# WHAT IS A LIBRARY?

- A library is a collection of pre-combined codes that can be used iteratively to reduce the time required to code.
- They are particularly useful for accessing the pre-written frequently used codes, instead of writing them from scratch every single time.
- Similar to the physical libraries, these are a collection of reusable resources, which means every library has a root source.
- This is the foundation behind the numerous open-source libraries available in Python.

# TERMINOLOGY

- **Scripts**
- **Modules**
- **Packages**
- **Libraries**

# SCRIPT

- A **script** is a Python file that's intended to be run directly. When you run it, it should do *something*. This means that scripts will often contain code written outside the scope of any classes or functions.

# MODULE

- A **module** is a Python file that's intended to be imported into scripts or other modules. It often defines members like classes, functions, and variables intended to be used in other files that import it.

# PACKAGE

A **package** is a collection of related modules that work together to provide certain functionality. These modules are contained within a folder and can be imported just like any other modules. This folder will often contain a special __init__ file that tells Python it's a package, potentially containing more modules nested within subfolders

# LIBRARY

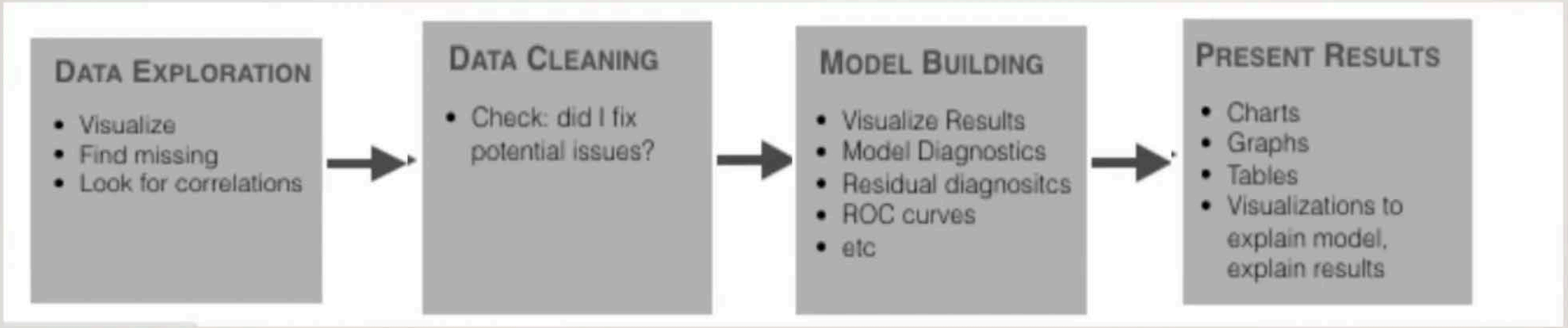
- A **library** is an umbrella term that loosely means “a bundle of code.”
- These can have tens or even hundreds of individual modules that can provide a wide range of functionality. Matplotlib is a plotting library.
- The Python Standard Library contains hundreds of modules for performing common tasks, like sending emails or reading JSON data.
- What’s special about the Standard Library is that it comes bundled with your installation of Python, so you can use its modules without having to download them from anywhere

# EXPLORATORY DATA ANALYSIS

DAMAN VIRDI

# EXPLORATORY DATA ANALYSIS

- Exploratory Data Analysis, or EDA, is essentially a type of storytelling for statisticians.
- It allows us to uncover patterns and insights, often with visual methods, within data. EDA is often the first step of the data modelling process
- . In this phase, data engineers have some questions in hand and try to validate those questions by performing EDA.
- EDA may sound exotic if you are new to the world of statistics.
- However, it's not actually very difficult to perform an EDA.
- Nor do you need to know special languages to do it.


DATA EXPLORATION
• Visualize
• Find missing
• Look for correlations
DATA CLEANING
• Check: did I fix potential issues?
MODEL BUILDING
• Visualize Results
• Model Diagnostics
• Residual diagnositcs
• ROC curves
• etc
PRESENT RESULTS
• Charts
• Graphs
• Tables
• Visualizations to explain model, explain results

```
pip install matplotlib
```

In [1]:

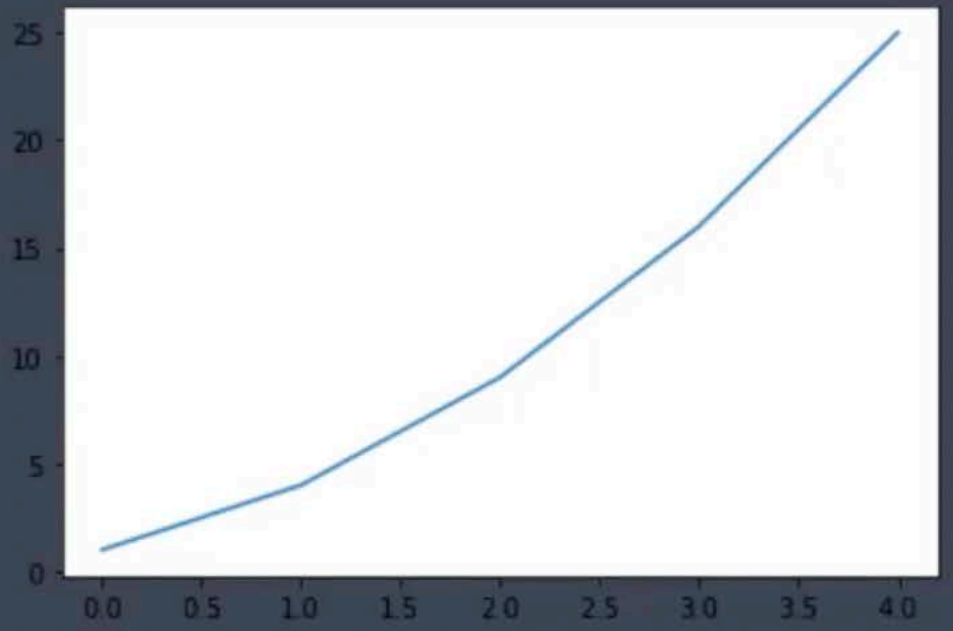
```
#importing matplotlib
import matplotlib.pyplot as plt
```

In [ ]:

```
#plotting a simple graph

list -[]
tuple -()
dict - {}
```

In [2]:

```
squares = [ 1,4,9,16,25]
plt.plot(squares)
plt.show()

#(0,1) (1, 4) (2 ,9 ) (3, 16) (4,25)
```

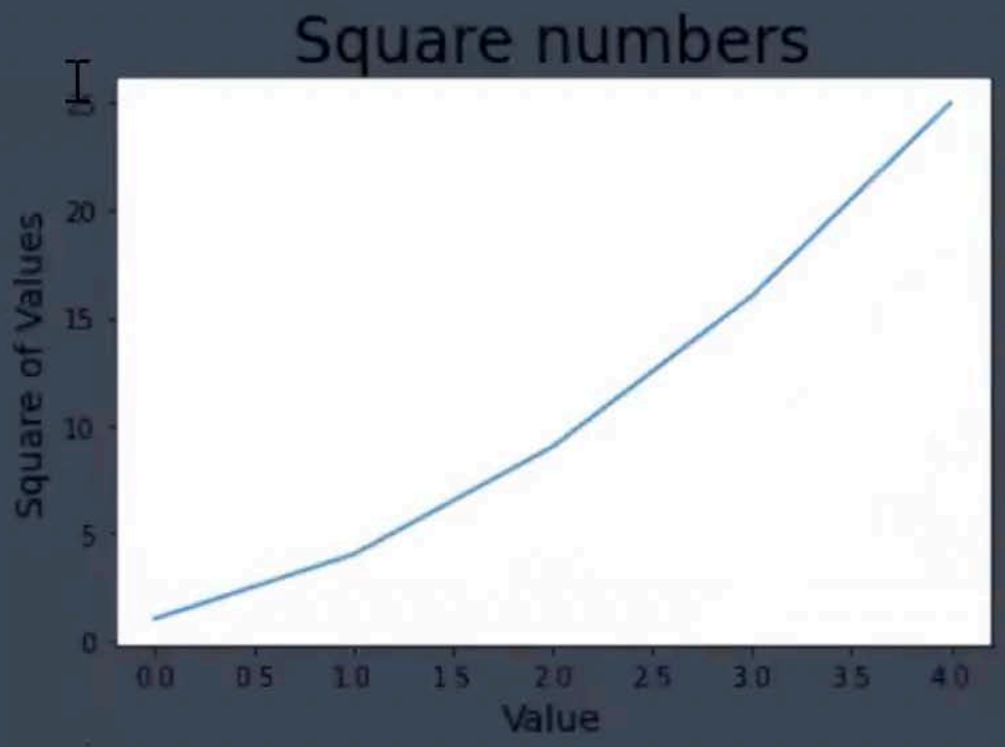


In [3]:

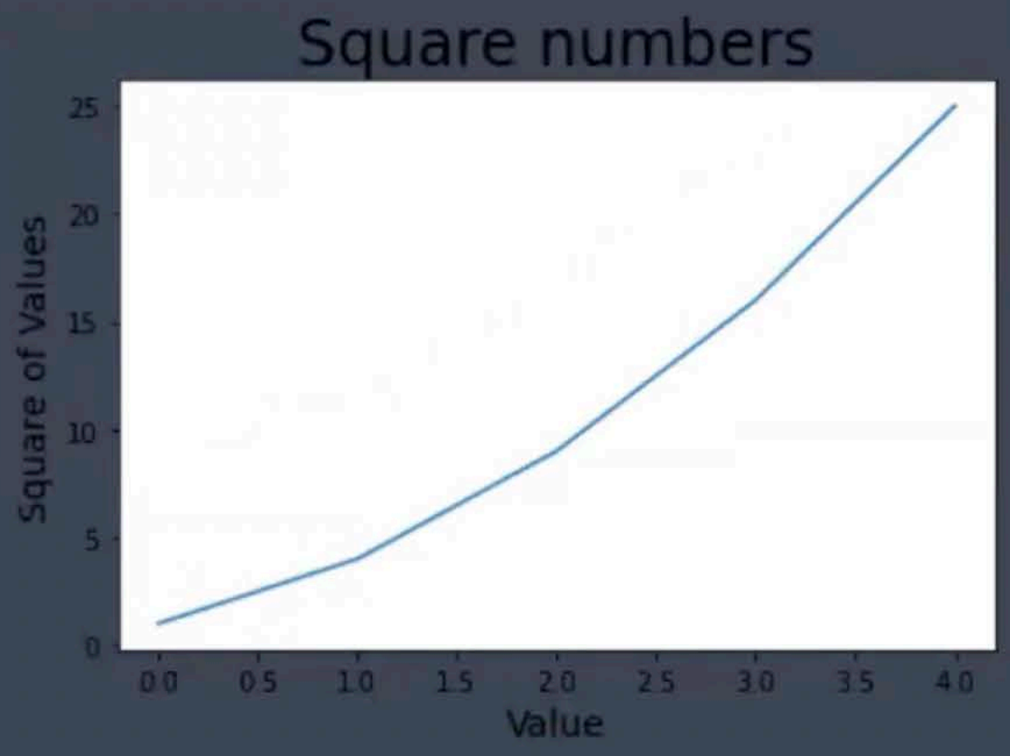
```
# changing thickness of graph
plt.plot(squares , linewidth = 10)
plt.show()
```

In [4]:

```
#set chart title and label axes
plt.title("Square numbers" , fontsize = 24)
plt.xlabel("Value" , fontsize = 14)
plt.ylabel("Square of Values" , fontsize = 14)
plt.plot(squares)

plt.show()
```



```python
#set chart title and label axes
plt.title("Square numbers" , fontsize = 24)
plt.xlabel("Value" , fontsize = 14)
plt.ylabel("Square of Values" , fontsize = 14)
plt.plot(squares , dash_capstyle = "round")

#set size of tick labels
plt.tick_params( axis = "both" , labelsize = 10)
plt.show()
```

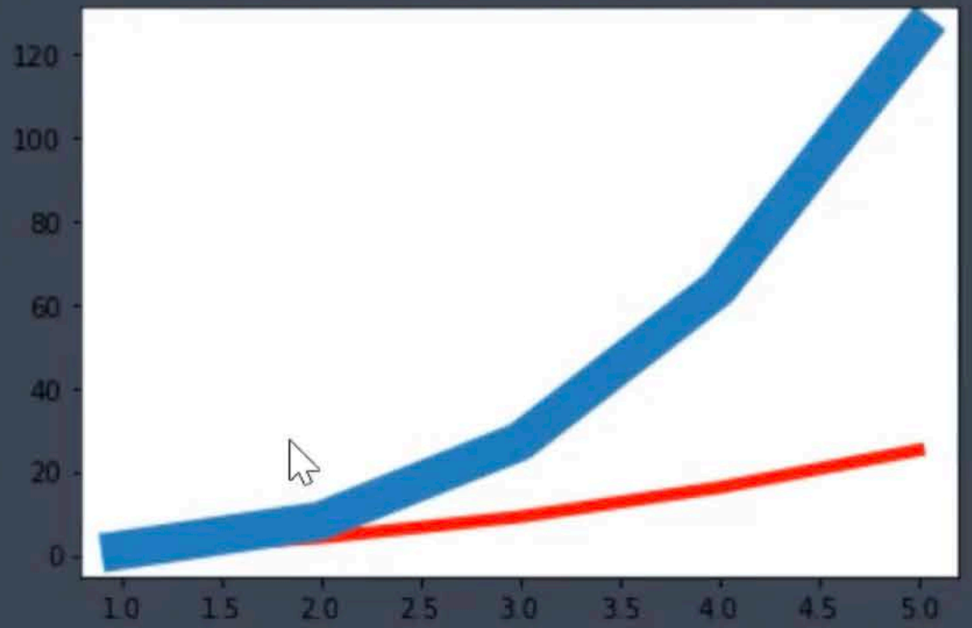


In [8]:

```
squares = [ 0, 1,4,9,16,25]
plt.plot(squares)
plt.show()

#(0,1) (1, 4) (2 ,9 ) (3, 16) (4,25)
```

In [12]:

```
#correcting the plot
input_values = [1,2,3,4,5]
squares = [1,4,9,16,25]
cubes = [1,8, 27,64 ,125]
plt.plot(input_values , squares, linewidth = 5 , c = "red")
plt.plot(input_values , cubes , linewidth = 15)

plt.show()
#(1,1) (2, 4) (3 ,9 ) (4, 16) (5,25)
```

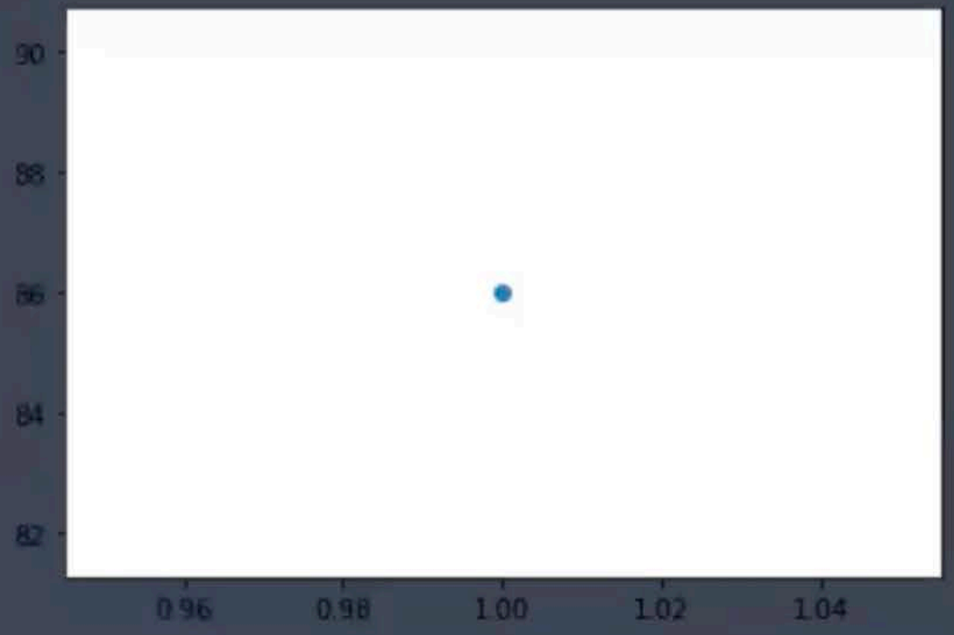


In [ ]:

```
1
```

In [3]:

```
#plotting points with scatter()
import matplotlib.pyplot

matplotlib.pyplot.scatter(1,86)
matplotlib.pyplot.show()
```



In [ ]:

```
1
```

In [5]:

```
#plotting points with scatter()
import matplotlib.pyplot as plt

#matplotlib.pyplot.scatter(1,86)
#matplotlib.pyplot.show()
```

In [ ]:

```
1
```

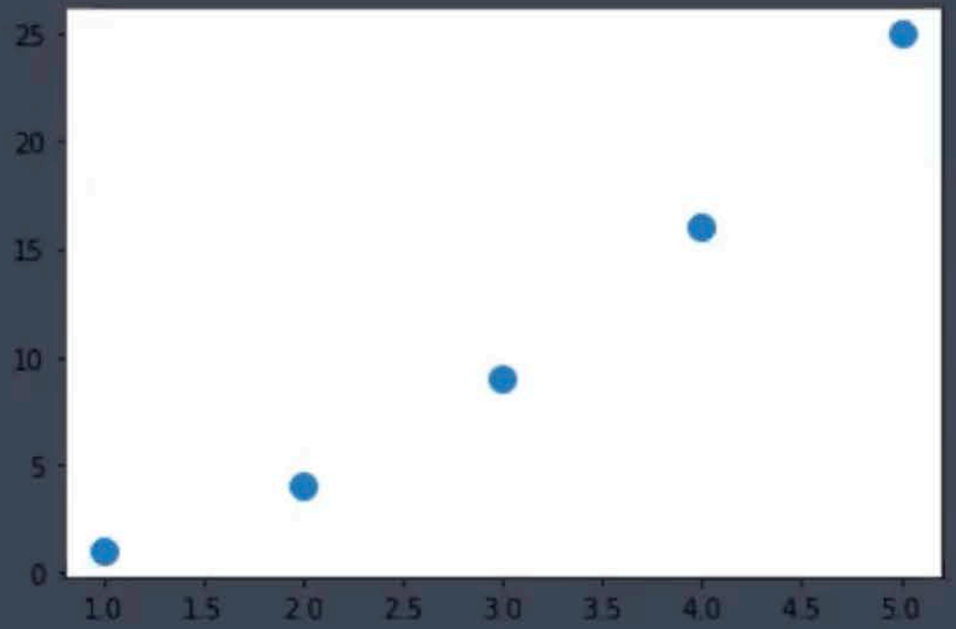
In [ ]:

```
1
```

In [4]:

```
#plotting series of Points

x_values = [1,2,3,4,5]
y_values = [1,4,9,16,25]

plt.scatter(x_values , y_values , s = 100)
plt.show()
```

```
2
3  x_values = [1,2,3,4,5]
4  y_values = [1,4,9,16,25]
5
6  plt.scatter(x_values , y_values , s = 100)
7  plt.show()
```



In [13]:

```
x_values = list(range(1 , 3))
x_values
```

```
[1, 2]
```

In [14]:

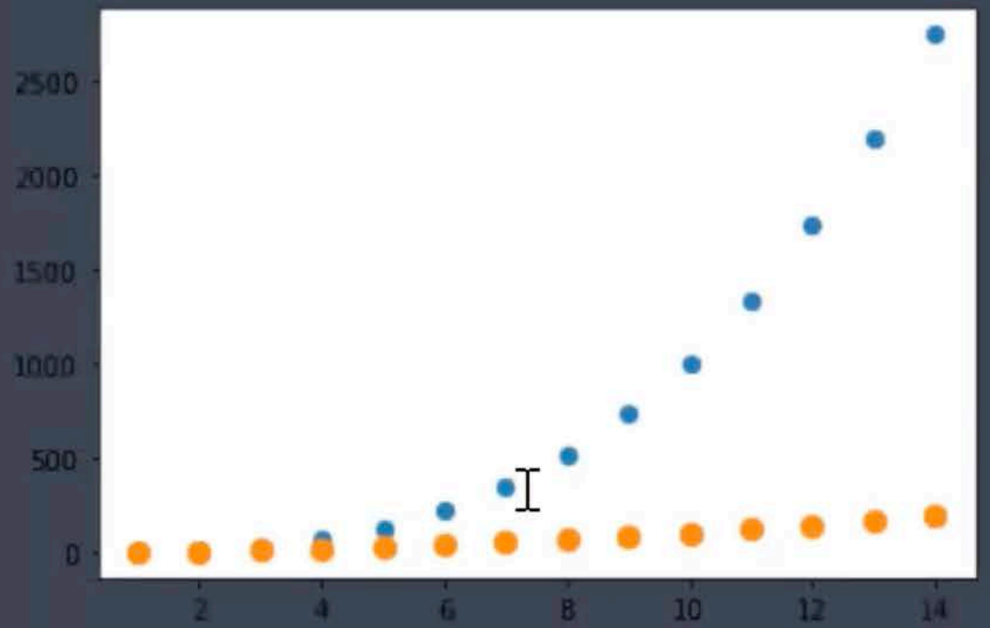
```
1
2 print(x_values)
3 y_values =[x**2 for x in x_values]
4 print(y_values)
5 #x**3 for x in x_values
```

```
[1, 2]
[1, 4]
```

In [7]:

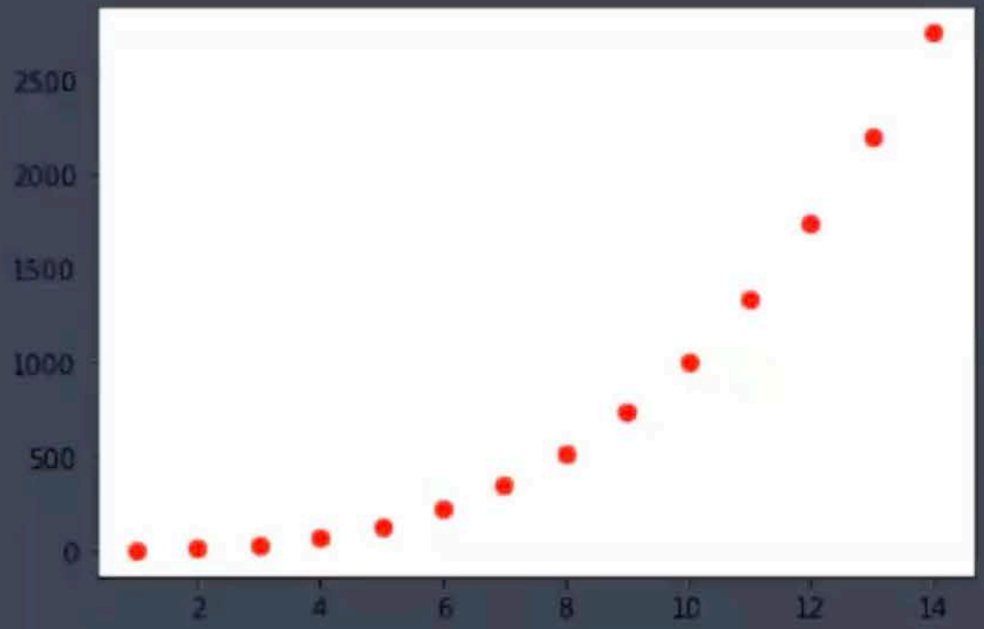
```
x_values = list(range(1 , 15))
print(x_values)
y_values = [x**3 for x in x_values]

y_values2 = [x**2 for x in x_values]
print(y_values)
plt.scatter(x_values , y_values , s =40)
plt.scatter(x_values , y_values2 , s = 70)
plt.show()
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197, 2744]
```



In [18]:

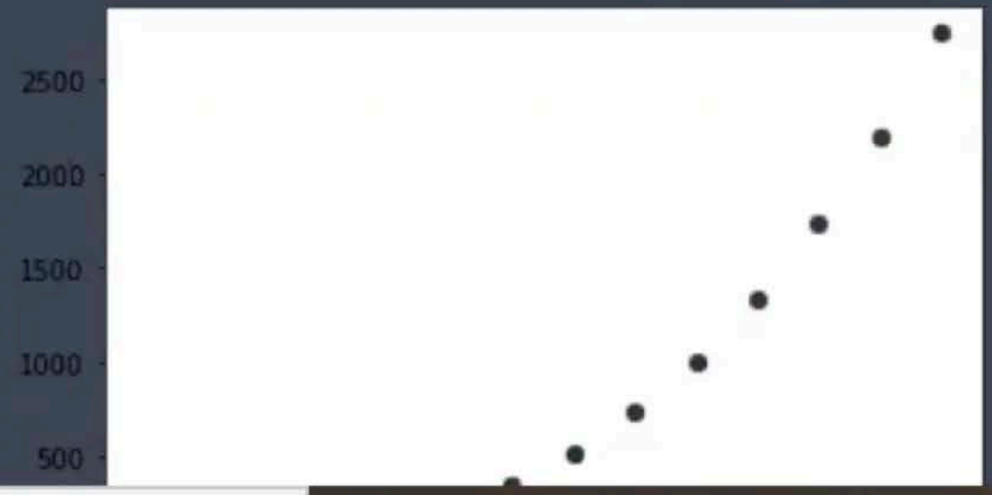
```
#Changing the color
plt.scatter(x_values , y_values , c = 'red' , s = 40)
plt.show()
```



In [20]:

```
plt.scatter(x_values , y_values , c = ( 0.1, 0.2, 0.1) , s = 40)
plt.show()

#values closer to 0 will produce dark colors

```

*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend t o specify the same RGB or RGBA value for all points.



In [22]:

```
#Using ColorMap

plt.scatter(x_values , y_values , c = y_values , cmap = plt.cm.Blues , s = 40)
plt.show()

```