Assignment 5

Group Members

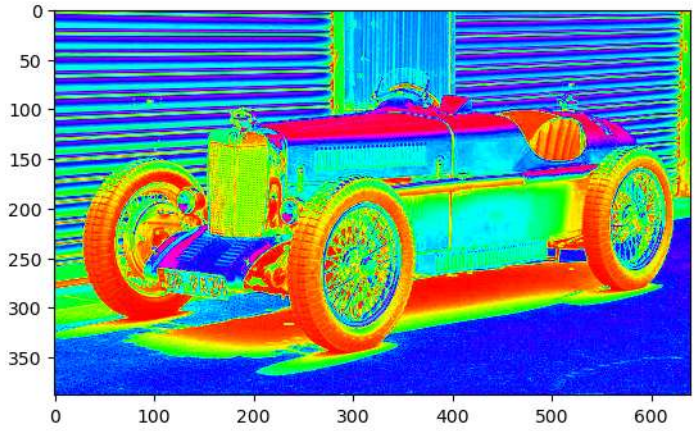
Abdul Wahab Madni, Hamza Badar, Aleksandr Semenikhin

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
import skimage, vintage
from sklearn.manifold import MDS
from PIL import Image, ImageEnhance, ImageDraw
from matplotlib.colors import LinearSegmentedColormap
```

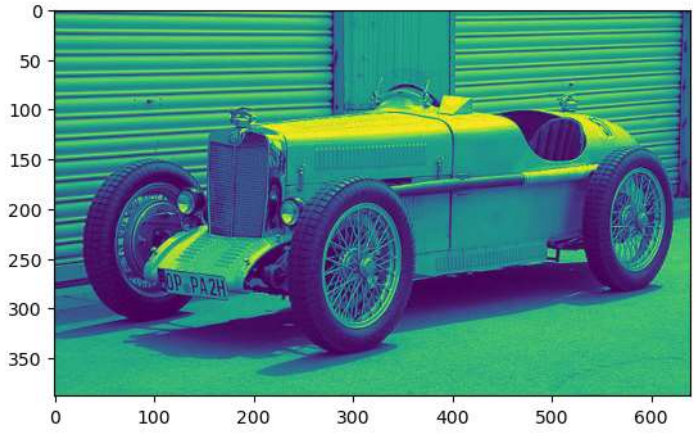
Ex.1: Working with HSV Color Space

a)

In [2]:
```
img = np.asarray(Image.open('oldtimer.png'))
lum_img = img[:, :, 0]
img_hsv = plt.imshow(lum_img,cmap="hsv")
plt.show()
```

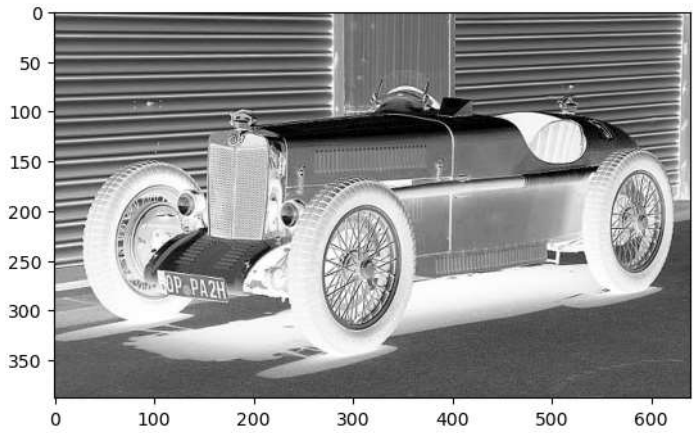


In [3]:
```
img_hsv = plt.imshow(lum_img)
```



b)

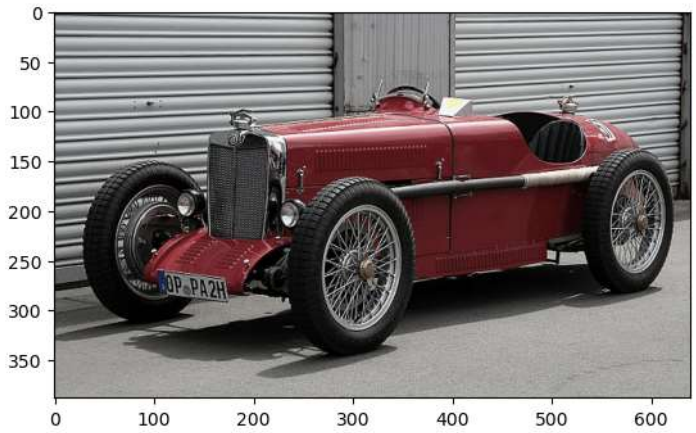
In [4]:
```
img_gr = plt.imshow(lum_img,cmap="binary")
plt.show()
```



It's the way to make it grayscale

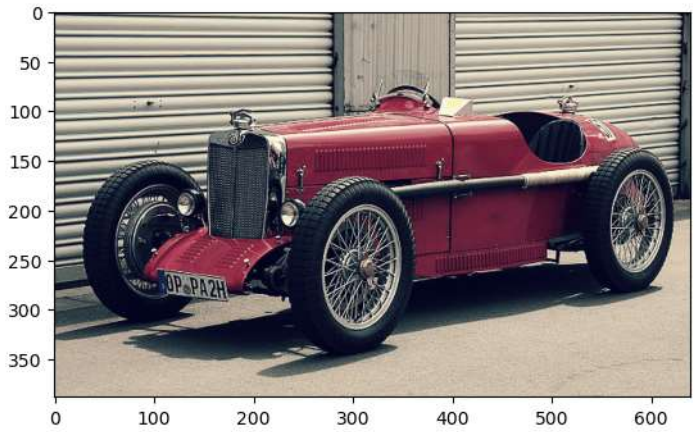
c)

In [5]:
```
img_s = Image.open("oldtimer.png")
img_satr = ImageEnhance.Color(img_s)
img_satr = np.array(img_satr.enhance(0.5))
plt.imshow(img_satr)
plt.show()
```



d)

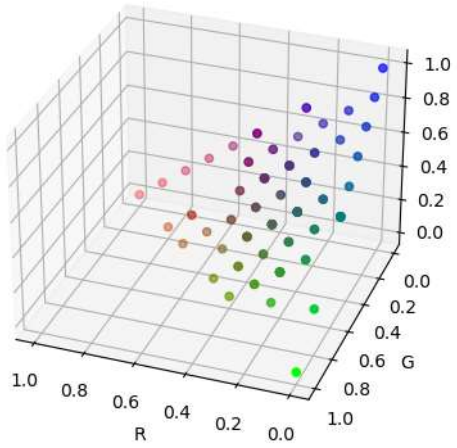
In [6]:
```
img_s = Image.open("oldtimer.png")
img_s = vintage.vintage_colors(img_s) #Python file!!
img_satr = ImageEnhance.Color(img_s)
img_satr = np.array(img_satr.enhance(0.5))
plt.imshow(img_satr)
plt.show()
```



Ex.2: Visualizing Color Spaces

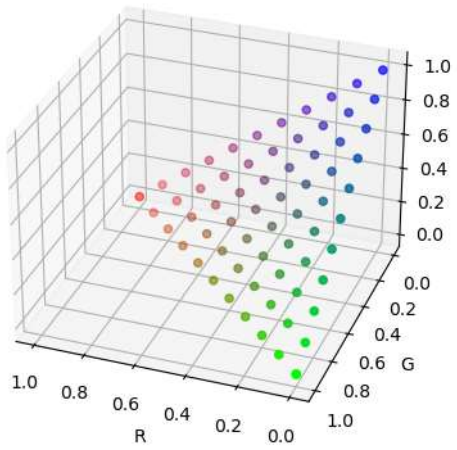
a)

In [7]:
```
np.random.seed(22)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
x_axis = [1,0,0]
y_axis = [0,1,0]
z_axis = [0,0,1]
hcolor = []
for i in range(100):
    x = np.round(np.random.random(),1)
    y = np.round(np.random.random(),1)
    z = np.round(np.random.random(),1)
    temp = x+y+z
    temp_x = np.round(x/temp,1)
    temp_y = np.round(y/temp,1)
    temp_z = np.round(z/temp,1)
    if temp_x+temp_y+temp_z != 1:
        temp_rand = np.random.choice([temp_x,temp_y,temp_z])
        if temp_rand == temp_x:
            temp_x += 1-(temp_x+temp_y+temp_z)
        elif temp_rand == temp_y:
            temp_y += 1-(temp_x+temp_y+temp_z)
        elif temp_rand == temp_z:
            temp_z += 1-(temp_x+temp_y+temp_z)
        else:
            temp_z += 1-(temp_x+temp_y+temp_z)
    x_axis.append(temp_x)
    y_axis.append(temp_y)
    z_axis.append(temp_z)
x_axis = np.round(x_axis,1)
y_axis = np.round(y_axis,1)
z_axis = np.round(z_axis,1)
for i in range(len(x_axis)):
    hcolor.append([x_axis[i],y_axis[i],z_axis[i]])
ax.scatter(x_axis,y_axis,z_axis,c= hcolor)
ax.set_xlabel('R')
ax.set_ylabel('G')
ax.set_zlabel('B')
ax.view_init(30,110,0)
plt.show()
```



In [8]:
```
np.random.seed(22)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
x_axis = [1,0,0]
y_axis = [0,1,0]
z_axis = [0,0,1]
hcolor = []
x,y,z =0,0,1
for i in range(10):
    x += 0.1
    y += 0.1
    z -= 0.1
    x_axis.append(x)
    y_axis.append(0)
    z_axis.append(z)
    x_axis.append(0)
    y_axis.append(y)
    z_axis.append(z)

x,y,z = 0.1,0.1,0.8

for i in range(9):
    for j in range(9-i):
        x_axis.append(x)
        y_axis.append(y)
        z_axis.append(z)
```

```
        y += 0.1
        z -= 0.1
    x += 0.1
    y = 0.1
    z = 0.1*(7-i)

x_axis = np.round(x_axis,1)
y_axis = np.round(y_axis,1)
z_axis = np.round(z_axis,1)
for i in range(len(x_axis)):
    hcolor.append([x_axis[i],y_axis[i],z_axis[i]])
ax.scatter(x_axis,y_axis,z_axis,c= hcolor)
ax.set_xlabel('R')
ax.set_ylabel('G')
ax.set_zlabel('B')
ax.view_init(30,110,0)
plt.show()
```

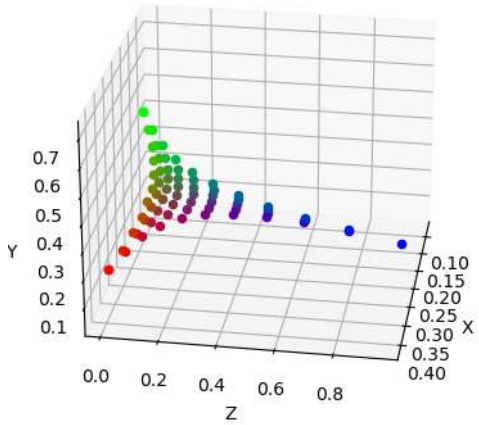


b)

In [9]:

```
np.random.seed(22)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

xyz = []

for i in range(len(x_axis)):
    xyz.append(skimage.color.rgb2xyz([x_axis[i],y_axis[i],z_axis[i]]))
    ax.scatter(xyz[i][0],xyz[i][2],xyz[i][1],c= skimage.color.xyz2rgb(xyz[i]))
ax.set_xlabel('X')
ax.set_ylabel('Z')
ax.set_zlabel('Y')
ax.view_init(25,9,0)
plt.show()
```

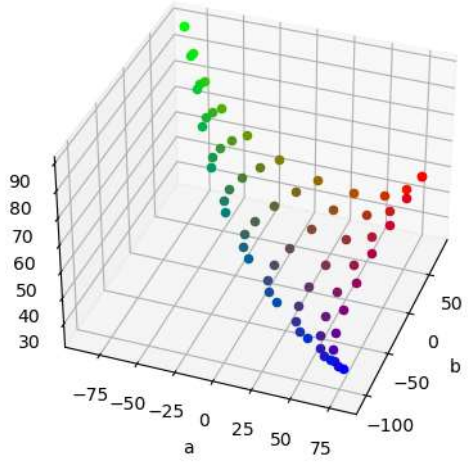
```
C:\Users\madni\AppData\Local\Temp\ipykernel_34336\3682823140.py:9: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence,
which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argume
nt or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
  ax.scatter(xyz[i][0],xyz[i][2],xyz[i][1],c= skimage.color.xyz2rgb(xyz[i]))
```



c)

In [10]:
```
np.random.seed(22)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

lab = []

for i in range(len(x_axis)):
    lab.append(skimage.color.rgb2lab([x_axis[i],y_axis[i],z_axis[i]]))
    ax.scatter(lab[i][2],lab[i][1],lab[i][0],c= skimage.color.lab2rgb(lab[i]))
ax.set_xlabel('b')
ax.invert_xaxis()
ax.set_ylabel('a')
ax.set_zlabel('L')
ax.view_init(35,20,0)
plt.show()
```

```
C:\Users\madni\AppData\Local\Temp\ipykernel_34336\3078182399.py:9: UserWarning: *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-argume
nt or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
  ax.scatter(lab[i][2],lab[i][1],lab[i][0],c= skimage.color.lab2rgb(lab[i]))
```

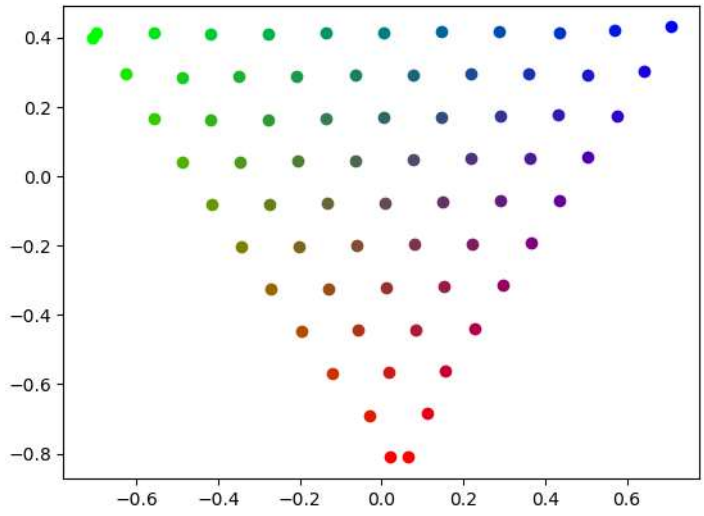


The brightest is XYZ and darkest is sRGB

d)

In [11]:
```
np.random.seed(13)
embedding = MDS(n_components=2, normalized_stress="auto")
rgb =np.array([x_axis,y_axis,z_axis])
rgb = np.transpose(rgb)
rgb_mds = embedding.fit_transform(rgb)
rgb_mds = np.transpose(rgb_mds)
for i in range(len(rgb_mds[0])):
    plt.scatter(rgb_mds[0][i],rgb_mds[1][i],c= rgb[i])
```

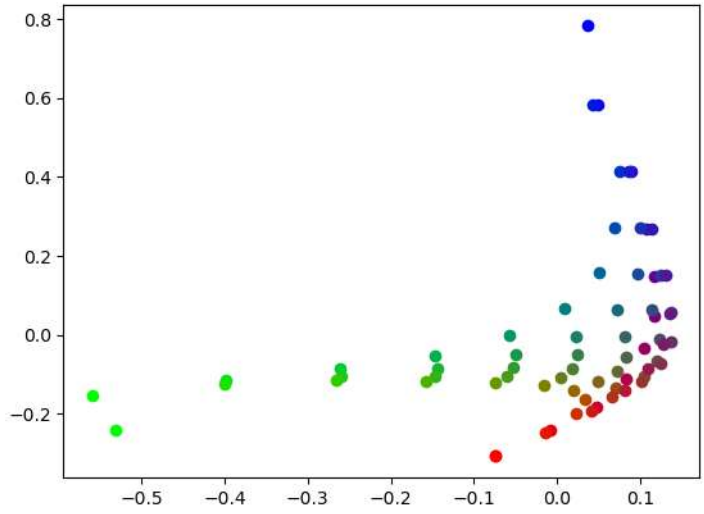
```
C:\Users\madni\AppData\Local\Temp\ipykernel_34336\3671114711.py:8: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence,
which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argume
nt or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(rgb_mds[0][i],rgb_mds[1][i],c= rgb[i])
```



In [12]:
```
np.random.seed(17)
embedding = MDS(n_components=2, normalized_stress="auto")
```

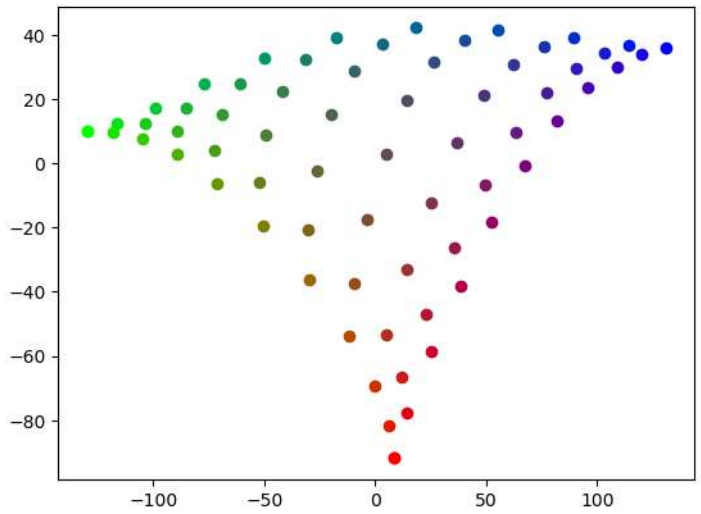
```
xyz_mds = embedding.fit_transform(xyz)
xyz_mds = np.transpose(xyz_mds)
for i in range(len(xyz_mds[0])):
    plt.scatter(xyz_mds[0][i],xyz_mds[1][i],c= skimage.color.xyz2rgb(xyz[i]))
```

```
C:\Users\madni\AppData\Local\Temp\ipykernel_34336\3249319595.py:6: UserWarning: *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-argume
nt or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(xyz_mds[0][i],xyz_mds[1][i],c= skimage.color.xyz2rgb(xyz[i]))
```



In [13]:
```
np.random.seed(32)
embedding = MDS(n_components=2, normalized_stress="auto")
lab_mds = embedding.fit_transform(lab)
lab_mds = np.transpose(lab_mds)
for i in range(len(lab_mds[0])):
    plt.scatter(lab_mds[0][i],lab_mds[1][i],c= skimage.color.lab2rgb(lab[i]))
```

```
C:\Users\madni\AppData\Local\Temp\ipykernel_34336\2496948907.py:6: UserWarning: *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-argume
nt or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(lab_mds[0][i],lab_mds[1][i],c= skimage.color.lab2rgb(lab[i]))
```



Honestly I have no idea what to comment about resulted scatter plots. MDS is used to translate "information about the pairwise 'distances' among a set of objects" into a configuration of points, so that is what we've got after doing it.

Ex.3: Face-based Luminance

a)

it's a human eye behavioral phenominon in which saturated colors appears brighter to us than their actual luminance. And this effect increases with the saturation of the color.

b)

Their goal was to create a task that is similar to the brightness method which is immune Helmholtz-Kohlrausch effect yet easir to carry out by users. Authors proposed luminance matching technique based on a thresholded image of a human face. The reason for this suggestion is that, as long as there is an appropriate luminance difference between shadowed and illuminated surfaces, humans are good at recognizing thresholded face images.

c)

They compared their newly purposed (double face luminance matching method) with The minimally distinct boundary method (MDB).

d)

The response rate of double face luminance method was superior to MDB. particepents were more consistant in the lightness value they chose.

e)

previously, authors had performed an informal study and additivity test that totally was self-contained which required no knowledge of the monitor's chromaticities or gamma. But for the colormaps Knowing the monitor gamma they could perform interpolation in what is essentially gamma corrected RGB space.

Ex.4: Color in Visualization

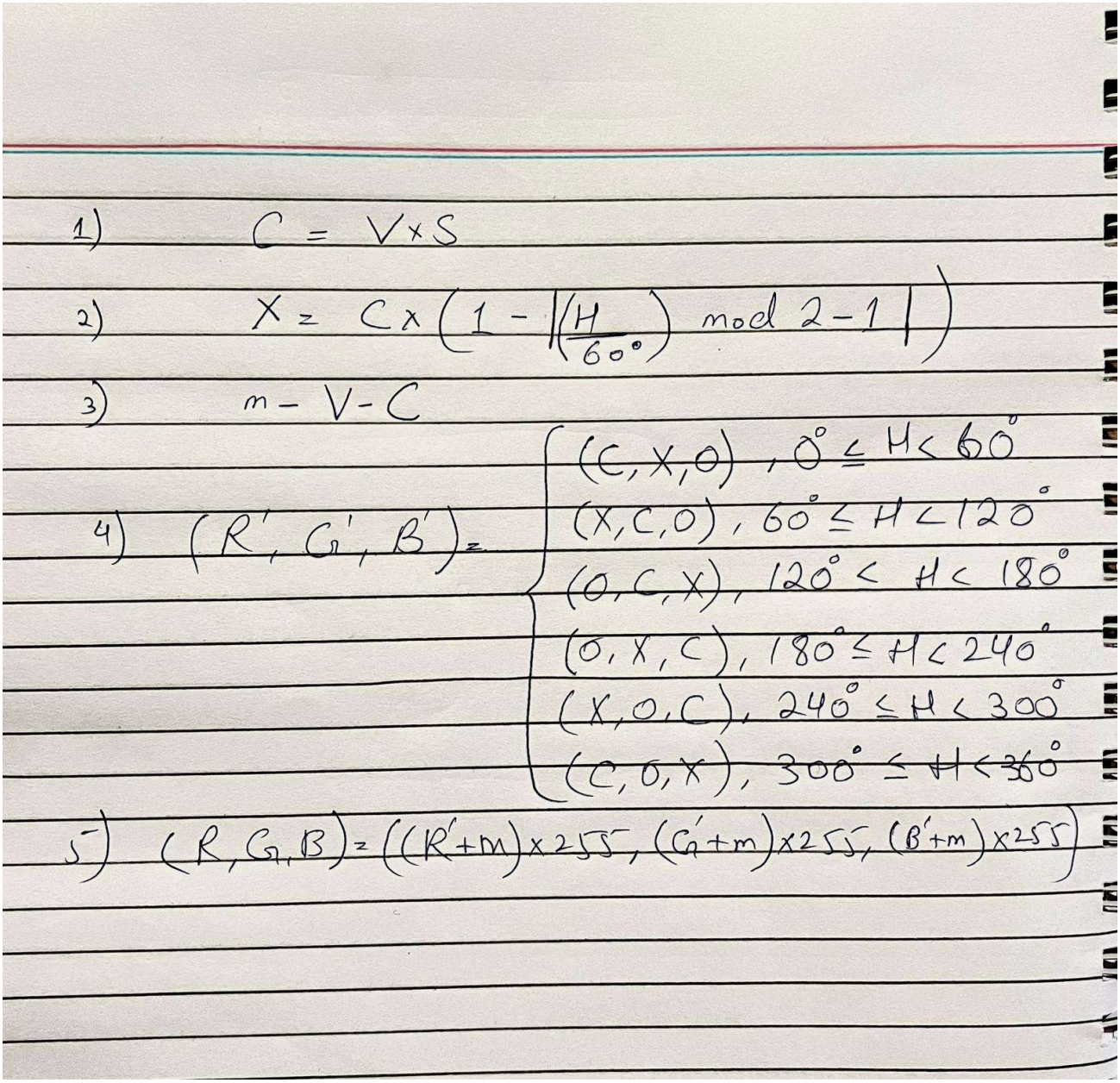
a)

In [14]:
```
x = ["Volkswagen", "Mercedes-Benz", "BMW", "Audi", "Ford"]
y = [2009, 2009, 2009, 2009, 2009]
z = [2500, 2000, 4000, 5000, 3500]
df= pd.DataFrame([x,y,z],['man','year','sold'])
df=df.T
#The color map legend
my_colormap = LinearSegmentedColormap.from_list ( 'my colormap',['purple','blue', 'yellow','orange','brown'],N=100)

pivot_table = df.pivot_table(values='man', index='man', columns='year')
sns.heatmap(pivot_table, annot=True, cmap= my_colormap, fmt= ".0f")

plt.xlabel('Year')
plt.ylabel('Manufacturer')
plt.title('Number of Vehicles Sold')
```

Out[14]:
```
Text(0.5, 1.0, 'Number of Vehicles Sold')
```

The data file was not given so we used a dummy data just to show our color map legend. The color legend can also be given to car companies.The given problem data was nominal. That's why we used the basic colors as described in the slides of color lecture. We didn't use the color red and green as around 10% of people are red/green color blind.

b)



Explaining step by step. In line 1:

we get the chroma value by multiplying saturation and value.

In line 2:

We defined an intermediate value X. We'll divide it to 60° hue in order to reduce it's hue value by that, take it's mod and then multiply to chroma value C(which contains V * S) in order to calculate value of 'X'.

In line 3:

we calculated the value of minimun RGB by subtracting chroma(C) by value(V) which will be useful in final step.

In step 4:

we perform the actual calculation which will convert the circular HSV form into the RGB form for each 60° interval of H(hue). For H value 60° to 180°, which contains the green color at the center(120°) and light blue and yellow at their each ends. So, the green will be present throughout this area and we'll give the chroma value for Green prime(G') throughout these angles. The yellow is present at 60° of H and we know it is obtained from equal mixing of red and green but we are taking value from 60° to onward. This means the value of red color will start reducing from here. That's why we reduced the value of Red color(which is obtained by X) from 60° to 120° in order to produce the accurate color. And the blue color will be absent here so we have given the blue prime(B') value of 0 from H 60° to 120°. Now taking about 120° to 180° of H, the color light blue(blue-green) is present at 180° of H which is obtained from equal mixing of green and blue but the angle is decreasing from 180° to backward. This means the value of blue color will start reducing from 180° to backward. That's why we reduced the value of blue color(obtained by X) from 120° to 180°. And we have given the blue prime(B') value X from 120° to 180°. The color red is completely absent here so we have given it the 0 value. Other angles are also obtained like this procedure.

In the final step:

We'll add it with minimum RGB value 'm' and to get the range 0 to 255, we'll multiply it with 255.

c)

For making a digital art for myself, I'll use HSV color space as it represent colors in a way that matches human perception. It also give a lot of options to change saturation, hue and value which is very difficult to do in CIEluv. On the other hand, if I have to share my digital art with others or take print out of it, I'll prefer CIEluv color space as it gives consistant colors and is independent of devices. Which means that it'll show same consistant colors on every other device.