# Importing Libraries

In [1]:
```
# import necessary libraries
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import pandas as pd
import tensorflow as tf
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score, cross_val_predict, valida
from sklearn.metrics import recall_score
warnings.filterwarnings("ignore")
```

# Reading Dataset

In [2]:
```
data_NN = pd.read_csv('NeuralNetwork.csv', header = None, names = ["X", "Y", "
```

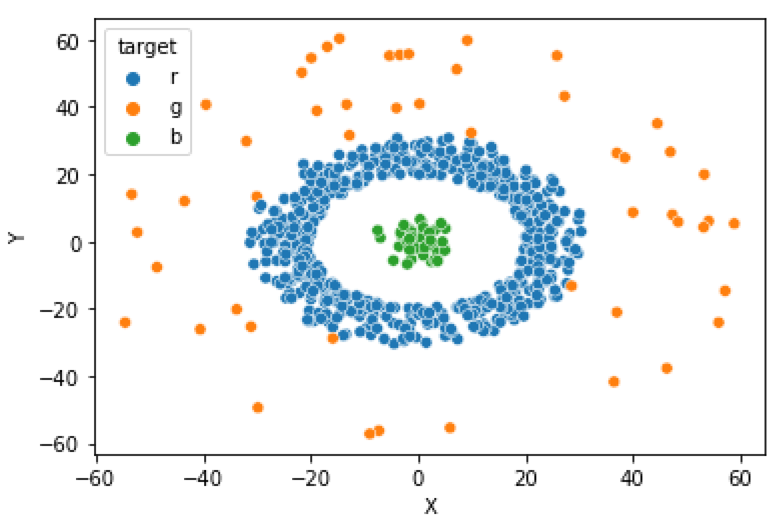
In [3]:
```
data_NN.head(5)
```

Out[3]:

| | X | Y | target |
|---|---|---|---|
| 0 | 16.263398 | 13.299206 | r |
| 1 | 0.775408 | 23.986692 | r |
| 2 | 29.170503 | -3.287474 | r |
| 3 | 6.739044 | -28.033329 | r |
| 4 | 3.216100 | 22.013695 | r |

In [4]:
```
sns.scatterplot(x = "X", y = "Y", data = data_NN, hue = "target")
```

Out[4]: <AxesSubplot:xlabel='X', ylabel='Y'>



In [5]:
```
pd.DataFrame(data_NN.describe()).style.format('{:.1f}')
```

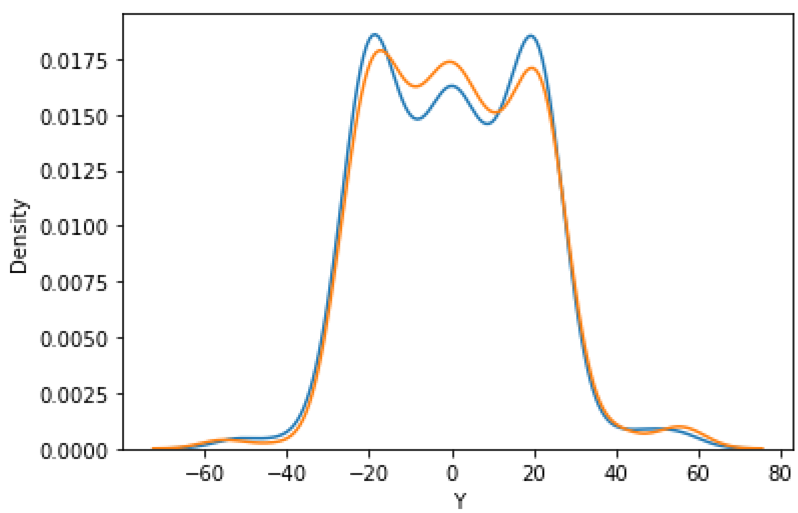
Out[5]:

| | X | Y |
|---|---|---|
| **count** | 685.0 | 685.0 |
| **mean** | 0.6 | 1.1 |
| **std** | 18.9 | 18.8 |
| **min** | -54.5 | -57.2 |
| **25%** | -16.4 | -14.8 |
| **50%** | 0.4 | 0.5 |
| **75%** | 17.3 | 16.6 |
| **max** | 58.9 | 60.3 |

Looking at the min, max, 25% and 75%, we can see that the data for X and Y both are using the same range is probably highly correlated.

In [6]:
```
X = data_NN.iloc[:,-3]
Y = data_NN.iloc[:,-2]
sns.distplot(X, hist = False)
sns.distplot(Y, hist = False)
```

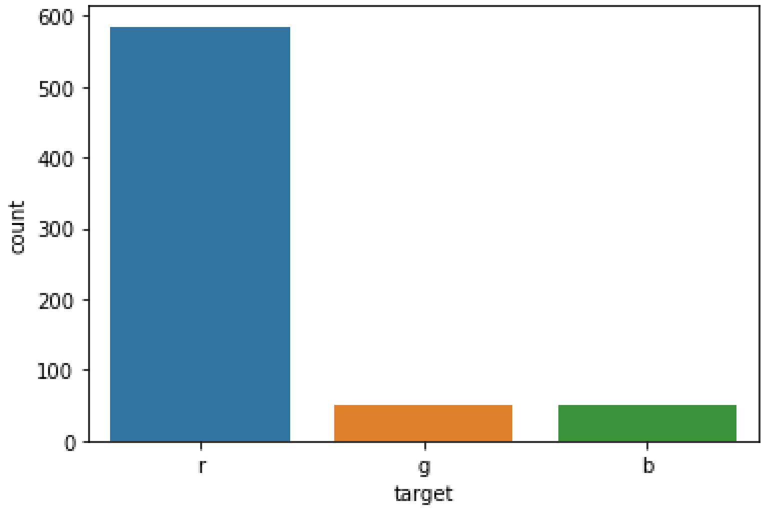
Out[6]: <AxesSubplot:xlabel='Y', ylabel='Density'>




The two input variable (X and Y) are highly correlated with each other as predicted by the description data.

In [7]:
```
sns.countplot(x= 'target', data = data_NN)
```

Out[7]: <AxesSubplot:xlabel='target', ylabel='count'>




There is class imbalance given that around 90% of data is in class "0." Within the model, we will implement class weights to help model better predict the target more accurately.

# Data Manipulation

In [8]:
```
#Replace target column with categorical numbers instead of letters
data_NN["target"].replace({"r":"0","g":"1","b":"2"}, inplace = True)
```

In [9]:
```
#Turn the target integer into string
data_NN['target'] = pd.Categorical(data_NN['target'])
```

In [10]:
```
#Seperate out input and output variables
X = data_NN.iloc[:, :-1]
y = data_NN.iloc[:, -1]
```

In [11]:
```
#Given the class imbalance, we will implement weights
counts = np.bincount(y)

weight_for_r = 1.0 / counts[0]
weight_for_g = 1.0 / counts[1]
weight_for_b = 1.0 / counts[2]

weight_for_r, weight_for_g, weight_for_b
```

Out[11]:
```
(0.0017094017094017094, 0.02, 0.02)
```

In [12]:
```
#Turn output variable into categorical variable so we can read it in the tenso
from tensorflow.keras.utils import to_categorical
y = to_categorical(y)
```

In [13]:
```
#split the database into test and trainning sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
```

In [14]:
```
#Standarize the dataset
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [15]:
```
#Making sure all the input and output variables are float64 for tensorflow mode
tf.keras.backend.set_floatx('float64')
```

In [16]:
```
#Create an array to store accuracy scores for all 12 models
acc_score = []
```

In [17]:
```
#Function for Neural Networks
def Neural_Network(layers, activationfunc, nueron):
    target_map_inverse = {0 : 'r', 1 : 'g', 2 : 'b'}
    if layers == 3:
        model = tf.keras.models.Sequential()
        model.add(tf.keras.layers.Dense(nueron*2, activation = 'relu'))
        model.add(tf.keras.layers.Dropout(0.3))
        model.add(tf.keras.layers.Dense(nueron, activation = 'relu'))
        model.add(tf.keras.layers.Dropout(0.3))
        model.add(tf.keras.layers.Dense(3, activation = activationfunc))
        model.compile(loss='categorical_crossentropy', optimizer='adam', metric
        history = model.fit(X_train, y_train, epochs=20, batch_size=16, valida
        y_pred = [target_map_inverse[i] for i in np.argmax(model.predict(X_tes
        cm = confusion_matrix([target_map_inverse[i] for i in np.argmax(y_test
        accuracyscore = accuracy_score([target_map_inverse[i] for i in np.argma
        acc_score.append(accuracyscore*100)
        print('Accuracy Score:', end = ' ')
        print(accuracyscore*100)
        print(' ')
        print('Confusion Matrix:')
        print(cm)

    if layers == 2:
        model = tf.keras.models.Sequential()
        model.add(tf.keras.layers.Dense(nueron, activation = 'relu'))
        model.add(tf.keras.layers.Dropout(0.3))
        model.add(tf.keras.layers.Dense(3, activation = activationfunc))
        model.compile(loss='categorical_crossentropy', optimizer='adam', metric
        history = model.fit(X_train, y_train, epochs=20, batch_size=16, valida
        y_pred = [target_map_inverse[i] for i in np.argmax(model.predict(X_tes
        cm = confusion_matrix([target_map_inverse[i] for i in np.argmax(y_test
        accuracyscore = accuracy_score([target_map_inverse[i] for i in np.argma
        acc_score.append(accuracyscore*100)
        print('Accuracy Score:', end = ' ')
        print(accuracyscore*100)
        print(' ')
        print('Confusion Matrix:')
        print(cm)

    if layers == 1:
        model = tf.keras.models.Sequential()
        model.add(tf.keras.layers.Dense(3, activation = activationfunc))
        model.compile(loss='categorical_crossentropy', optimizer='adam', metric
        history = model.fit(X_train, y_train, epochs=20, batch_size=16, valida
        y_pred = [target_map_inverse[i] for i in np.argmax(model.predict(X_tes
        cm = confusion_matrix([target_map_inverse[i] for i in np.argmax(y_test
        accuracyscore = accuracy_score([target_map_inverse[i] for i in np.argma
        acc_score.append(accuracyscore*100)
        print('Accuracy Score:', end = ' ')
        print(accuracyscore*100)
```

```
        print(' ')
        print('Confusion Matrix:')
        print(cm)
```

# Model Deployment

### The goal is to find the simplest model. We will different activations, number of nuerons and layers to see which model has the best accuracy and confusion matrix

MODEL 1: Layers - 1, activation function - softmax, nuerons - 32

In [18]:
```
Neural_Network(1, "softmax",32)
```

```
WARNING:tensorflow:From C:\Users\NPatel\Anaconda3\lib\site-packages\tensorflow
\python\ops\array_ops.py:5043: calling gather (from tensorflow.python.ops.arra
y_ops) with validate_indices is deprecated and will be removed in a future ver
sion.
Instructions for updating:
The `validate_indices` argument has no effect. Indices are always validated on
CPU and never validated on GPU.
Accuracy Score: 9.48905109489051

Confusion Matrix:
[[ 8  0  0]
 [ 3  1  2]
 [79 40  4]]
```

MODEL 2: Layers - 1, activation function - softmax, nuerons - 64

In [19]:
```
Neural_Network(1, "softmax",64)
```

```
Accuracy Score: 23.357664233576642

Confusion Matrix:
[[ 5  0  3]
 [ 4  1  1]
 [65 32 26]]
```

MODEL 3: Layers - 1, activation function - sigmoid, nuerons - 32

In [20]:
```
Neural_Network(1, "sigmoid",32)
```

```
Accuracy Score: 20.437956204379564

Confusion Matrix:
[[ 5  3  0]
 [ 3  2  1]
 [48 54 21]]
```

MODEL 4: Layers - 1, activation function - sigmoid, nuerons - 64

In [21]:
```
Neural_Network(1, "sigmoid",64)
```

```
Accuracy Score: 19.708029197080293

Confusion Matrix:
[[ 5  3  0]
 [ 3  2  1]
 [42 61 20]]
```

MODEL 5: Layers - 2, activation function - softmax, nuerons - 32,3

In [22]:
```
Neural_Network(2, "softmax",32)
```

```
Accuracy Score: 56.20437956204379

Confusion Matrix:
[[ 8  0  0]
 [ 0  2  4]
 [ 9 47 67]]
```

MODEL 6: Layers - 2, activation function - softmax, nuerons - 64,3

In [23]:
```
Neural_Network(2, "softmax",64)
```

```
Accuracy Score: 86.13138686131386

Confusion Matrix:
[[  8   0   0]
 [  0   5   1]
 [  0  18 105]]
```

MODEL 7: Layers - 2, activation function - softmax, nuerons - 128,3

In [24]:
```
Neural_Network(2, "softmax",128)
```

```
Accuracy Score: 96.35036496350365

Confusion Matrix:
[[  8   0   0]
 [  0   4   2]
 [  0   3 120]]
```

MODEL 8: Layers - 2, activation function - sigmoid, nuerons - 32,3

In [25]:
```
Neural_Network(2, "sigmoid",32)
```

```
Accuracy Score: 73.72262773722628

Confusion Matrix:
[[ 8  0  0]
 [ 0  4  2]
 [ 4 30 89]]
```

MODEL 9: Layers - 2, activation function - sigmoid, nuerons - 64,3

In [26]:
```
Neural_Network(2, "sigmoid",64)
```

```
Accuracy Score: 76.64233576642336

Confusion Matrix:
[[ 8  0  0]
 [ 0  4  2]
 [ 3 27 93]]
```

MODEL 10: Layers - 2, activation function - sigmoid, nuerons - 128,3

In [27]:
```
Neural_Network(2, "sigmoid",128)
```

```
Accuracy Score: 97.08029197080292

Confusion Matrix:
[[  8   0   0]
 [  0   5   1]
 [  0   3 120]]
```

MODEL 11: Layers - 3, activation function - softmax, nuerons - 64,32,3

In [28]:
```
Neural_Network(3,"softmax",32)
```

```
Accuracy Score: 99.27007299270073

Confusion Matrix:
[[  8   0   0]
 [  0   5   1]
 [  0   0 123]]
```

MODEL 12: Layers - 3, activation function - softmax, nuerons - 128,64,3

In [29]:
```
Neural_Network(3, "softmax",64)
```

```
Accuracy Score: 98.54014598540147

Confusion Matrix:
[[  8   0   0]
 [  0   4   2]
 [  0   0 123]]
```

MODEL 13: Layers - 3, activation function - sigmoid, nuerons - 64,32,3

In [30]:
```
Neural_Network(3, "sigmoid",32)
```

```
Accuracy Score: 94.16058394160584

Confusion Matrix:
[[  8   0   0]
 [  0   4   2]
 [  0   6 117]]
```

MODEL 14: Layers - 3, activation function - sigmoid, nuerons - 128,64,3

In [31]:
```
Neural_Network(3, "sigmoid",64)
```

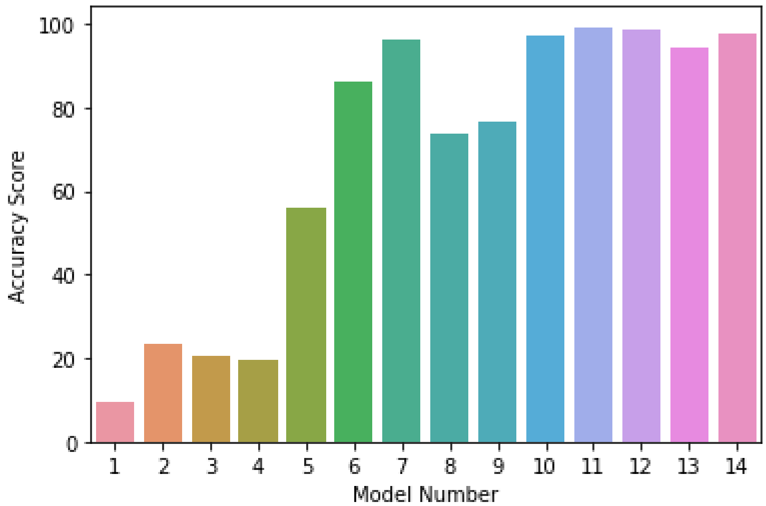
```
Accuracy Score: 97.8102189781022

Confusion Matrix:
[[  8   0   0]
 [  0   3   3]
 [  0   0 123]]
```

# Model Evaluation and Selection

In [32]:
```
Model = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
sns.barplot(x= Model, y=acc_score)
plt.xlabel('Model Number')
plt.ylabel("Accuracy Score")
```

Out[32]:
```
Text(0, 0.5, 'Accuracy Score')
```




Models 7 and 10 are the best models. These are our semi-complex models. It is using two layers (4 layers counting the drop out layers to prevent overfitting), the softmax or sigmoid function and 128 and 3 nuerons. Looking at the confusion matrix for these models, we can see less than ten instance are misclassified. The strategy used to find the simplest model was to start with a simple model and moniter the accuracy score. At every stage, we added more layers or nuerons. Once we found a model with high accuracy of 95% or above, we selected that model. More complex models with three layers have higher accuracy but aren't the simplest. We tested between the two activation fuctions but those aren't making a huge overall impact on the accuracy of the model for the given data. Note to avoid overfitting, 30% of drop out layer after each layer is added.

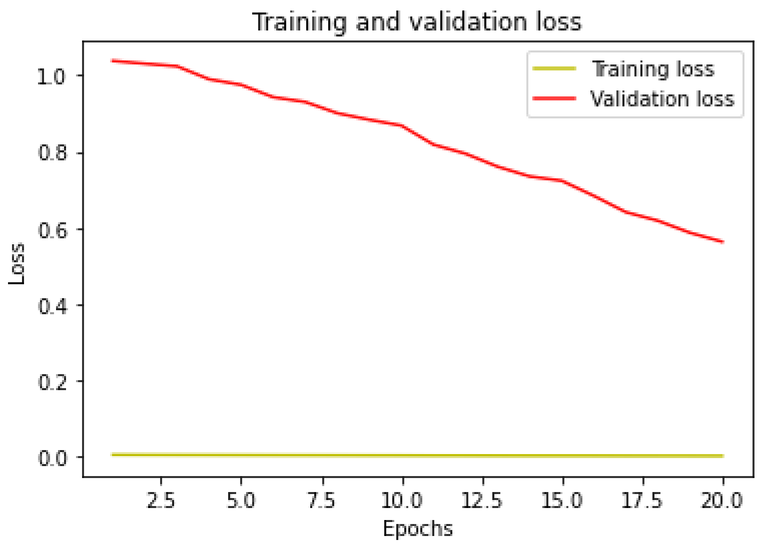
# Model Selection and Finalization (Sensitivity and Accuracy Matrix)

In [34]:
```
target_map_inverse = {0 : 'r', 1 : 'g', 2 : 'b'}
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(128, activation = 'relu'))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Dense(3, activation = 'sigmoid'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accu
history = model.fit(X_train, y_train, epochs=20, batch_size=16, validation_spli
y_pred = [target_map_inverse[i] for i in np.argmax(model.predict(X_test), axis=
cm = confusion_matrix([target_map_inverse[i] for i in np.argmax(y_test, axis =
accuracyscore = accuracy_score([target_map_inverse[i] for i in np.argmax(y_tes
acc_score.append(accuracyscore*100)
print('Accuracy Score:', end = ' ')
print(accuracyscore*100)
print(' ')
print('Confusion Matrix:')
print(cm)
```

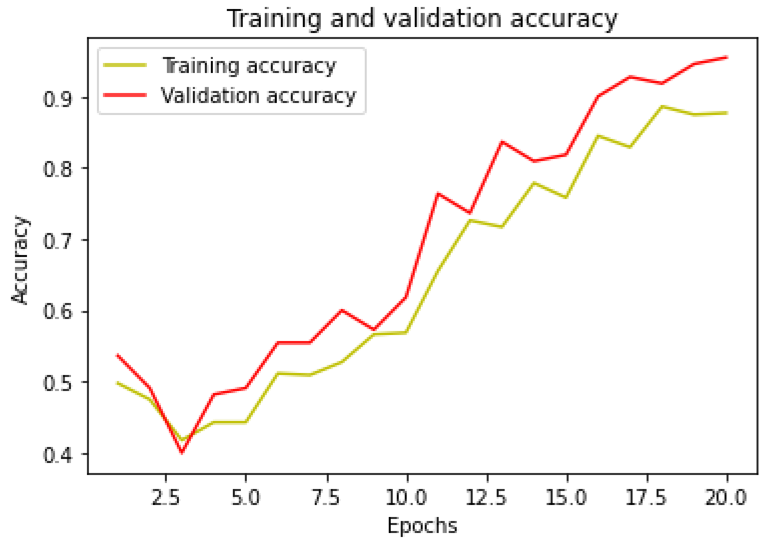
```
Accuracy Score: 96.35036496350365

Confusion Matrix:
[[  8   0   0]
 [  0   5   1]
 [  0   4 119]]
```

In [35]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




In [36]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, 'y', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




# New Attribute Testing

We will create four new attributes using the X and Y data we were provided in out original dataset. Then we are going to run these new attributes through our "best selected model" in section above.

In [37]:
```
#Create New Attributes
X3 = np.zeros(len(X))
X3 = data_NN['X']**2
X4 = np.zeros(len(X))
X4 = data_NN['Y']**2
X5 = np.zeros(len(X))
X5 = data_NN['X']*data_NN['Y']
```

In [38]:
```
#empty out acc_score array for new attributes
acc_score= []
```

New Attribute: Model 1

In [39]:
```
#Create attribute inputs columns, split trainning and test sets, standardize tl
X = np.vstack((X3,X4)).T
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ranc
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
Neural_Network(2, "softmax",128)
```

```
Accuracy Score: 99.27007299270073

Confusion Matrix:
[[  8   0   0]
 [  0   5   1]
 [  0   0 123]]
```

New Attribute: Model 2

In [40]:
```
#Create attribute inputs columns, split trainning and test sets, standardize tl
X = np.vstack((X3,X5)).T
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ranc
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
Neural_Network(2, "softmax",128)
```

```
Accuracy Score: 83.21167883211679

Confusion Matrix:
[[  8   0   0]
 [  0   4   2]
 [ 15   6 102]]
```

New Attribute: Model 3

In [41]:
```
#Create attribute inputs columns, split trainning and test sets, standardize th
X = X = np.vstack((X3,X4,X5)).T
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ranc
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
Neural_Network(2, "softmax",128)
```

```
Accuracy Score: 98.54014598540147

Confusion Matrix:
[[  8   0   0]
 [  0   4   2]
 [  0   0 123]]
```

New Attribute: Model 4

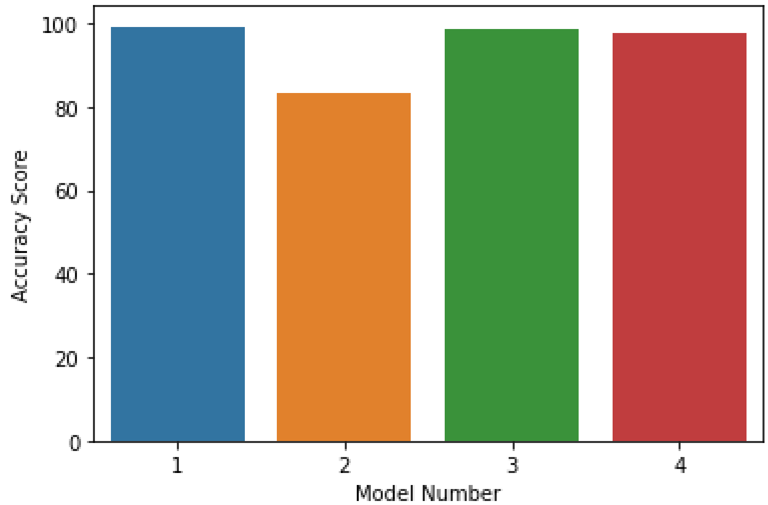
In [42]:
```
#Create attribute inputs columns, split trainning and test sets, standardize th
X1 = data_NN['X']
X2 = data_NN['Y']
X = np.vstack((X1,X2,X3,X4,X5)).T
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ranc
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
Neural_Network(2, "softmax",128)
```

```
Accuracy Score: 97.8102189781022

Confusion Matrix:
[[  8   0   0]
 [  0   4   2]
 [  0   1 122]]
```

In [43]:
```
Model = [1, 2, 3, 4]
sns.barplot(x= Model, y=acc_score)
plt.xlabel('Model Number')
plt.ylabel("Accuracy Score")
```

Out[43]:
```
Text(0, 0.5, 'Accuracy Score')
```




The accuracy scores for all the 4 new attributes as input has the best accuracy greater than 80%. This is predicted given that all the new attributes are combinations of, X1 and X2, the original variables used to select our "best model" . Note, we have added in the drop layer to prevent overfitting. Also, the accuracy score won't be the exactly same everytime the model is ran since it is dependend on GPU.

In [ ]: