MACHINE LEARNING

Assignment Report

BITS F464 (ML)

*Submitted By:*

Amit Anjandeb Ghosh 2021A7PS0181H

Soumil Ray 2021A7PS2652H

Aditya Aggarwal 2021A7PS2380H



**Naive Bayes Classifier**

We first loaded the dataset into a pandas data frame. We then split the dataset into training (67%) and testing data frames (33%).

We counted the number of <=50s and >50s in the target column of the training data. These two values were used to calculate the prior probabilities of the target classes. The prior probability of <=50K came out to be 0.753 whereas it was 0.246 for >50K.

We then implemented a function to calculate the conditional probabilities of each feature given to each class in the training set. We handled continuous data and categorical data separately. For categorical data, we substituted the training data into the Naive Bayes formula. For each continuous data, we split the training data into two groups based on the target value. Then we calculated the mean and variance of each group and made a normal distribution to generate the probability.

Finally, we use the prior probabilities and the conditional probabilities found earlier to calculate the probabilities of each row of the testing data corresponding to each of the 2 target classes. Based on whichever probability is higher, we classify the testing data.

We ran this through two models, one with Laplace Smoothing, and the other without Laplace Smoothing. In Laplace smoothing, we increment the numerator by ‘alpha’ and the denominator by ‘alpha \* k’ before computing the conditional probability, where alpha is a variable and k is the number of features. Practically, we assume that we have gone over every outcome ‘alpha’ times extra. In our case, k is 14, and we took alpha to be 1.

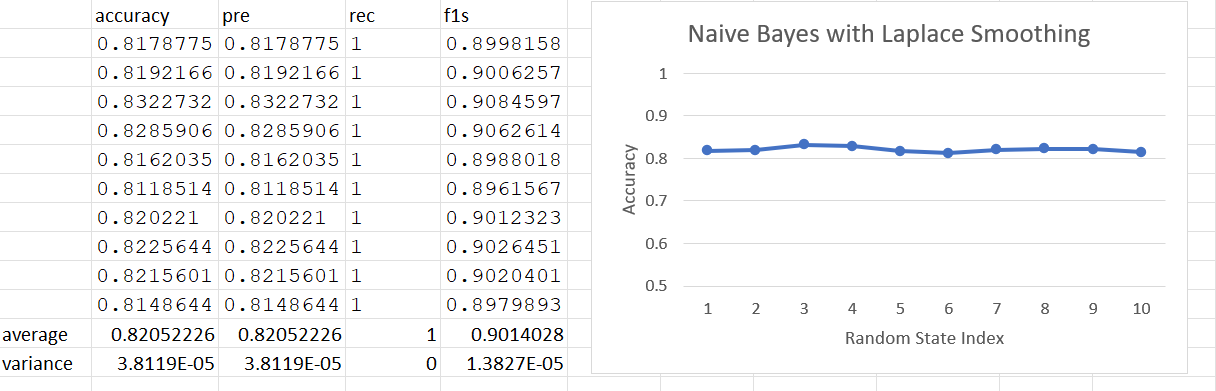
After classifying the testing data, we calculated the accuracy, precision, recall and F1-score.

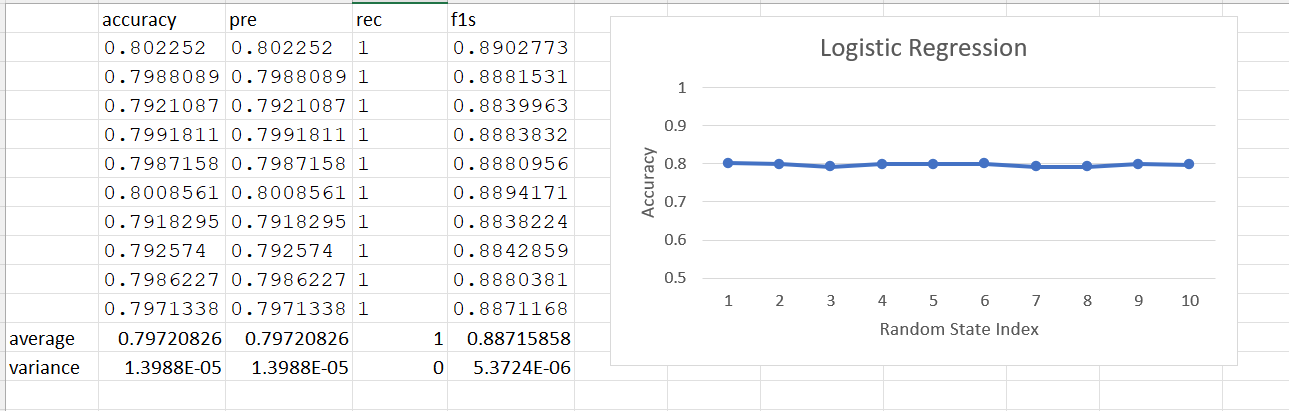
We also implemented Logistic Regression and k-nearest neighbors using scikit-learn and compared them with the Naive Bayes classifier. For KNN, we chose the value of k to be 5. We set the value of p to 2 because we wanted to use Euclidean distance as our distance metric.

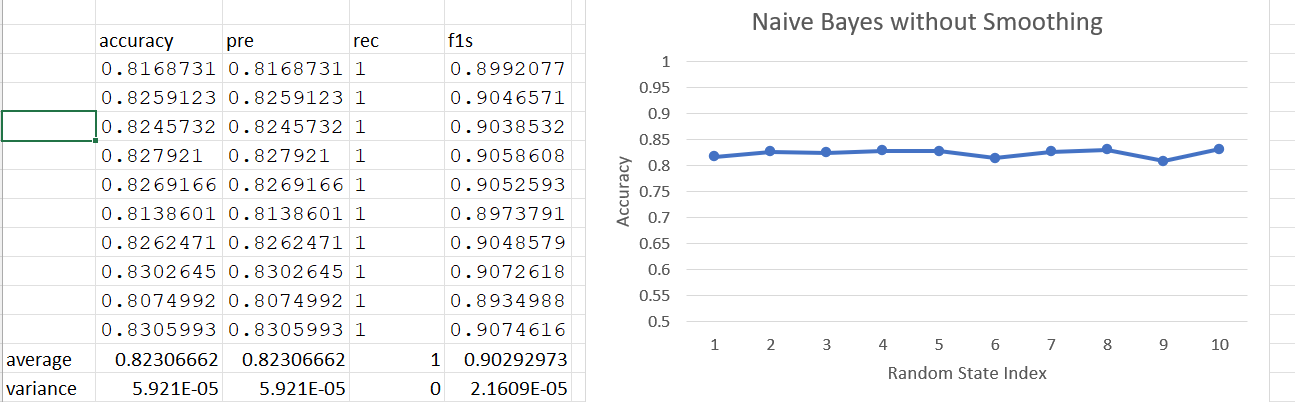
**Inferences**

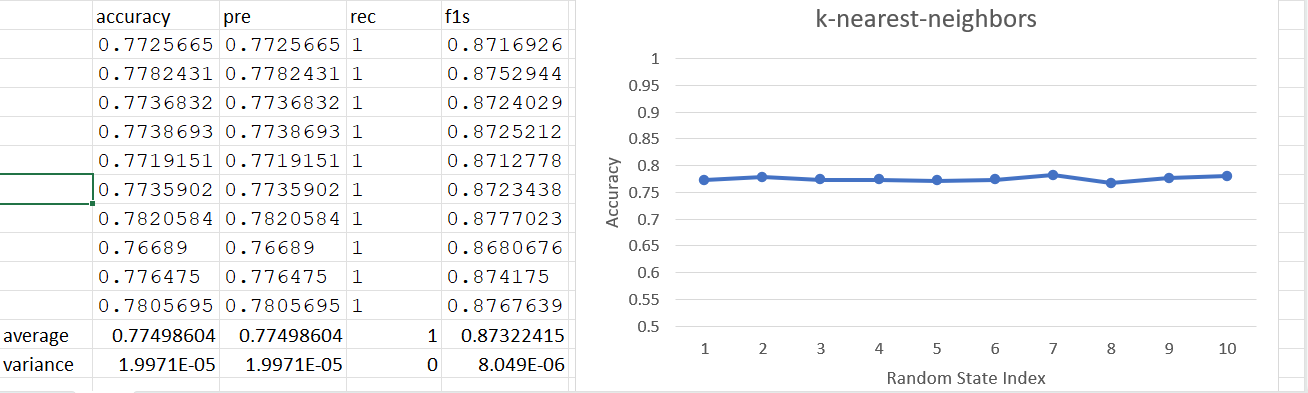
We observed that on average, Naive Bayes model performed better than logistic regression and k-nearest neighbors. The performance metrics are shown below.

The differences between the Naïve Bayes Classifier with Laplace Smoothing and the one without were largely statistically insignificant. This could be due to the nature of the data at large.









**Artificial Neural Network**

**Implementation:**

At first, we manipulated the given dataset so as to implement a 67:33 train-test split, as opposed to the 85:15 in built train test split that TensorFlow was providing with the dataset. To do this, we had to merge the training and test arrays, shuffle, then proceed to split into train and test again.

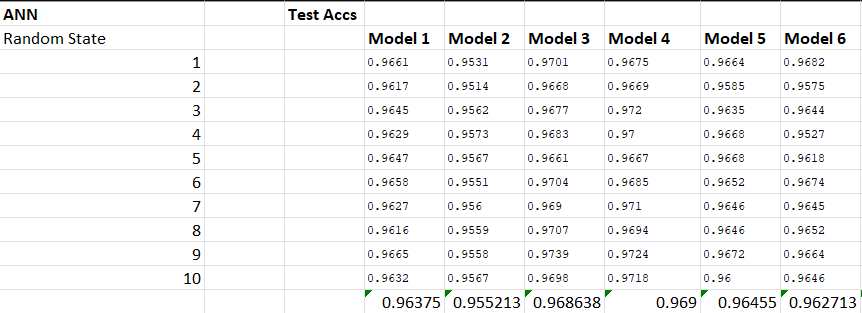
We performed the required preprocessing by dividing all the feature values by 255 in order to scale them uniformly given the RGB color values. Thereafter, we chose to flatten the dataset in order to make the first input layer of the neural network have 784 values. This is because the dataset, as it was, contained data in the form of 28\*28 arrays, which would be unfeasible to put as the first layer of a neural network. Thus, we chose to convert it all to 1D.

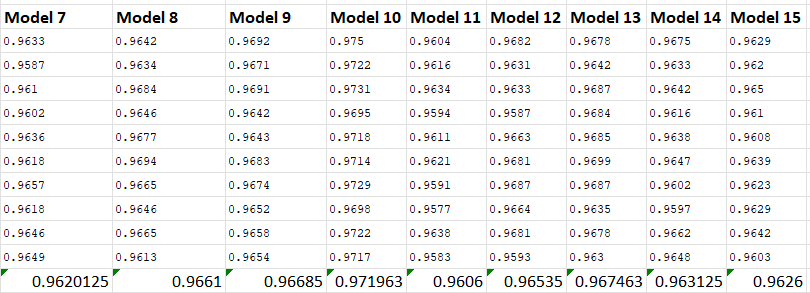
We then proceeded to implement the Neural Networks using TensorFlow. We set the epochs to 5 – implying that the Neural Networks would train themselves over the complete training data 5 times.

We constructed 15 such Neural Network models, each having 100 - 150 total neurons in their hidden layers, and each utilizing sigmoid, tanh, or ReLU as their activation functions. We varied the activation functions and the number of neurons in the models in a phased manner so as to draw better inferences.

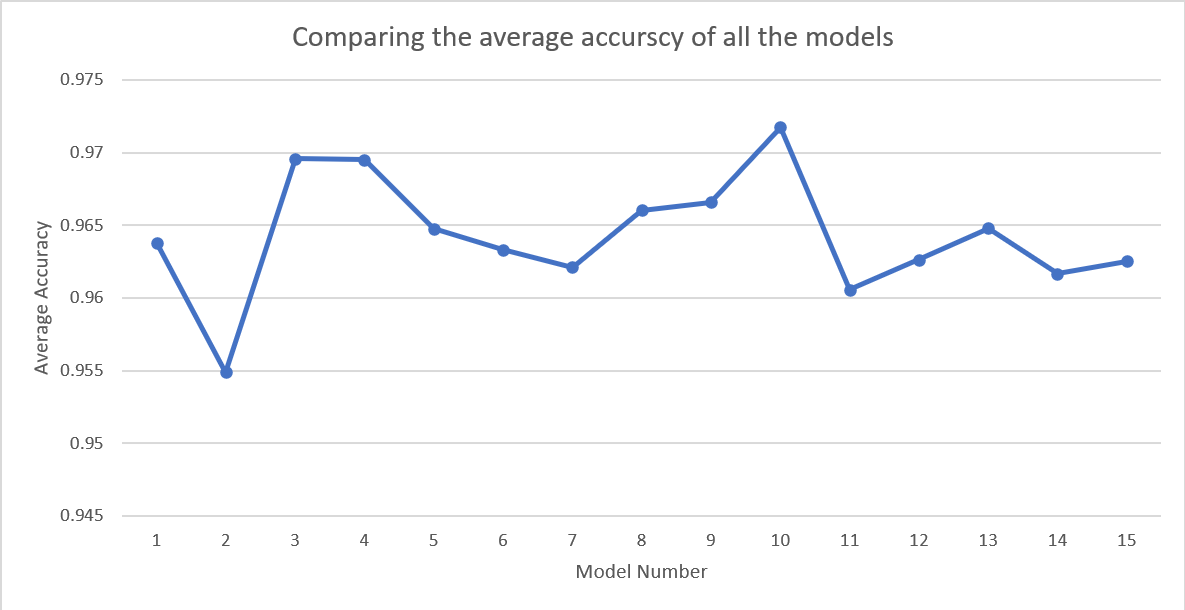
For each of these neural networks, we then proceeded to evaluate the performance over the test data, and then found out the predicted digit value using the argmax function. These predicted values were later used to construct a Confusion Matrix with the help of the inbuilt TensorFlow function tf.math.confusion\_matrix

**Cumulative data for all the models**



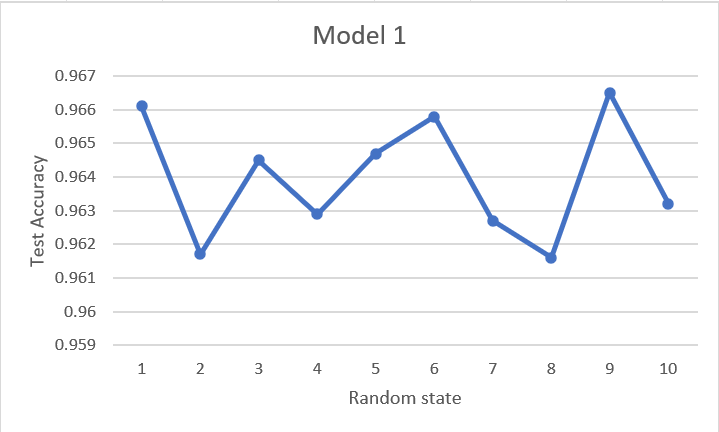
****

**AVERAGE ACCURACIES OVER ALL THE MODELS**

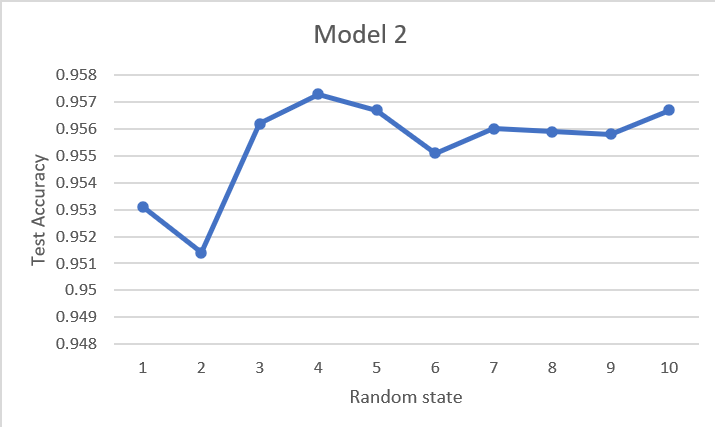
****

**Model by Model specifications + performance -**

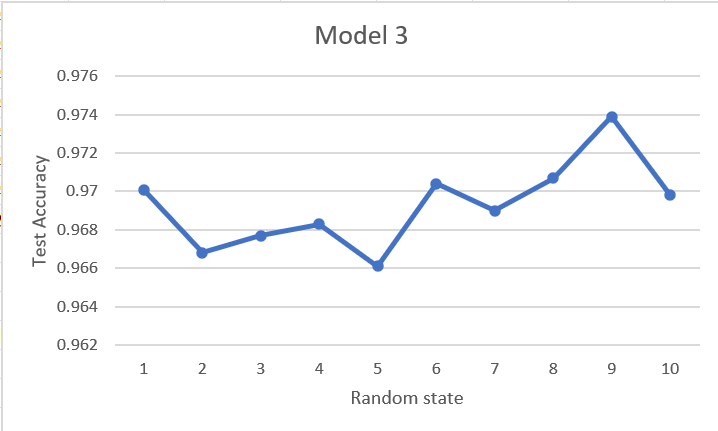
# MODEL 1 -- 150 neurons, 2 Hidden layers, both having sigmoid activation function. Both Hidden Layers with 75 Neurons. Output layer activation function Softmax.



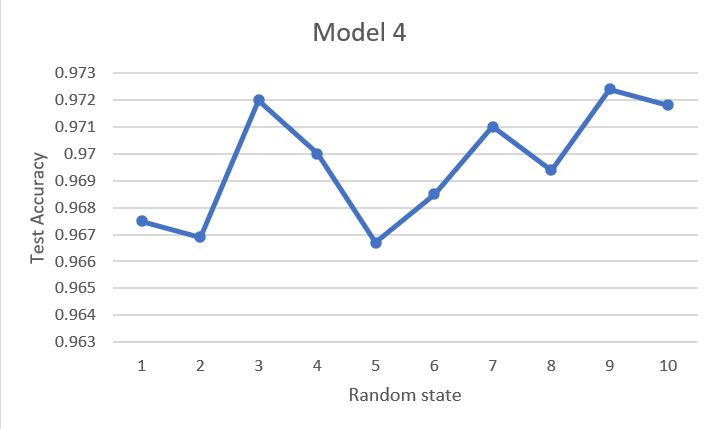
# MODEL 2 -- 100 Neurons, 3 Hidden Layers, All having sigmoid apart from the output layer, which uses Softmax.



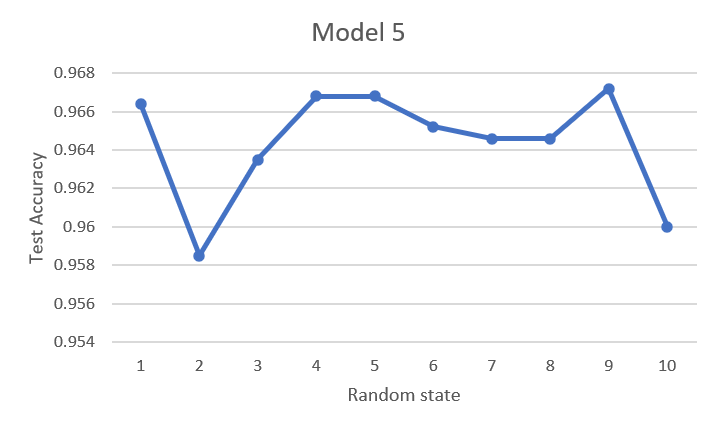
# MODEL 3 -- 150 Neurons, 2 Hidden Layers, both layers having ReLU. Each hidden layer has 75 neurons each. Output layer has Softmax Activation.



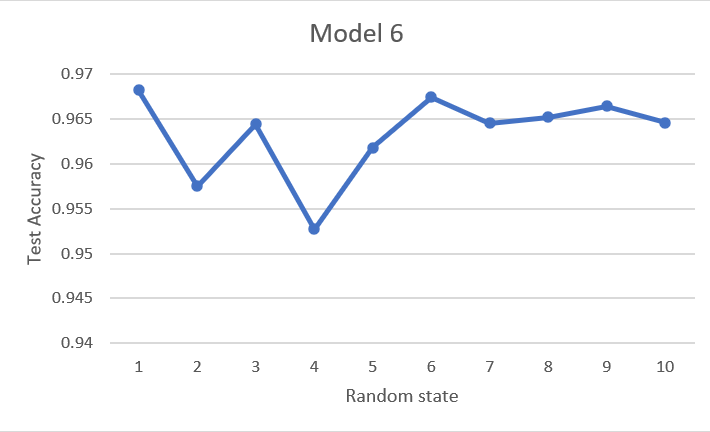
# MODEL 4 -- 150 Neurons, 2 Hidden Layers, both of 75 neurons each. Both hidden layers have the **tanh activation function**. The output layer has the softmax activation function



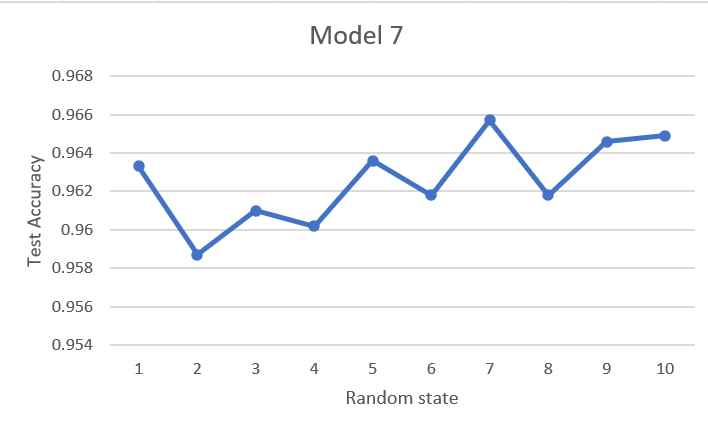
# **MODEL 5** -- 100 Neurons, 3 Hidden Layers – first layer having 50 neurons, second layer having 30 neurons, third layer having 20 neurons. All hidden layers have the ReLU activation function. Output layer has the **softmax** activation function.



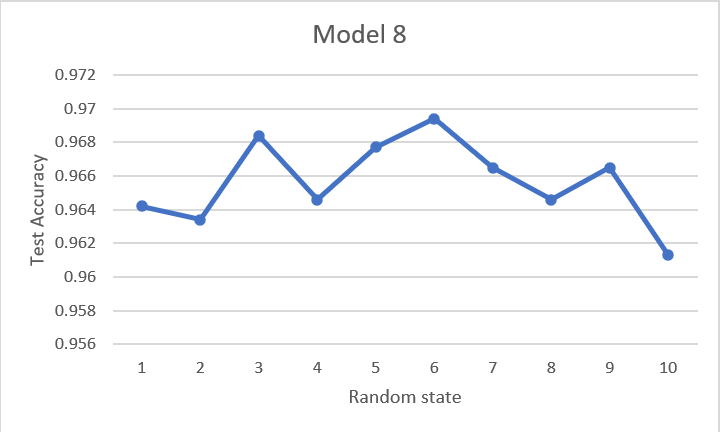
# MODEL 6 -- 100 Neurons, 3 Hidden Layers – first layer having 50 neurons, second layer having 30 neurons, third layer having 20 neurons. All hidden layers having tanh activation function. Output layer has the softmax activation function.



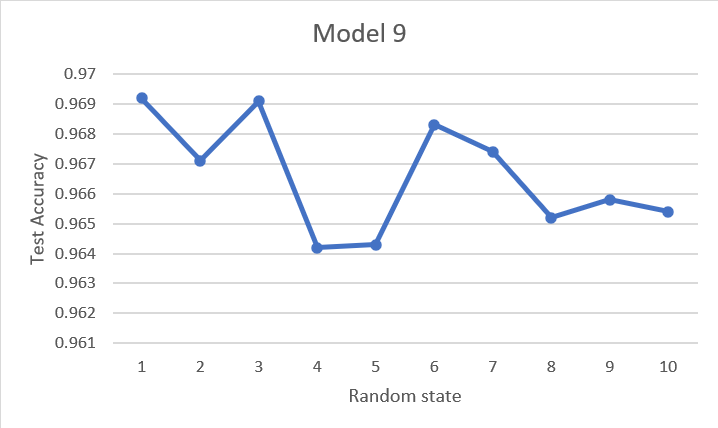
# MODEL 7 -- 100 Neurons, 3 Hidden Layers – 50 Neurons in layer 1, 30 in layer 2, 20 in layer 3. **Sigmoid used as the activation function for the first layer, ReLU for the second, tanh for the third.** Output layer has softmax activation.



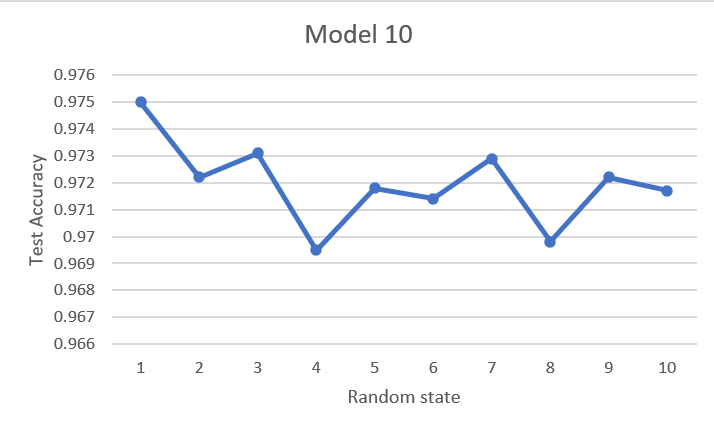
# MODEL 8 -- 150 Neurons, 2 Hidden Layers – Each hidden layer having 75 Neurons each. **Sigmoid activation for the first hidden layer, ReLU for the second.** Output layer has softmax activation.



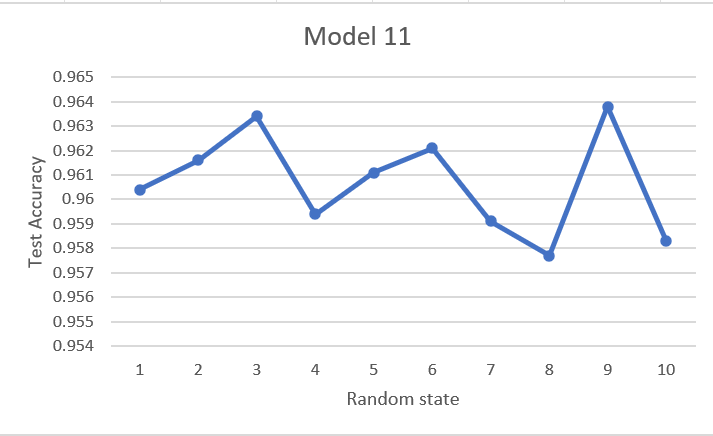
# MODEL 9 -- 150 Neurons, 2 Hidden Layers. Each hidden layer has 75 neurons each. First layer has **sigmoid activation,** second layer has **tanh activation.** Output layer has softmax activation.



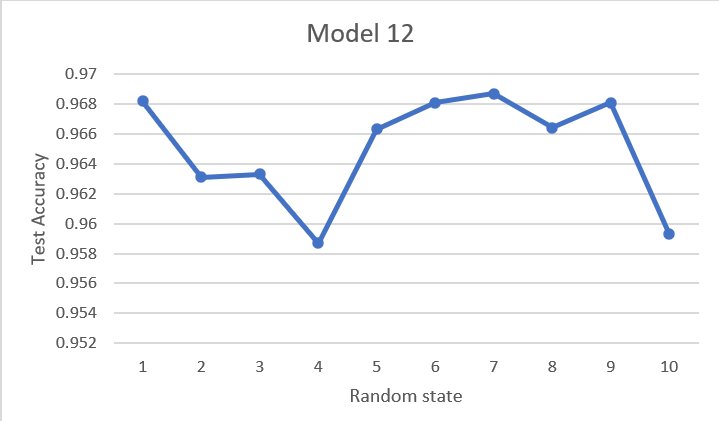
# MODEL 10 -- 150 Neurons, 2 Hidden Layers. Each hidden layer has 75 neurons each. **ReLU used as the activation function for the first layer, tanh for the second.** Output layer has softmax activation.



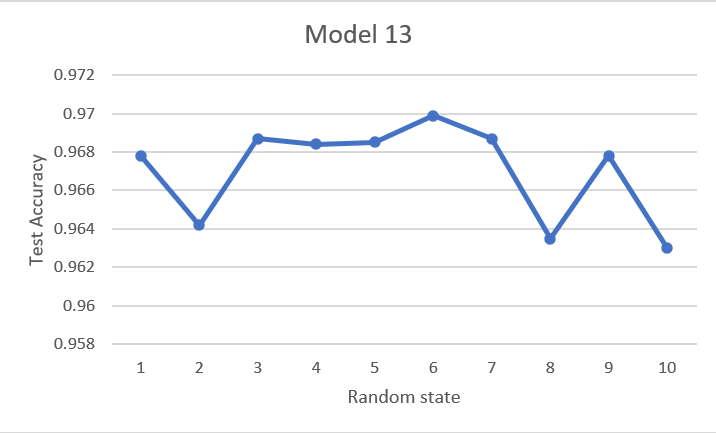
# MODEL 11 -- 150 Neurons, 3 Hidden Layers. **First hidden layer - 60 neurons, second hidden layer - 50 neurons, third hidden layer - 40 neurons.** All activation functions for hidden layers **sigmoid.** Output layer has softmax activation.



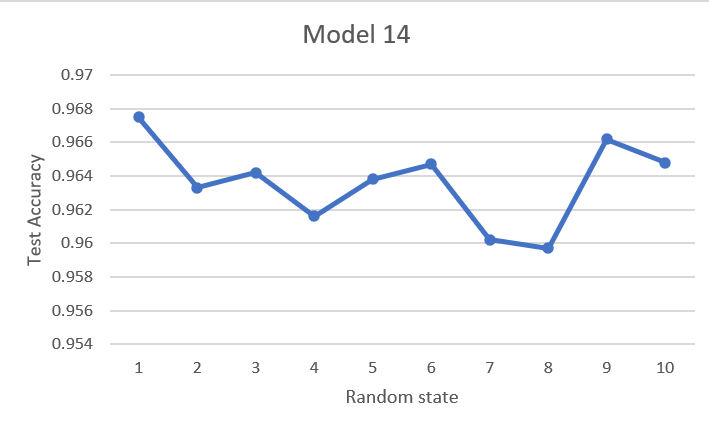
# MODEL 12 -- 150 Neurons, 3 Hidden Layers – first hidden layer having 60 neurons, second hidden layer having 50 neurons, third hidden layer having 40 neurons. **All hidden layers have ReLU activation function.** Output layer has softmax activation.



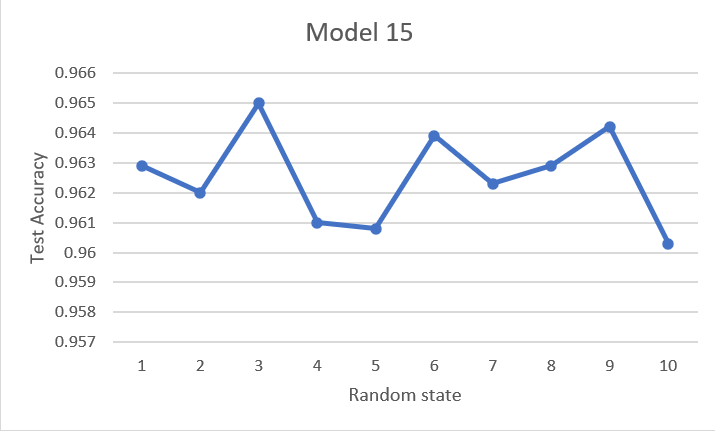
# MODEL 13 -- 150 Neurons, 3 Hidden Layers – first hidden layer having 60 neurons, second hidden layer having 50 neurons, third hidden layer having 40 neurons. **All hidden layers have tanh activation function.** Output layer has softmax activation.



# MODEL 14 -- 150 Neurons, 3 Hidden Layers – first hidden layer having 60 neurons, second hidden layer having 50 neurons, third hidden layer having 40 neurons. **First hidden layer has Sigmoid activation, second has ReLU, third has tanh.** Output layer has softmax activation.

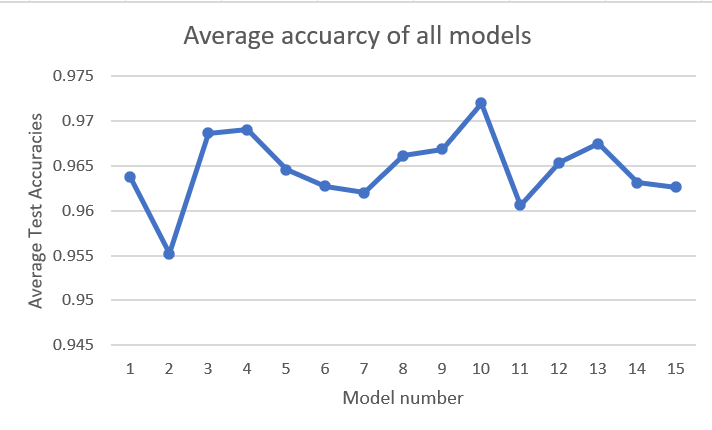


# MODEL 15 -- 100 Neurons, 2 Hidden Layers, First hidden layer has 50 neurons and sigmoid activation, second has 50 neurons and ReLU activation. Output layer has softmax activation.



**INFERENCES**

As seen from the graph comparing the average test accuracies (pasted again below for reference) of all the models for each of the 10 random states, **model 10** achieves the highest average test accuracy – at 97.196 percent.

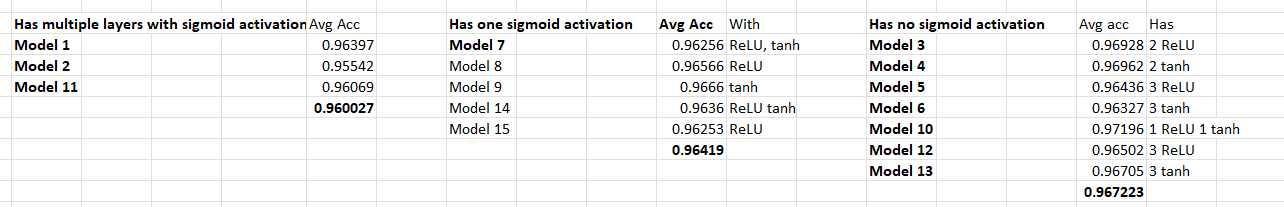


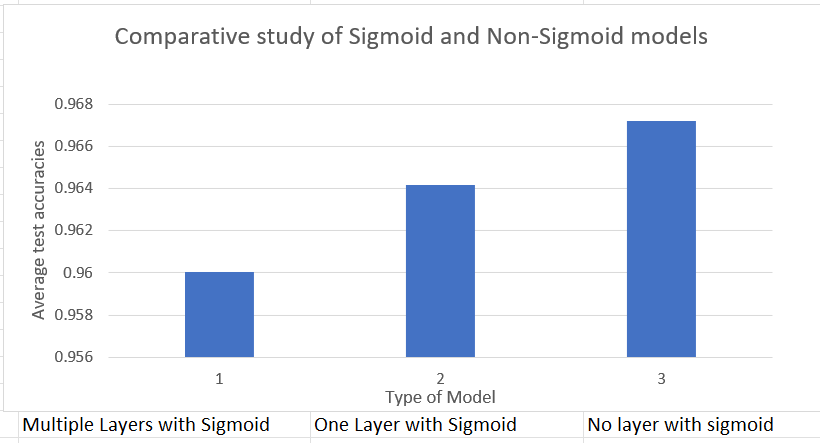
**Models 3 and 4** are not statistically significant from **Model 10 –** having average test accuracies of **0.96864** and **0.969** respectively.

Model 10 utilizes **ReLU and tanh activation over 150 neurons,** while Model 3 uses only **ReLU** over 150 neurons, and Model 4 uses only **tanh** over 150 neurons.

A comparative look at the accuracies of the models with 150 neurons and 100 neurons shows that the accuracies of models with 150 neurons seems to be higher.

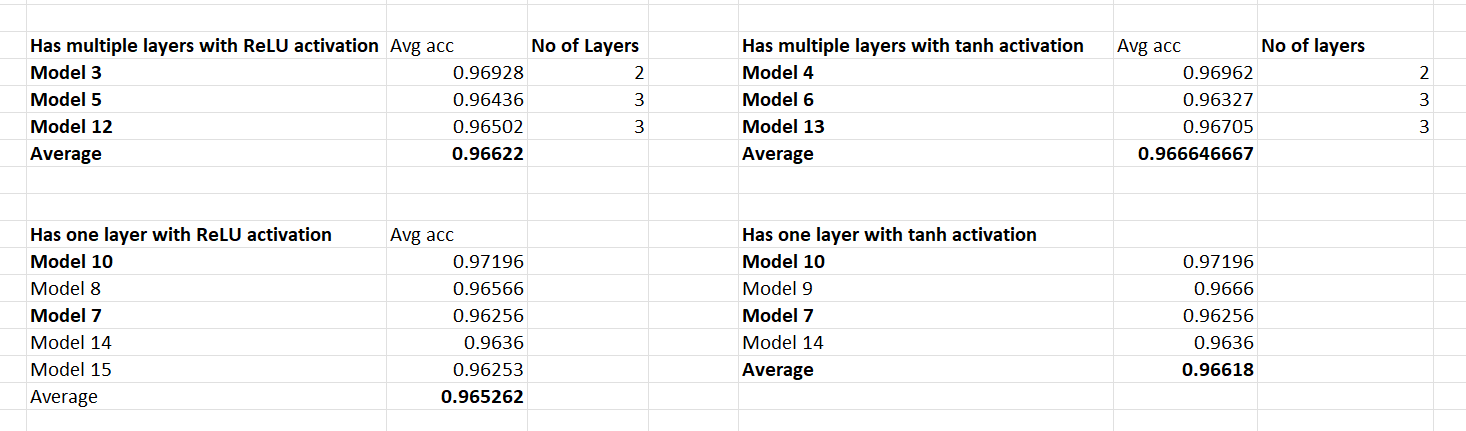
Also, through the best performing models, it appears that ReLU and tanh are more fitting activation functions for this data than sigmoid is at least, since all the best performing models which are not statistically significant from each other involve only the ReLU and the tanh activation function.

****



The above table and graph showcases the average accuracies for models having multiple layers with sigmoid activation function, one layer with sigmoid activation function, and no sigmoid activation function at all. A steady increase in the average accuracies is visible.

We can note some variations in individual test accuracies model by model, but that can largely be laid down to the fact that there are multiple factors contributing. However, on an average, we can conclude that the sigmoid activation function is performing the worst of the three.



From the above table, we can conclude that the difference between the models utilizing tanh and the models utilizing ReLU activation function is largely statistically insignificant

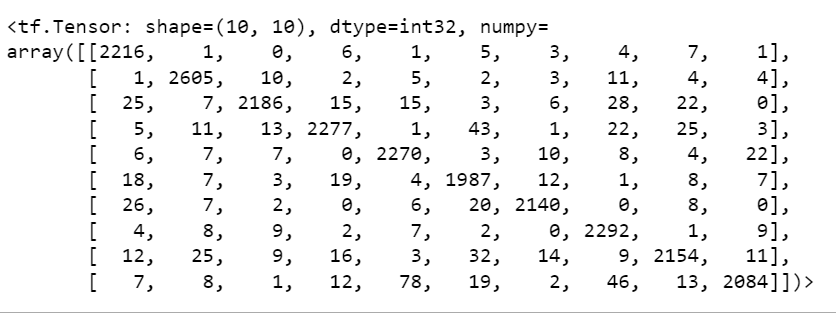
**CONFUSION MATRICES**

Notation: Y-axis: **Predictions. (0-9)** X-axis: **Actual (0-9)**

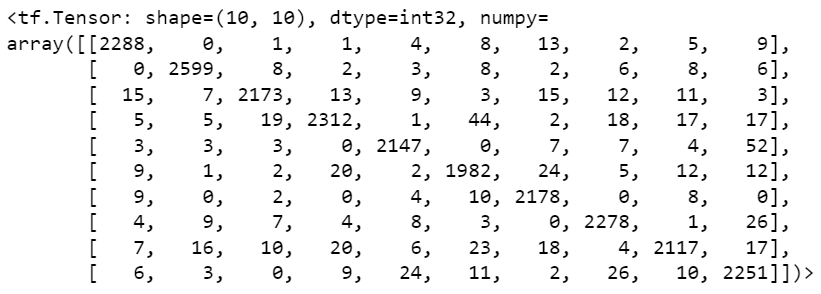
**Model 1**

========

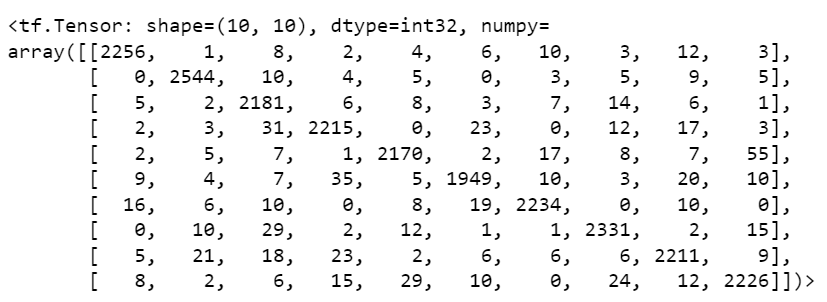
**Random State: 10**



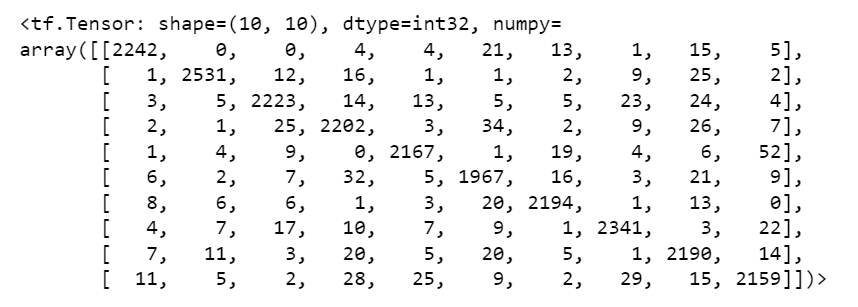
**Random State: 9**

****

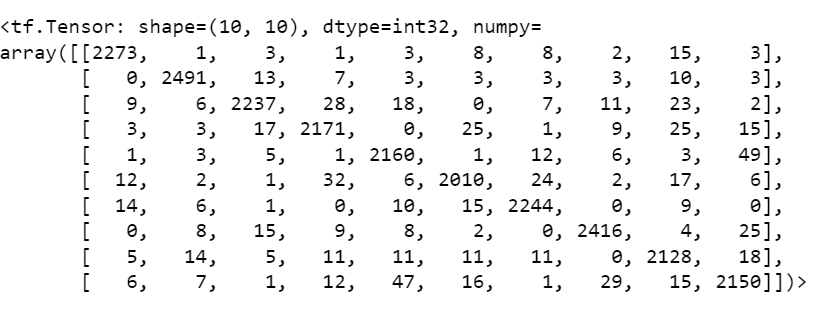
**Random State: 1**

****

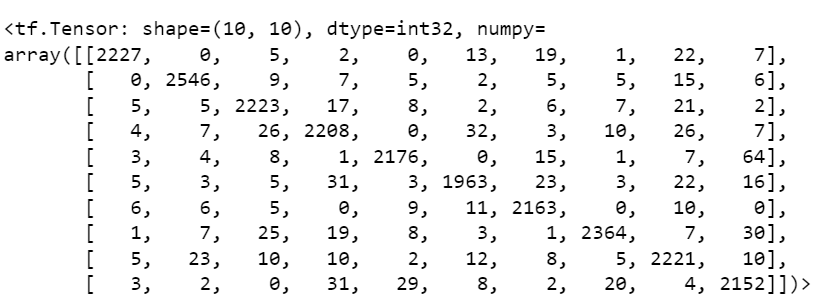
**Random State: 2**

****

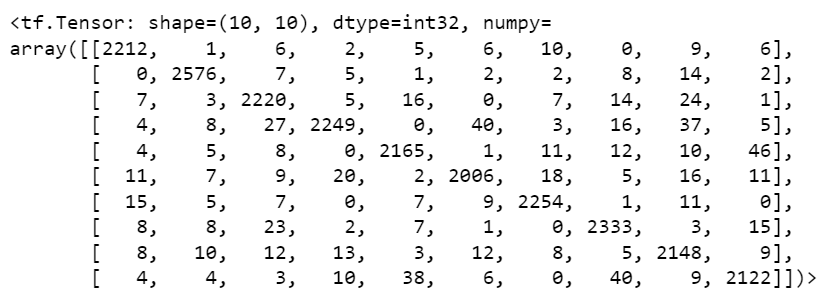
**Random State 3:**

****

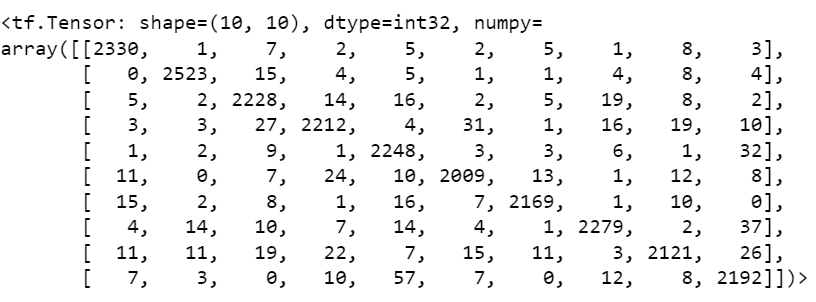
**Random State 4:**

****

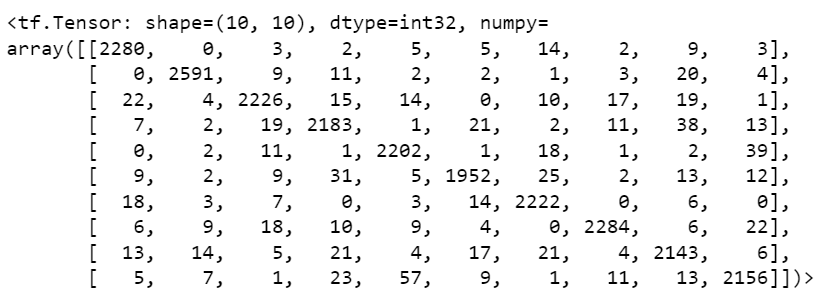
**Random State 5:**

****

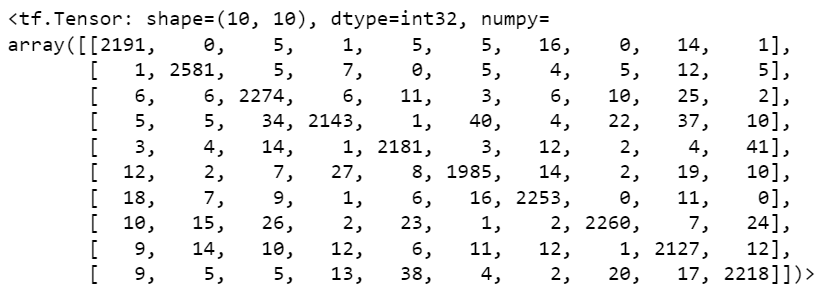
**Random State 6:**

****

**Random State 7:**

****

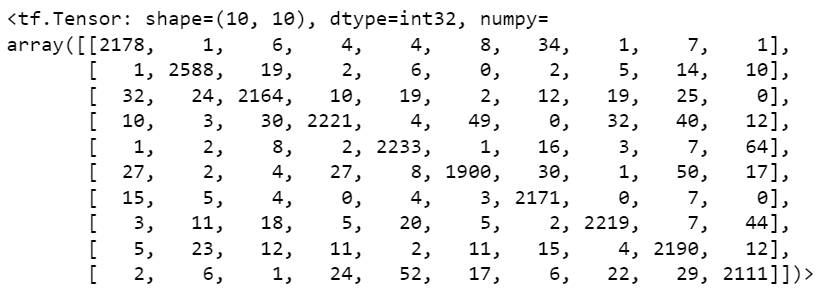
**Random State 8:**

****

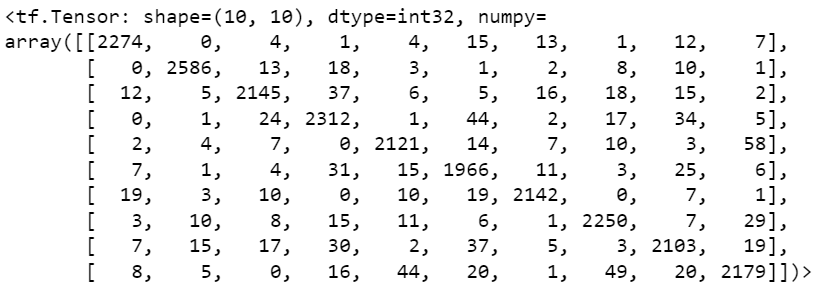
**MODEL 2**

**=======**

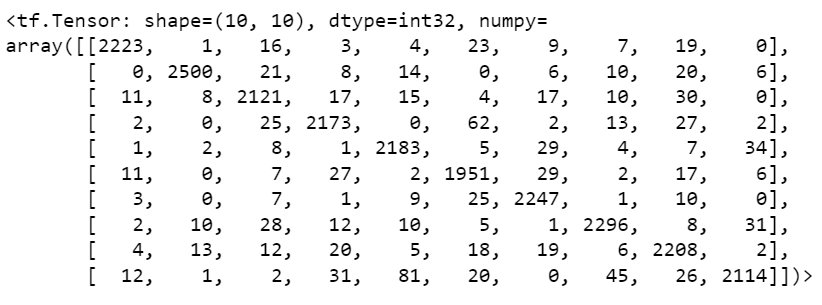
**Random State: 10**

****

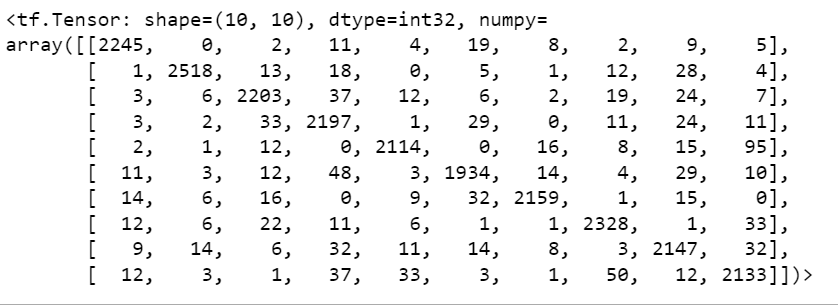
**Random State: 9**

****

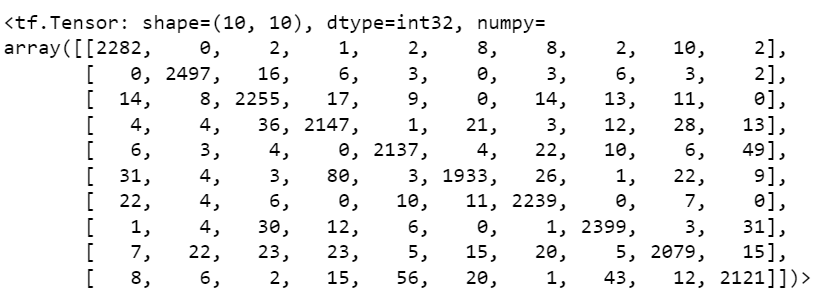
**Random State 1:**

****

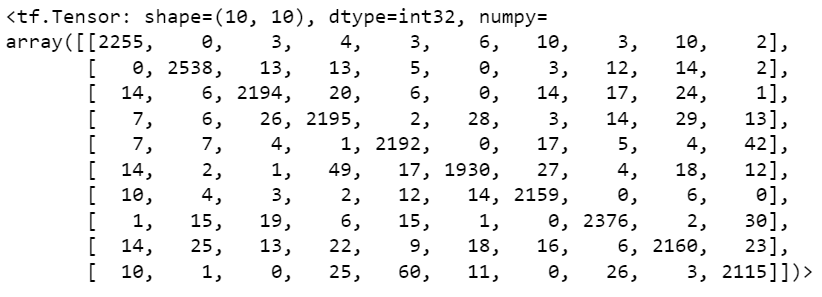
**Random State 2:**

****

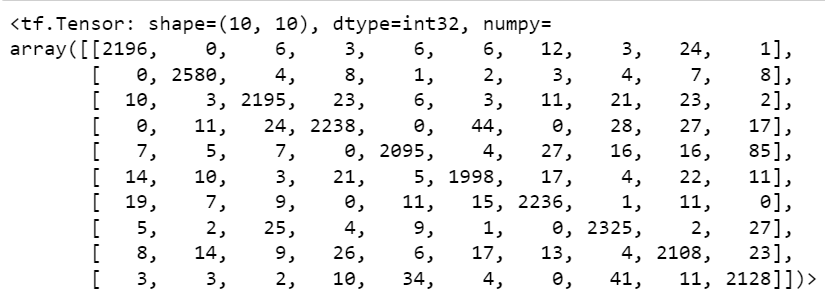
**Random State 3:**

****

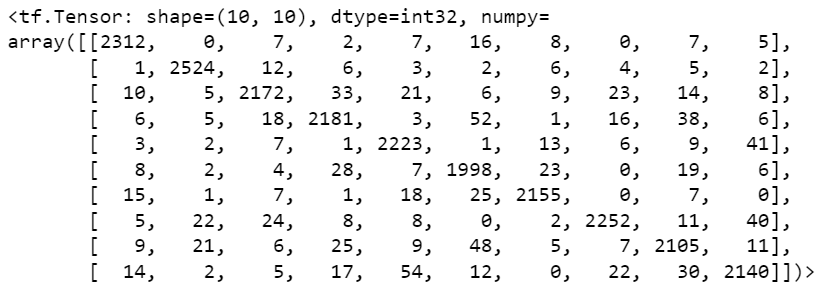
**Random State 4:**

****

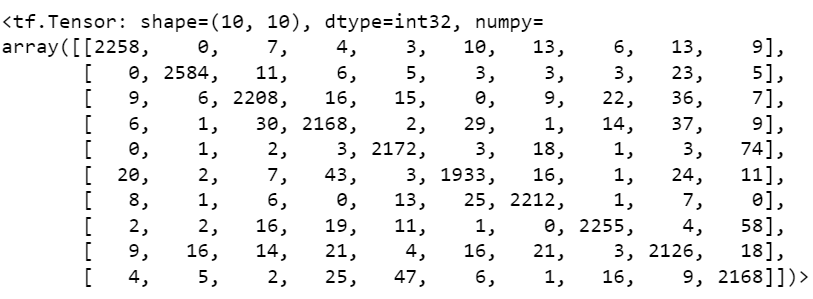
**Random State 5:**

****

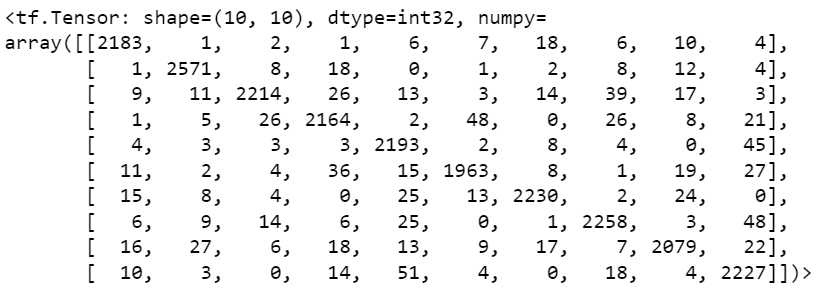
**Random State 6:**

****

**Random State 7:**

****

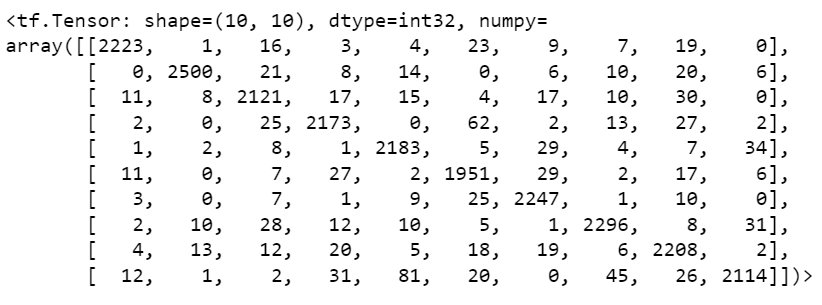
**Random State 8:**

****

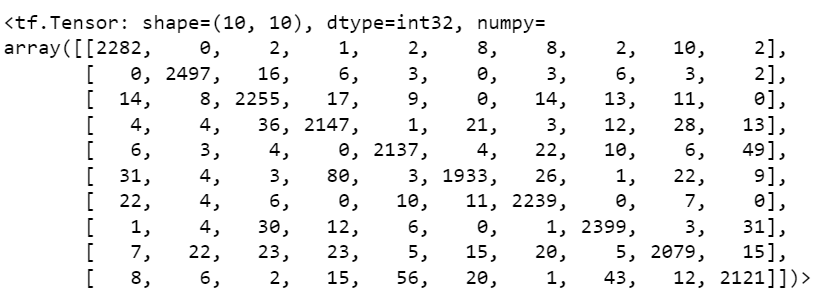
**Model 3**

========

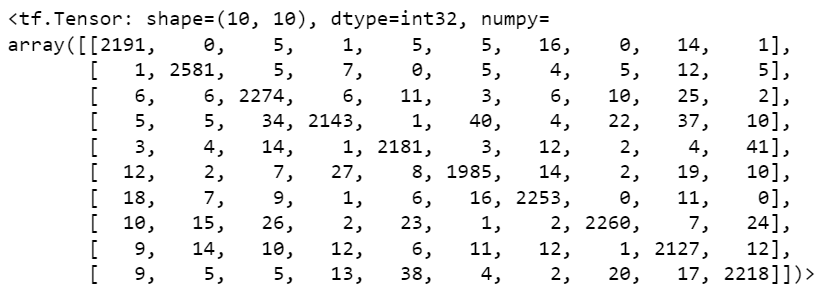
**Random State: 10**

****

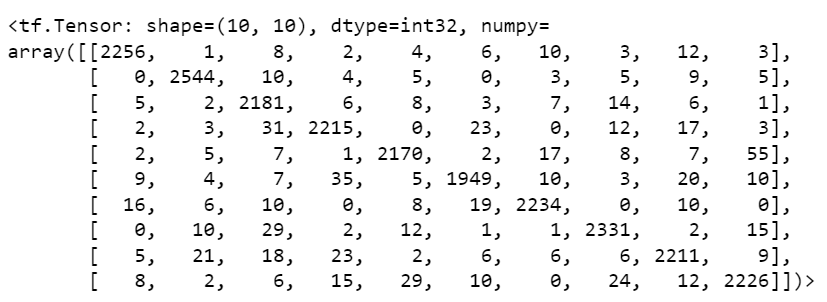
**Random State: 9**

****

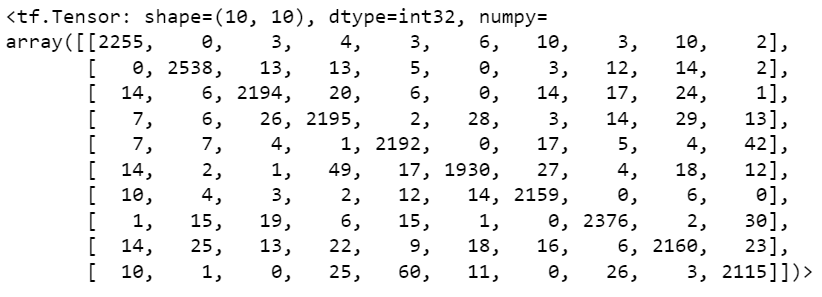
**Random State: 1**

****

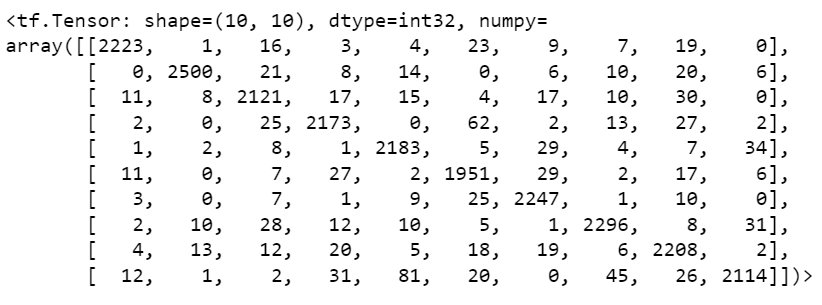
**Random State: 2**

****

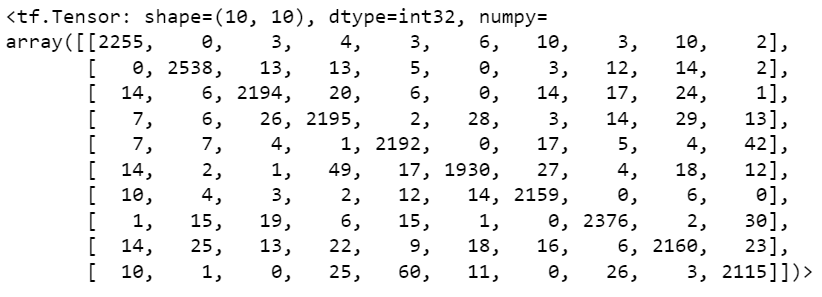
**Random State 3:**

****

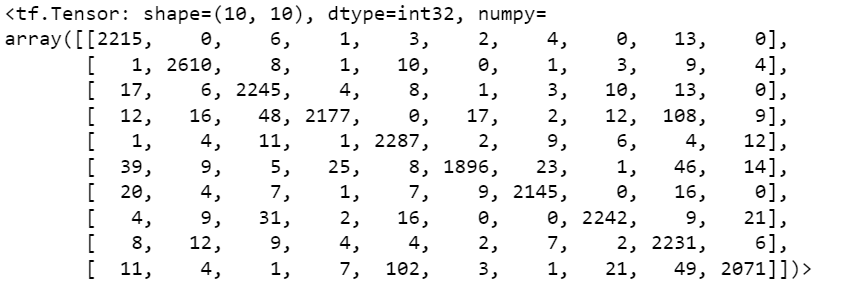
**Random State 4:**

****

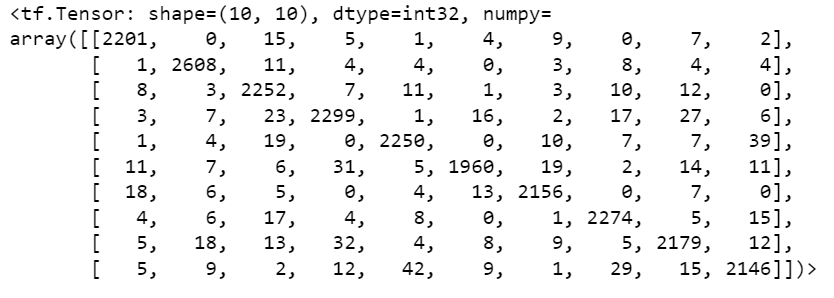
**Random State 5:**

****

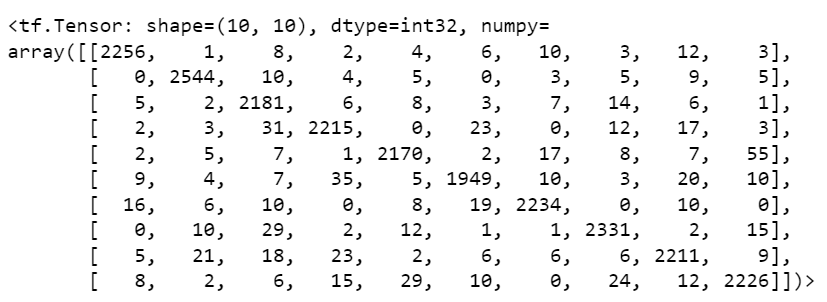
**Random State 6:**

****

**Random State 7:**

****

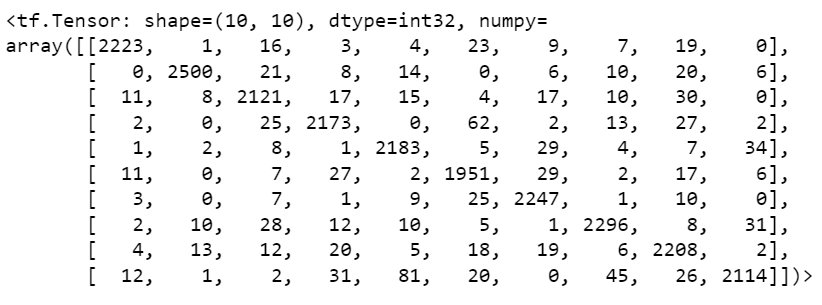
**Random State 8:**

****

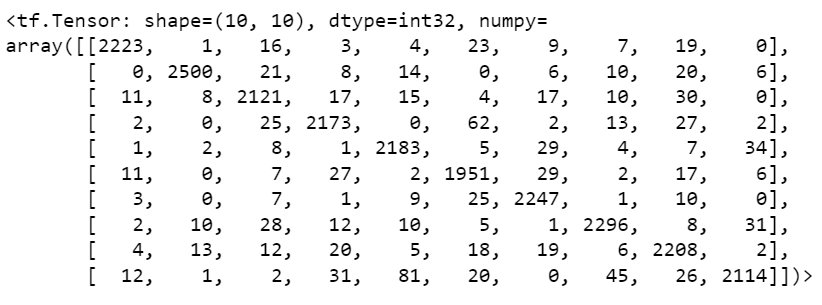
**MODEL 4**

**=======**

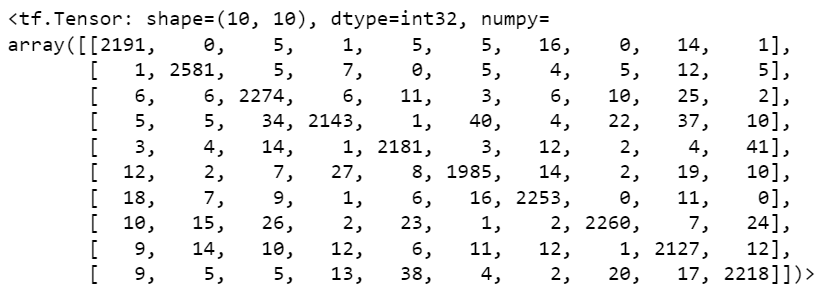
**Random State: 10**

****

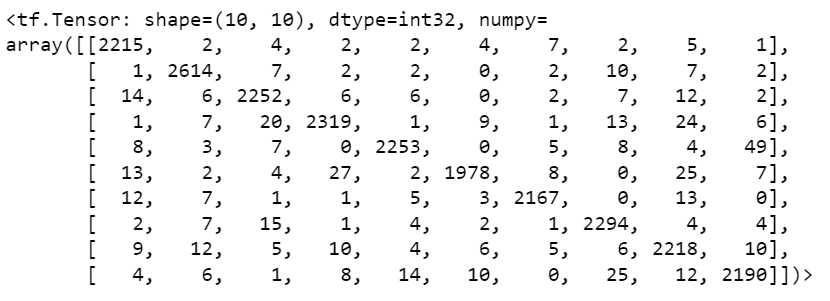
**Random State: 9**

****

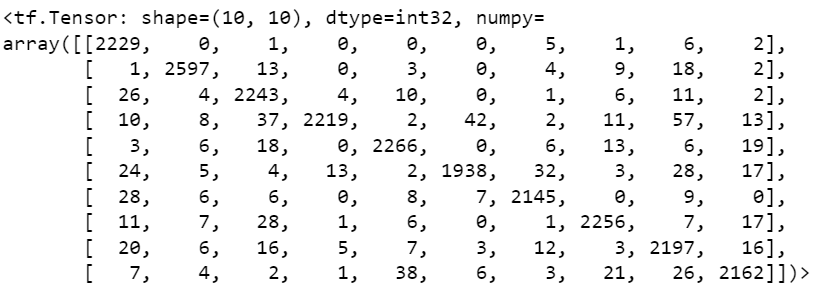
**Random State 1:**

****

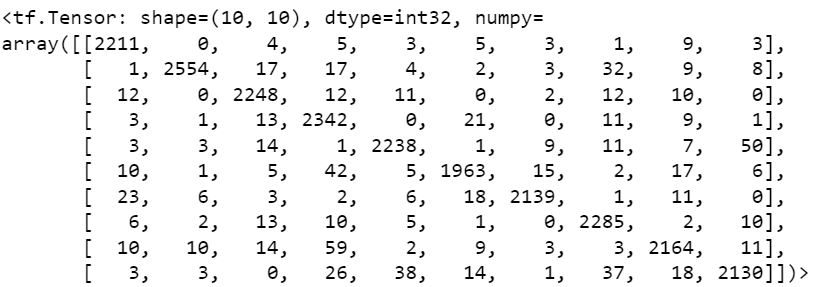
**Random State 2:**

****

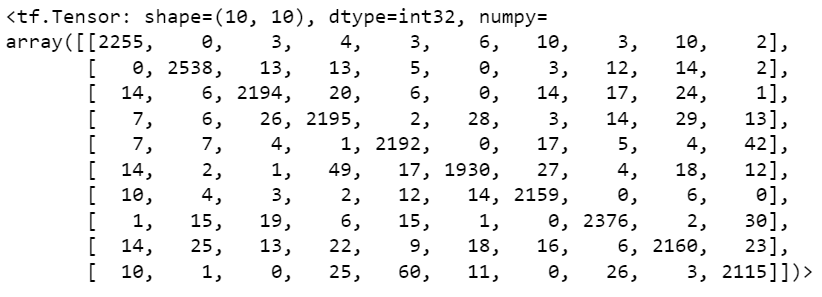
**Random State 3:**

****

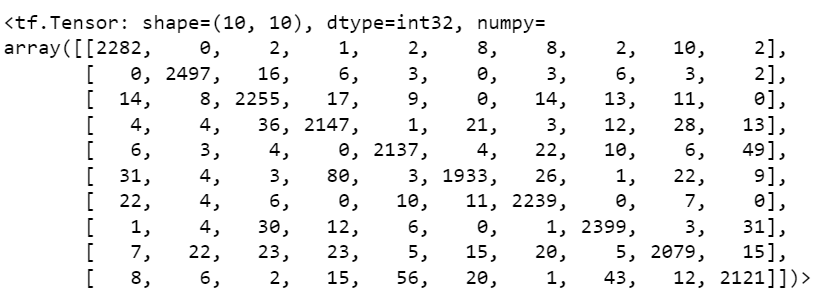
**Random State 4:**

****

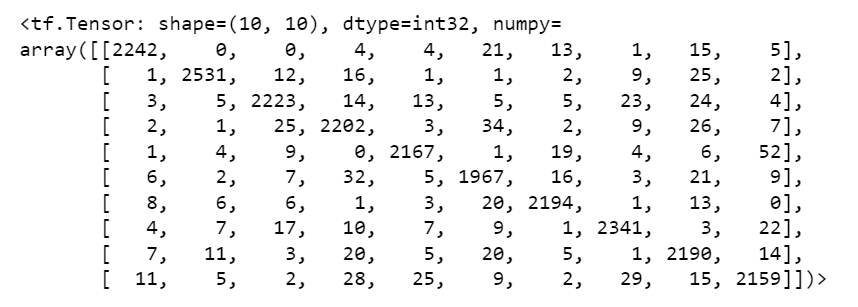
**Random State 5:**

****

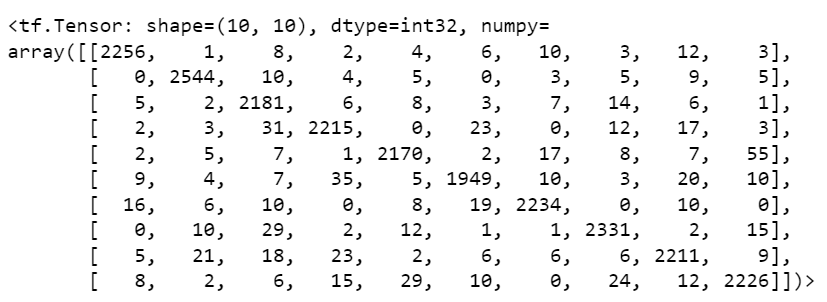
**Random State 6:**

****

**Random State 7:**

****

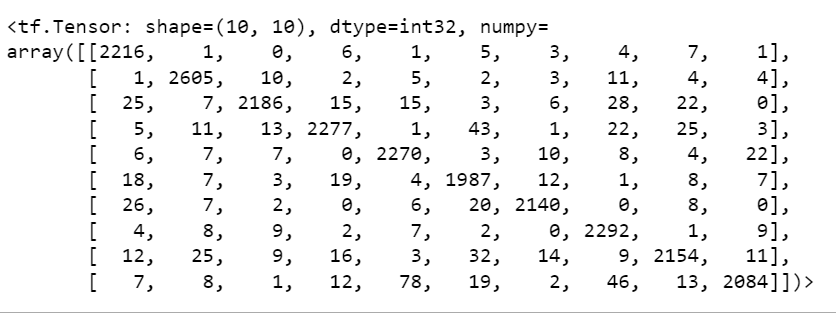
**Random State 8:**

****

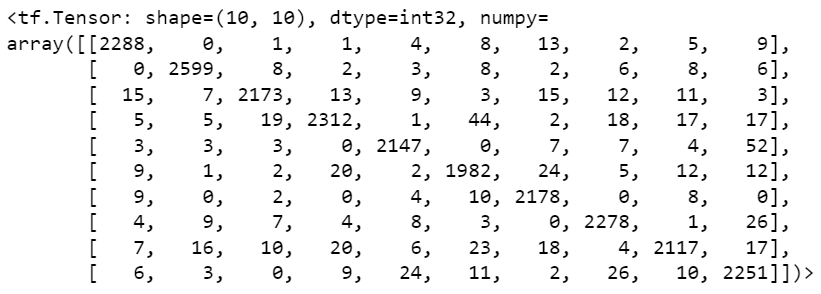
**Model 5**

========

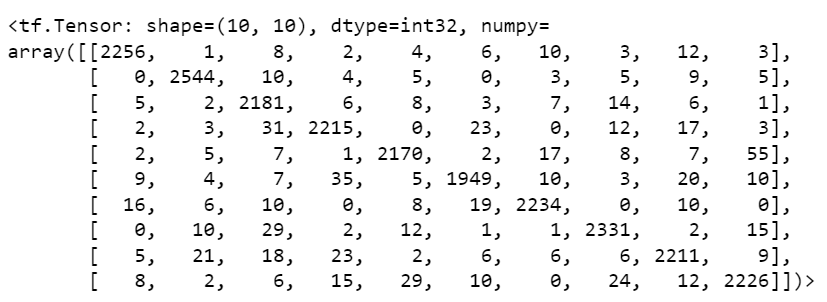
**Random State: 10**



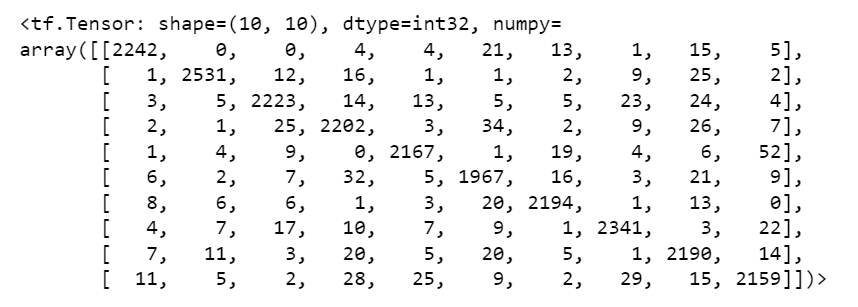
**Random State: 9**

****

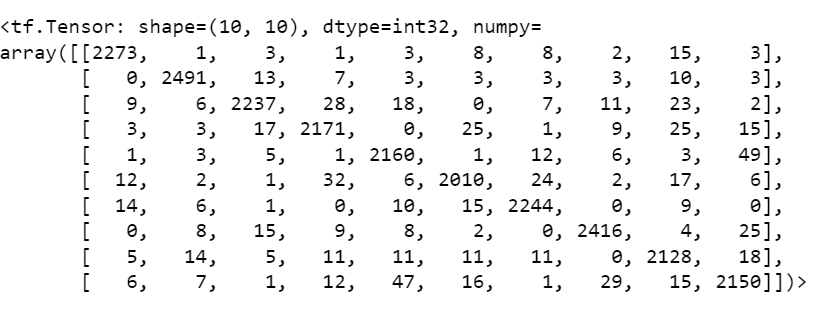
**Random State: 1**

****

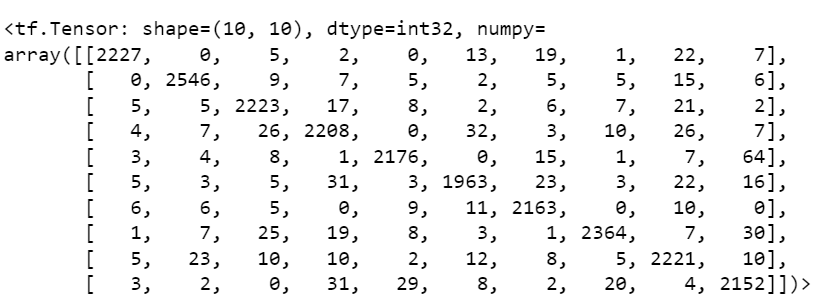
**Random State: 2**

****

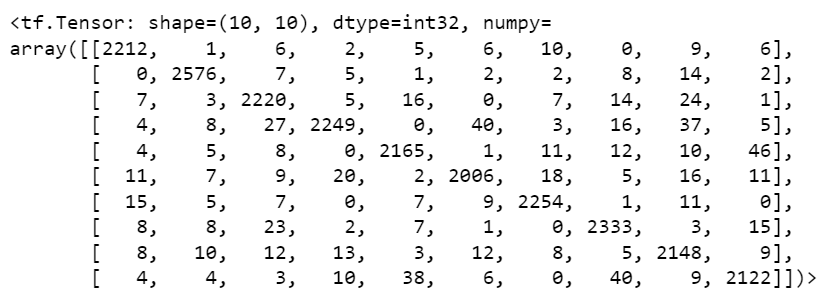
**Random State 3:**

****

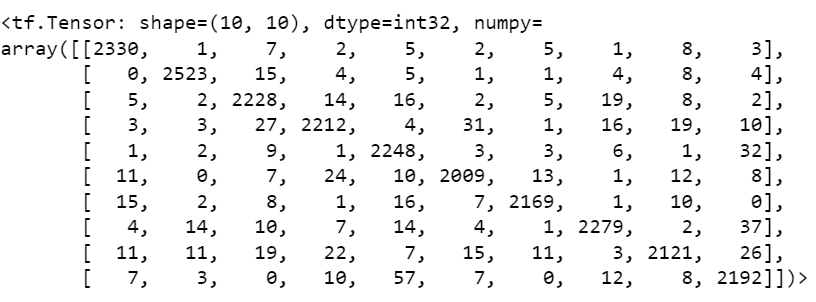
**Random State 4:**

****

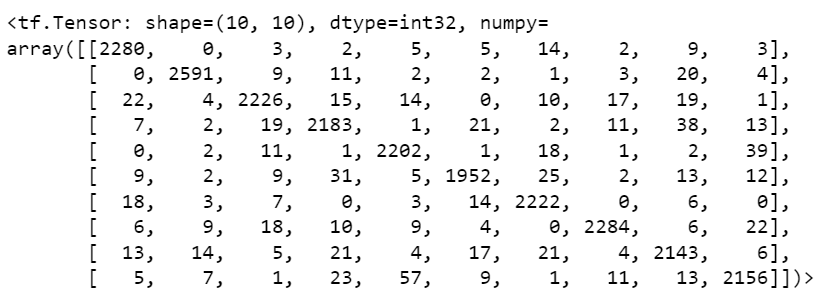
**Random State 5:**

****

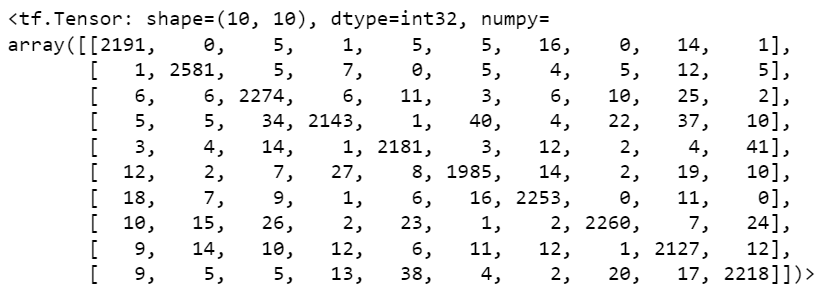
**Random State 6:**

****

**Random State 7:**

****

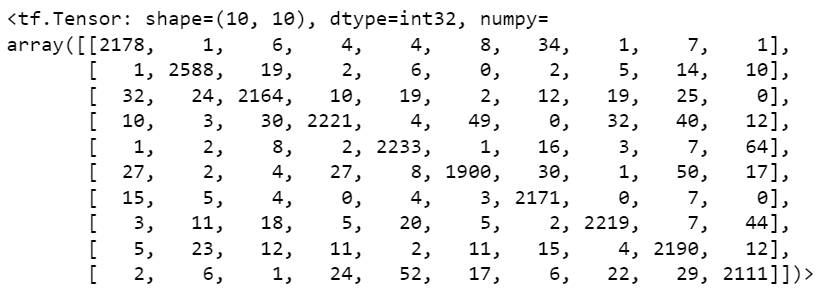
**Random State 8:**

****

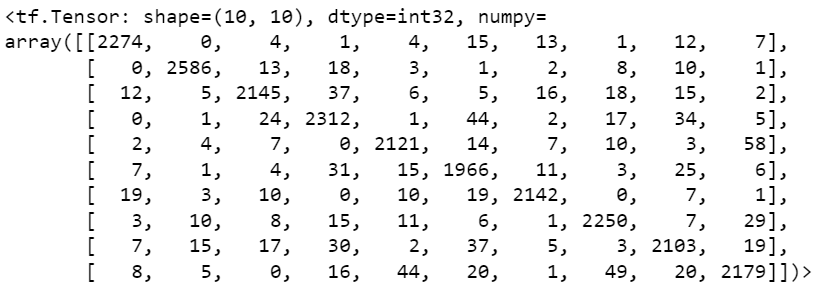
**MODEL 6**

**=======**

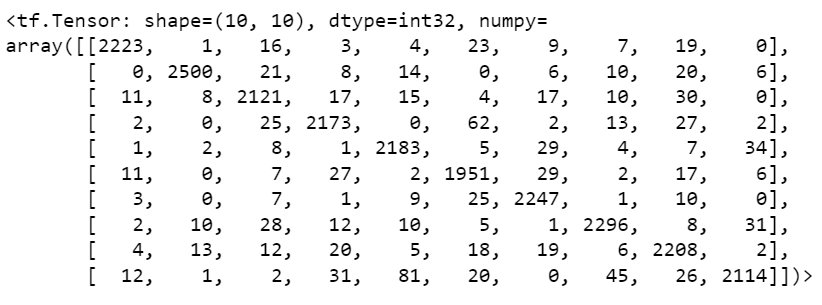
**Random State: 10**

****

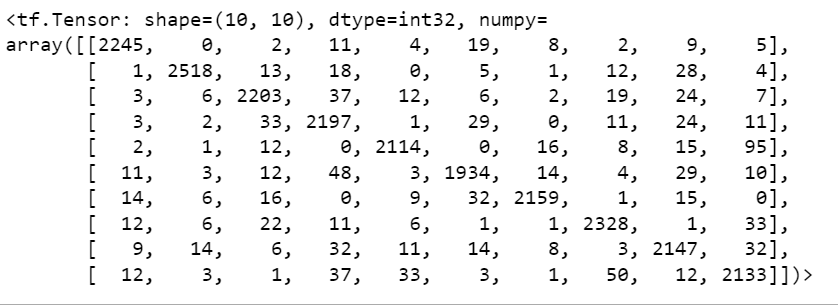
**Random State: 9**

****

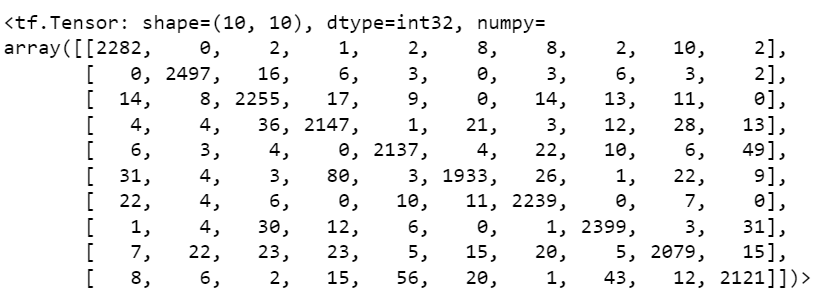
**Random State 1:**

****

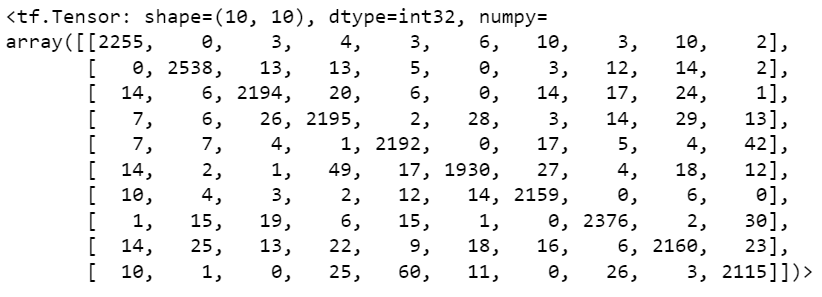
**Random State 2:**

****

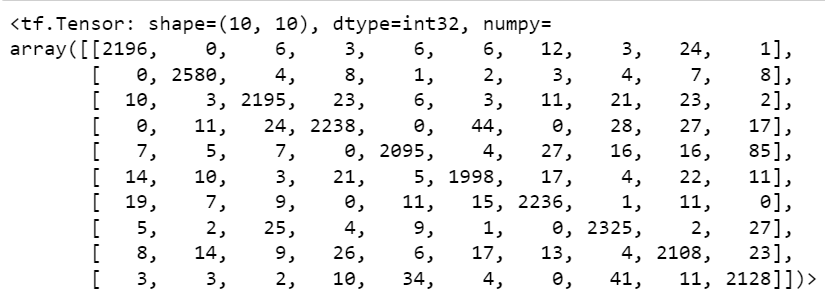
**Random State 3:**

****

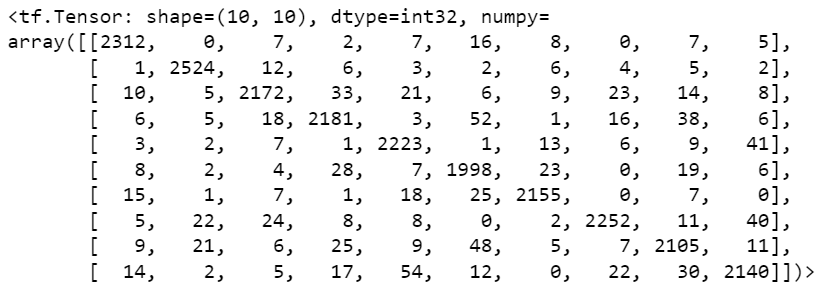
**Random State 4:**

****

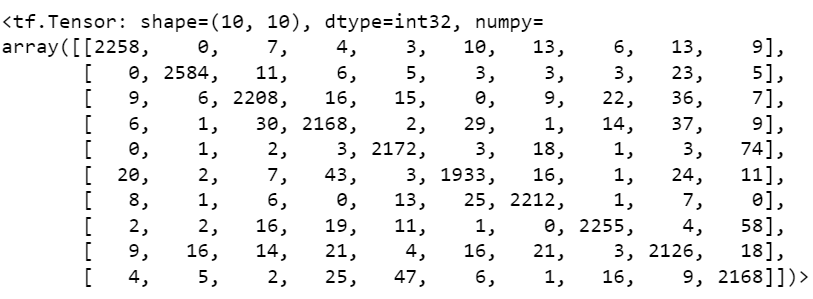
**Random State 5:**

****

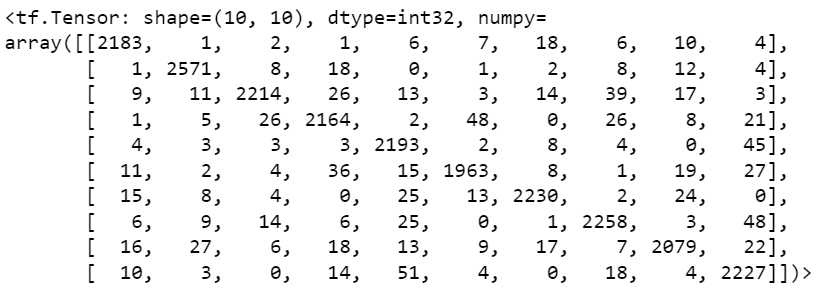
**Random State 6:**

****

**Random State 7:**

****

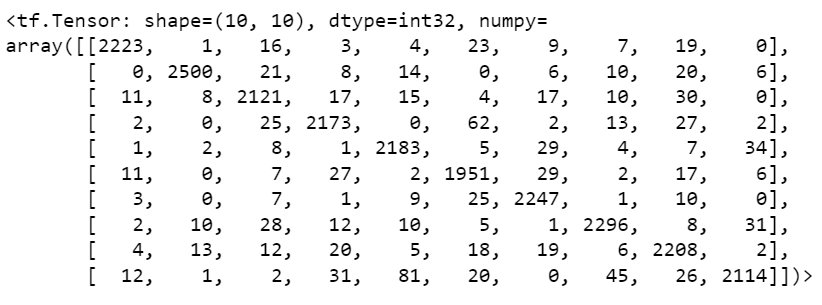
**Random State 8:**

****

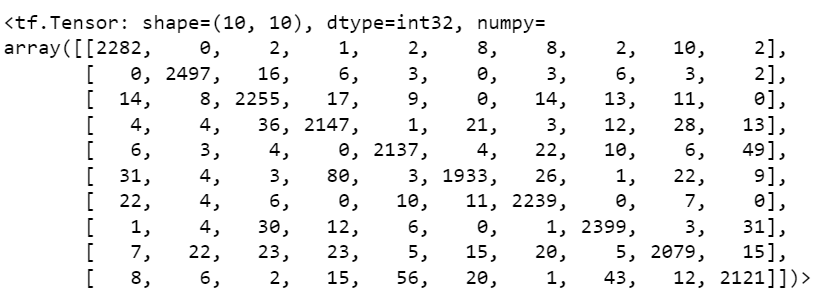
**Model 7**

========

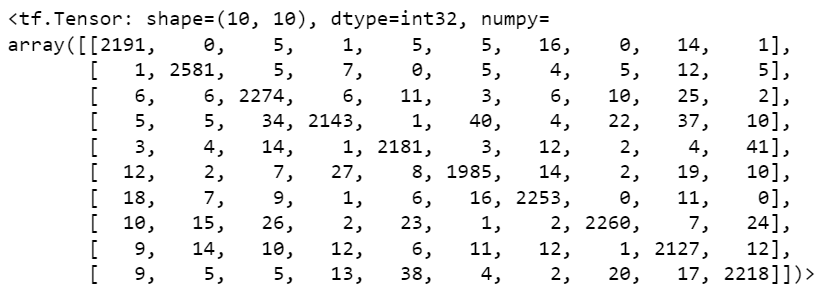
**Random State: 10**

****

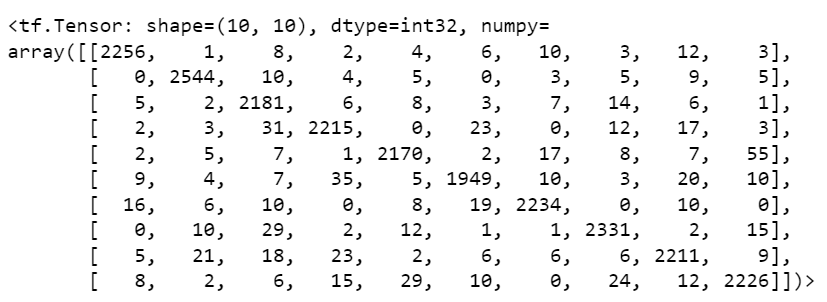
**Random State: 9**

****

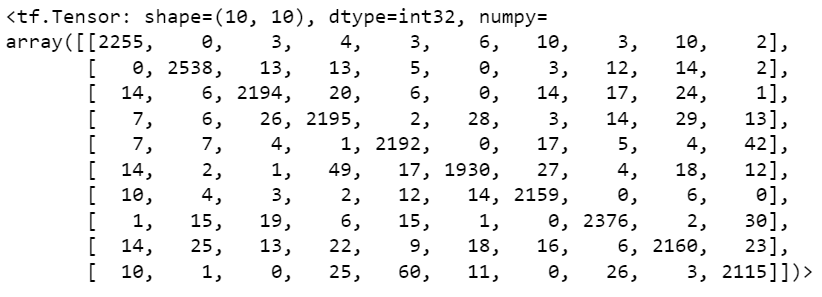
**Random State: 1**

****

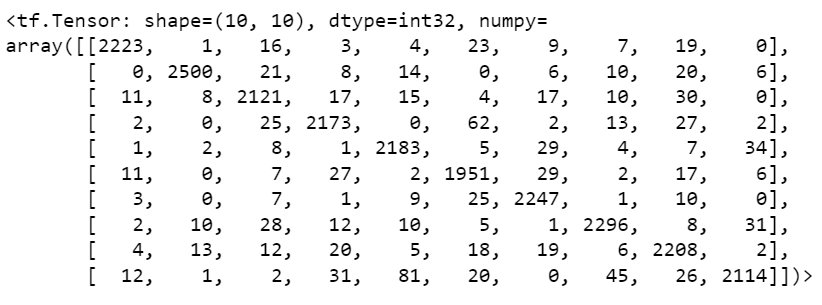
**Random State: 2**

****

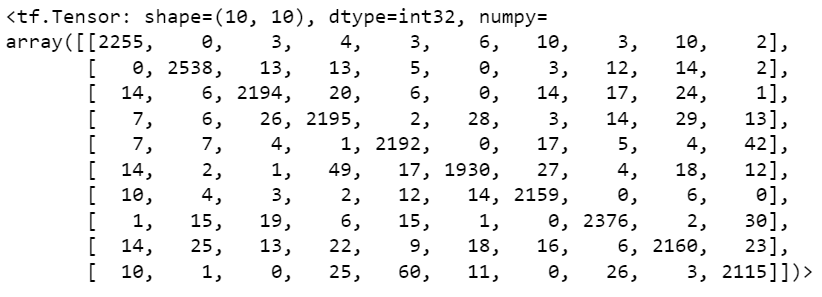
**Random State 3:**

****

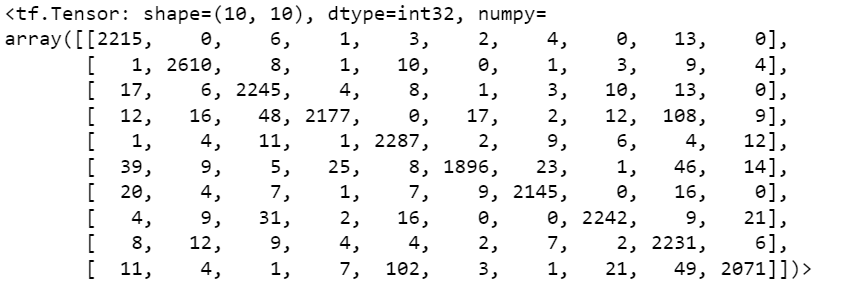
**Random State 4:**

****

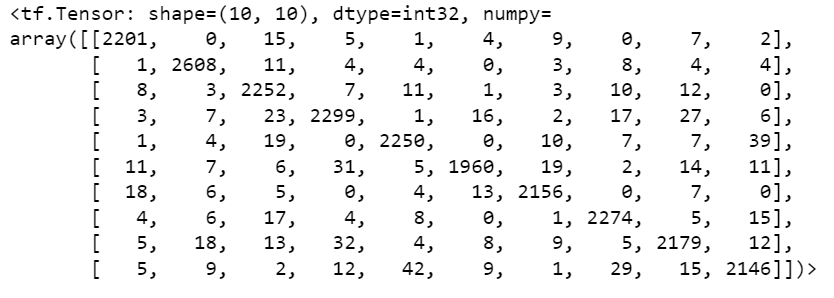
**Random State 5:**

****

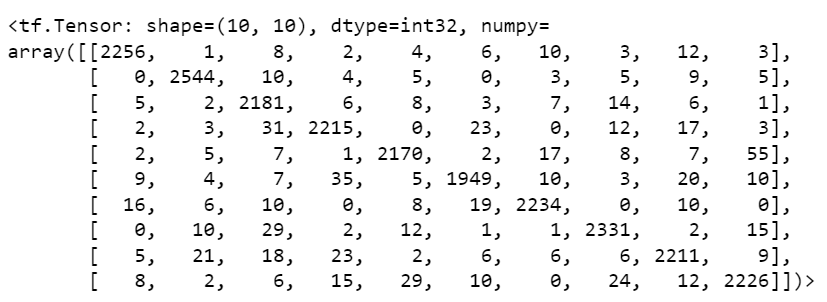
**Random State 6:**

****

**Random State 7:**

****

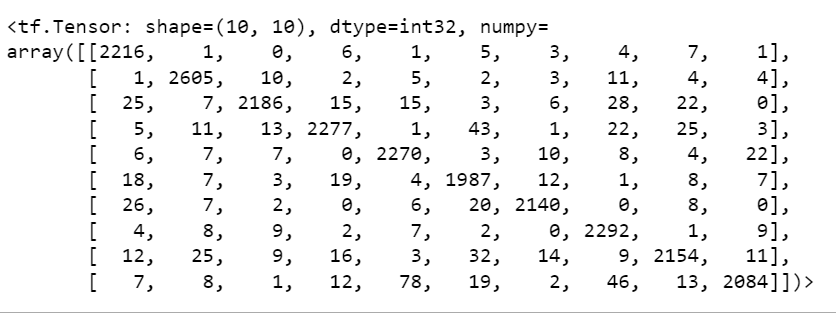
**Random State 8:**

****

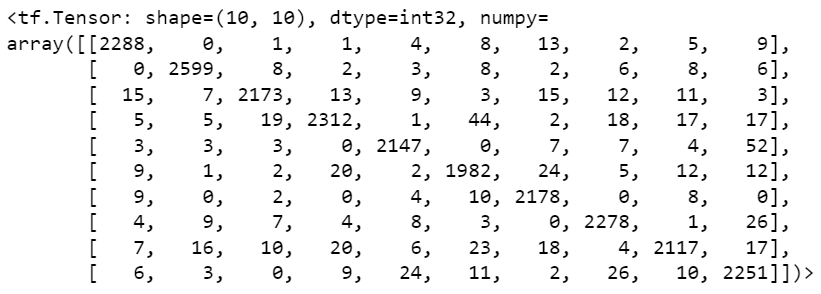
**Model 8**

========

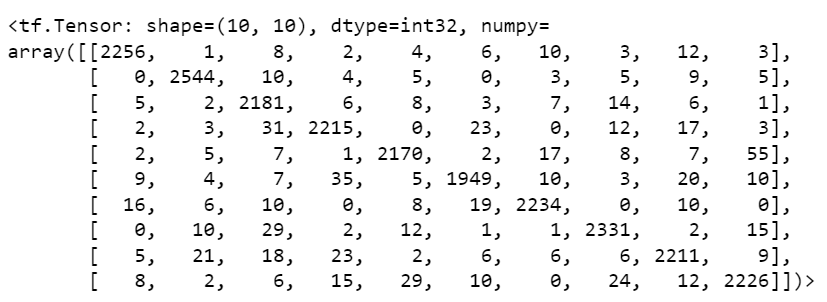
**Random State: 10**



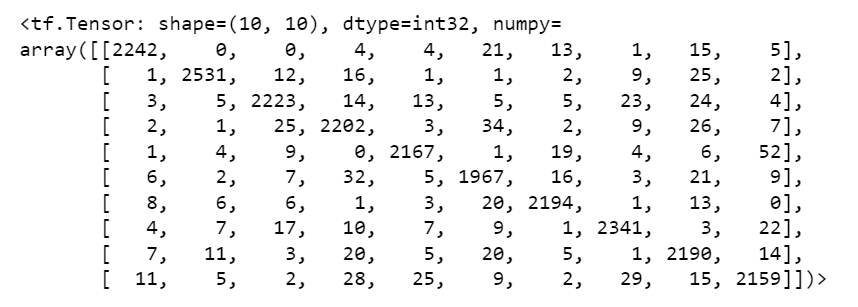
**Random State: 9**

****

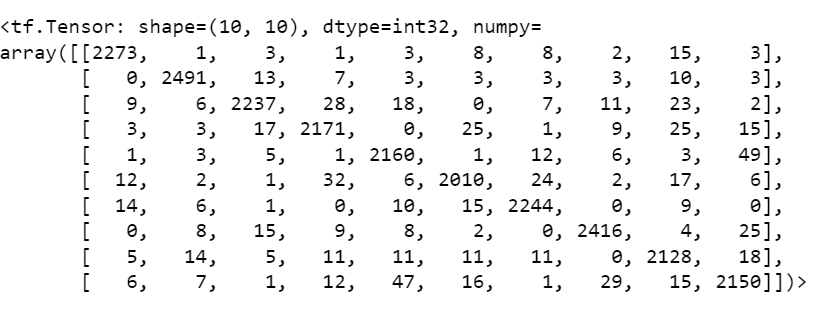
**Random State: 1**

****

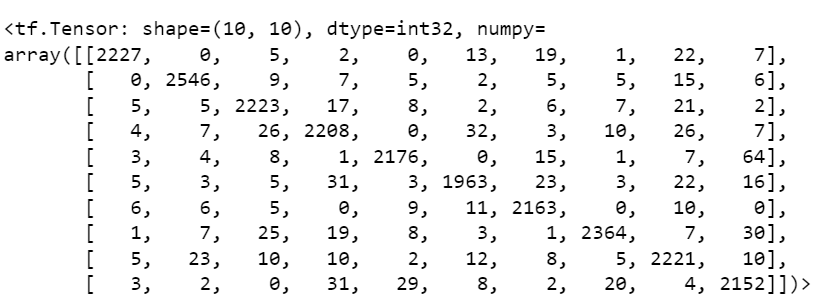
**Random State: 2**

****

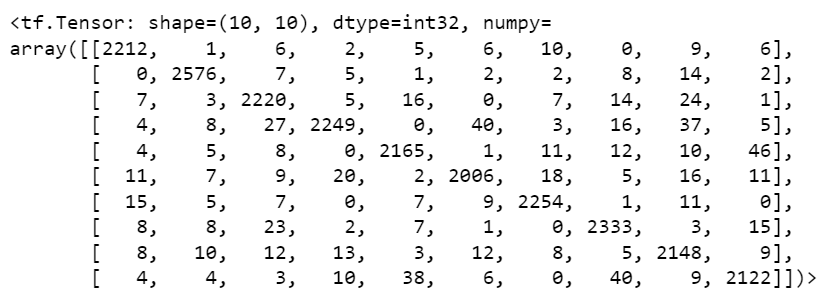
**Random State 3:**

****

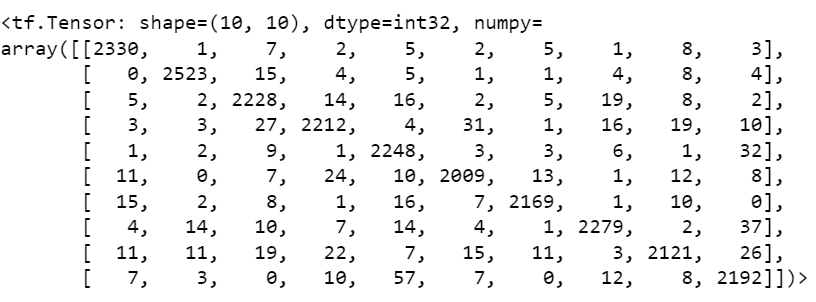
**Random State 4:**

****

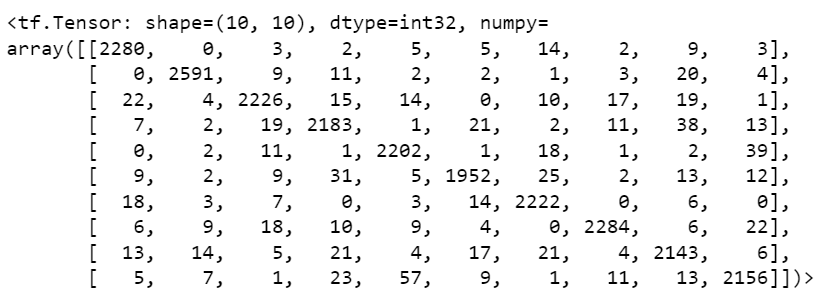
**Random State 5:**

****

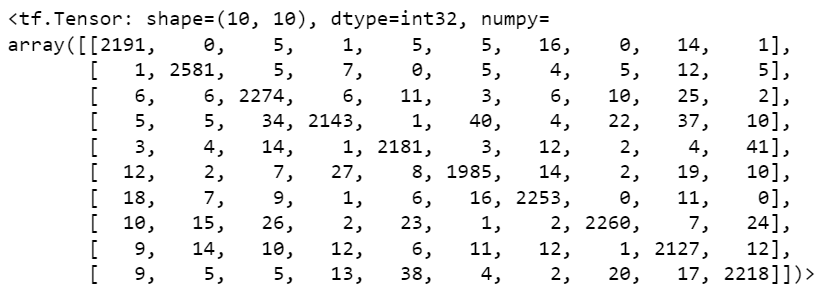
**Random State 6:**

****

**Random State 7:**

****

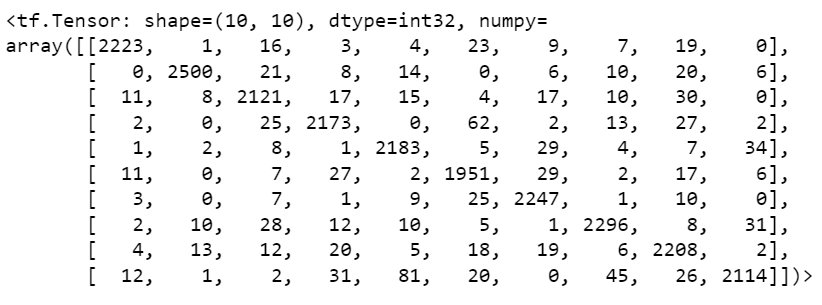
**Random State 8:**

****

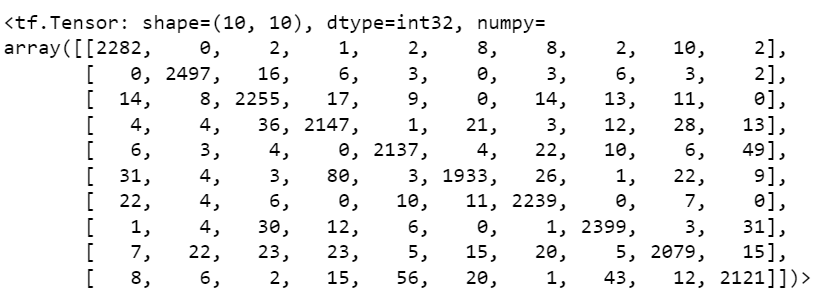
**Model 9**

========

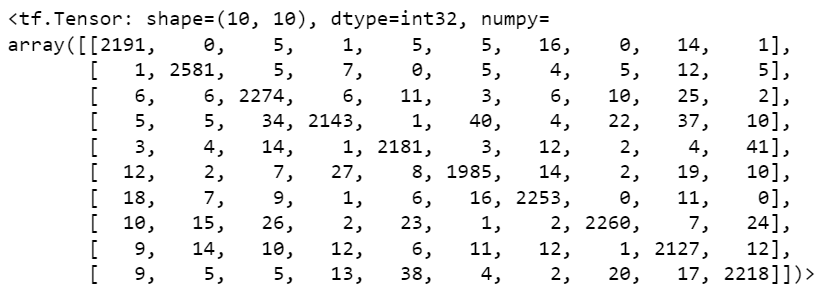
**Random State: 10**

****

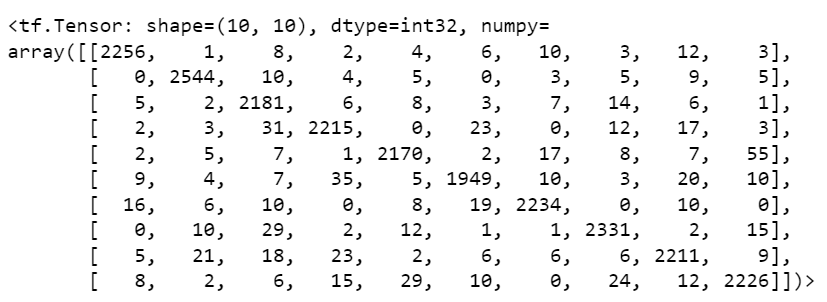
**Random State: 9**

****

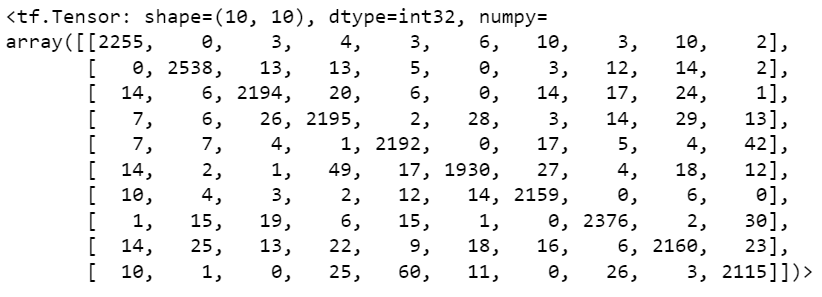
**Random State: 1**

****

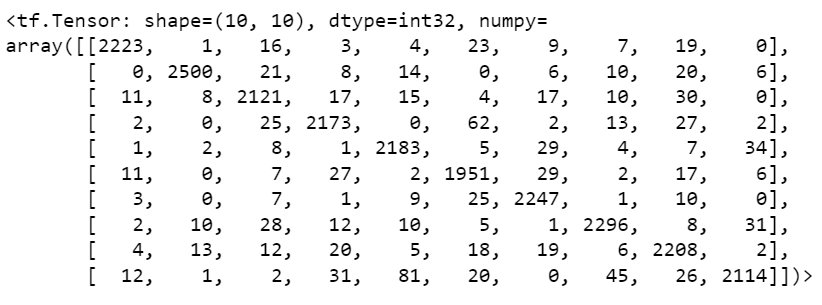
**Random State: 2**

****

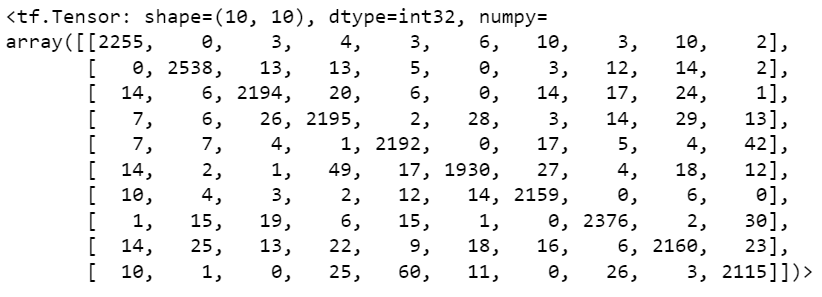
**Random State 3:**

****

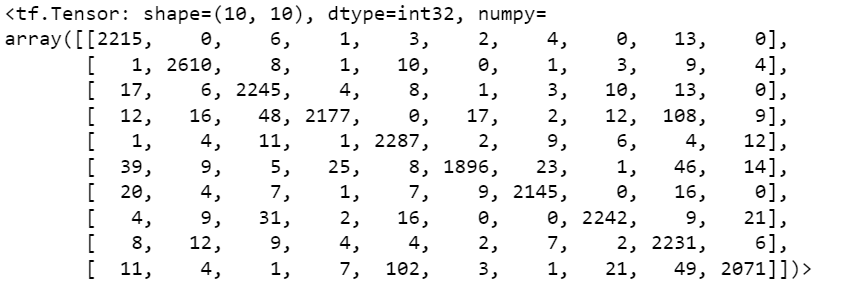
**Random State 4:**

****

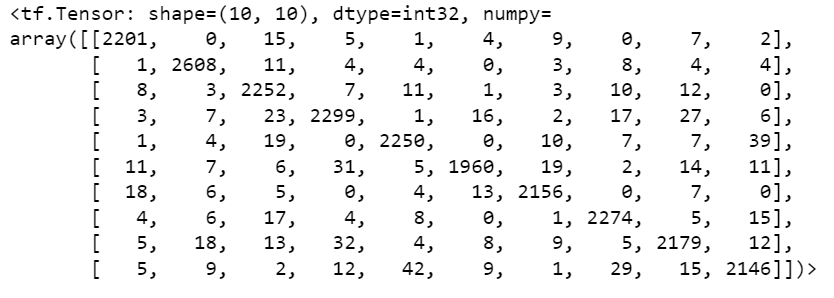
**Random State 5:**

****

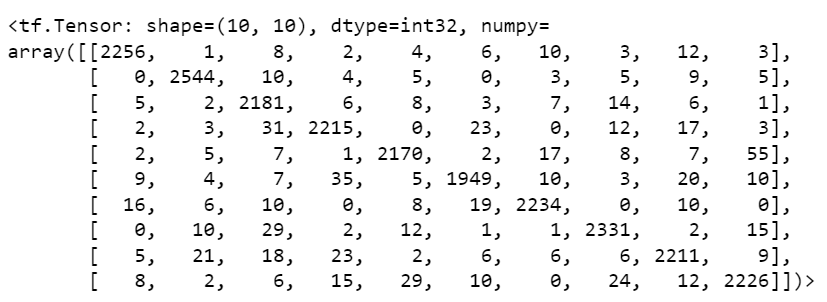
**Random State 6:**

****

**Random State 7:**

****

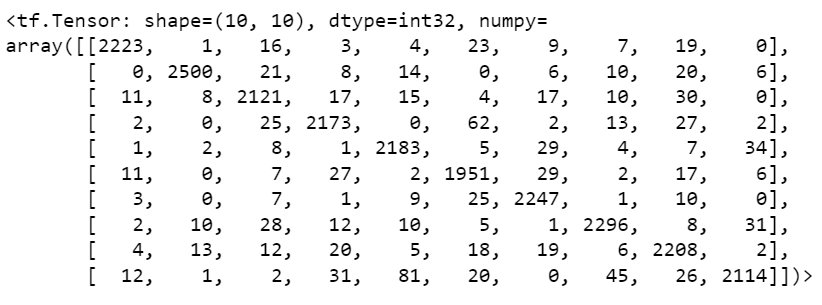
**Random State 8:**

****

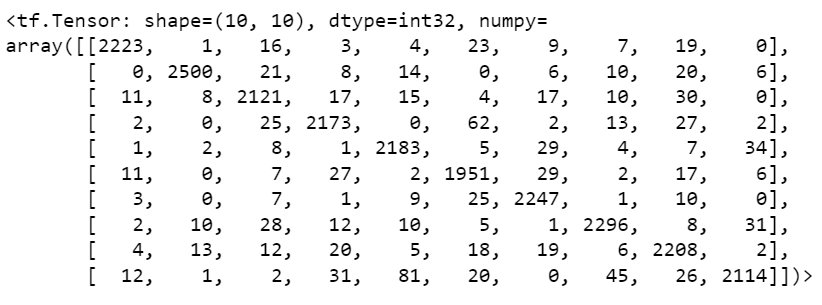
**MODEL 10**

**=======**

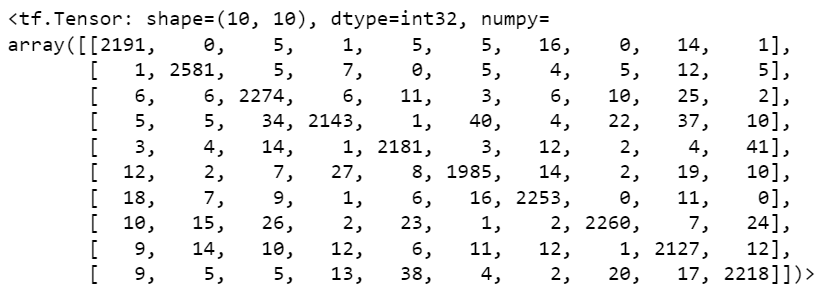
**Random State: 10**

****

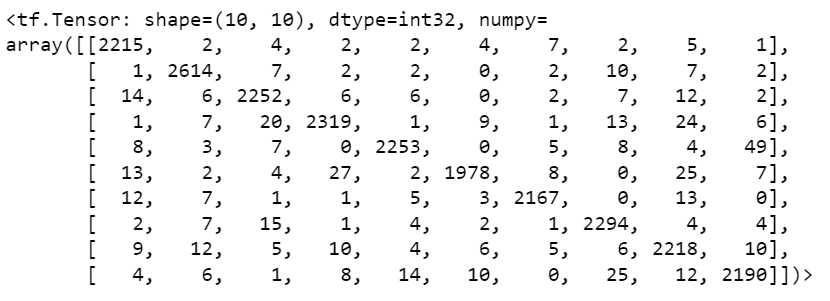
**Random State: 9**

****

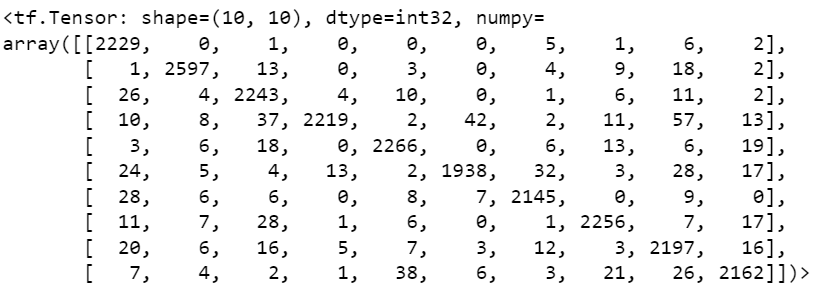
**Random State 1:**

****

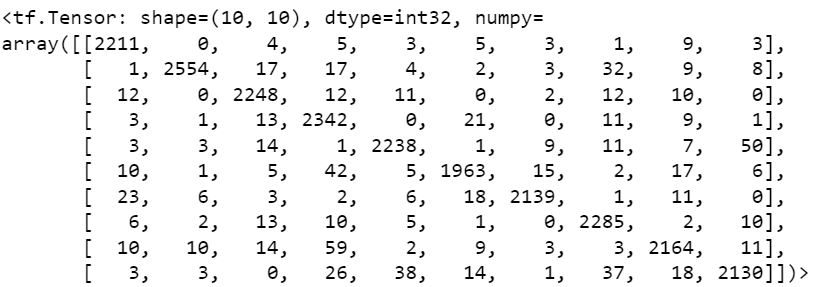
**Random State 2:**

****

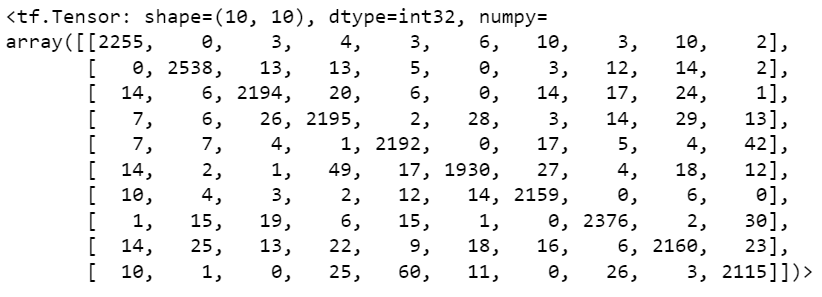
**Random State 3:**

****

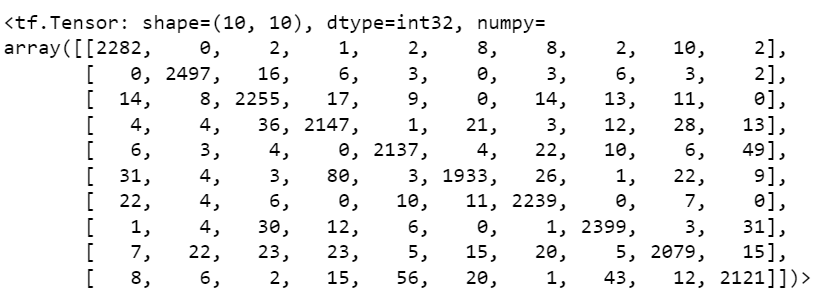
**Random State 4:**

****

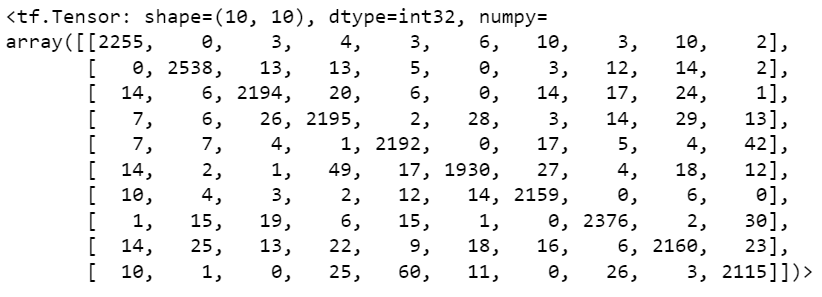
**Random State 5:**

****

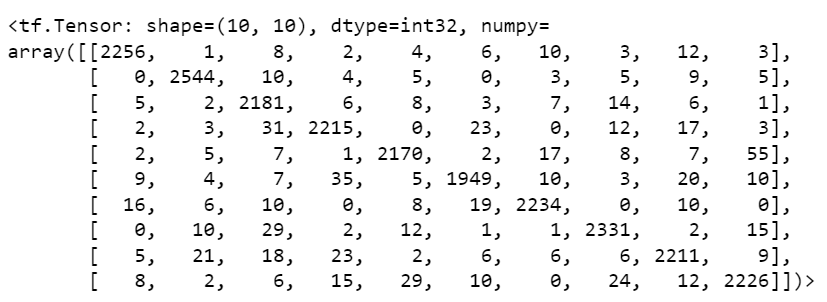
**Random State 6:**

****

**Random State 7:**

****

**Random State 8:**

****