PROJECT REPORT ON

PYTHON DISTRIBUTION PACKAGE

Submitted to

Department of Computer Applications

in partial fulfilment for the award of the degree of

MASTER OF COMPUTER APPLICTIONS

Batch (2020-2022)

Submitted by

Janhavi Babber

Enrolment Number: GE-17220375

Under the Guidance of

Mr. Sanjiv Chauhan

GRAPHIC ERA DEEMED TO BE UNIVERSITY DEHRADUN

CANDIDATE'S DECLARATION

I hereby certify that the work presented in this project report entitled "Python Distribution Package" in partial fulfilment of the requirements for the award of the degree of Master of Computer Applications is a bonafide work carried out by me during the period of February 2022 to July 2022 under the supervision of Mr. Sanjiv Chauhan, Department of Computer Application, Graphic Era Deemed to be University, Dehradun, India.

This work has not been submitted elsewhere for the award of a degree/diploma/certificate.

Janhavi Babber

Name and Signature of Candidate

This is to certify that the above mentioned statement in the candidate's declaration is correct to the best of my knowledge.

Mr. Sanjiv Chauhan

Date: **Name and Signature of Guide**

Signature of Supervisor **Signature of External Examiner**

HOD

ACKNOWLEDGEMENT

Management is a profession wherein no work can be accomplished without the help and assistance of a large number of people, be it your superiors or subordinates. Completing a task is never a one man's effort. It is often the result of direct or indirect invaluable contribution of a number of individuals.

I would like to thank **Graphic Era 'Deemed to be University'** for providing me with this great opportunity to work on this report.

It is indeed a great pleasure to take the opportunity to extend my sincere thanks to all those whose help and guidance made this endeavor a successful one.

I wish to express my sincere gratitude to **Mr. Sanjiv Chauhan,** for his guidance and support during the study and preparation of the project and report.

Name: Janhavi Babber

Course: MCA(Sem:4)

Roll No.: 1101965

CERTIFICATE OF ORIGINALITY

This is to certify that the project report entitled **Python Distribution Package** submitted to **Graphic Era University, Dehradun** in partial fulfilment of the requirement for the award of the degree of **MASTER OF COMPUTER APPLICATIONS (MCA)**, is an authentic and original work carried out by Ms. Ilarika Rautela with enrolment number GE-20111965 under my supervision and guidance.

The matter embodied in this project is genuine work done by the student and has not been submitted whether to this University or to any other University / Institute for the fulfilment of the requirements of any course of study.

Signature of the Student: Signature of the Guide

Date: Date:

Enrolment No.: GE-20111965

Name of the Student: Name and Designation of the Guide:

Janhavi Babber Mr. Sanjiv Chauhan

SD

Shikha, Deep (Contractor)

To: Babber, Janhavi (Contractor)

Sat 14-05-2022 17:32

Hello Janhavi Babber,

Employee ID: 2153790
Employee Name: Babber, Janhavi
Project ID: 1000288128
Project Name: CBE Campus Interns
Assignment Start Date: 2022-02-18
Assignment End Date: 2022-08-29

CDE Java FSE
Integrated Learning Start Date: 2022-03-03
Integrated Learning End Date: 2022-07-27
Total Number of Weeks: 19
Weeks Completed: 12
Weeks left: 7

Regards,
Deepshikha
GenC Coach, CDE
Human Resources - GenC
M +91 9528290855

15-Jan-2022

Janhavi Babber

MCA Computer Application

Graphic Era University

Dear Janhavi Babber,

Further to our Letter of Intent / Offer for the position of Programmer Analyst Trainee / Programmer Analyst aligned to the hiring category and in response to your subsequent confirmation for Internship Program with us, we are pleased to offer you an Internship with us for **a period of 3 to 6 months.** Your Internship onboarding will be scheduled anytime between now, through end of March 2022 based on your availability factoring your college exam schedule and our business requirements.

During this period, you will be provided with a stipend of INR 12,000 per month equated to the planned duration of the Internship curriculum and will be paid only subject to successful completion of milestones as defined in the curriculum prior to the monthly stipend processing window for a given month based on your performance and attendance.

Actual commencement of Internship dates and duration would be shortly communicated to you and the internship would be based on the business demand aligned to your skill tracks.

Though Cognizant Internship being a pre- requisite skill and capability development program, it does not guarantee employment. However, the successful completion of internship will form a critical part of your employment with Cognizant if an opportunity arises in future.

You will undergo a learning curriculum as per the learning track assigned to you. The learning path will include in-depth sessions, hands on exercise and project work. There will also be series of webinars, quizzes, SME interactions, mentor connects, code challenges, assessments etc. to

accelerate your learning. The outcome during Internship would be monitored through formal evaluations.

Prior to joining on the rolls of Cognizant, you must have successfully completed the prescribed Internship program. In the event of unsatisfactory Internship, Cognizant reserves rights at its sole discretion to revoke its employment offer.

Please also note that:

- The Internship timings would be for 9 hours per day from Monday through Friday aligned to the working timings followed in Cognizant
- Interns are covered under Cognizant's calendar holidays of the respective location of internship and you would need to adhere with minimum attendance requirements. Prior approvals are must towards any unavoidable leave or break requests during the program.
- There would be zero tolerance to plagiarisms and misconduct during the internship. Any such incident reported will lead to immediate cancellation of internship without any notice.
- You would be required to ensure timely completion and submission of assignments, project work and preparation required prior to the sessions.
- You may be required, to travel to other locations within India if there is a business need as per your internship program
- Cognizant reserves rights regarding IT infra as applicable and access to information and material of Cognizant during the internship period and may modify or amend the Cognizant GenC program terms and conditions from time to time
- Stipend payment will be done for the prescribed Internship Curriculum period only and no additional payment will be done for any delay in completion.
- Attendance and successful completion of Milestone(s) are the eligible factors for processing stipend

payment and tenure spent will not guarantee your monthly stipend payment.

At the time of your reporting for the internship, you will be required to sign a Non - Disclosure Agreement with the company. During the course of your Internship and after completion of the same, you are required to maintain strictest confidentiality with respect to company proprietary or products that you access or come into contact with, during your project as an Intern, at all times as per our Policy. Use of company proprietary information or products shall not be made without prior permission from the concerned authority. Any breach of information security will be dealt as per Company Policy.

You will also be required to submit the following documents at the time of reporting;

- Photocopy of your Passport & Visa
- Photocopy of your Certificates / Mark Sheets in support of your Educational Qualification(s)
- 2 Passport-size photographs
- Pan Card
- Aadhar Card
- Personal individual bank account from a nationalized bank for processing stipend

Please do not hesitate to call us for any information you may need.

We wish you good luck.

Yours sincerely,

For Cognizant Technology Solutions India Pvt. Ltd.,

Maya Sreekumar

Vice President - Human Resource

I accept the terms and conditions of the internship program as mentioned above.

Signature: **Date:**

TABLE OF CONTENTS

1. INTRODUCTION

Probability denotes the possibility of something happening. It is a mathematical concept that predicts how likely events are to occur. The probability values are expressed between 0 and 1. The definition of probability is the degree to which something is likely to occur. This fundamental theory of probability is also applied to probability distributions.

Probability Distributions

A probability distribution is a statistical function that describes all the possible values and probabilities for a random variable within a given range. This range will be bound by the minimum and maximum possible values, but where the possible value would be plotted on the probability distribution will be determined by a number of factors. The mean (average), standard deviation, skewness, and kurtosis of the distribution are among these factors.

There are two types of probability distribution which are used for different purposes and various types of the data generation process.

1. Normal or Cumulative Probability Distribution
2. Binomial or Discrete Probability Distribution

1.1. IDENTIFICATION OF NEED

This package will contain code to analyse Gaussian distribution you can also use a package to read a dataset and calculate the mean and standard deviation. You will also include code to plot a histogram of the dataset, as well as a probability density function of a Gaussian distribution. Finally, this package will be able to add two Gaussian distribution together. You can install it by using pip install using terminal. It is not mandatory to upload the package to pypi(repository where all python package are stored for the general public to download and install).

1.2. SOFTWARE REQUIREMENT SPECIFICATION

Platform, Hardware and Software Requirement Specifications:

- **Tools for Development**

  PyCharm

- **Platform:**

  Python

- **Hardware and Software Requirements:**

  RAM: 4GB

  System Type: 64bit OS

  Processor: i5

1.3. SOFTWARE DEVELOPMENT LIFE CYCLE MODEL

Waterfall Model

In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The sequential phases in Waterfall model are −

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next

phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

2. THEORITICAL BACKGROUND

2.1. OBJECT-ORIENTED PROGRAMMING

Object-oriented programming, also known as OOP, is a type of programming based on *objects*. Object are bundles of data and associated logic. For example, you might have a "dog" object that consists of some data (the dog's name or favorite treat) and associated logic (for example, instructions on how to bark).

Objects are made from templates called *classes* that define what kinds of data the object can hold and what kinds of things the object can do. These are known as the object's *properties* and *methods*, respectively.

Methods are functions that represent something you can ask the object to do. For example, the statement *car.drive()* can be interpreted as telling the object in the "car" variable to "drive". Properties are variables that belong to an object. Continuing the example, your car object might have a property called *gas*, and the statement *car.gas=100* would set the car's gas to 100. These two statements manipulate a car object that already exists. Recall that the car's class is the template that defines how to make a car object and what a car is by defining its properties and methods. Within the definitions of those methods, you will find the code that manipulates the car from the inside.

For instance, instead of *car.gas=100*, you might find *self.gas=100*, which is a car object telling itself – *self*, get it? – to set its own gas to 100. OOP is a large topic but the basics above are all you need to get started.

Your code will describe the Boxes game as the interaction of various objects. Those objects all have properties and methods, which you will define in the object's class. And when you write a piece of code, you should remember

whether you’re writing the class code that defines what an object can do from the “inside” of the object, or code that manipulates an object from the “outside” of that object.

2.2. PYTHON

Python 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
- 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 development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- 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 Syntax compared to 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.

2.3. PROBABLITY DISTRIBUTION

In Statistics, the **probability distribution** gives the possibility of each outcome of a random experiment or event. It provides the probabilities of different possible occurrences.

To recall, the **probability is a measure of uncertainty of various phenomena**. Like, if you throw a dice, the possible outcomes of it, is defined by the probability. This distribution could be defined with any random experiments, whose outcome is not sure or could not be predicted. Let us discuss now its definition, function, formula and its types here, along with how to create a table of probability based on random variables.

Probability Distribution

Probability distribution yields the possible outcomes for any random event. It is also defined based on the underlying sample space as a set of possible outcomes of any random experiment. These settings could be a set of real numbers or a set of vectors or a set of any entities. It is a part of probability and statistics.

Random experiments are defined as the result of an experiment, whose outcome cannot be predicted. Suppose, if we toss a coin, we cannot predict, what outcome it will appear either it will come as Head or as Tail. The possible result of a random experiment is called an outcome. And the set of outcomes is called a sample point. With the help of these experiments or events, we can always create a probability pattern table in terms of variables and probabilities.

Types of Probability Distribution

The probability distribution is divided into two parts:

1. Discrete Probability Distributions
2. Continuous Probability Distributions

1. Discrete Probability Distribution

A discrete distribution describes the probability of occurrence of each value of a discrete random variable. The number of spoiled apples out of 6 in your refrigerator can be an example of a discrete probability distribution.

Each possible value of the discrete random variable can be associated with a non-zero probability in a discrete probability distribution.

Let's discuss some significant probability distribution functions.

2.3.1. Binomial Distribution

The binomial distribution is a discrete distribution with a finite number of possibilities. When observing a series of what are known as Bernoulli trials, the binomial distribution emerges. A Bernoulli trial is a scientific experiment with only two outcomes: success or failure.

Consider a random experiment in which you toss a biased coin six times with a 0.4 chance of getting head. If 'getting a head' is considered a ‘success’, the binomial distribution will show the probability of r successes for each value of r.

The binomial random variable represents the number of successes (r) in n consecutive independent Bernoulli trials.

Binomial distribution formulas

mean

$$\mu = n * p$$

In other words, a fair coin has a probability of a positive outcome (heads) p = 0.5. If you flip a coin 20 times, the mean would be 20 * 0.5 = 10; you'd expect to get 10 heads.

variance

$$\sigma^2 = np(1-p)$$

Continuing with the coin example, n would be the number of coin tosses and p would be the probability of getting heads.

standard deviation

$$\sigma = \sqrt{np(1-p)}$$

In other words, the standard deviation is the square root of the variance.

probability density function

$$f(k, n, p) = \frac{n!}{k!(n-k)!} p^k (1-p)^{(n-k)}$$

Binomial Distribution Examples

As we already know, binomial distribution gives the possibility of a different set of outcomes. In the real-life, the concept is used for:

- To find the number of used and unused materials while manufacturing a product.
- To take a survey of positive and negative feedback from the people for anything.
- To check if a particular channel is watched by how many viewers by calculating the survey of YES/NO.
- The number of men and women working in a company.
- To count the votes for a candidate in an election and many more.

2. Continuous Probability Distributions

A continuous distribution describes the probabilities of a continuous random variable's possible values. A continuous random variable has an infinite and uncountable set of possible values (known as the range). The mapping of time can be considered as an example of the continuous probability distribution. It can be from 1 second to 1 billion seconds, and so on.

The area under the curve of a continuous random variable's PDF is used to calculate its probability. As a result, only value ranges can have a non-zero probability. A continuous random variable's probability of equalling some value is always zero.

Now, look at some varieties of the continuous probability distribution.

2.3.2. Normal Distribution

Normal Distribution is one of the most basic continuous distribution types. Gaussian distribution is another name for it. Around its mean value, this probability distribution is symmetrical. It also demonstrates that data close to the mean occurs more frequently than data far from it. Here, the mean is 0, and the variance is a finite value.

In the example, you generated 100 random variables ranging from 1 to 50. After that, you created a function to define the normal distribution formula to calculate the probability density function. Then, you have plotted the data points and probability density function against X-axis and Y-axis, respectively.

Gaussian distribution formulas

probability density function

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where: μ is the mean σ is the standard deviation σ^2 is the variance

Normal Distribution Examples

Since the normal distribution statistics estimates many natural events so well, it has evolved into a standard of recommendation for many probability queries. Some of the examples are:

- Height of the Population of the world
- Rolling a dice (once or multiple times)
- To judge the Intelligent Quotient Level of children in this competitive world
- Tossing a coin
- Income distribution in countries economy among poor and rich
- The sizes of females shoes
- Weight of newly born babies range
- Average report of Students based on their performance

3. BACKGROUND STUDY

3.1. Advanced OOP topics

Inheritance is the last object-oriented programming topic in the lesson. Thus far you've been exposed to:

- Classes and objects
- Attributes and methods
- Magic methods
- Inheritance

Classes, object, attributes, methods, and inheritance are common to all object-oriented programming languages.

Knowing these topics is enough to start writing object-oriented software. What you've learned so far is all you need to know to complete this OOP lesson. However, these are only the fundamentals of object-oriented programming.

3.1.1 MAGIC METHODS :

The init function which initializes a new object is a magic method. It lets you override and customize default python behaviour of the initialized methods of python:

1. init method lets you override or customize how python instantiates an object.
2. add method overrides the behaviour of the '+' sign.
3. Representation method overrides the default representation behaviour of printing output. The mean and

standard deviation of the Gaussian object. Usually representation method controls what gets printed on the screen.

3.1.2. ORGANISING CODE IN MODULES:-

In Python, a module is a single python file that contains a colletion of functions, class or a global variables, they are called modules because they’re modular.

If you are using macOS, you can open an application called Terminal and use the same commands that you use in the workspace. That is because Linux and MacOS are related.

If you are using Windows, the analogous application is called Console. The Console commands can be somewhat different than the Terminal commands.

Jupyter Notebooks are especially useful for data science applications because you can wrangle data, analyze data, and share a report all in one document. However, they're not ideal for writing modular programs, which require separating code into different files.

Look at how the distribution class and Gaussian class are modularized into different files.

The Gaussiandistribution.py imports the Distribution class from the Generaldistribution.py file. Note the following line of code:

from Generaldistribution import Distribution

This code essentially pastes the distribution code to the top of the Gaussiandistribution file when you run the code. You can see in the example_code.py file an example of how to use the Gaussian class.

The example_code.py file then imports the Gaussian distribution class.

For the rest of the lesson, you'll work with modularized code rather than a Jupyter Notebook. Go through the code in the modularized_code folder to understand how everything is organized.

3.1.3. INHERITANCE:-

The distribution class takes care of the initialization and the read_data_file method. The rest of the Gaussian code is in the Gaussian class and Binomial code in the binomial Class. The Distribution class is the parent class for both Binomial Class as well as Gaussian Object.

3.2. MAKING A PACKAGE:-

The distribution and Gaussian code was refactored into individual modules. A *Python module* is just a Python file containing code.

You'll convert the distribution code into a Python package. A *package* is a collection of Python modules. Although the previous code might already seem like it was a Python package because it contained multiple files, a Python package also needs an __init__.py file. In this section, you'll learn how to create this __init__.py file and then pip install the package into your local Python installation.

3.2.1. What is pip?

pip is a Python package manager that helps with installing and uninstalling Python packages. You might have used pip to install packages using the command line: pip install numpy. When you execute a command like pip install numpy, pip downloads the package from a Python package repository called PyPi.

You'll use pip to install a Python package from a local folder on your computer. The last part of the lesson will focus on uploading packages to PyPi so that you can share your package with the world.

If you want to develop a package locally on your computer, you should consider setting up a virtual environment. That way, if you install your package on your computer, the package won't install into your main Python installation. Before starting the next exercise, the next part of the lesson will discuss what virtual environments are and how to use them.

Object-oriented programming and Python packages

A Python package does not need to use object-oriented programming. You could simply have a Python module with a set of functions. However, most—if not all—of the popular Python packages take advantage of object-oriented programming for a few reasons:

1. Object-oriented programs are relatively easy to expand, especially because of inheritance.
2. Object-oriented programs obscure functionality from the user. Consider scipy packages. You don't need to know how the actual code works in order to use its classes and methods.

3.2.2. __init__.py

This file tells the python that this folder contains a package. A package always needs to have an init file even if the file is completely empty. The code inside the init file gets run whenever you import a package inside a python program.

In this case the init filr is importing the Gaussian class from the Gaussian distribution module because I want to be able to import the Gaussian class directly by writing :

from distributions import Gaussian

from distributions import Binomial

This will work with including this line in init file as well but any program that uses the package would have to import the Gaussian and binomial class indirectly from distributions.

Gaussiandistribution import Gaussian

Binomialdistribution import Binomial

So, including the line is like making a shortcut for importing the Gaussian or binomial class directly.

3.2.3. setup.py

This file is necessary for pip installing. Pip will automatically look for this file. So, this file contain information or metadata about the packge like package name, version, description, etc.

Now to install the package:-

Step1: Go to terminal in pycharm

Step2: Check if your in the same directory with setup.py file

Step3: In terminal type pip install .

Step4: Python package installed

Now package can be used in the code.

To import Gaussian class:

from distributions import Gaussian

from distributions import Binomial

3.3. Virtual Environment

Now here, you can upload files into a Python package and pip install the package. If you decide to install your package on your local computer, you'll want to create a virtual environment. A virtual environment is a silo-ed Python installation apart from your main Python installation. That way you can install packages and delete the virtual environment without affecting your main Python installation.

Let's talk about two different Python environment managers: conda and venv. You can create virtual environments with either one. The following sections describe each of these environment managers, including some advantages and disadvantages. If you've taken other data science, machine learning, or artificial intelligence courses at Udacity, you're probably already familiar with conda.

3.3.1. Conda

Conda does two things: manages packages and manages environments.

As a package manager, conda makes it easy to install Python packages, especially for data science. For instance, typing conda install numpy installs the numpy package.

As an environment manager, conda allows you to create silo-ed Python installations. With an environment manager, you can install packages on your computer without affecting your main Python installation.

The command line code looks something like the following:

conda create --*name environmentname*

source activate environmentname

conda install numpy

3.3.2. pip and Venv

There are other environmental managers and package managers besides conda. For example, venv is an environment manager that comes preinstalled with Python 3. pip is a package manager.

pip can only manage Python packages, whereas conda is a language agnostic package manager. In fact, conda was invented because pip could not handle data science packages that depended on libraries outside of Python. If you look at the history of conda, you'll find that the software engineers behind conda needed a way to manage data science packages (such as NumPy and Matplotlib) that relied on libraries outside of Python.

conda manages environments *and* packages. pip only manages packages.

To use venv and pip, the commands look something like the following:

python3 -m venv environmentname

source environmentname/bin/activate

pip install numpy

Which to choose

Whether you choose to create environments with venv or conda will depend on your use case. conda is very helpful for data science projects, but conda can make generic Python software development a bit more confusing; that's the case for this project.

If you create a conda environment, activate the environment, and then pip install the distributions package, you'll find that the system installs your package globally rather than in your local conda environment. However, if you create the conda environment and install pip simultaneously, you'll find

that pip behaves as expected when installing packages into your local environment:

conda create --*name environmentname pip*

On the other hand, using pip with venv works as expected. pip and venv tend to be used for generic software development projects including web development. You can use conda or venv if you want to develop locally on your computer and install your package.

I'll be using venv, which is what we recommend for this project.

3.3.3. Instructions for venv

For instructions about how to set up virtual environments on a macOS, Linux, or Windows machine using the terminal, see Installing packages using pip and virtual environments.

Refer to the following notes for understanding:

- If you are using Python 2.7.9 or later (including Python 3), the Python installation should already come with the Python package manager called pip. There is no need to install it.
- env is the name of the environment you want to create. You can call env anything you want.
- Python 3 comes with a virtual environment package preinstalled. Instead of typing python3 -m virtualenv env, you can type python3 -m venv env to create a virtual environment.

Once you've activated a virtual environment, you can then use terminal commands to go into the directory where your Python library is stored. Then, you can run pip install.

You'll see that creating a virtual environment actually creates a new folder containing a Python installation. Deleting this folder removes the virtual environment.

3.3.4. Making a Package and pip installing

On your local computer, you need to create a folder called python_distribution_package. Inside this folder, you need to create a few folders and files:

- A setup.py file, which is required in order to use pip install.
- A subfolder called distributions, which is the name of the Python package.
- Inside the distributions folder, you need:
    - The Gaussiandistribution.py file (provided).
    - The Generaldistribution.py file (provided).
    - The __init__.py file (you need to create this file).

Once everything is set up, in order to actually create the package, use your terminal window to navigate into the python_distribution_package folder.

Enter the following:

cd python_distribution_package

pip install

If everything is set up correctly, pip installs the distributions package into the workspace. You can then start the Python interpreter from the terminal by entering:

python

Then, within the Python interpreter, you can use the distributions package by entering the following:

from distributions import Gaussian

gaussian_one = Gaussian(25, 2)

gaussian_one.mean

gaussian_one + gaussian_one

In other words, you can import and use the Gaussian class because the distributions package is now officially installed as part of your Python installation.

If you want to install the Python package locally on your computer, you might want to set up a virtual environment first. A virtual environment is a silo-ed Python installation apart from your main Python installation. That way you can easily delete the virtual environment without affecting your Python installation.

If you want to try using virtual environments in this workspace first, follow these instructions:

1. There is an issue with the Ubuntu operating system and Python3, in which the venv package isn't installed correctly. In the workspace, one way to fix this is by running this command in the workspace terminal: conda update python.

For more information, see venv doesn't create activate script python3. Then, enter y when prompted. It might take a few minutes for the workspace to update. If you are not using Anaconda on your local computer, you can skip this first step.

2. Enter the following command to create a virtual environment: python -m venv venv_name where venv_name is the name you want to give to your virtual environment. You'll see a new folder appear with the Python installation named venv_name.
3. In the terminal, enter source venv_name/bin/activate. You'll notice that the command line now shows (venv_name)at the beginning of the line to indicate you are using the venv_name virtual environment.
4. Enter pip install python_package/. That should install your distributions Python package.

Inside the folder called python_distribution_package , there is another folder and these files:

- distributions, which contains the code for the distributions package including Gaussiandistribution.py and Generaldistribution.py code.
- setup.py, a file needed for building Python packages with pip.
- test.py unit tests to help you debug your code.
- numbers.txt and numbers_binomial.txt, which are data files used as part of the unit tests.
- Binomialdistribution.py and Binomialdistribution_challenge.py. Choose one of these files for completing the exercise. Binomialdistribution.py includes more of the code already set up

for you. In Binomialdistribution_challenge.py, you'll have to write all of the code from scratch. Both files contain instructions with TODOS to fill out.

In these files, you only need to change the following:

- __init__.py, inside the distributions folder. You need to import the binomial package.
- Either Binomialdistribution.py or Binomialdistribution_challenge.py You also need to put your Binomialdistribution.py file into the distributions folder.

When you're ready to test out your code, follow these steps:

1. **pip**** install your distributions package**. In the terminal, make sure you are in the 4a_binomial_package directory. If not, navigate there by entering the following at the command line:

cd python_distribution_package

pip install

2. **Run the unit tests**. Enter the following.

python -m unittest test

Modify the Binomialdistribution.py code until all the unit tests pass.

If you change the code in the distributions folder after pip installing the package, Python will not know about the changes.

When you make changes to the package files, you'll need to run the following:

pip install --upgrade

To know where pythion package is installed or stored locally:

distributions.__file__

3.4. PyPi vs. test PyPi

It is optional to put your package on pypi.

3.4.1. Putting code on Pypi

Note that pypi.org and test.pypy.org are two different websites. You'll need to register separately at each website. If you only register at pypi.org, you will not be able to upload to the test.pypy.org repository.

Remember that your package name must be unique. If you use a package name that is already taken, you will get an error when trying to upload the package.

Summary of the terminal commands used in the video

cd binomial_package_files

python setup.py sdist

pip install twine

commands to upload to the pypi test repository

*twine upload --repository-url https://test.pypi.org/legacy/ dist/**

pip install --index-url https://test.pypi.org/simple/ dsnd-probability

command to upload to the pypi repository

*twine upload dist/**

pip install dsnd-probability

More PyPi resources

We explains how to distribute Python packages, including more configuration options for your setup.py file. You'll notice that the Python command to run the setup.py is slightly different, as shown in the following example:

python3 setup.py sdist bdist_wheel

This command still outputs a folder called dist. The difference is that you will get both a .tar.gz file and a .whl file. The .tar.gz file is called a *source archive*, whereas the .whl file is a *built distribution*. The .whl file is a newer type of installation file for Python packages. When you pip install a package, pip firsts look for a .whl file (wheel file); if there isn't one, it looks for the .tar.gz file.

A .tar.gz file (an sdist) contains the files needed to compile and install a Python package. A .whl file (a *built distribution*) only needs to be copied to the proper place for installation. Behind the scenes, pipinstalling a .whl file has fewer steps than installing a .tar.gz file.

Other than this command, the rest of the steps for uploading to PyPi are the same.

3.4.2. Upload to PyPi

You'll be uploading a package to PyPi.

You need to create three files:

setup.cfg

README.md

license.txt

You also need to create accounts for the pypi test repository and pypi repository.

Don't forget to keep your passwords; you'll need to type them into the command line.

Once you have all the files set up correctly, you can use the following commands on the command line. You need to make the name of the package unique, so change the name of the package from distributions to something else. That means changing the information in setup.py and the folder name.

In the terminal, make sure you are in the python_distribution_package_to_pypi directory. If not, navigate there by entering the following at the command line:

cd python_distribution_package_to_pypi

python setup.py sdist

pip install twine

Commands to upload to the PyPi test repository

*twine upload --repository-url https://test.pypi.org/legacy/ dist/**

pip install --index-url https://test.pypi.org/simple/ distributions

Command to upload to the PyPi repository

*twine upload dist/**

pip install distributions

4. LIBRARIES USED

4.1. MatplotLib in Python

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Installation :

Windows, Linux and macOS distributions have matplotlib and most of its dependencies as wheel packages. Run the following command to install matplotlib package :

```
python -mpip install -U matplotlib
```

Importing matplotlib :

```
from matplotlib import pyplot as plt
```
or
```
import matplotlib.pyplot as plt
```

Importing matplotlib :
Python has a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.

The Math Module

Python has also a built-in module called math, which extends the list of mathematical functions.

To use it, you must import the math module:

```
import math
```

When you have imported the math module, you can start using methods and constants of the module.

The math.sqrt() method for example, returns the square root of a number:

5. METHODOLOGY

5.1. FOR DISTRIBUTION CLASS

Generic distribution class for calculating and visualizing a probability distribution.

Attributes:

mean (float) representing the mean value of the distribution

stdev (float) representing the standard deviation of the distribution.

data_list (list of floats) a list of floats extracted from the data file.

METHODS:-

S.No.	***METHOD NAME***	***EXPLANATION***	***ARGS***	***RETURN TYPE***
1.	read_data_file	Function to read in data from a txt file. The txt file should have one number (float) per line. The numbers are stored in the data attribute.	file_name (string): name of a file to read from	None

5.2. FOR GAUSSIAN CLASS

Gaussian distribution class for calculating and visualizing a Gaussian distribution. Here __init__, __add__, __repr__ are magic methods.

Attributes:-

mean (float) representing the mean value of the distribution

stdev (float) representing the standard deviation of the distribution

data_list (list of floats) a list of floats extracted from the data file.

METHODS:-

S.No.	***METHOD NAME***	***EXPLANATION***	***ARGS***	***RETURN TYPE***
1.	calculate_mean	Function to calculate the mean of the data set.	None	float: mean of the data set
2.	calculate_stdev	Function to calculate the standard deviation of the data set.	sample (bool): whether the data represents a sample or population	float: standard deviation of the data set
3.	plot_histogram	Function to output a histogram of the instance variable data using matplotlib pyplot library.	None	None
4.	pdf	Probability density function calculator for the gaussian distribution.	x (float): point for calculating the probability density function	float: probability density function output
5.	plot_histogram_pdf	Function to plot the normalized histogram of the data and a plot of the probability density function along the same range	n_spaces (int): number of data points	list: x values for the pdf plot list: y values for the pdf plot

6.	__add__	Function to add together two Gaussian distributions	other (Gaussian): Gaussian instance	Gaussian: Gaussian distribution
7.	__repr__	Function to output the characteristics of the Gaussian instance	None	string: characteristics of the Gaussian

5.3. FOR BINOMIAL CLASS

Binomial distribution class for calculating and visualizing a Binomial distribution.

Attributes:-

mean (float) representing the mean value of the distribution

stdev (float) representing the standard deviation of the distribution

data_list (list of floats) a list of floats to be extracted from the data file

p (float) representing the probability of an event occurring

n (int) number of trials

METHODS:-

S.No.	*METHOD NAME*	*EXPLANATION*	*ARGS*	*RETURN TYPE*
1.	calculate_mean	Function to calculate the mean from p and n.	None	float: mean of the data set
2.	calculate_stdev	Function to calculate the standard deviation from p and n.	sample (bool): whether the data represents a sample or population	float: standard deviation of the data set
3.	replace_stats_with_data	Function to calculate p and n from the data set.	None	float: the p value float: the n value
4.	plot_bar	Function to output a histogram of the instance variable data using matplotlib pyplot library.	None	None

5.	plot_bar_pdf	Function to plot the pdf of the binomial distribution	None	list: x values for the pdf plot list: y values for the pdf plot
6.	pdf	Probability density function calculator for the binomial distribution.	x (float): point for calculating the probability density function	float: probability density function output
7.	__add__	Function to add together two Binomial distributions with equal p	No other (Binomial): Binomial instance ne	Binomial: Binomial distribution
8.	__repr__	Function to output the characteristics of the Binomial instance	None	string: characteristics of the Gaussian

6. IMPLEMENTATION

6.1. Code:-

__init__.py

```
from .Gaussiandistribution import Gaussian
from .Binomialdistribution import Binomial
```

Generaldistribution.py

```
class Distribution:

    def __init__(self, mu=0, sigma=1):


            """ Generic distribution class for calculating and

            visualizing a probability distribution.


            Attributes:

                    mean (float) representing the mean value of the distribution

                    stdev (float) representing the standard deviation of the
distribution
```

```
            data_list (list of floats) a list of floats extracted from the data
file

            """


        self.mean = mu

        self.stdev = sigma

        self.data = []



    def read_data_file(self, file_name):


        """Function to read in data from a txt file. The txt file should have

        one number (float) per line. The numbers are stored in the data
attribute.


        Args:

            file_name (string): name of a file to read from


        Returns:
```

```
        None


"""


with open(file_name) as file:
        data_list = []
        line = file.readline()
        while line:
                data_list.append(int(line))
                line = file.readline()
file.close()
self.data = data_list
```

Gaussiandistribution.py

```
import math

import matplotlib.pyplot as plt

from .Generaldistribution import Distribution


class Gaussian(Distribution):
    """ Gaussian distribution class for calculating and

    visualizing a Gaussian distribution.


    Attributes:

            mean (float) representing the mean value of the distribution

            stdev (float) representing the standard deviation of the distribution

            data_list (list of floats) a list of floats extracted from the data file


    """

    def __init__(self, mu=0, sigma=1):


            Distribution.__init__(self, mu, sigma)
```

```
def calculate_mean(self):

        """Function to calculate the mean of the data set.

        Args:

            None

        Returns:

            float: mean of the data set

        """

        avg = 1.0 * sum(self.data) / len(self.data)

        self.mean = avg

        return self.mean
```

```
    def calculate_stdev(self, sample=True):

        """Function to calculate the standard deviation of the data set.

        Args:
            sample (bool): whether the data represents a sample or
population

        Returns:
            float: standard deviation of the data set

        """

        if sample:
            n = len(self.data) - 1
        else:
            n = len(self.data)
```

```
        mean = self.calculate_mean()

        sigma = 0

        for d in self.data:
            sigma += (d - mean) ** 2

        sigma = math.sqrt(sigma / n)

        self.stdev = sigma

        return self.stdev


    def plot_histogram(self):
        """Function to output a histogram of the instance variable data
using
```

```
        matplotlib pyplot library.

        Args:
            None

        Returns:
            None
        """
        plt.hist(self.data)
        plt.title('Histogram of Data')
        plt.xlabel('data')
        plt.ylabel('count')

    def pdf(self, x):
        """Probability density function calculator for the gaussian
distribution.
```

```
        Args:

            x (float): point for calculating the probability density
function



        Returns:

            float: probability density function output

        """


        return (1.0 / (self.stdev * math.sqrt(2*math.pi))) * math.exp(-
0.5*((x - self.mean) / self.stdev) ** 2)



    def plot_histogram_pdf(self, n_spaces = 50):


        """Function to plot the normalized histogram of the data and a plot
of the

        probability density function along the same range
```

```
Args:

        n_spaces (int): number of data points


Returns:

        list: x values for the pdf plot

        list: y values for the pdf plot


"""


mu = self.mean

sigma = self.stdev


min_range = min(self.data)

max_range = max(self.data)


 # calculates the interval between x values

interval = 1.0 * (max_range - min_range) / n_spaces
```

```
        x = []
        y = []

        # calculate the x values to visualize
        for i in range(n_spaces):
            tmp = min_range + interval*i
            x.append(tmp)
            y.append(self.pdf(tmp))

        # make the plots
        fig, axes = plt.subplots(2,sharex=True)
        fig.subplots_adjust(hspace=.5)
        axes[0].hist(self.data, density=True)
        axes[0].set_title('Normed Histogram of Data')
        axes[0].set_ylabel('Density')

        axes[1].plot(x, y)
        axes[1].set_title('Normal Distribution for \n Sample Mean and
Sample Standard Deviation')
```

```
        axes[0].set_ylabel('Density')

        plt.show()


        return x, y


def __add__(self, other):


        """Function to add together two Gaussian distributions


        Args:

                other (Gaussian): Gaussian instance


        Returns:

                Gaussian: Gaussian distribution


        """


        result = Gaussian()
```

```
        result.mean = self.mean + other.mean
        result.stdev = math.sqrt(self.stdev ** 2 + other.stdev ** 2)

        return result



def __repr__(self):

        """Function to output the characteristics of the Gaussian instance

        Args:
            None

        Returns:
            string: characteristics of the Gaussian

        """
```

```
        return "mean {}, standard deviation {}".format(self.mean,
self.stdev)
```

Bionomialdistribution.py

```
import math
import matplotlib.pyplot as plt
from .Generaldistribution import Distribution
class Binomial(Distribution):
    """ Binomial distribution class for calculating and
    visualizing a Binomial distribution.
    Attributes:
        mean (float) representing the mean value of the distribution
        stdev (float) representing the standard deviation of the distribution
        data_list (list of floats) a list of floats to be extracted from the data file
        p (float) representing the probability of an event occurring
        n (int) number of trials
    """
```

```
def __init__(self, prob=.5, size=20):
    self.n = size
    self.p = prob
    Distribution.__init__(self, self.calculate_mean(), self.calculate_stdev())
   def calculate_mean(self):
    """Function to calculate the mean from p and n
    Args:
        None
    Returns:
        float: mean of the data set
    """
    self.mean = self.p * self.n
    return self.mean
def calculate_stdev(self):
    """Function to calculate the standard deviation from p and n.
           Args:
        None
    Returns:
```

```
        float: standard deviation of the data set

    """

    self.stdev = math.sqrt(self.n * self.p * (1 - self.p))

    return self.stdev

def replace_stats_with_data(self):

    """Function to calculate p and n from the data set

          Args:  None

    Returns:

        float: the p value

        float: the n value


    """

    self.n = len(self.data)

    self.p = 1.0 * sum(self.data) / len(self.data)

    self.mean = self.calculate_mean()

    self.stdev = self.calculate_stdev()


    return self.p, self.n
```

```
def plot_bar(self):
    """Function to output a histogram of the instance variable data using
    matplotlib pyplot library.
    Args:
        None
    Returns:
        None
    """

    plt.bar(x = ['0', '1'], height = [(1 - self.p) * self.n, self.p * self.n])
    plt.title('Bar Chart of Data')
    plt.xlabel('outcome')
    plt.ylabel('count')

def pdf(self, k):
    """Probability density function calculator for the binomial distribution.

    Args:
```

```
            x (float): point for calculating the probability density function



        Returns:

            float: probability density function output

        """



        a = math.factorial(self.n) / (math.factorial(k) * (math.factorial(self.n -
k)))

        b = (self.p ** k) * (1 - self.p) ** (self.n - k)

        return a * b

    def plot_bar_pdf(self):



        """Function to plot the pdf of the binomial distribution

        Args:

            None

        Returns:

            list: x values for the pdf plot

            list: y values for the pdf plot
```

```
"""

x = []

y = []

# calculate the x values to visualize

for i in range(self.n + 1):

    x.append(i)

    y.append(self.pdf(i))

# make the plots

plt.bar(x, y)

plt.title('Distribution of Outcomes')

plt.ylabel('Probability')

plt.xlabel('Outcome')

plt.show()

return x, y
```

```
def __add__(self, other):

    """Function to add together two Binomial distributions with equal p

    Args:
        other (Binomial): Binomial instance

    Returns:
        Binomial: Binomial distribution
    """
    try:
        assert self.p == other.p, 'p values are not equal'
    except AssertionError as error:
        raise
    result = Binomial()
    result.n = self.n + other.n
    result.p = self.p
```

```
        result.calculate_mean()
        result.calculate_stdev()
        return result
    def __repr__(self):

        """Function to output the characteristics of the Binomial instance
        Args:
            None
        Returns:
            string: characteristics of the Gaussian

        "
        return "mean {}, standard deviation {}, p {}, n {}".\
        format(self.mean, self.stdev, self.p, self.n)
```

test.py

```
# Any changes to the distributions library should be reinstalled with

#  pip install --upgrade .



# For running unit tests, use

# /usr/bin/python -m unittest test



import unittest



from distributions import Gaussian

from distributions import Binomial



class TestGaussianClass(unittest.TestCase):

    def setUp(self):

        self.gaussian = Gaussian(25, 2)

        self.gaussian.read_data_file('numbers.txt')



    def test_initialization(self):
```

```
        self.assertEqual(self.gaussian.mean, 25, 'incorrect mean')
        self.assertEqual(self.gaussian.stdev, 2, 'incorrect standard deviation')

    def test_readdata(self):
        self.assertEqual(self.gaussian.data,\
         [1, 3, 99, 100, 120, 32, 330, 23, 76, 44, 31], 'data not read in correctly')

    def test_meancalculation(self):
        self.assertEqual(self.gaussian.calculate_mean(),\
         sum(self.gaussian.data) / float(len(self.gaussian.data)), 'calculated
mean not as expected')

    def test_stdevcalculation(self):
        self.assertEqual(round(self.gaussian.calculate_stdev(), 2), 92.87,
'sample standard deviation incorrect')
        self.assertEqual(round(self.gaussian.calculate_stdev(0), 2), 88.55,
'population standard deviation incorrect')

    def test_pdf(self):
```

```
        self.assertEqual(round(self.gaussian.pdf(25), 5), 0.19947,\
         'pdf function does not give expected result')
        self.gaussian.calculate_mean()
        self.gaussian.calculate_stdev()
        self.assertEqual(round(self.gaussian.pdf(75), 5), 0.00429,\
        'pdf function after calculating mean and stdev does not give expected result')


    def test_add(self):
        gaussian_one = Gaussian(25, 3)
        gaussian_two = Gaussian(30, 4)
        gaussian_sum = gaussian_one + gaussian_two

        self.assertEqual(gaussian_sum.mean, 55)
        self.assertEqual(gaussian_sum.stdev, 5)


class TestBinomialClass(unittest.TestCase):
    def setUp(self):
        self.binomial = Binomial(0.4, 20)
```

```
    self.binomial.read_data_file('numbers_binomial.txt')


def test_initialization(self):
    self.assertEqual(self.binomial.p, 0.4, 'p value incorrect')
    self.assertEqual(self.binomial.n, 20, 'n value incorrect')


def test_readdata(self):
    self.assertEqual(self.binomial.data,\
     [0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0], 'data not read in correctly')


def test_calculatemean(self):
    mean = self.binomial.calculate_mean()
    self.assertEqual(mean, 8)


def test_calculatestdev(self):
    stdev = self.binomial.calculate_stdev()
    self.assertEqual(round(stdev,2), 2.19)
```

```
def test_replace_stats_with_data(self):
    p, n = self.binomial.replace_stats_with_data()
    self.assertEqual(round(p,3), .615)
    self.assertEqual(n, 13)

def test_pdf(self):
    self.assertEqual(round(self.binomial.pdf(5), 5), 0.07465)
    self.assertEqual(round(self.binomial.pdf(3), 5), 0.01235)

    self.binomial.replace_stats_with_data()
    self.assertEqual(round(self.binomial.pdf(5), 5), 0.05439)
    self.assertEqual(round(self.binomial.pdf(3), 5), 0.00472)

def test_add(self):
    binomial_one = Binomial(.4, 20)
    binomial_two = Binomial(.4, 60)
    binomial_sum = binomial_one + binomial_two
```

```
        self.assertEqual(binomial_sum.p, .4)

        self.assertEqual(binomial_sum.n, 80)

if __name__ == '__import__main':

    unittest.main()
```

setup.py

```
    from setuptools import setup

    setup(name='distributions',
          version='0.1',
          description='Gaussian distributions',
          packages=['distributions'],
          zip_safe=False)
```

requirement.txt

```
distributions~=0.1

matplotlib~=3.5.2

setuptools~=58.1.0
```

6.2. Input:-

numbers.txt

1

3

99

100

120

32

330

23

76

44

3

numbers_bionomial.txt

0

1

1

1

1

1

0

1

0

1

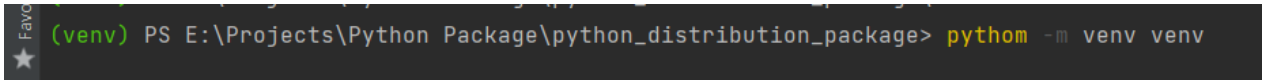
0

1

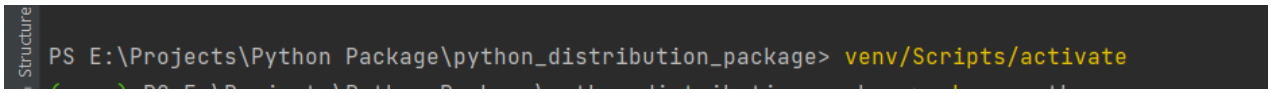
0

6.3. Output:-

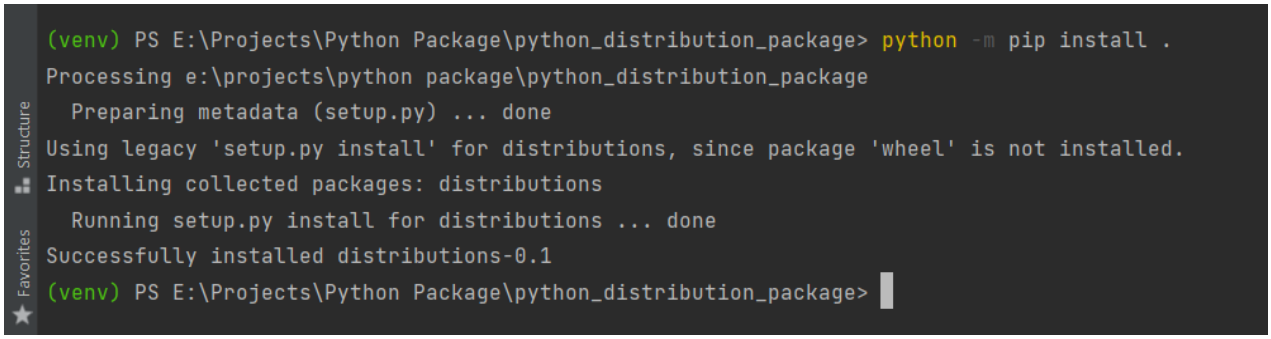
Step 1: In this step, we create a virtual environment



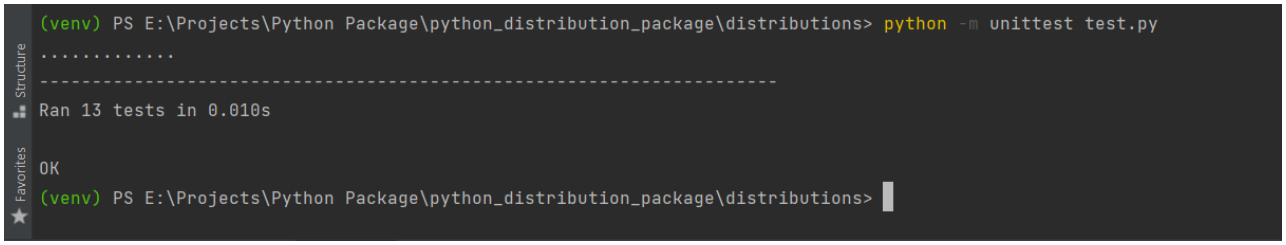
Step 2: In this step, we activate the created virtual environment



Step 3: Installing the distribution package on virtual environment



Step 4: Running the test case file to check the functionality of the package



6.4. STEPS TO INSTALL AND RUN THE PACKAGE

Step 1: In this step, we create a virtual environment

python –m venv venv

Step 2: In this step, we activate the created virtual environment

venv/Scripts/activate

Step 3: Installing the distribution package on virtual environme

python –m pip install .

Step 4: Running the test case file to check the functionality of the package

Python –m unittest test.py

7. FUTURE SCOPE

Furthermore, you can add more classes related to Probability distribution to enhance the functionality of the proposed package that can be developed to handle or perform more operations related to probability distribution. Also this package can be uploaded to official python repository pypi so that it can be available globally to all users as there is no such package exists to perform operations related to probability distribution.

8. CONCLUSION

The main objective of this package is to perform the Gaussian and binomial Probability distribution operations on the given dataset. The proposed package can be achieved by developing appropriate methods for calculating mean, standard deviation, plotting graphs, etc. From the results of the project, by passing a test file for checking all the operations are correct we can come to a conclusion that the both the classes operations were successfully executed from the test cases passed. Gaussian and Binomial both classes have calculated the correct answers with respect to the operations they are performing and all test cases are also passed as well.

9. BIBLIOGRAPHY

- Binomial distribution - Wikipedia
- Normal distribution - Wikipedia
- Installing packages using pip and virtual environments
- pypi.org
- test.pypy.org
- https://docs.python.org/3/distutils/packageindex.html