Python & Colab intro

```
a = 10
```

```
print(a)
```

```
10
```

BMI counting code

```
height = 180
weight = 75
bmi = weight / pow((height / 100), 2)
print(bmi)
```

```
23.148148148148145
```

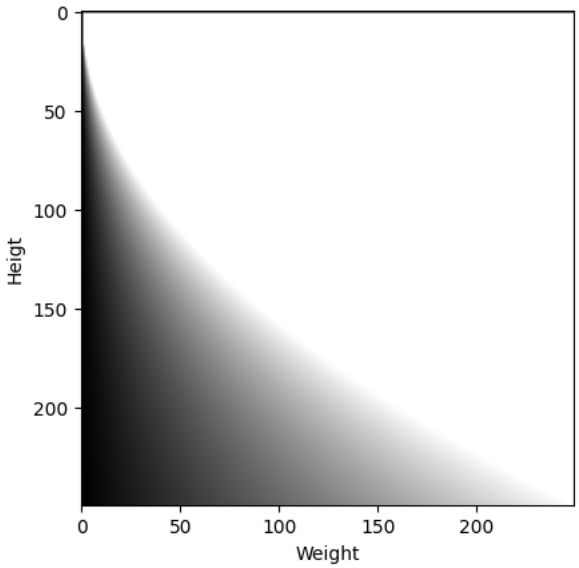
```
import sklearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
def bmi(height_in_centimeters, weight_in_kilograms):
  return weight_in_kilograms / ((height_in_centimeters / 100) ** 2)

print(bmi(170, 80))
```

```
27.68166089965398
```

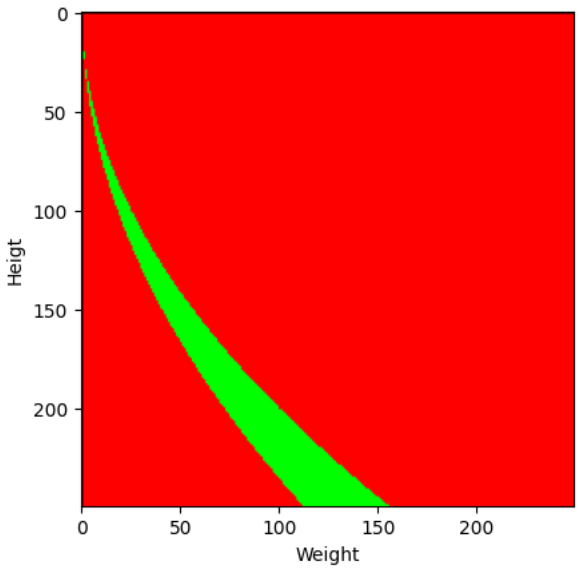
```
result = np.zeros([250, 250, 3])
for x in range(1, 250):
  for y in range(1, 250):
    if bmi(x, y) < 40:
      result[x][y] = bmi(x,y)
    else:
      result[x][y] = 40

# normalisation to achieve the range 0 - 1
result = (result - np.min(result)) / (np.max(result) - np.min(result))

plt.xlabel("Weight")
plt.ylabel("Heigt")
plt.imshow(result)
```

```
<matplotlib.image.AxesImage at 0x7c654e01f820>
```



```
result = np.zeros([250, 250, 3])
for x in range(1, 250):
  for y in range(1, 250):
    if bmi(x, y) > 18 and bmi(x, y) < 25:
      result[x][y] = [0, 1, 0]
    else:
      result[x][y] = [1, 0, 0]

# normalisation to achieve the range 0 - 1
result = (result - np.min(result)) / (np.max(result) - np.min(result))

plt.xlabel("Weight")
plt.ylabel("Heigt")
plt.imshow(result)
```

```
<matplotlib.image.AxesImage at 0x7c654dff3a90>
```

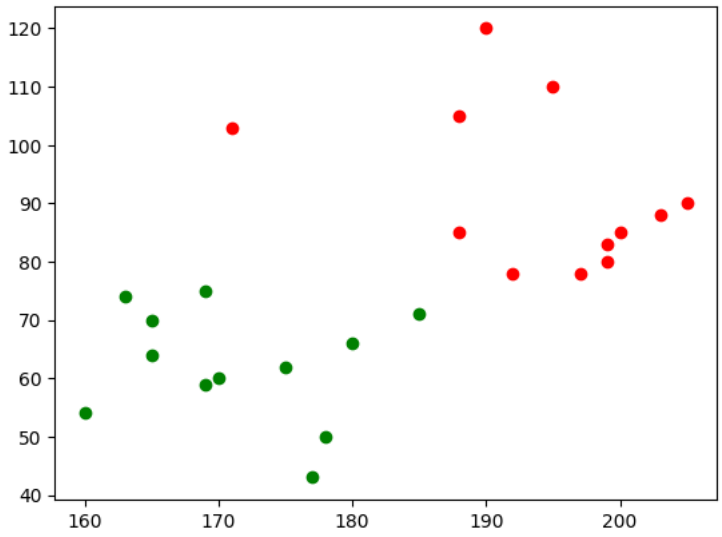


Prediction

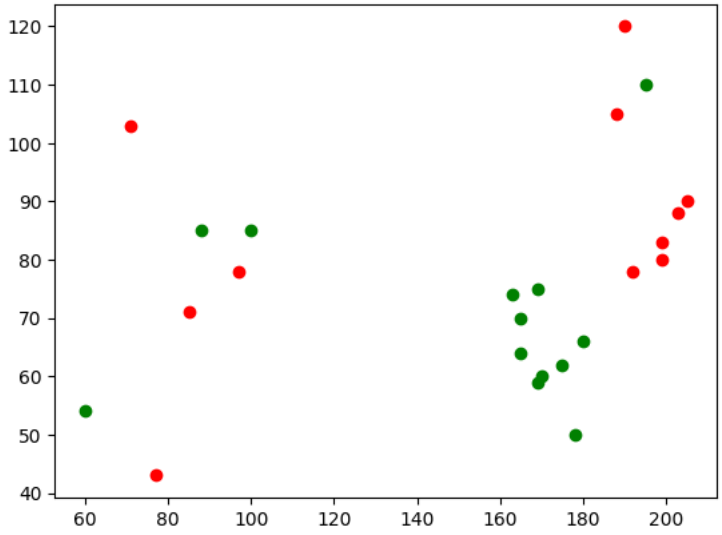
```
data_1 = np.array([[160,54,1],[170,60,1],[165,70,1],[190,120,0],[203,88,0],[192,78,0],[163,74,1],[177,43,1],[175,62,1],[195,
data_2 = np.array([[60,54,1],[170,60,1],[165,70,1],[190,120,0],[203,88,0],[192,78,0],[163,74,1],[77,43,0],[175,62,1],[195,11
```

```
def show_points(data):
  for x, y, c in data:
    if c == 0:
      plt.plot(x, y, marker = 'o', color = 'red')
    else:
      plt.plot(x, y, marker = 'o', color = 'green')
```

```
show_points(data_1)
```

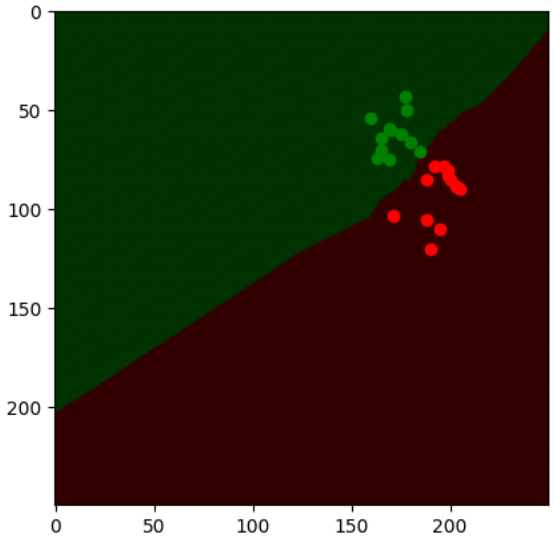



```
show_points(data_2)
```

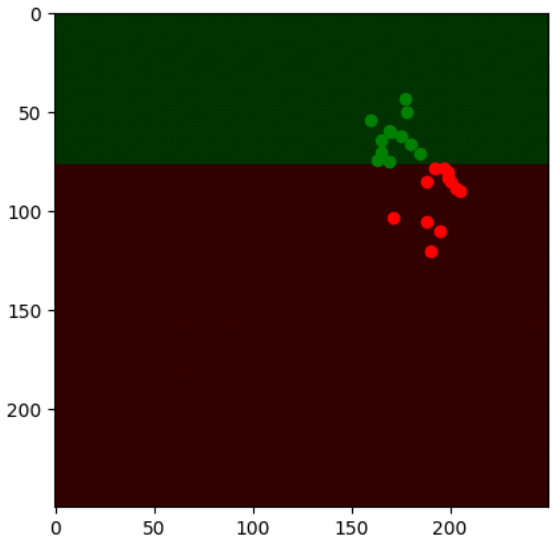


Building predicting model

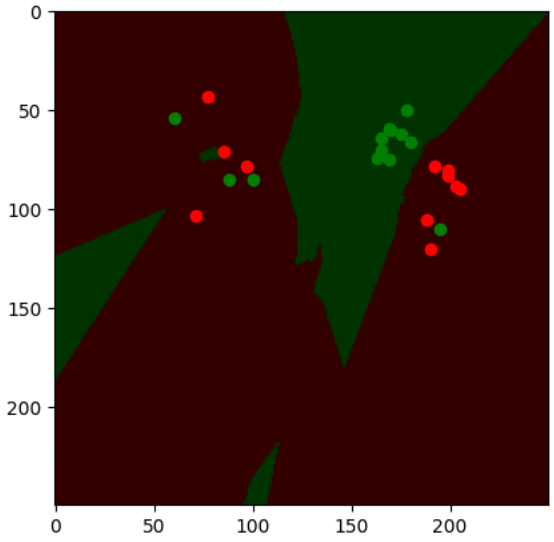
```
def classify_all_points(model, max = 250):
  # create all pairs (x,y) in range 0-max
  samples = []
  for x in range(max):
    for y in range(max):
      samples.append((x, y))
  samples = np.array(samples)

  # use the model for classification of all pairs
  r = model.predict(samples)

  # create image witch all pairs (value > 0,5 - green, value <= 0.5 - red)
  result = np.zeros([max, max, 3])
  for i in range(len(samples)):
    if r[i] > 0.5:
      result[samples[i, 1], samples[i, 0]] = [0, 0.2, 0]
    else:
      result[samples[i, 1], samples[i, 0]] = [0.2, 0, 0]

  return result


model = KNeighborsClassifier() # DecisionTreeClassifier()
samples = data_1[:, 0:2]
labels = data_1[:, 2]
model.fit(samples, labels)

result = classify_all_points(model)
plt.imshow(result)
show_points(data_1)
```



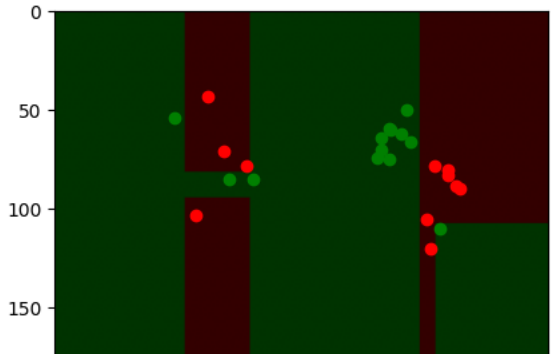
```
model = DecisionTreeClassifier()
samples = data_1[:, 0:2]
labels = data_1[:, 2]
model.fit(samples, labels)

result = classify_all_points(model)
plt.imshow(result)
show_points(data_1)
```




```
model = KNeighborsClassifier() # DecisionTreeClassifier()
samples = data_2[:, 0:2]
labels = data_2[:, 2]
model.fit(samples, labels)

result = classify_all_points(model)
plt.imshow(result)
show_points(data_2)
```




```
model = DecisionTreeClassifier()
samples = data_2[:, 0:2]
labels = data_2[:, 2]
model.fit(samples, labels)

result = classify_all_points(model)
plt.imshow(result)
show_points(data_2)
```



Using neuronal network

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential()
model.add(Dense(64, input_dim = 2, activation = 'sigmoid'))
model.add(Dense(64, activation = 'sigmoid'))
model.add(Dense(1, activation = 'sigmoid'))
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 64)                192

 dense_7 (Dense)             (None, 64)                4160

 dense_8 (Dense)             (None, 1)                 65

=================================================================
Total params: 4417 (17.25 KB)
Trainable params: 4417 (17.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
samples = data_2[:, 0:2]
labels = data_2[:, 2]
model.fit(samples, labels, epochs = 200, verbose = 0)

result = classify_all_points(model)
plt.imshow(result)
show_points(data_2)
```

```
1954/1954 [==============================] - 3s 1ms/step
```