Outline

- 3D Graphs
  - np.meshgrid()
  - Working of meshgrid
  - Plot_surface
- Pie charts
- Linechart in case of multiple values

3-D Graphs

- Matplotlib allows us to plot 3-D graphs

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

a = np.array([0, 1, 2, 3])
b = np.array([0, 1, 2])

a, b = np.meshgrid(a, b) # We'll use numpy's meshgrid
```

In [ ]:

```
a
```

Out[18]:

```
array([[0, 1, 2, 3],
       [0, 1, 2, 3],
       [0, 1, 2, 3]])
```

In [ ]:

```
b
```

Out[19]:

```
array([[0, 0, 0, 0],
       [1, 1, 1, 1],
       [2, 2, 2, 2]])
```

Notice here:

- `meshgrid()` allows you to **convert a 1-D array into 2-D**

How is `meshgrid()` working?

- It **takes dimensions of both arrays `a` and `b`**
  - Dimension of `a` is (4,)
  - Dimension of `b` is (3,)
- **Creates two 3×4 matrices**

- In **first matrix**, array of `a` **repeats 3 times** and gets **stacked vertically**
- In **second matrix**, array of `b` **repeats 4 times** and gets **stacked horizontally**

So, we get all possible combinations as coordinates

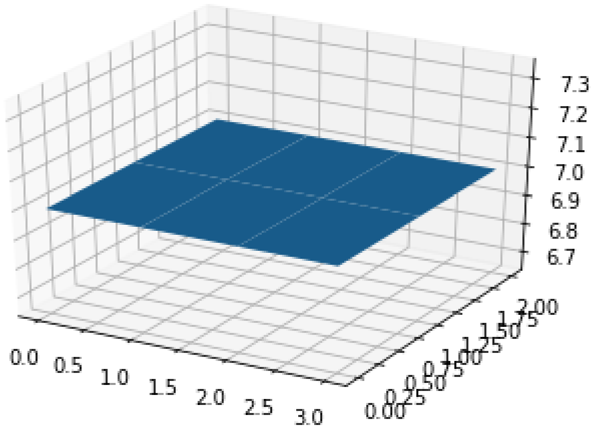
```
 (0,0)  (1, 0)  (2, 0)  (3, 0)
(0,1)  (1, 1)  (2, 1)  (3, 1)
(0,2)  (1, 2)  (2, 2)  (3, 2)
```

- **Two 1-D arrays get converted into a meshgrid**

We can use the meshgrid to plot our 3-D graph

In [ ]:

```python
fig = plt.figure()  # Let's save plt.figure() in fig
ax = fig.add_subplot(projection='3d')
ax.plot_surface(a, b, np.array([[7]]) )  # 3rd argument is the elevation
plt.show()
```



Observe that:

- **Range on the axes of our plane surface** is taken from `a` **and** `b`
  - **x-axis is represented by** `a` ---> 0 to 3

```
- **y-axis is represented by `b`** ---> 0 to 2
```

- The **3rd dimension is the height from the floor we passed in**
  - ○ **z-axis is represented by the elevation** we provided ---> 7

Question: Did everyone get this output?

Let's build a more complex 3-D plot

In [ ]:

```python
a = np.arange(-1, 1, 0.005)  # I am going to use a very small step-size
b = a
a, b = np.meshgrid(a, b)


# Since `a` and `b` are some, this time we'll get a square matrix
```

In [ ]:

```python
a.shape
```

Out[22]:

```
(400, 400)
```

In [ ]:

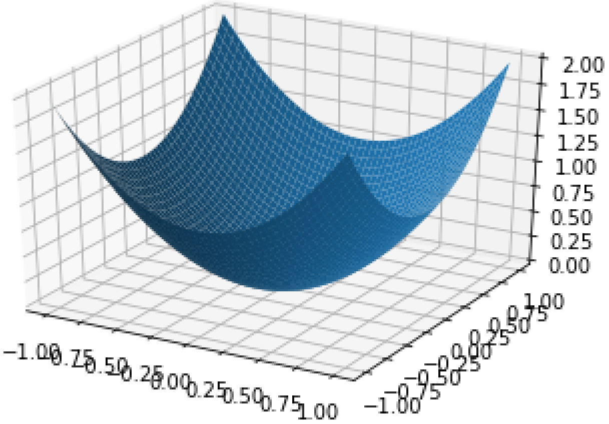
```python
b.shape
```

Out[23]:

```
(400, 400)
```

In [ ]:

```python
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(a, b, a**2 + b**2) # We'll NOT keep a constant z-axis ele
plt.show()
```



Notice that:

- **Higher the values of `a` and `b`, higher will be z-axis value**
- At **`(a, b) = (0, 0)`**, `z` is **lowest at 0**
- The **dimensions of `a`, `b` and `a^2 + b^2` are all same**
  - 400×400
- That is why it is able to map all **(x, y, z) coordinates**

In [ ]:

```python
(a**2).shape
```

Out[25]:

```
(400, 400)
```

In [ ]:
```
(a**2 + b**2).shape
```

Out[26]:
```
(400, 400)
```

Pie charts

Finally Matplotlib also provide you with **Pie charts**

We won't go into too much details of it - We know what a Pie Chart is

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/021/
```

```
--2022-12-13 02:48:57--  https://d2beiqkhq929f0.cloudfront.net/public_asset
s/assets/000/021/299/original/final_vg1_-_final_vg_%281%29.csv?1670840166 (h
ttps://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/021/299/origin
al/final_vg1_-_final_vg_%281%29.csv?1670840166)
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 1
8.165.94.81, 18.165.94.181, 18.165.94.193, ...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|
18.165.94.81|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2041483 (1.9M) [text/plain]
Saving to: ‘final_vg.csv’

final_vg.csv        100%[===================>]   1.95M  --.-KB/s    in 0.06s

2022-12-13 02:48:57 (32.5 MB/s) - ‘final_vg.csv’ saved [2041483/2041483]
```

In [ ]:
```
data = pd.read_csv('final_vg.csv')
```

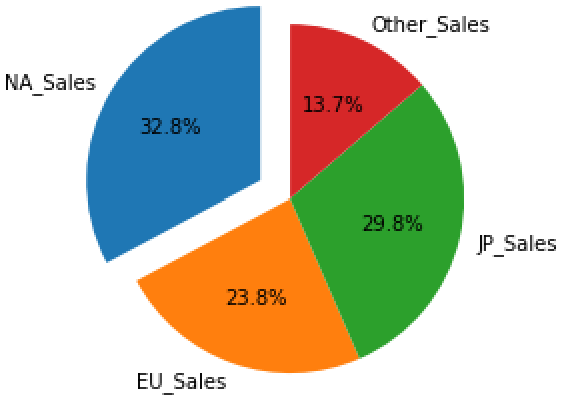
If you remember, we had made a sales across regions piechart in the lecture.

Let's plot the same using matplotlib now

In [ ]:
```
sales_data = data[['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
region_sales = sales_data.T.sum(axis='columns')
```

In [ ]:

```
plt.pie(region_sales,
       labels=region_sales.index,
        autopct='%1.1f%%',
       startangle=90,
       explode=(0.2,0,0,0))
plt.show()
```


Other_Sales
NA_Sales
JP_Sales
EU_Sales