**ABSTRACT**

Automated Results Bot is an application makes ease to the person who finds it to store and retrieve results with just a click. It becomes even more necessary for lecturers as it is a time saving app.

Our “Automated Results Bot” helps you in doing exactly that. Automated Results Bot is a Web Scraping application. An application like this collects data from a website on the internet and stores the information at a desired place where user can access it.

In this application, a results website is scraped, and the results related to any specified person are taken. Then, the person’s results in the sheet are served to the client in an organised way .

This is helpful for any user who wants to keep a track of their results or to keep a record of them. It saves the time of the student by collecting all the important information. This project is coded in Python using several modules and designed by PyQtDesigner.

**TABLE OF CONTENTS**

1. Introduction........................................................................................................1
   1. Overview..........................................................................................1
   2. Motivation………………………………………………………....1
   3. ProblemDefinition............................................................................1
   4. Objectives………………………………………………….………2

2. SystemDesign…………………………………………… 3

3. PseudoAlgorithm/Implementation………………………… 6

4. Results ……… 20

5. Conclusion & Future Work …………………………. 23

6. References.23

**INTRODUCTION**

What’s Automated Results Bot ? …

* Automated Results Bot is an application of automation and web scraping combined together to store results of a desired class in an organised way.

**OVERVIEW**

A Selenium based project which scraps a Results website and via a web page enables the user to access all the details.

**MOTIVATION**

* In Our mini project we try to place all the related information of a selected person accessible by any individual effectively.
* The end user should get or access the results information very easily and use it for specific things.

**PROBLEM DEFINATION**

Prepare a Selenium Based project error-free which creates excel sheets which has the data using python programming language and scrape the wed data by the help of selenium modules and several others too.

**OBJECTIVES**

* Project UI.
* A website to scrape.
* Excel modules to store and modify information.
* A web scraper build to get the data from College Results website.

**System Design**

* Our project completely deals and revolves around the vital concept ***“*Web Scraping*”*** which means  the process of using bots to extract content and data from a **website**. Then presenting the data in one location.
* Data scraping is a technique where a computer program extracts data from human-readable output coming from another program.

1. **Selenium Framework – Usage of chromedriver and scraping with automation.**

**Theory:**

Selenium is a portable framework for testing web applications. Selenium provides a playback tool for authoring functional tests without the need to learn a test scripting language.

Selenium Python bindings provides a simple API to write functional/acceptance tests using Selenium WebDriver. Through Selenium Python API you can access all functionalities of Selenium WebDriver in an intuitive way.

Selenium Python bindings provide a convenient API to access Selenium WebDrivers like Firefox, Ie, Chrome, Remote etc. The current supported Python versions are 3.5 and above.

The “Automated Results Bot” folder is just the project folder, it actually contains four files –

* **AutomatedResultsBot.py** − This file has all the UI design which earlier had an .ui extension then converted to .py thereafter the full development has been done.
* **main.py** **–** Execution of the given data and scraping is done in this file.
* **Roll\_No.xlsx**- This file has all the roll numbers of the particular college which are used in the project.
* **Results\_File\_Name.xlsx-** An excel file which stores all the gathered information based on the options selected by the user.

**Installation of selenium:**

*Pip install selenium*

**Instructions for Windows users**

Install Python 3 using the MSI available in python.org download page.

Start a command prompt using the cmd.exe program and run the pip command as given below to install selenium.

C:\Python39\Scripts\pip.exe install selenium

Now you can run your test scripts using Python. For example, if you have created a Selenium based script and saved it inside C:\my\_selenium\_script.py, you can run it like this:

C:\Python39\python.exe C:\my\_selenium\_script.py

**Drivers**

Selenium requires a driver to interface with the chosen browser. Firefox, for example, requires geckodriver, which needs to be installed before the below examples can be run. Make sure it’s in your PATH, e. g., place it in /usr/bin or /usr/local/bin.

Failure to observe this step will give you an error selenium.common.exceptions.WebDriverException: Message: ‘geckodriver’ executable needs to be in PATH.

Other supported browsers will have their own drivers available. Links to some of the more popular browser drivers follow.

Chrome: <https://sites.google.com/a/chromium.org/chromedriver/downloads>

Edge: <https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/>

Firefox: <https://github.com/mozilla/geckodriver/releases>

Safari: <https://webkit.org/blog/6900/webdriver-support-in-safari-10/>

**Simple Usage**

If you have installed Selenium Python bindings, you can start using it from Python like this.

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

driver = webdriver.Firefox()

driver.get("http://www.python.org")

assert "Python" in driver.title

elem = driver.find\_element\_by\_name("q")

elem.clear()

elem.send\_keys("pycon")

elem.send\_keys(Keys.RETURN)

assert "No results found." not in driver.page\_source

driver.close()

The above script can be saved into a file (eg:- python\_org\_search.py), then it can be run like this:

python python\_org\_search.py

The python which you are running should have the selenium module installed.

**Interacting with the page**

Just being able to go to places isn’t terribly useful. What we’d really like to do is to interact with the pages, or, more specifically, the HTML elements within a page. First of all, we need to find one. WebDriver offers a number of ways to find elements. For example, given an element defined as:

<input type="text" name="passwd" id="passwd-id" />

you could find it using any of:

element = driver.find\_element\_by\_id("passwd-id")

element = driver.find\_element\_by\_name("passwd")

element = driver.find\_element\_by\_xpath("//input[@id='passwd-id']")

element = driver.find\_element\_by\_css\_selector("input#passwd-id")

You can also look for a link by its text, but be careful! The text must be an exact match! You should also be careful when using XPATH in WebDriver. If there’s more than one element that matches the query, then only the first will be returned. If nothing can be found, a NoSuchElementException will be raised.

WebDriver has an “Object-based” API; we represent all types of elements using the same interface. This means that although you may see a lot of possible methods you could invoke when you hit your IDE’s auto-complete key combination, not all of them will make sense or be valid. Don’t worry! WebDriver will attempt to do the Right Thing, and if you call a method that makes no sense (“setSelected()” on a “meta” tag, for example) an exception will be raised.

So, you’ve got an element. What can you do with it? First of all, you may want to enter some text into a text field:

element.send\_keys("some text")

You can simulate pressing the arrow keys by using the “Keys” class:

element.send\_keys(" and some", Keys.ARROW\_DOWN)

It is possible to call send\_keys on any element, which makes it possible to test keyboard shortcuts such as those used on GMail. A side-effect of this is that typing something into a text field won’t automatically clear it. Instead, what you type will be appended to what’s already there. You can easily clear the contents of a text field or textarea with the clear method:

element.clear()

**Locating Elements**

There are various strategies to locate elements in a page. You can use the most appropriate one for your case. Selenium provides the following methods to locate elements in a page:

find\_element\_by\_id

find\_element\_by\_name

find\_element\_by\_xpath

find\_element\_by\_link\_text

find\_element\_by\_partial\_link\_text

find\_element\_by\_tag\_name

find\_element\_by\_class\_name

find\_element\_by\_css\_selector

**Locating by XPath**

XPath is the language used for locating nodes in an XML document. As HTML can be an implementation of XML (XHTML), Selenium users can leverage this powerful language to target elements in their web applications. XPath supports the simple methods of locating by id or name attributes and extends them by opening up all sorts of new possibilities such as locating the third checkbox on the page.

One of the main reasons for using XPath is when you don’t have a suitable id or name attribute for the element you wish to locate. You can use XPath to either locate the element in absolute terms (not advised), or relative to an element that does have an id or name attribute. XPath locators can also be used to specify elements via attributes other than id and name.

Absolute XPaths contain the location of all elements from the root (html) and as a result are likely to fail with only the slightest adjustment to the application. By finding a nearby element with an id or name attribute (ideally a parent element) you can locate your target element based on the relationship. This is much less likely to change and can make your tests more robust.

For instance, consider this page source:

<html>

<body>

<form id="loginForm">

<input name="username" type="text" />

<input name="password" type="password" />

<input name="continue" type="submit" value="Login" />

<input name="continue" type="button" value="Clear" />

</form>

</body>

</html>

The form elements can be located like this:

login\_form = driver.find\_element\_by\_xpath("/html/body/form[1]")

login\_form = driver.find\_element\_by\_xpath("//form[1]")

login\_form = driver.find\_element\_by\_xpath("//form[@id='loginForm']")

**Chrome WebDriver**

class selenium.webdriver.chrome.webdriver.WebDriver(executable\_path='chromedriver', port=0, options=None, service\_args=None, desired\_capabilities=None, service\_log\_path=None, chrome\_options=None, keep\_alive=True)

Bases: selenium.webdriver.remote.webdriver.WebDriver

Controls the ChromeDriver and allows you to drive the browser.

You will need to download the ChromeDriver executable from http://chromedriver.storage.googleapis.com/index.html

\_\_init\_\_(executable\_path='chromedriver', port=0, options=None, service\_args=None, desired\_capabilities=None, service\_log\_path=None, chrome\_options=None, keep\_alive=True)

Creates a new instance of the chrome driver.

Starts the service and then creates new instance of chrome driver.

Args:

executable\_path - path to the executable. If the default is used it assumes the executable is in the $PATH

port - port you would like the service to run, if left as 0, a free port will be found.

options - this takes an instance of ChromeOptions

service\_args - List of args to pass to the driver service

desired\_capabilities - Dictionary object with non-browser specific capabilities only, such as “proxy” or “loggingPref”.

service\_log\_path - Where to log information from the driver.

chrome\_options - Deprecated argument for options

keep\_alive - Whether to configure ChromeRemoteConnection to use HTTP keep-alive.

**(ii) PyQt5 – Development of the UI**

**Theory: PyQt5** is one of the most **used** modules in building GUI apps in Python, and that's due to its simplicity as you will see. Another great feature that encourages developers to **use PyQt5** is the **PyQt5** designer, which makes it so easy to develop complex GUI apps in a short time. You just drag your widgets to build your form.

**Installation:**

First, we need to install PyQt5 library. For this, type the following command in the terminal or command prompt:

*pip install pyqt5*

If successfully installed one can verify it by running the code:

>>>*import PyQt5*

PyQt5 provides lots of tools and QtDesigner is one of them. For this run this command:

*pip install PyQt5-tools*

**PyQt5 Components**

PyQt5 comprises a number of different components. First of all there are a number of Python extension modules. These are all installed in the PyQt5 Python package and are described in the list of modules.

Each extension module has a corresponding PEP 484 defined stub file containing type hints for the module’s API. This can be used by static type checkers such as mypy.

PyQt5 contains plugins that enable Qt Designer and qmlscene to be extended using Python code. See Writing Qt Designer Plugins and Integrating Python and QML respectively for the details.

PyQt5 also contains a number of utility programs.

pyuic5 corresponds to the Qt uic utility. It converts QtWidgets based GUIs created using Qt Designer to Python code.

pyrcc5 corresponds to the Qt rcc utility. It embeds arbitrary resources (eg. icons, images, translation files) described by a resource collection file in a Python module.

pylupdate5 corresponds to the Qt lupdate utility. It extracts all of the translatable strings from Python code and creates or updates .ts translation files. These are then used by Qt Linguist to manage the translation of those strings.

**Converting .ui file into .py file:**

For this we have to go to sitpackages directory in terminal or command prompt and run the command as shown below. Getting the location of sitepackages is mentioned previously.

>>> cd “C:\\Users\\……\\Programs\\Python\\Python36-32\\lib\\site-packages” [Location of sitepackages]

>>> pyuic5 “C:\Users\……\FILENAME.ui”[Exact location of .ui file] -o ” C:\Users\…….\FILENAME.py” [Location where want to put .py file]

**Using Qt Designer**

Qt Designer is the Qt tool for designing and building graphical user interfaces. It allows you to design widgets, dialogs or complete main windows using on-screen forms and a simple drag-and-drop interface. It has the ability to preview your designs to ensure they work as you intended, and to allow you to prototype them with your users, before you have to write any code.

Qt Designer uses XML .ui files to store designs and does not generate any code itself. Qt includes the uic utility that generates the C++ code that creates the user interface. Qt also includes the QUiLoader class that allows an application to load a .ui file and to create the corresponding user interface dynamically.

PyQt5 does not wrap the QUiLoader class but instead includes the uic Python module. Like QUiLoader this module can load .ui files to create a user interface dynamically. Like the uic utility it can also generate the Python code that will create the user interface. PyQt5’s pyuic5 utility is a command line interface to the uic module. Both are described in detail in the following sections.

**(iii) openpyxl - openpyxl** is a Python library to read/write Excel 2010 xlsx/xlsm/xltx/xltm files. It was born from lack of existing library to read/write natively from Python the Office Open XML format.

**Openpyxl Working Process**

The Openpyxl library is used to write or read the data in the excel file and many other tasks. An excel file that we use for operation is called Workbook that contains a minimum of one Sheet and a maximum of tens of sheets.

Sheets consist of Rows (horizontal series) starting from 1 and Columns (vertical series) starting from A.

Row and column together make a grid and form a cell that may store some data. Data can be of any type, such as numeric, string.

Openpyxl provides flexibility to read data from the individual cell or write data to it.

**Openpyxl Read Data from cell**

We can read the data that we have previously written in the cell. There are two methods to read a cell, firstly we can access it by cell name, and secondly, we can access it by the cell() function. For example, we are reading the data from the sample\_file.xlrs file.

import openpyxl

wb = openpyxl.load\_workbook('sample\_file.xlsx')

sheet = wb.active

x1 = sheet['A1']

x2 = sheet['A2']

#using cell() function

x3 = sheet.cell(row=3, column=1)

print("The first cell value:",x1.value)

print("The second cell value:",x2.value)

print("The third cell value:",x3.value)

**(iv) xlsxwriter - XlsxWriter** can be used to write text, numbers, formulas and hyperlinks to multiple worksheets and it supports features such as formatting and many more, including: 100% compatible Excel XLSX files.

**Installing XlsxWriter**

The first step is to install the XlsxWriter module. There are several ways to do this.

Using PIP

The pip installer is the preferred method for installing Python modules from PyPI, the Python Package Index:

$ pip install XlsxWriter

# Or to a non system dir:

$ pip install --user XlsxWriter

**Create a simple XLSX file**

Let’s start by creating a simple spreadsheet using Python and the XlsxWriter module.

Say that we have some data on monthly outgoings that we want to convert into an Excel XLSX file:

expenses = (

['Rent', 1000],

['Gas', 100],

['Food', 300],

['Gym', 50],

)

To do that we can start with a small program like the following:

import xlsxwriter

# Create a workbook and add a worksheet.

workbook = xlsxwriter.Workbook('Expenses01.xlsx')

worksheet = workbook.add\_worksheet()

# Some data we want to write to the worksheet.

expenses = (

['Rent', 1000],

['Gas', 100],

['Food', 300],

['Gym', 50],

)

# Start from the first cell. Rows and columns are zero indexed.

row = 0

col = 0

# Iterate over the data and write it out row by row.

for item, cost in (expenses):

worksheet.write(row, col, item)

worksheet.write(row, col + 1, cost)

row += 1

# Write a total using a formula.

worksheet.write(row, 0, 'Total')

worksheet.write(row, 1, '=SUM(B1:B4)')

workbook.close()

**Workbook Class**

The Workbook class is the main class exposed by the XlsxWriter module and it is the only class that you will need to instantiate directly.

The Workbook class represents the entire spreadsheet as you see it in Excel and internally it represents the Excel file as it is written on disk.

Constructor

Workbook(filename[, options])

Create a ew XlsxWriter Workbook object.

Parameters:

filename (string) – The name of the new Excel file to create.

options (dict) – Optional workbook parameters. See below.

Return type:

A Workbook object.

The Workbook() constructor is used to create a new Excel workbook with a given filename:

import xlsxwriter

workbook = xlsxwriter.Workbook('filename.xlsx')

worksheet = workbook.add\_worksheet()

worksheet.write(0, 0, 'Hello Excel')

workbook.close()

**Pseudo Algorithm and Implementation**

**UI CODE:**

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(529, 303)

MainWindow.setMouseTracking(False)

MainWindow.setWindowTitle("AutomatedResultsBot")

MainWindow.setStyleSheet("color: rgb(0, 255, 0);")

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.pushButton = QtWidgets.QPushButton(self.centralwidget)

self.pushButton.setGeometry(QtCore.QRect(230, 200, 75, 23))

self.pushButton.setObjectName("pushButton")

self.pushButton.setStyleSheet("""

QPushButton

{

transition: all .5s ease;

color:#e7eeef;

border: 3px solid white;

font-family:'Montserrat', sans-serif;

text-transform: uppercase;

text-align: center;

line-height: 1;

font-size: 12px;

font-weight:bold;

background-color : #1ba94c;

padding: 1px;

outline: none;

border-radius: 2px;

border-color:#FFFFFF;

}

QPushButton::hover

{

color: #1ba94c;

background-color: #e7eeef;

font-weight:bold;

border-radius:1px;

border-color: #1ba94c;

}

QPushButton::focus

{

box-shadow: 0 0.5em 0.5em -0.4em var(--hover);

transform: translateY(-0.25em);

}

"""

)

self.comboBox = QtWidgets.QComboBox(self.centralwidget)

self.comboBox.setGeometry(QtCore.QRect(50, 110, 91, 22))

self.comboBox.setStyleSheet("bg-color:red;\n"

"color:#1ba94c ;\n"

"shadow:5px;\n"

"\n"

"")

self.comboBox.setObjectName("comboBox")

self.comboBox.addItem("")

self.comboBox.addItem("")

self.comboBox.addItem("")

self.comboBox.addItem("")

self.comboBox.addItem("")

self.comboBox.addItem("")

self.comboBox.setStyleSheet("QComboBox{color: rgb(255, 255, 255 );\n"

"background-color: #1ba94c;\n"

"background-image:url(background.png);}\n"

"text-align:center;}\n"

"QComboBox QAbstractItemView{border: 0px;color:black}")

self.comboBox\_2 = QtWidgets.QComboBox(self.centralwidget)

self.comboBox\_2.setGeometry(QtCore.QRect(170, 110, 91, 22))

self.comboBox\_2.setObjectName("comboBox\_2")

self.comboBox\_2.addItem("")

self.comboBox\_2.addItem("")

self.comboBox\_2.addItem("")

self.comboBox\_2.addItem("")

self.comboBox\_2.setStyleSheet("QComboBox{color: rgb(255, 255, 255 );\n"

"background-color: #1ba94c;\n"

"background-image:url(background.png);}\n"

"QComboBox QAbstractItemView{border: 0px;color:black}")

self.comboBox\_3 = QtWidgets.QComboBox(self.centralwidget)

self.comboBox\_3.setGeometry(QtCore.QRect(290, 110, 81, 22))

self.comboBox\_3.setObjectName("comboBox\_3")

self.comboBox\_3.addItem("")

self.comboBox\_3.addItem("")

self.comboBox\_3.setStyleSheet("QComboBox{color: rgb(255, 255, 255 );\n"

"background-color: #1ba94c;\n"

"background-image:url(background.png);}\n"

"QComboBox QAbstractItemView{border: 0px;color:#000000}")

self.comboBox\_4 = QtWidgets.QComboBox(self.centralwidget)

self.comboBox\_4.setGeometry(QtCore.QRect(400, 110, 81, 22))

self.comboBox\_4.setObjectName("comboBox\_4")

self.comboBox\_4.addItem("")

self.comboBox\_4.addItem("")

self.comboBox\_4.addItem("")

self.comboBox\_4.setStyleSheet("QComboBox{color: rgb(255, 255, 255 );\n"

"background-color: #1ba94c;\n"

"background-image:url(background.png);}\n"

"QComboBox QAbstractItemView{border: 0px;color:#000000}")

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(0, 0, 541, 41))

self.label.setStyleSheet("background-color: #1ba94c ;\n"

"color:#ffffff;\n"

"font-align:center;")

self.label.setObjectName("label")

self.label\_2 = QtWidgets.QLabel(self.centralwidget)

self.label\_2.setGeometry(QtCore.QRect(150, 10, 221, 20))

self.label\_2.setStyleSheet("color:#FFFFFF;")

self.label\_2.setObjectName("label\_2")

MainWindow.setCentralWidget(self.centralwidget)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

self.pushButton.setText(\_translate("MainWindow", "SUBMIT"))

self.comboBox.setItemText(0, \_translate("MainWindow", "CIV"))

self.comboBox.setItemText(1, \_translate("MainWindow", "CSE"))

self.comboBox.setItemText(2, \_translate("MainWindow", "EEE"))

self.comboBox.setItemText(3, \_translate("MainWindow", "ECE"))

self.comboBox.setItemText(4, \_translate("MainWindow", "MEC"))

self.comboBox.setItemText(5, \_translate("MainWindow", "IT"))

self.comboBox\_2.setItemText(0, \_translate("MainWindow", "1 YEAR"))

self.comboBox\_2.setItemText(1, \_translate("MainWindow", "2 YEAR"))

self.comboBox\_2.setItemText(2, \_translate("MainWindow", "3 YEAR"))

self.comboBox\_2.setItemText(3, \_translate("MainWindow", "4 YEAR"))

self.comboBox\_3.setItemText(0, \_translate("MainWindow", "1 SEM"))

self.comboBox\_3.setItemText(1, \_translate("MainWindow", "2 SEM"))

self.comboBox\_4.setItemText(0, \_translate("MainWindow", "A"))

self.comboBox\_4.setItemText(1, \_translate("MainWindow", "B"))

self.comboBox\_4.setItemText(2, \_translate("MainWindow", "C"))

self.label.setText(\_translate("MainWindow", "<html><head/><body><p><br/></p></body></html>"))

self.label\_2.setText(\_translate("MainWindow",

"<html><head/><body><p><span style=\" font-size:11pt; font-weight:600;\">Automated Results Bot</span></p><p><span style=\" font-size:11pt;\"><br/></span></p></body></html>"))

self.pushButton.clicked.connect(self.call)

def call(self):

branch = self.comboBox.currentText()

year = self.comboBox\_2.currentText()

sem = self.comboBox\_3.currentText()

section = self.comboBox\_4.currentText()

code = year[0] + '\_' + branch + '-' + section # 2\_CIV-A

filename = year + '\_' + branch + '\_' + sem + '-' + section # eg: 2Year\_CIV\_1SEM-A

# print(filename)

main.find\_sub(code)

main.collect(code, filename)

**EXECUTION CODE:**

from selenium import webdriver

import openpyxl as op

import xlsxwriter as xl

from selenium.common.exceptions import NoSuchElementException

from xlsxwriter import Workbook

wb = op.load\_workbook('Roll\_No.xlsx') # To open Roll Numbers Sheet

Sheet = wb['Sheet1']

subjects = [] # List for storing Subjects

count\_dict1 = {} # Dictionary for storing A+ Grades in each Subjects

count\_dict2 = {} # Dictionary for storing A Grades in each Subjects

Classes = [] # List for storing All Classes Names

colNum = 0

def find\_sub(code):

print("In find\_sub function")

for i in Sheet[1]: # To find the Total Classes

if i.value:

Classes.append(i.value)

print(Classes)

print(code)

colNum = Classes.index(code) + 1 # To find column number

print(colNum)

driver = webdriver.Chrome()

driver.get("https://sis.vce.ac.in/BE\_Results\_367\_Sem\_27-03-2021/")

searchBox = driver.find\_element\_by\_xpath('//\*[@id="txtHTNO"]')

searchButton = driver.find\_element\_by\_xpath('//\*[@id="btnResults"]')

roll = Sheet.cell(row=2, column=colNum).value

searchBox.send\_keys(roll)

searchButton.click()

def check\_exists\_by\_xpath(xpath): # To check xpath exists or not

try:

driver.find\_element\_by\_xpath(xpath)

except NoSuchElementException:

return False

return True

if check\_exists\_by\_xpath('//\*[@id="AutoNumber3"]/tbody/tr[17]/td[1]'):

num = 15

elif check\_exists\_by\_xpath('//\*[@id="AutoNumber3"]/tbody/tr[16]/td[1]'):

num = 14

elif check\_exists\_by\_xpath('//\*[@id="AutoNumber3"]/tbody/tr[15]/td[1]'):

num = 13

elif check\_exists\_by\_xpath('//\*[@id="AutoNumber3"]/tbody/tr[14]/td[1]'):

num = 12

elif check\_exists\_by\_xpath('//\*[@id="AutoNumber3"]/tbody/tr[13]/td[1]'):

num = 11

elif check\_exists\_by\_xpath('//\*[@id="AutoNumber3"]/tbody/tr[12]/td[1]'):

num = 10

elif check\_exists\_by\_xpath('//\*[@id="AutoNumber3"]/tbody/tr[11]/td[1]'):

num = 9

elif check\_exists\_by\_xpath('//\*[@id="AutoNumber3"]/tbody/tr[10]/td[1]'):

num = 8

elif check\_exists\_by\_xpath('//\*[@id="AutoNumber3"]/tbody/tr[9]/td[1]'):

num = 7

elif check\_exists\_by\_xpath('//\*[@id="AutoNumber3"]/tbody/tr[8]/td[1]'):

num = 6

else:

num = 5

st = '(//\*[@id="AutoNumber3"]/tbody/tr[3]/td[3])[2]' # Manually taking Subject 1&2 Names

sub\_box = driver.find\_element\_by\_xpath(st)

sub\_name = sub\_box.text

subjects.append(sub\_name)

st = '(//\*[@id="AutoNumber3"]/tbody/tr[4]/td[3])[2]'

sub\_box = driver.find\_element\_by\_xpath(st)

sub\_name = sub\_box.text

subjects.append(sub\_name)

for i in range(5, num + 3): # For getting Subject Names

v = str(i)

st = '(//\*[@id="AutoNumber3"]/tbody/tr[' + v + ']/td[3])'

sub\_box = driver.find\_element\_by\_xpath(st)

sub\_name = sub\_box.text

subjects.append(sub\_name)

print(subjects, "\n\n")

driver.close()

def collect(code, filename):

print('In collect function')

List\_Roll = [] # List to store Roll Numbers

file = str(filename) + '.xlsx' # Storing xlsx file name as String

workbook: Workbook = xl.Workbook(file) # Creating a xlsx File

ws = workbook.add\_worksheet() # Creating a WorkSheet

for i in Sheet[1]: # For Column Number

if i.value:

Classes.append(i.value)

colNum = Classes.index(code) + 1

print(colNum)

i = 0

while Sheet.cell(row=i + 1, column=colNum).value: # collects and stores roll numbers in List based on colNum

Roll\_No = Sheet.cell(row=i + 1, column=colNum).value

List\_Roll.append(Roll\_No)

i = i + 1

class\_length = len(List\_Roll) # To find class length to Iterate

print(class\_length)

for a in range(len(subjects)): # To initialize dictionary key(Subject\_Name):Value(0)

count\_dict1[subjects[a]] = 0 # For A+ Grades

count\_dict2[subjects[a]] = 0 # For A Grades

driver = webdriver.Chrome()

driver.get("https://sis.vce.ac.in/BE\_Results\_367\_Sem\_27-03-2021/")

ws.write(0, 0, 'Roll Number')

ws.write(0, 1, 'Name')

ws.write(0, 2, 'SGPA')

ws.write(0, 3, 'CGPA')

for i in range(1, class\_length): # For Iterating through number of students

searchBox = driver.find\_element\_by\_xpath('//\*[@id="txtHTNO"]')

searchButton = driver.find\_element\_by\_xpath('//\*[@id="btnResults"]')

Roll\_No = List\_Roll.\_\_getitem\_\_(i)

searchBox.send\_keys(Roll\_No)

searchButton.click()

name = driver.find\_element\_by\_xpath('//\*[@id="lblStudName"]').text

SGPA = driver.find\_element\_by\_xpath('//\*[@id="lblSGPa"]').text

CGPA = driver.find\_element\_by\_xpath('//\*[@id="lblCGPA"]').text

lis = list(SGPA.split(" ")) # For last string i.e., SGPA

lis2 = list(CGPA.split(" ")) # For last string i.e., CGPA

if lis[0] == '':

sgpa = 0

else:

sgpa = float(lis[6])

if lis2[0] == '':

cgpa = 0

else:

cgpa = float(lis2[6])

for j in range(len(subjects)): # For counting A+ & A Grades in each subjects

v = str(j + 3)

gd = '(//\*[@id="AutoNumber3"]/tbody/tr[' + v + ']/td[7])'

grade\_point = driver.find\_element\_by\_xpath(gd).text

if grade\_point == "A+":

count\_dict1[subjects[j]] = count\_dict1[subjects[j]] + 1

if grade\_point == "A":

count\_dict2[subjects[j]] = count\_dict2[subjects[j]] + 1

# For writing data into sheet

ws.write(i, 0, Roll\_No)

ws.write(i, 1, name)

ws.write(i, 2, sgpa)

ws.write(i, 3, cgpa)

# For Printing Details of each student

print(Roll\_No)

print(name)

print("SGPA : ", sgpa)

print("CGPA : ", cgpa)

print(count\_dict1)

print(count\_dict2, "\n\n")

driver.back()

driver.refresh()

# print('After Iterating N students Times')

def sub\_grade(): # For write number of A+ & A Grades in each Subject into sheet

length = class\_length + 10

ws.write(length, 0, 'Subjects')

ws.write(length, 1, 'Number of A+ Grades')

ws.write(length, 2, 'Number of A Grades')

for k in range(len(subjects)):

ws.write(k + length + 1, 0, subjects[k])

ws.write(k + length + 1, 1, count\_dict1[subjects[k]])

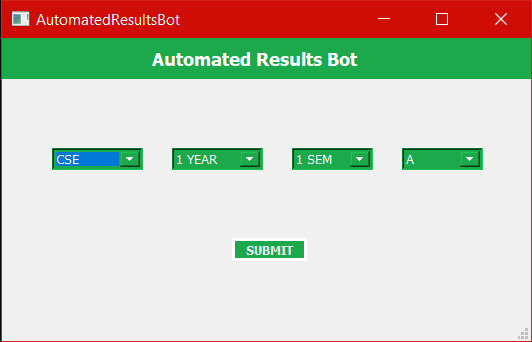
ws.write(k + length + 1, 2, count\_dict2[subjects[k]])

sub\_grade()

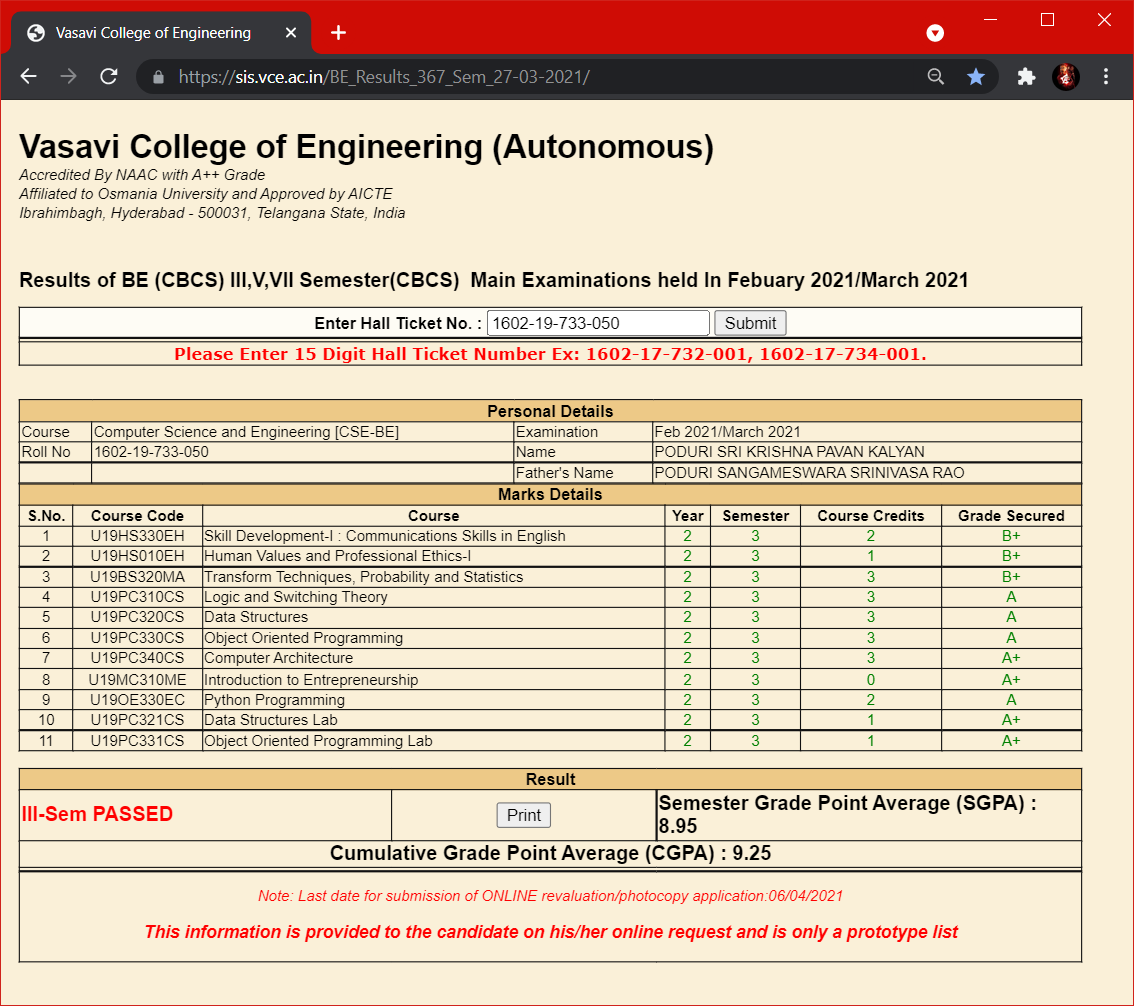
workbook.close()

driver.close()

**RESULTS**

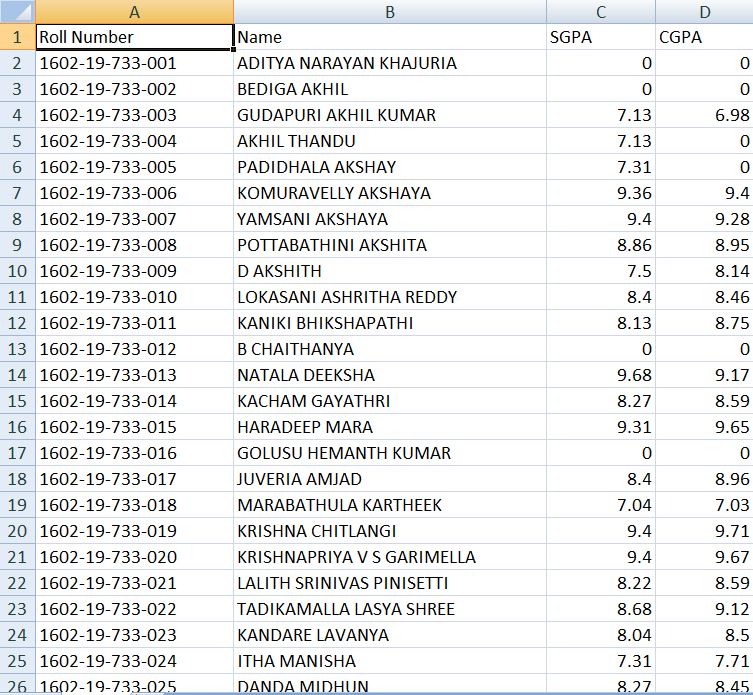


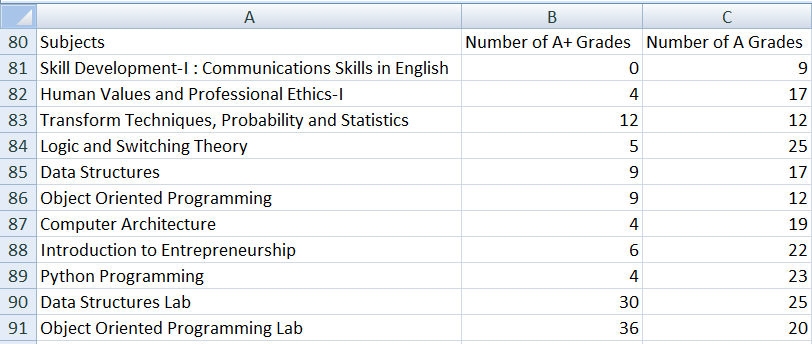
**SAMPLE RESULTS WEBSITE**



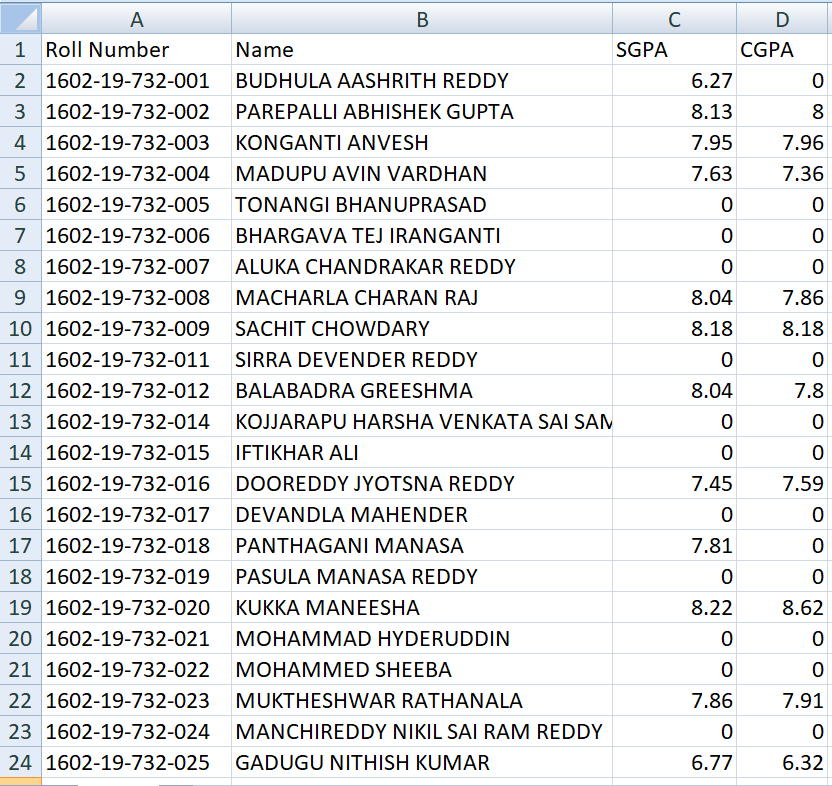
**END RESULTS**

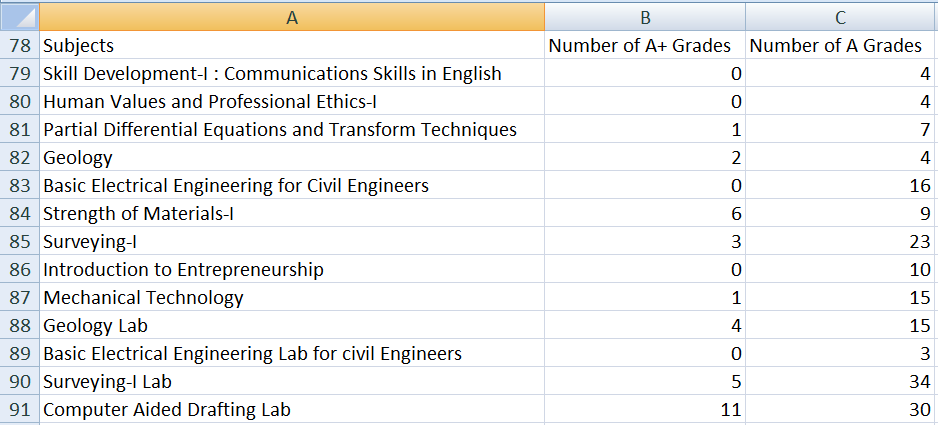
2-Year 1-SEM CSE-A RESULTS:



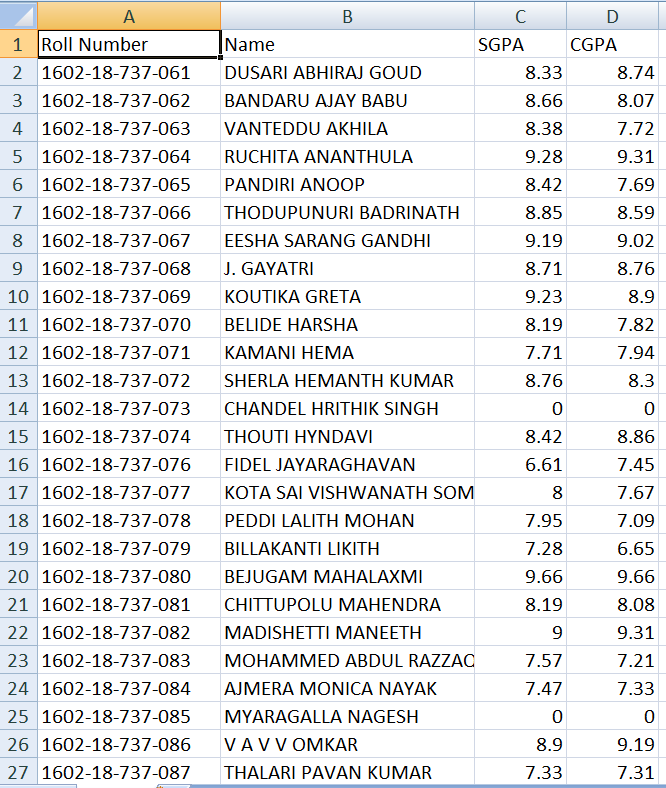


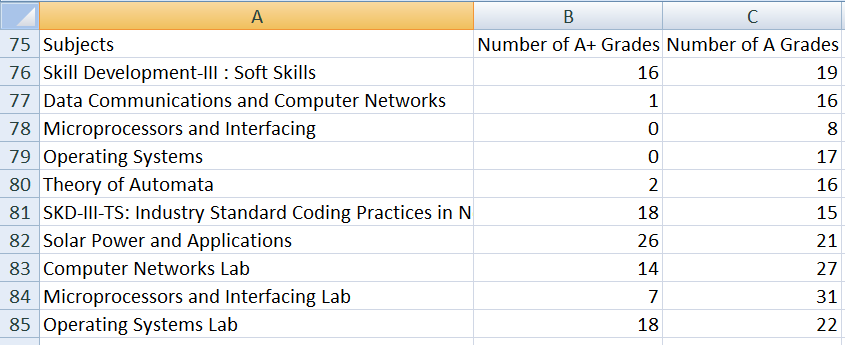
2-Year 1-SEM CIV-A RESULTS:

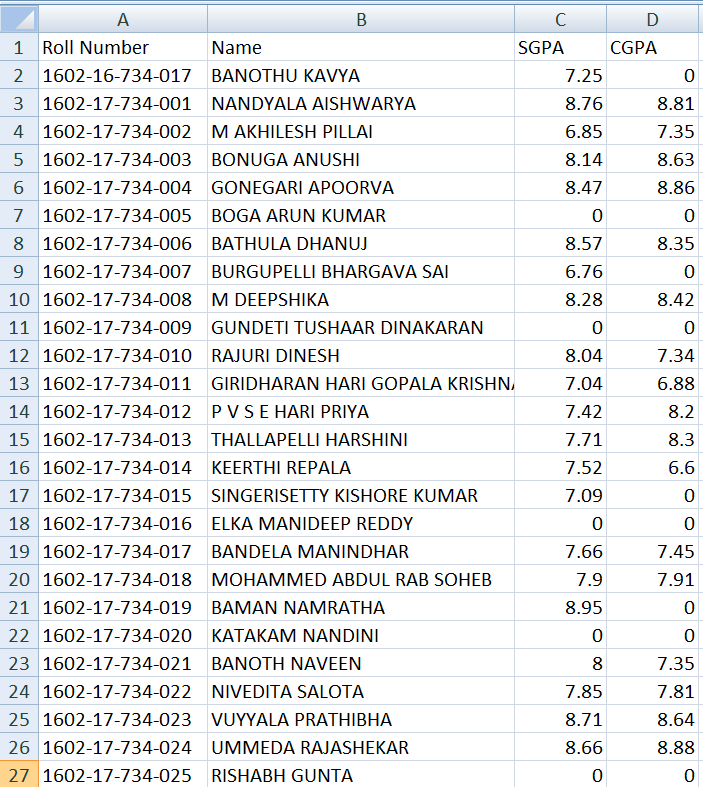


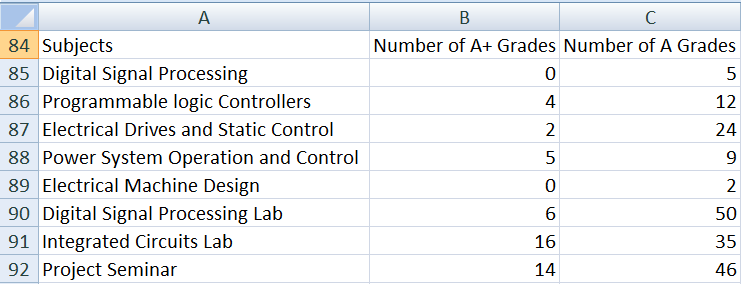


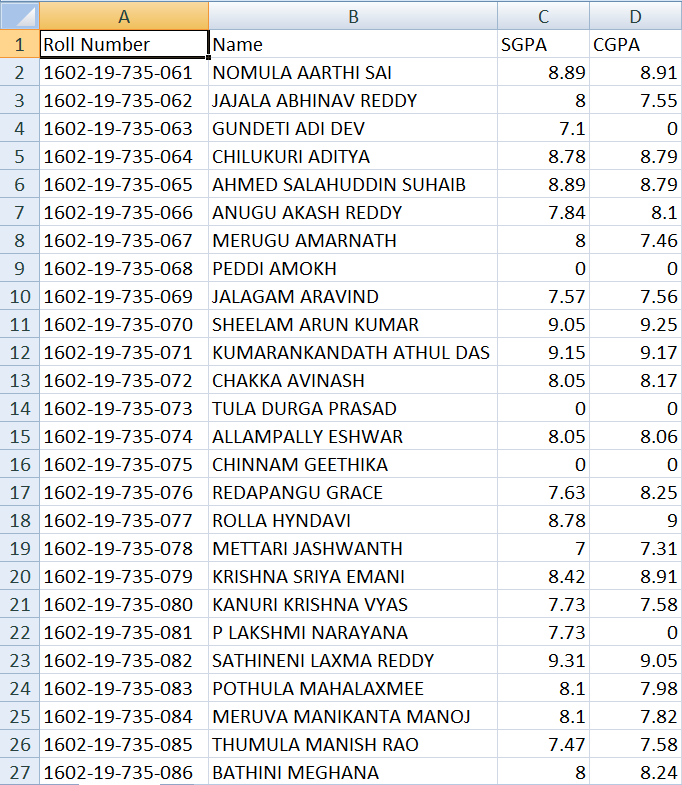
3-Year 1-SEM IT-B RESULTS:

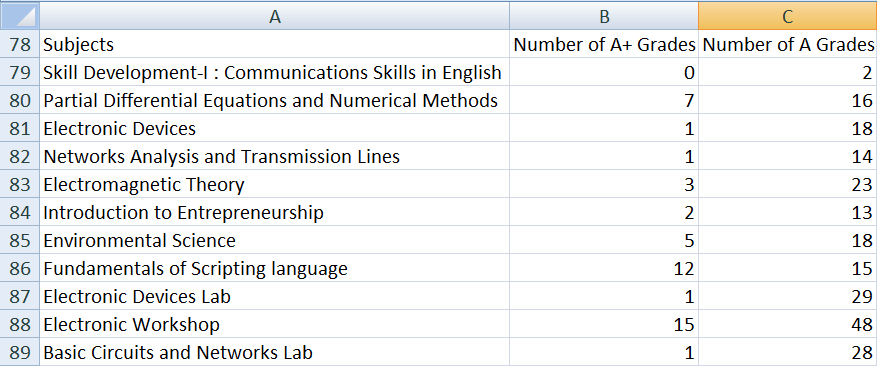


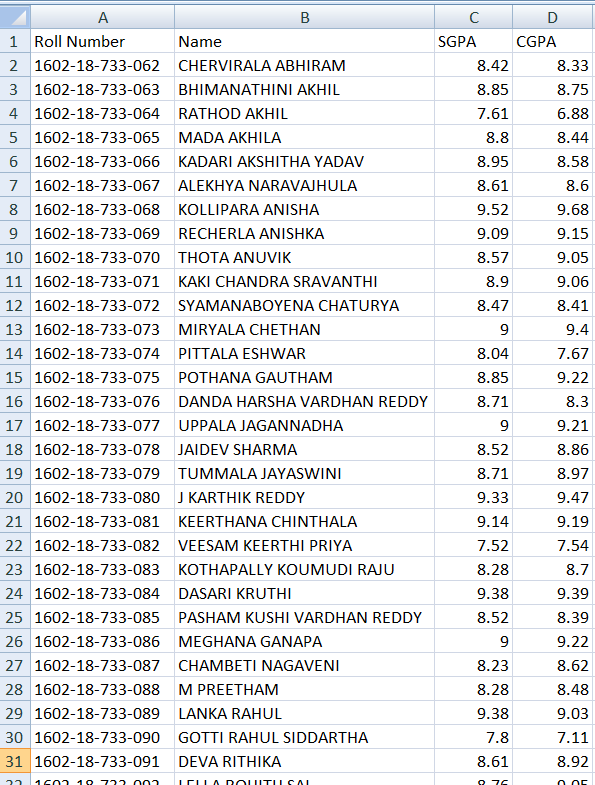


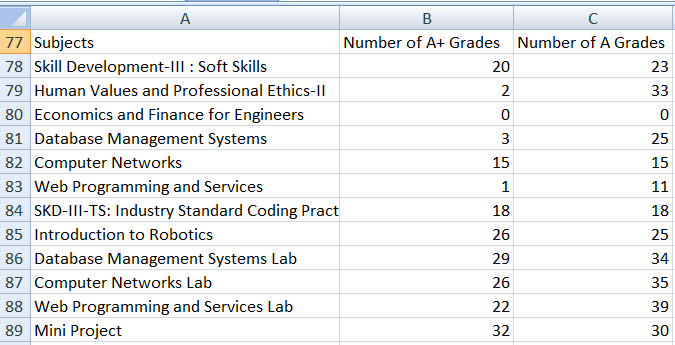
4-Year 1-SEM EEE-A RESULTS: 

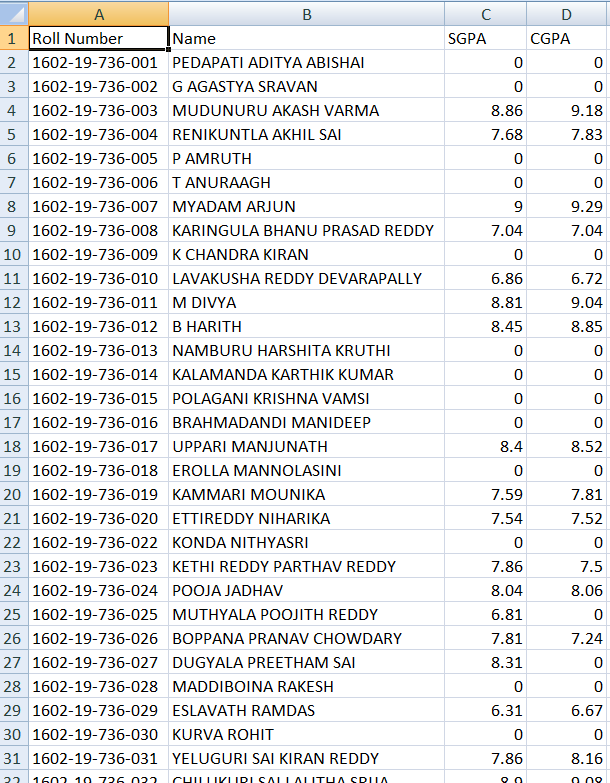


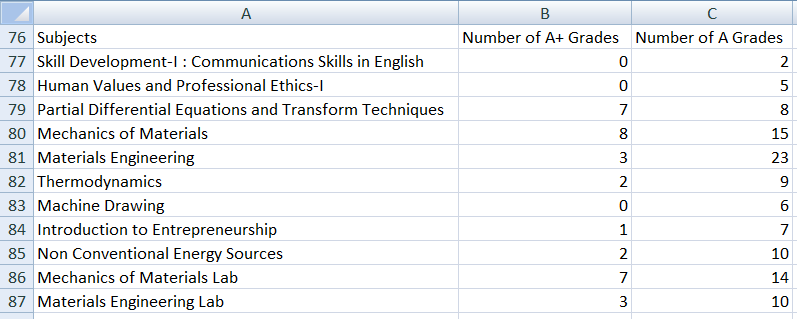
2-Year 1-SEM ECE-B RESULTS: 



3-Year 1-SEM CSE-B RESULTS: 



2-Year 1-SEM MEC-A RESULTS: 



**CONCLUSION AND FUTURE WORK**

As we built python a project which automates the simple task, it is very easy to use. Its Graphical User Interface makes users to access results in single touch.

Future works are,

* 1. At present it is static project works only for 3rd, 5th & 7th semester results. We will try to make it dynamic to change results links and work for any results.
  2. 1st year student’s results Roll Numbers should include in the Roll\_No.xlsx sheet.
  3. Can add extra features like Graph Representation of Marks in a excel sheet.

**REFERENCES**

1. Python – Documentation

(<https://docs.python.org/3/> )

2. Selenium – Documentation

(<https://selenium-python.readthedocs.io/>)

3. XlsxWriter – Documentation

(<https://xlsxwriter.readthedocs.io/>)

4. Openpyxl – Documentation

(<https://openpyxl.readthedocs.io/en/stable/>)