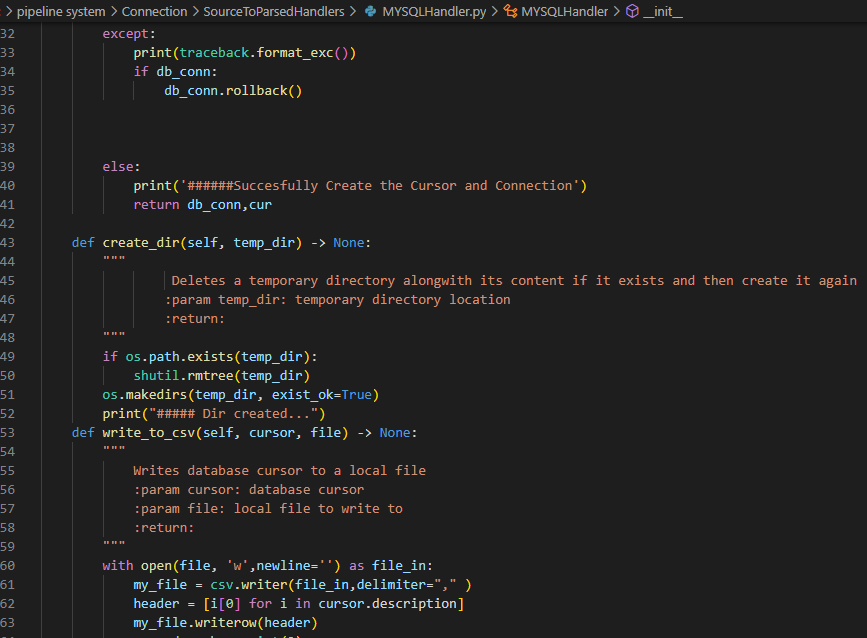
For connection with MySQL, there are two python files. That are:-

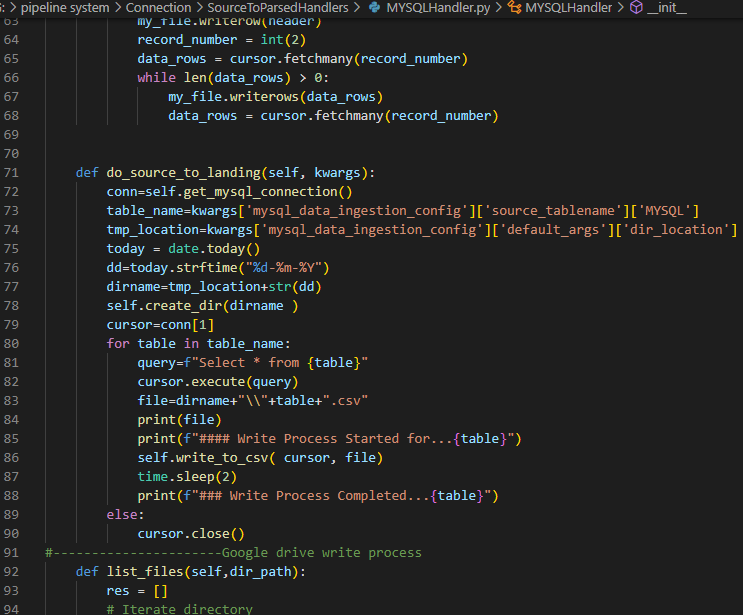
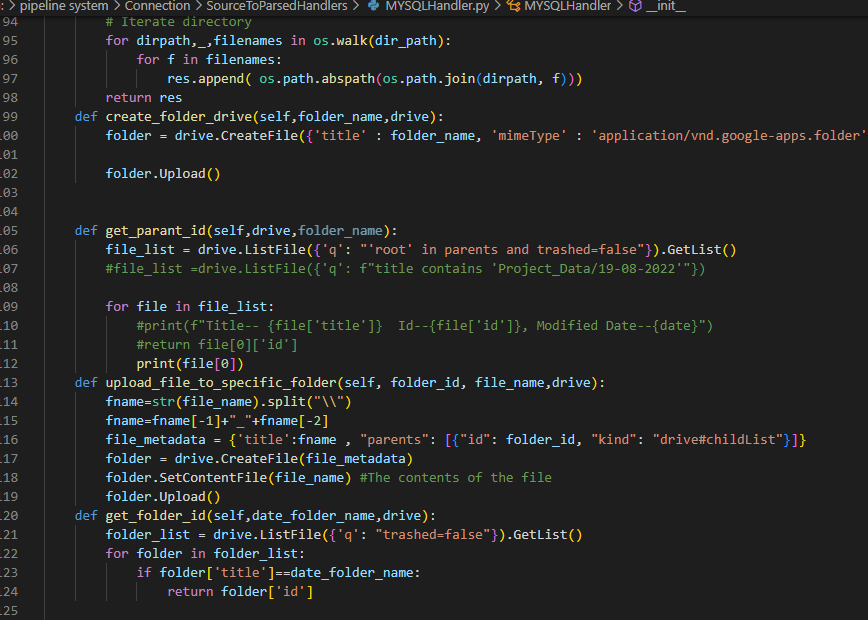
* BaseHandler:

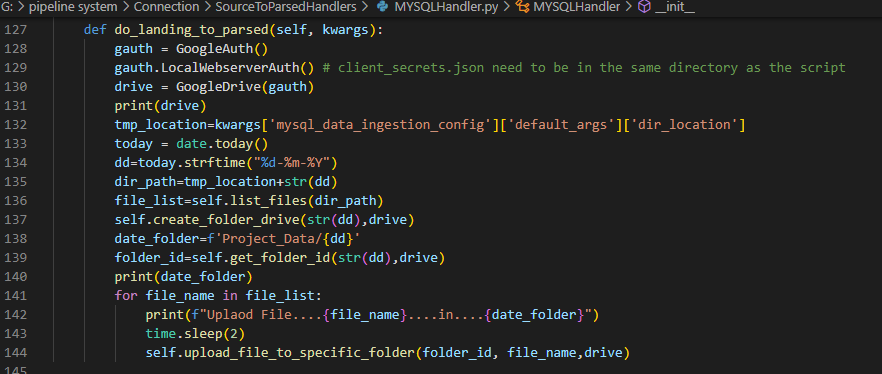
**Fig.5.10** : 

* MySQL Handler:

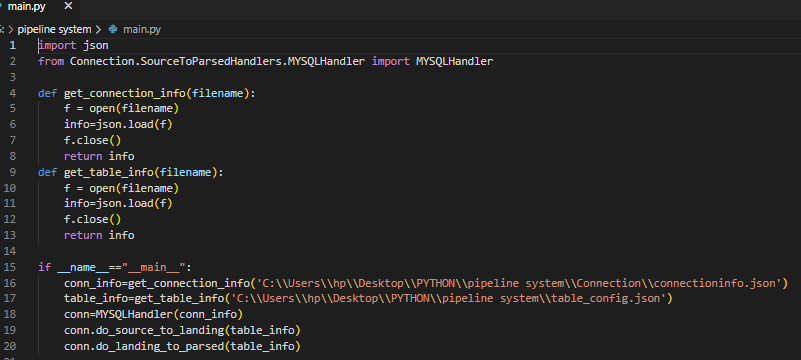
**Fig.5.11 :** 

**Fig.5.12 :** 

**Fig.5.13 :** **Fig.5.14 :** 

**Fig.5.15 :** 

After all the configurations, ther is main code for running our project. I.e.

**Fig.5.16 :** 

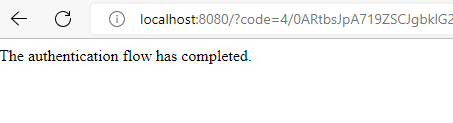
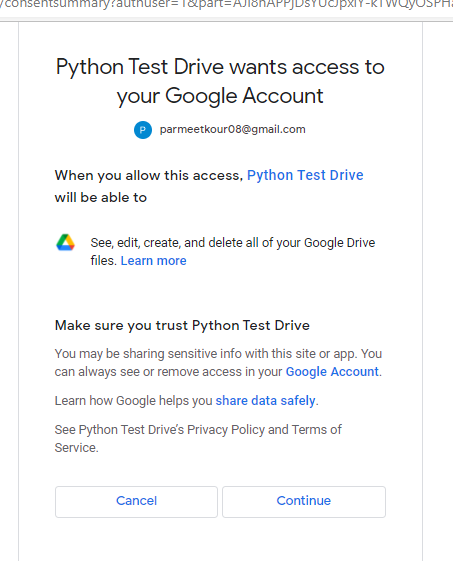
**14.7 Project Result -**

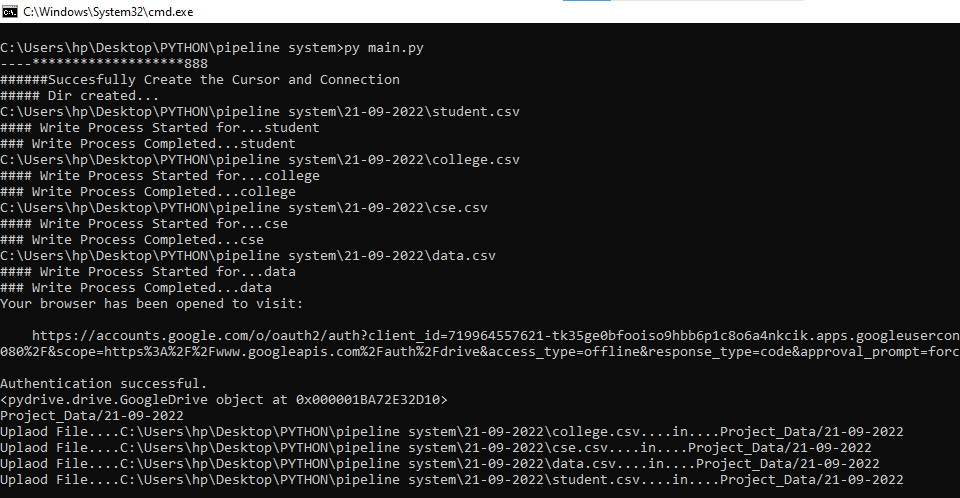
For authentication, firstly it will ask for login google id once.

Because of the YAML file, we don’t need to do this authentication process, it will automatically log in afterward.

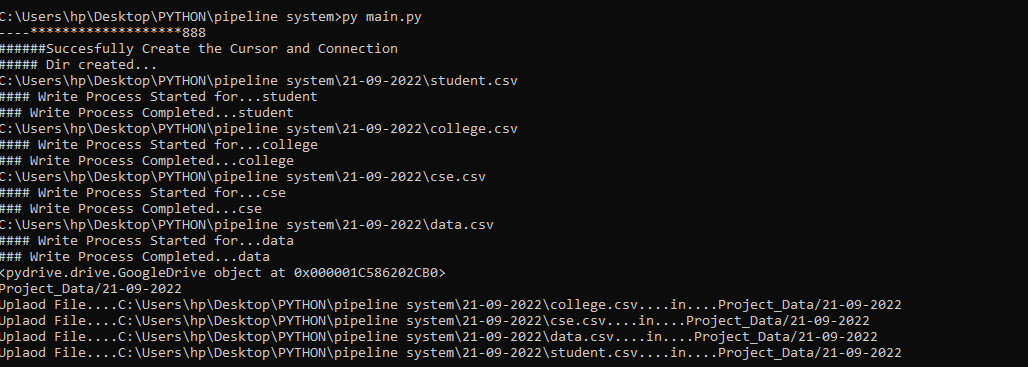
Here is the screenshot of the page where we completing the authentication process.

**Fig.5.17&18 :**

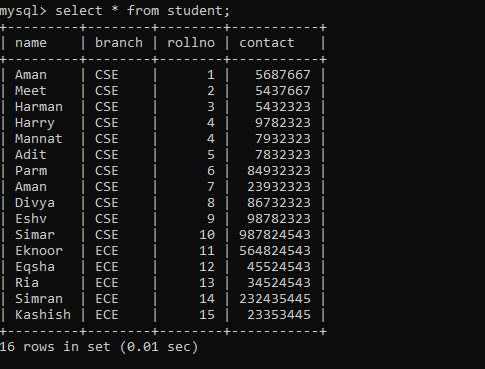
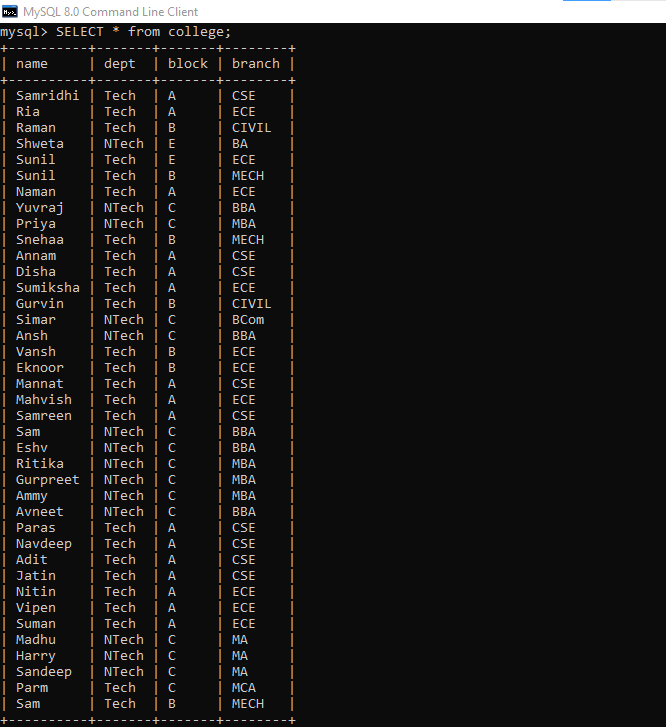
****

**Fig.5.19 : **

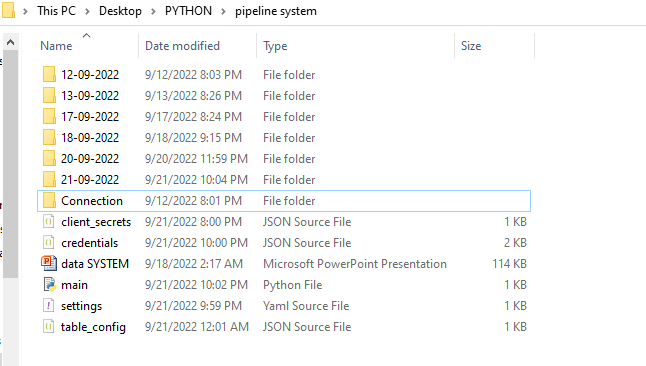
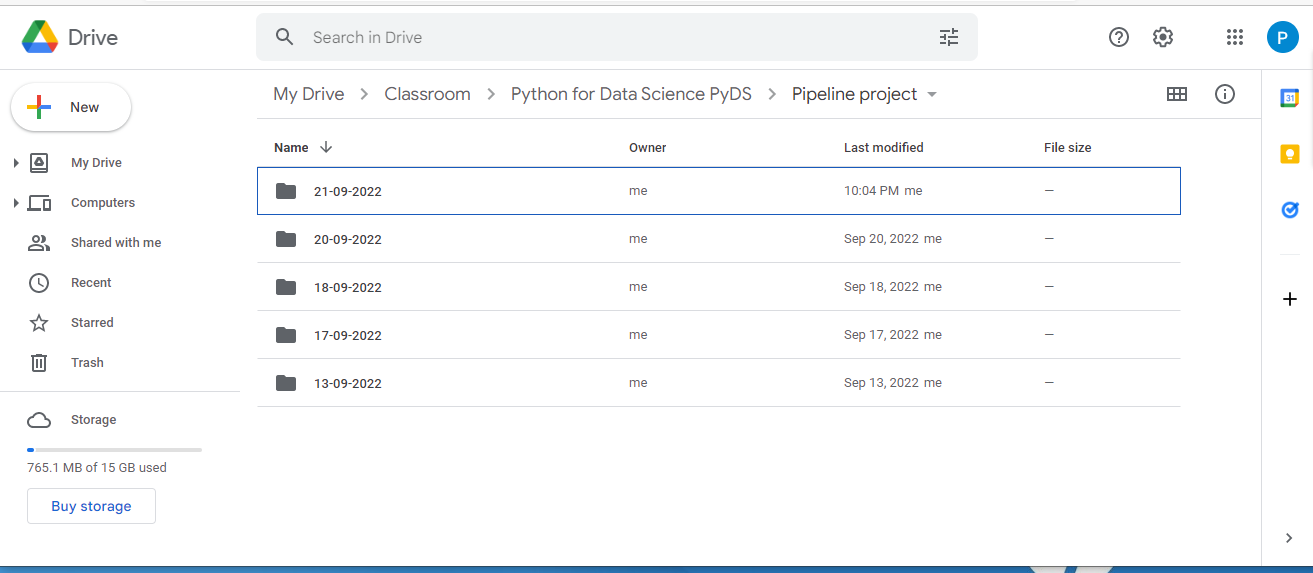
Here, the authentication process.

**Fig.5.20 : **

* For eg. We received data from some other company in SQL. This data needs to be stored in our system and google drive, so that business intelligence team picks the data in accordance with their needs.

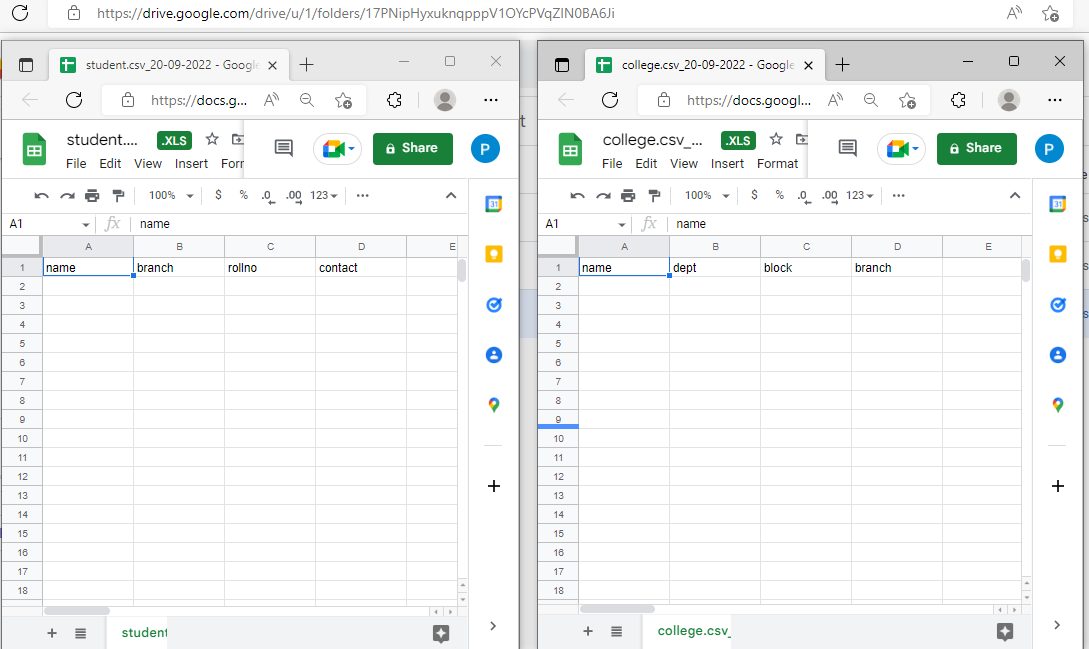
**Fig.21&22 :** 

* After running the main code, data from the SQL source will get saved in our system and in the linked google drive in the form of an excel-sheet.

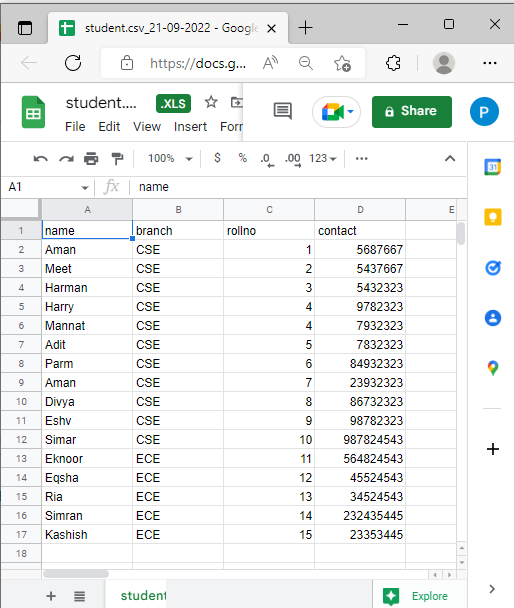
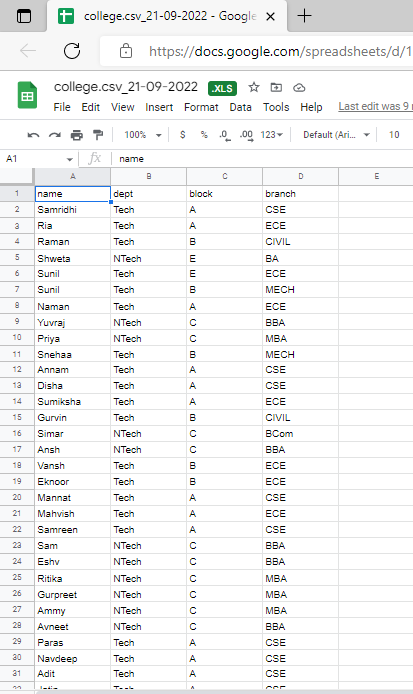
 

**Fig.5.23,24&25 :** 

* Status of the database server prior to the updating of data from the companies end (20.09.22).

**Fig.5.26 :** 

* Status of the database server after the update of data from the company's end (21.09.22).



**Fig.5.27&28 :**

**14.8 Conclusion -**

This project mainly provides benefits across the organization:-

* Quickly migrate data from on-premises to the cloud.
* Reliably replicate key data sources for disaster recovery and backup.
* Easily consolidate multiple data sources into a single data destination.
* Consistently transform data to make it analytics-ready.
* Automate the data movement process.

***References -***

* [***Data Pipeline- Definition, Architecture, Examples, and Use Cases (projectpro.io)***](https://www.projectpro.io/article/data-pipeline-definition-architecture-examples/528#mcetoc_1fmrm9kbac)
* [***https://www.bing.com/images/search***](https://www.bing.com/images/search)
* [***Data Pipeline- Definition, Architecture, Examples, and Use Cases (projectpro.io)***](https://www.projectpro.io/article/data-pipeline-definition-architecture-examples/528#mcetoc_1fmrm9kbac)
* [***Google Cloud console***](https://console.cloud.google.com/apis/credentials?authuser=1&project=clientsecrets-09)
* [***WHAT IS DATA PIPELINE? | BEST TOOLS FOR OPERATIONS WITH DATA PIPELINES - YouTube***](https://www.youtube.com/watch?v=0qJgz5A3ND8)
* [***https://www.geeksforgeeks.org/***](https://www.geeksforgeeks.org/)
* [***What is a Data Pipeline? Why to Use Them, Different Types, and Standard Components (matillion.com)***](https://www.matillion.com/resources/blog/what-is-a-data-pipeline-why-to-use-them-different-types-and-standard-components)

**—---------------------------------------------------------------------------------------------------------------**

**THANK YOU!**

**—--------------------------------------------------------------------------------------------------------------**