ML Engineering Tutorial Part 1

Tutor: Ralf Mayet (mayet@campus.tu-berlin.de)
Adaptive Systems Group, Humboldt University Berlin

Content

- Python Basics (calculations, variables, functions, conditionals, loops and packages)
- Numpy (arrays, properties, calculations and operations)
- Tensorflow (basics, linear regression)

Goals

- Familiarize yourself with the basics of Python, Numpy and Tensorflow.
- Be able to build simple regression models with Tensorflow.
- Build a Sequential multi-layered model with Tensorflow and optimize it.

Sources

- [1] The Python Tutorial
- [2] Numpy Quickstart
- [3] Tensorflow and Keras Basic Regression
- [4] A line-by-line layman's guide to Linear Regression using TensorFlow (adapted to TF2)

Part 1A: Python Basics

Here we will look at some fundamentals of Python like using it as a calculator, variables, functions, conditionals, loops and packages.

In [1]:
```
# Python as a calculator
40 + 2
```

Out[1]:
```
42
```

In [2]:

```
# Variables

# Initialize some numerical variables
someNumber = 42
result = 0

# Perform a calculation and assign to result
result = someNumber * 2

# Print the result
print(result)
```

```
84
```

In [3]:

```
# The most important variable types: Strings, Floats, Integers, Lists

# Strings of characters
someString = "Hello World!"
someOtherString = "I have to stay at home 😱 "

print("Strings: ")
print(someString)
print(someOtherString * 5)
print("---" * 6)

# Floating point numbers and integers (whole numbers)
someFloat = 4.2
someOtherFloat = 5 / 2

print("Floats and Integers: ")
print(someFloat)
print(someOtherFloat)
print(int(someOtherFloat))
print("---" * 6)

# Lists
someList = [1, 2, 3]
someEmptyList = []

print("Lists:")
print(someList)
print(someEmptyList)

# Lists have an append function:
someEmptyList.append(42)
someEmptyList.append(someList)
someEmptyList.append(someString)
print(someEmptyList)
print("---" * 6)
```

```
Strings:
Hello World!
I have to stay at home 😱 I have to stay at home 😱 I have to stay at home 😱 I have to stay at home 😱 I have to stay at home 😱
------------------
Floats and Integers:
4.2
2.5
2
------------------
Lists:
[1, 2, 3]
[]
[42, [1, 2, 3], 'Hello World!']
------------------
```

In [4]:

```python
# Functions

# Some built-in functions exist in Python (https://docs.python.org/3/libra
# We already used print, int, append.

# We can also define new functions:
def printTwice(whatToPrint):
  print(whatToPrint)
  print(whatToPrint)

printTwice("Hello World!")

# Functions like int or str can return new values:
def returnTwice(whatToReturn):
  return whatToReturn + whatToReturn

print(returnTwice("Hello World!"))
```

```
Hello World!
Hello World!
Hello World!Hello World!
```

In [5]:

```python
# Conditionals
if "Hello" in someString:
  print(someString + " contains the word Hello")

if "Goodbye" not in someString:
  print(someString + " does NOT contain the word Goodbye")

if 42 > 12:
  print("fourty two is bigger than twelve.")
elif 42 == 42:
  print("fourty two is equal to fourty two")
else:
  print("error in the matrix..")
```

```
Hello World! contains the word Hello
Hello World! does NOT contain the word Goodbye
fourty two is bigger than twelve.
```

In [6]:

```
# Loops
i = 0
while i < 10:
  print("in the loop for the " + str(i) + "th time")
  i = i + 1

print("---" * 6)

for i in range(10):
  print("in the loop for the " + str(i) + "th time")
```

```
in the loop for the 0th time
in the loop for the 1th time
in the loop for the 2th time
in the loop for the 3th time
in the loop for the 4th time
in the loop for the 5th time
in the loop for the 6th time
in the loop for the 7th time
in the loop for the 8th time
in the loop for the 9th time
------------------
in the loop for the 0th time
in the loop for the 1th time
in the loop for the 2th time
in the loop for the 3th time
in the loop for the 4th time
in the loop for the 5th time
in the loop for the 6th time
in the loop for the 7th time
in the loop for the 8th time
in the loop for the 9th time
```

In [7]:

```
# Importing Packages

# There is built-in functions, we can define functions,
# but we can also use packages that contain functions!

import random # we only need to do this once in a script.

for i in range(10):
  print(random.randint(0,6))

# Packages can be installed on the system-level
# For an overview see 272k projects on https://pypi.org

# In development you will frequently end up reading the manuals
# of packages to figure out how to use their functions.
```

```
0
6
4
2
3
6
1
5
0
2
```

Part 1A Exercises

In [8]:
```
# Task 1A_a
# Implement a function that prints the string "WIN" or "LOOSE" with 50% cha
```

In [9]:
```
# Task 1A_b
# Implement a function that returns "WIN" or "LOOSE" with 50% chance each,
# then use it in a loop to populate a list with ten entries from the functi
```

In [10]:
```
# Task 1A_c
# Remove all "WIN" entries from the list generated above.
```

Part 1B: *Numpy*

Scientific Computing in Python. We look at the numpy array, its properties, using it for calculations and some operations.

In [11]:
```
import numpy as np
```

In [12]:
```
# The numpy array

# numpy's main object is the multidimensional array.
# You can instantiate it in several ways:

## using np.zeros:
zeroArray = np.zeros(10)
print("Zero Array (10):")
print(zeroArray)

zeroMatrix = np.zeros((10,10))
print("Zero Matrix (10,10):")
print(zeroMatrix)

## from a python list:
someList = [1,2,3,4,5,6]
someArray = np.array(someList)
print("Some Array:")
print(someList)

## ... and many more
```

```
Zero Array (10):
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Zero Matrix (10,10):
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
Some Array:
[1, 2, 3, 4, 5, 6]
```

In [13]:

```
# Numpy array properties
zeroMatrix = np.zeros((10,10))
print("Zero Matrix (10,10):")
print(zeroMatrix)

print("ndim:")
print(zeroMatrix.ndim)
print("shape:")
print(zeroMatrix.shape)
print("size:")
print(zeroMatrix.size)
print("data type:")
print(zeroMatrix.dtype)
```

```
Zero Matrix (10,10):
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
ndim:
2
shape:
(10, 10)
size:
100
data type:
float64
```

In [14]:

```
# Numpy array arithmetic

# operations apply elementwise
a = np.array( [9,3,4,5] )
b = np.arange( 4 )
c = a-b

print("a:")
print(a)

print("b:")
print(b)

print("c=a-b:")
print(c)

print("b**2:")
print(b**2)

print("10*np.sin(a)")
print(10*np.sin(a))

print("a<35")
print(a<4)
```

```
a:
[9 3 4 5]
b:
[0 1 2 3]
c=a-b:
[9 2 2 2]
b**2:
[0 1 4 9]
10*np.sin(a)
[ 4.12118485  1.41120008 -7.56802495 -9.58924275]
a<35
[False  True False False]
```

In [15]:

```
# Numpy array operations

# Shape manipulation:
someArray = np.array([1,2,3,4,5,6])
print("Some Array:")
print(someArray)

someMatrix = someArray.reshape((2,3))
print("Some Matrix:")
print(someMatrix)

backToArray = someMatrix.flatten()
print("Back To Array:")
print(backToArray)

# Many more things like concatenation, mirroring, scaling, etc. all possibl
# Ref: https://numpy.org/devdocs/docs/index.html
```

```
Some Array:
[1 2 3 4 5 6]
Some Matrix:
[[1 2 3]
 [4 5 6]]
Back To Array:
[1 2 3 4 5 6]
```

In [16]:

```
# Numpy array broadcasting
# This describes how numpy treats operations on arrays with different sizes

# Elementwise when arrays have the same size
a = np.array([1.0, 2.0, 3.0])
b = np.array([2.0, 2.0, 2.0])
print("a*b")
print(a * b)

# Scalar multiplications
a = np.array([1.0, 2.0, 3.0])
alpha = 2.0
print("a*alpha")
print(a * alpha)

# Operating on arrays, numpy starts with the last dimension and compares.
# Two dimensions are compatible when a) they are equal, or b) one of them i

# Example:
images = np.ones((150,4,4,3)) # Could be a stack of 150 4x4 pixel RGB image
print("first image as-is:")
print(images[0])

color_scaling = np.array([0.5,0.8,1.0]) # Scale red by half, green by 80%,
scaled_images = images * color_scaling
print("first image color graded:")
print(scaled_images[0])

# Dimension mismatch:
#a = np.array([1.0, 2.0, 3.0, 4.0])
#b = np.array([2.0, 2.0, 2.0])
#print("a*b")
#print(a * b)
```

```
a*b
[2. 4. 6.]
a*alpha
[2. 4. 6.]
first image as-is:
[[[1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]]]
first image color graded:
[[[0.5 0.8 1. ]
  [0.5 0.8 1. ]
  [0.5 0.8 1. ]
  [0.5 0.8 1. ]]

 [[0.5 0.8 1. ]
  [0.5 0.8 1. ]
  [0.5 0.8 1. ]
  [0.5 0.8 1. ]]

 [[0.5 0.8 1. ]
  [0.5 0.8 1. ]
  [0.5 0.8 1. ]
  [0.5 0.8 1. ]]

 [[0.5 0.8 1. ]
  [0.5 0.8 1. ]
  [0.5 0.8 1. ]
  [0.5 0.8 1. ]]]
```

Part 1B *Exercises*

In [17]:

```
# Task 1B_a
# Implement a function that returns an array length 12 with random numbers.
```

In [18]:

```
# Task 1B_b
# Use the function from 1B_a to make a 12x12 matrix of random numbers itera
```

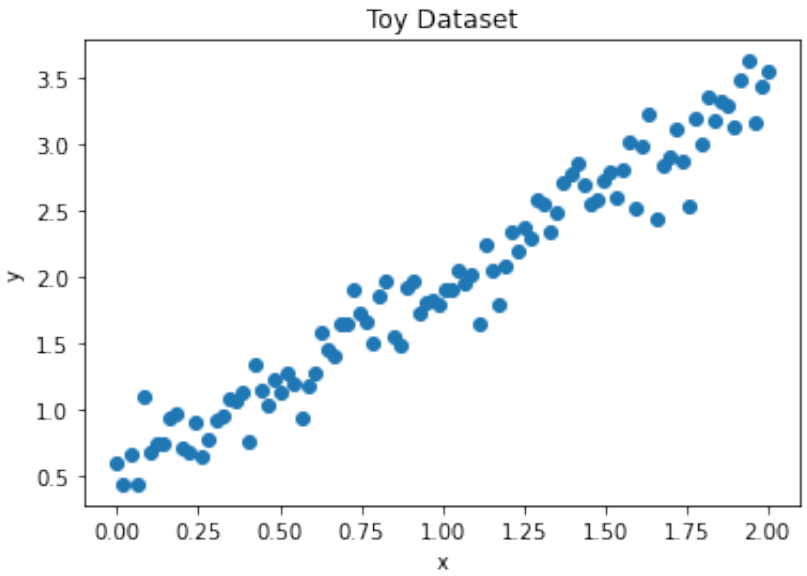
In [19]:
```
# Task 1B_c
# Reshape the matrix from 1B_b to a (6,6,4) 6x6pixel RGBA image and
# give it 100% opacity and a red-tint.


# To visualize the result:
#import matplotlib.pyplot as plt
#plt.imshow(RESULT_ARRAY)
```

Part 1C: Tensorflow

Using Tensorflow for a simple regression task.

In [20]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
import matplotlib.pyplot as plt
```

In [21]:
```
# Generating toy dataset
X = np.linspace(0, 2, 100)
y = 1.5 * X + np.random.randn(*X.shape) * 0.2 + 0.5

# Plot using matplotlib scatter function
plt.scatter(X, y)
plt.title("Toy Dataset")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



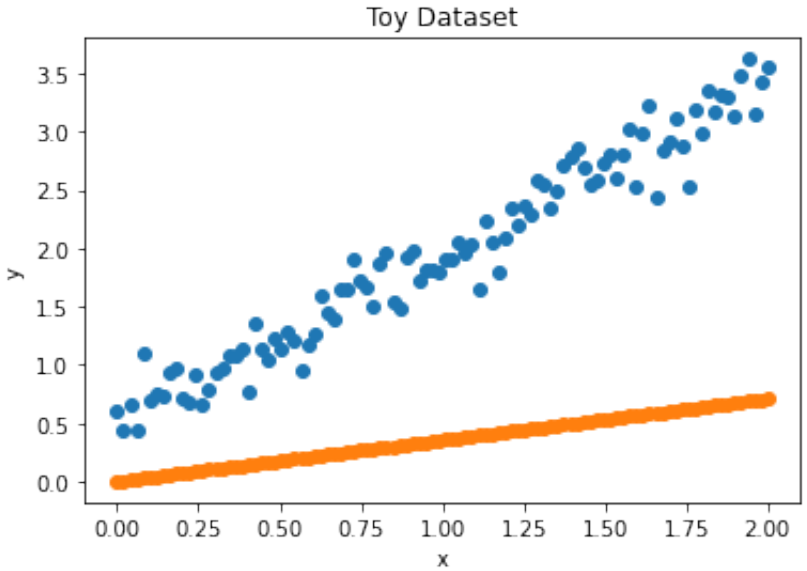
In [22]:

```
# Making a model
model = tf.keras.Sequential([
    layers.Dense(input_shape=[1,], units=1)
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

```
# Get some example predictions
predictions = model.predict(X)

# Plot using matplotlib scatter function
plt.scatter(X, y)
plt.scatter(X, predictions)
plt.title("Toy Dataset")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



In [24]:

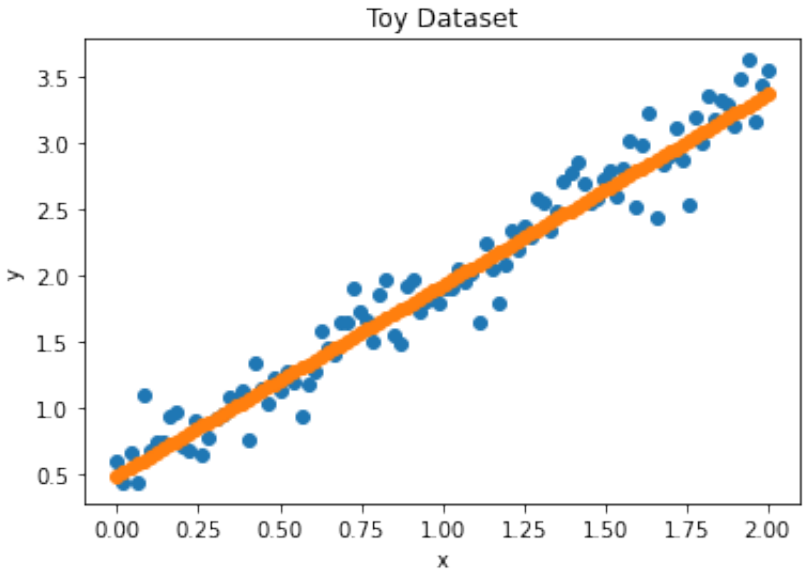
```
# Train the model
model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.1), loss='mean_a
model.fit(X,y, epochs=100)
```

```
Epoch 1/100
4/4 [==============================] - 0s 940us/step - loss: 1.3804
Epoch 2/100
4/4 [==============================] - 0s 684us/step - loss: 0.6072
Epoch 3/100
4/4 [==============================] - 0s 787us/step - loss: 0.2549
Epoch 4/100
4/4 [==============================] - 0s 842us/step - loss: 0.4310
...
Epoch 97/100
4/4 [==============================] - 0s 878us/step - loss: 0.1443
Epoch 98/100
4/4 [==============================] - 0s 860us/step - loss: 0.1451
Epoch 99/100
4/4 [==============================] - 0s 693us/step - loss: 0.1442
Epoch 100/100
4/4 [==============================] - 0s 822us/step - loss: 0.1454
```

Out[24]: `<tensorflow.python.keras.callbacks.History at 0x7fdf201a5280>`

In [25]:

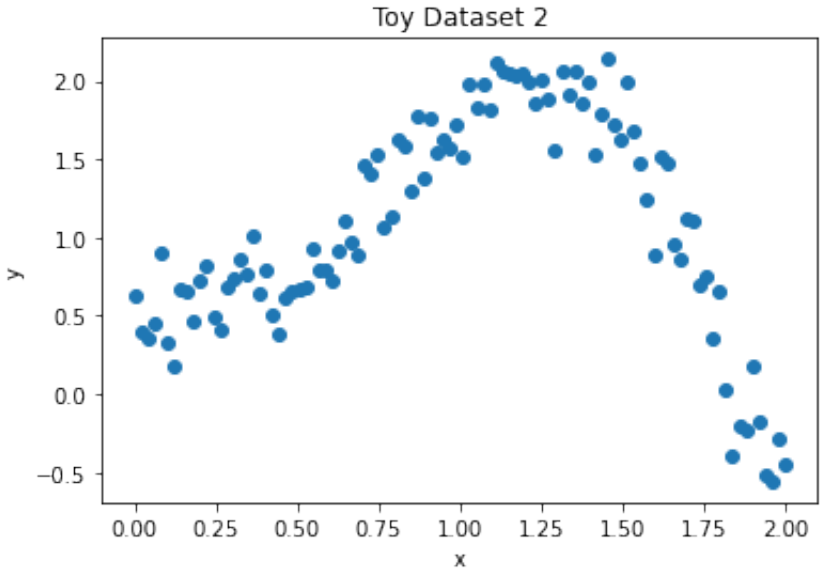
```
# Evaluate the final predictions
predictions = model.predict(X)

# Plot using matplotlib scatter function
plt.scatter(X, y)
plt.scatter(X, predictions)
plt.title("Toy Dataset")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



Part 1C Exercises

In [26]:

```
# Task 1C_a
# Try to use the same model we used in the linear regression example to mod

X = np.linspace(0, 2, 100)
y = 1.5 * np.sin(X**2) + np.random.randn(*X.shape) * 0.2 + 0.5

# Plot using matplotlib scatter function
plt.scatter(X, y)
plt.title("Toy Dataset 2")
plt.xlabel("x")
plt.ylabel("y")
plt.show()

# Solution
```



In [27]:

```
# Task 1C_b
# Make changes to the model used in the linear regression example to model

X = np.linspace(0, 2, 100)
y = 1.5 * np.sin(X**2) + np.random.randn(*X.shape) * 0.2 + 0.5

# Plot using matplotlib scatter function
plt.scatter(X, y)
plt.title("Toy Dataset 2")
plt.xlabel("x")
plt.ylabel("y")
plt.show()

# Solution
```

ML Engineering Tutorial Part 2

Tutor: Ralf Mayet (mayet@campus.tu-berlin.de)
Adaptive Systems Group, Humboldt University Berlin

Content / Goals

- Build simple regression models with Tensorflow.
- Build a simple machine learning pipeline (preprocessing, learning, evaluation)
- Build a classification multi-layer perceptron with Tensorflow.
- Extend it to use convolutional layers and observe the difference.

Sources

- [1] Tensorflow and Keras Basic Regression
- [2] A line-by-line layman's guide to Linear Regression using TensorFlow (adapted to TF2)
- [3] Tensorflow Documentation: Loading MNIST
- [4] Basic classification: Classify images of clothing

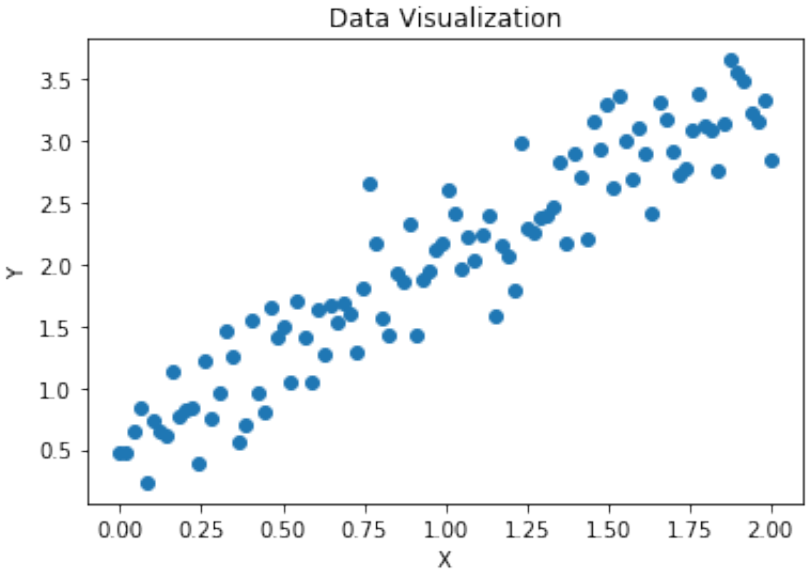
In [1]:
```
# Importing packages we'll be using
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
import matplotlib.pyplot as plt
```

In [2]:
```
# Helper functions for plotting:
def plotData(X, Y, predictions=None, title="Data Visualization"):
  plt.scatter(X, Y)
  if predictions is not None:
    plt.scatter(X, predictions)
  plt.title(title)
  plt.xlabel("X")
  plt.ylabel("Y")
  plt.show()

def plotImage(image):
  plt.imshow(image)
  plt.colorbar()
  plt.show()
```

Part 2A: Re-visiting regression

In [3]:

```
# Generating toy dataset
X = np.linspace(0, 2, 100)

# linear
y = 1.5 * X + np.random.randn(*X.shape) * 0.3 + 0.5
# sinusoidal
# y = 1.5 * np.sin(X**2) + np.random.randn(*X.shape) * 0.2 + 0.5

# Plot using our utility function
plotData(X, y)
```



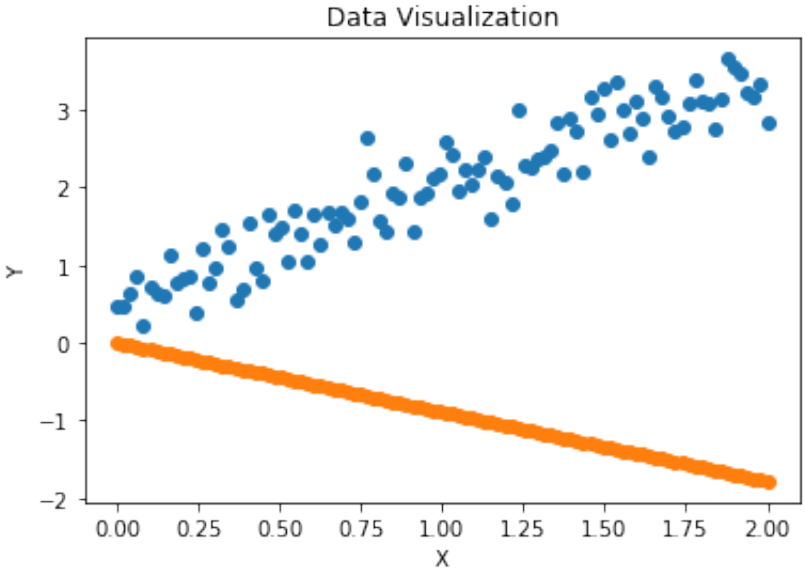
In [4]:

```
# Making a model
# Ref activation functions: https://www.researchgate.net/profile/Junxi_Feng

model = tf.keras.Sequential([
    layers.Dense(input_shape=[1,], units=1)
])

# model = tf.keras.Sequential([
#     layers.Dense(input_shape=[1,], units=1, activation="tanh"),
#     layers.Dense(units=4, activation="tanh"),
#     layers.Dense(units=1)
# ])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

In [5]:

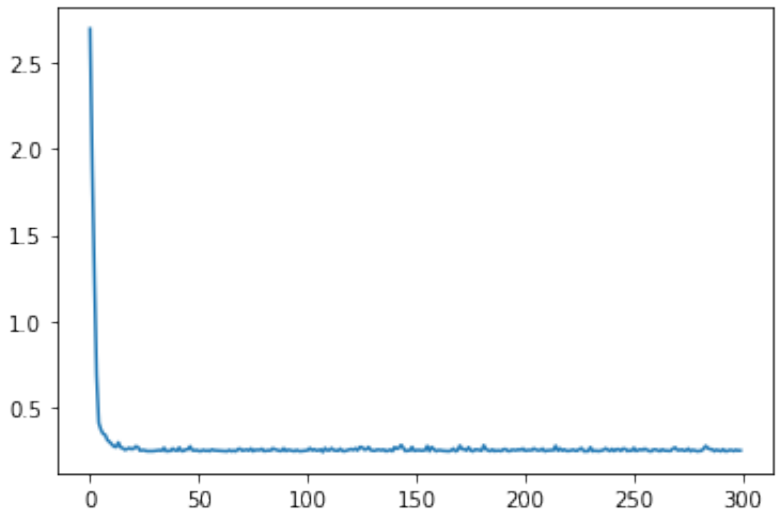
```
# Get some example predictions
predictions = model.predict(X)
plotData(X,y,predictions)

meanSquareError = ((y-predictions)**2).mean()
print("Mean Square Error: %.2f" % meanSquareError)
```




```
Mean Square Error: 9.56
```

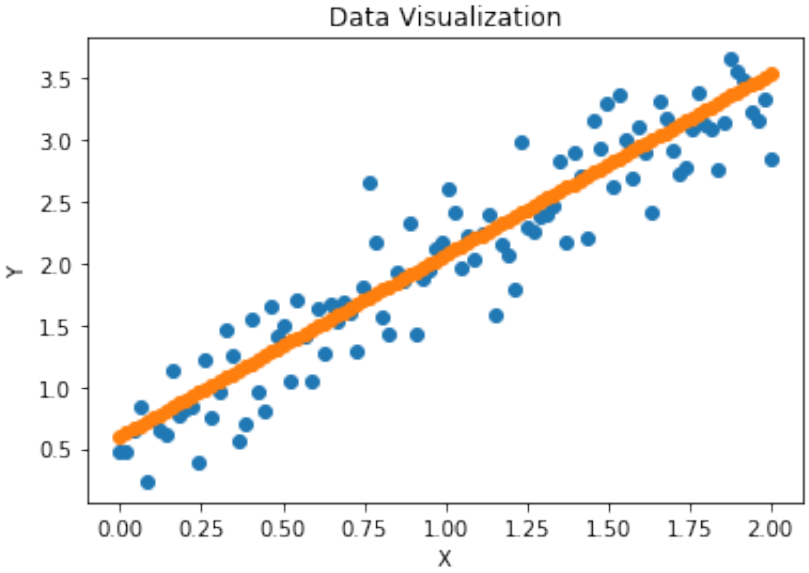
In [6]:

```
# Train the model
model.compile(optimizer=tf.optimizers.SGD(learning_rate=0.1), loss='mean_ab
history = model.fit(X,y, epochs=300, verbose=0)

# Plot the loss
plt.plot(history.history['loss'])
plt.show()
```



In [7]:

```
# Evaluate the final predictions
predictions = model.predict(X)
plotData(X,y,predictions)

meanSquareError = ((y-predictions)**2).mean()
print("Mean Square Error: %.2f" % meanSquareError)
```




```
Mean Square Error: 1.54
```

Part 2B: Loading Images

In [8]:

```
# Loading MNIST using built-in TF function
# Ref https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist/lo
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

In [9]:

```
# Inspect dataset
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

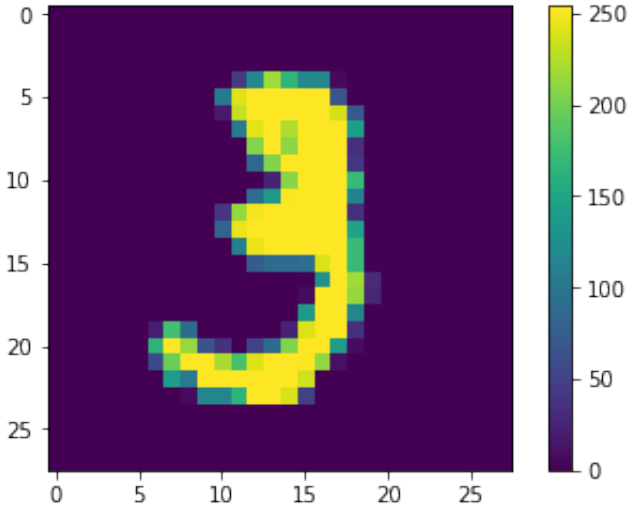
In [10]:

```
# Printing individual image and label
print(x_train[0])
print(y_train[0])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
   78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
5
```

In [11]:

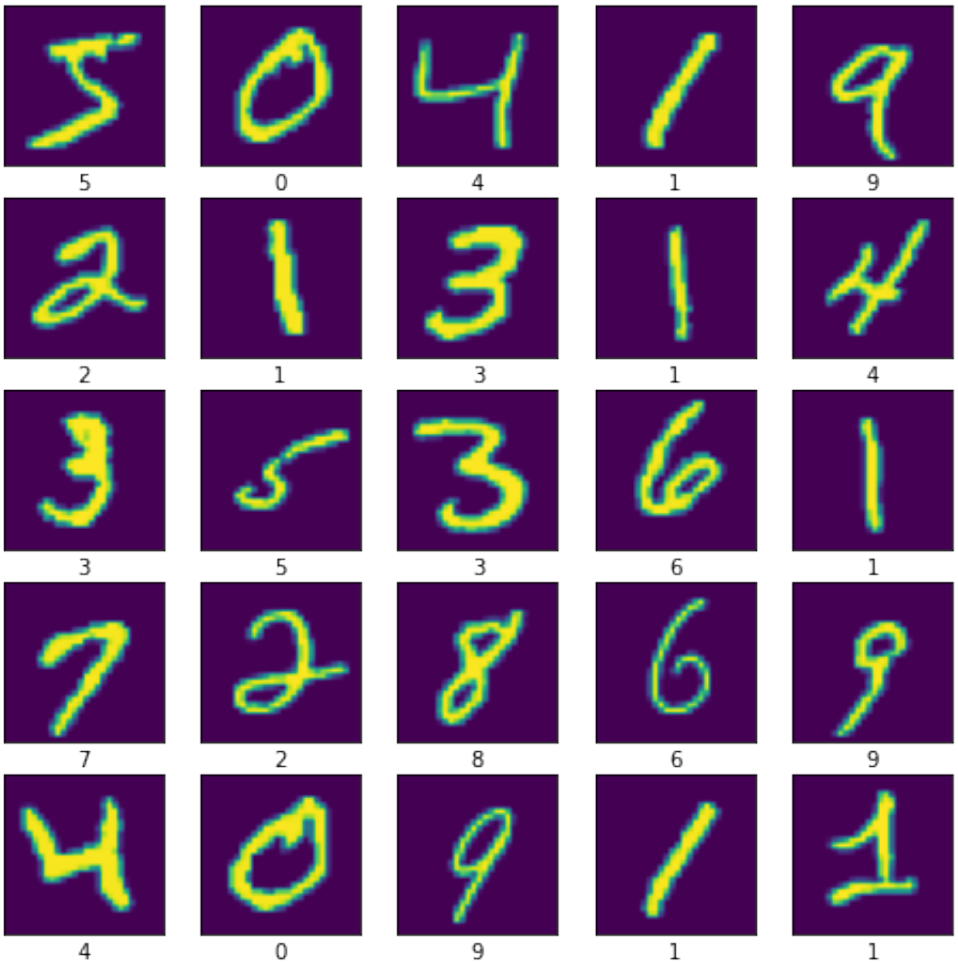
```python
# Plotting individual image and label
plotImage(x_train[10])
print(y_train[10])
```



3

In [12]:

```
# Plot 25 examples
plt.figure(figsize=(8,8))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i])
    plt.xlabel(y_train[i])
plt.show()
```



In [13]:
```
# LIVE DEMO: Loading Images on a Desktop/Server w/o Google Colab
```

Part 2C: Classification of Handwritten Digits

In [14]:
```
# We have our data in x_train, y_train (see above)

# Let's build a model for classification:
model = tf.keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu'),
    layers.Dense(10)
])
```

In [15]:
```
# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logit
              metrics=['accuracy'])
```

In [16]:
```
# Evaluate Accuracy on test data
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

```
313/313 - 0s - loss: 163.1000 - accuracy: 0.1670

Test accuracy: 0.16699999570846558
```

In [17]:
```
# Fit to data
model.fit(x_train, y_train, epochs=3)
```

```
Epoch 1/3
1875/1875 [==============================] - 2s 1ms/step - loss: 2.6813 - a
ccuracy: 0.8555
Epoch 2/3
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3736 - a
ccuracy: 0.9094
Epoch 3/3
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2842 - a
ccuracy: 0.9284
```

Out[17]:
```
<tensorflow.python.keras.callbacks.History at 0x7fd2ee9195b0>
```

In [18]:
```
# Evaluate Accuracy on test data
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

```
313/313 - 0s - loss: 0.3514 - accuracy: 0.9225

Test accuracy: 0.9225000143051147
```

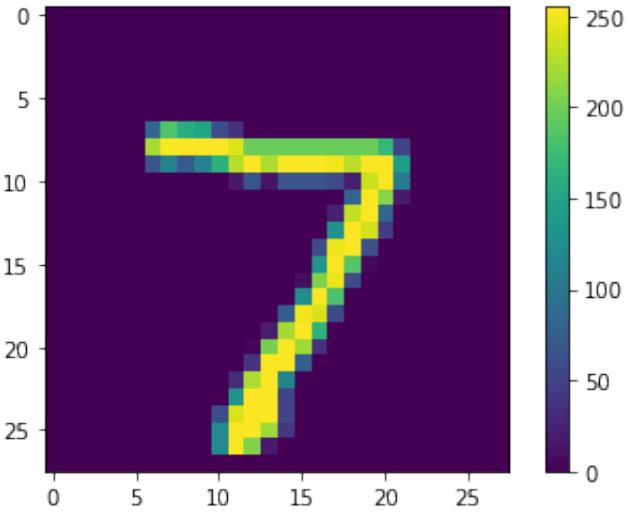
In [19]:

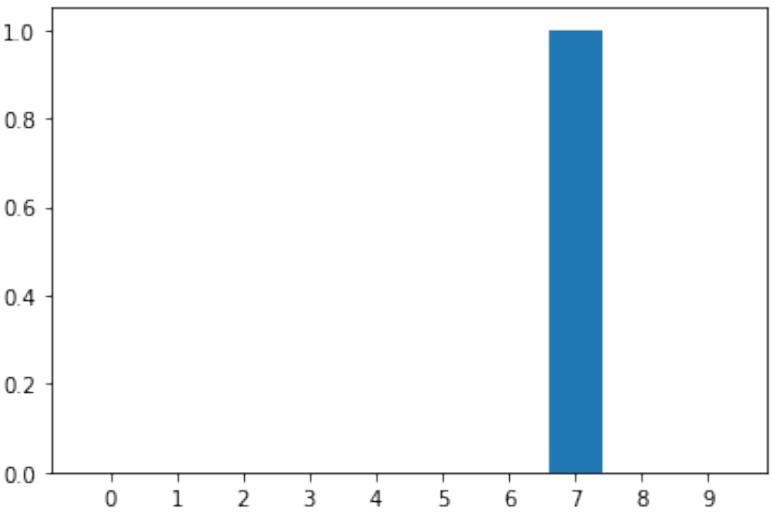
```
# Get some predictions
probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
predictions = probability_model.predict(x_test)
print(predictions.shape)
print(predictions[0])
```

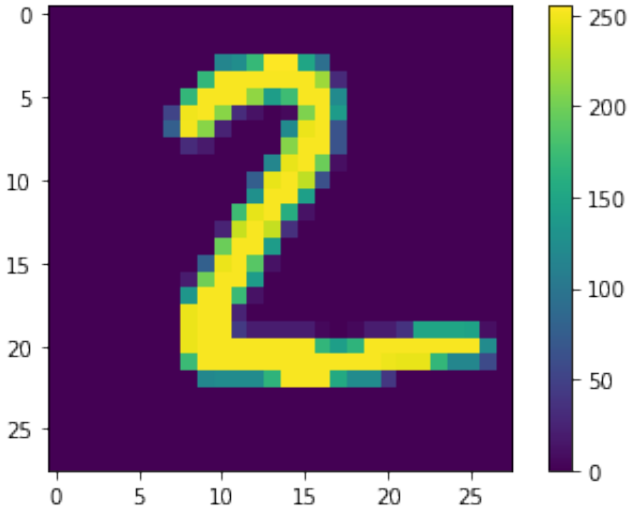
```
(10000, 10)
[0.0000000e+00 0.0000000e+00 4.0486893e-12 3.0670396e-14 0.0000000e+00
 1.9584209e-22 0.0000000e+00 1.0000000e+00 2.4825537e-35 1.5775345e-24]
```

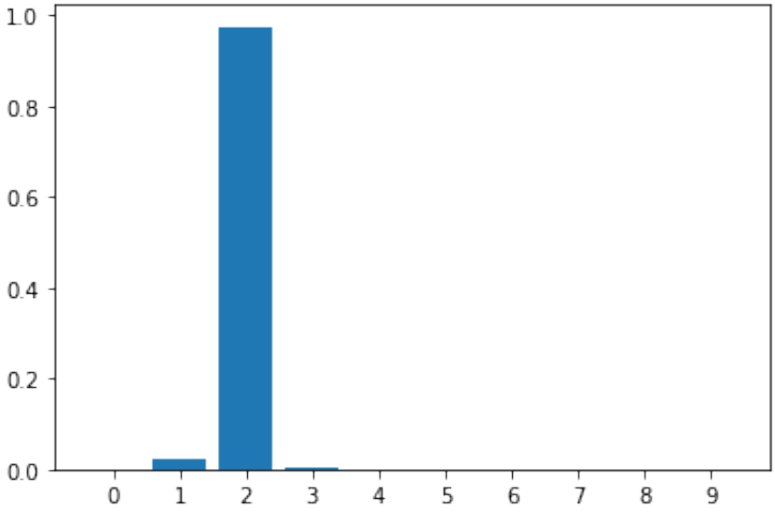
In [20]:

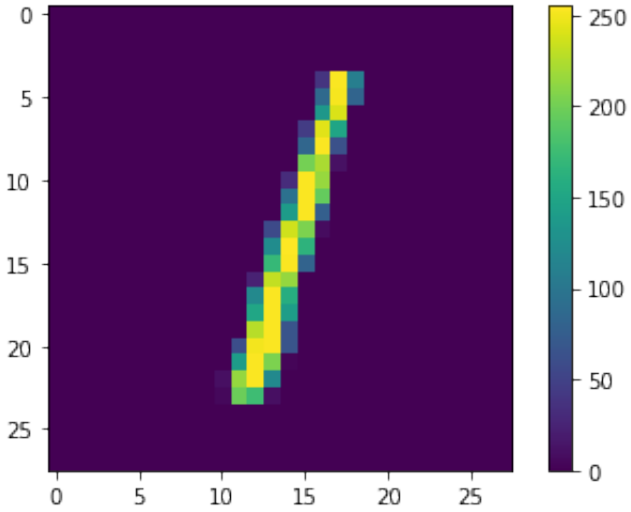
```
# Plot classification results
for i in range(3):
  plotImage(x_test[i])
  plt.xticks(range(10))
  plt.bar(range(10), predictions[i])
  plt.show()
```

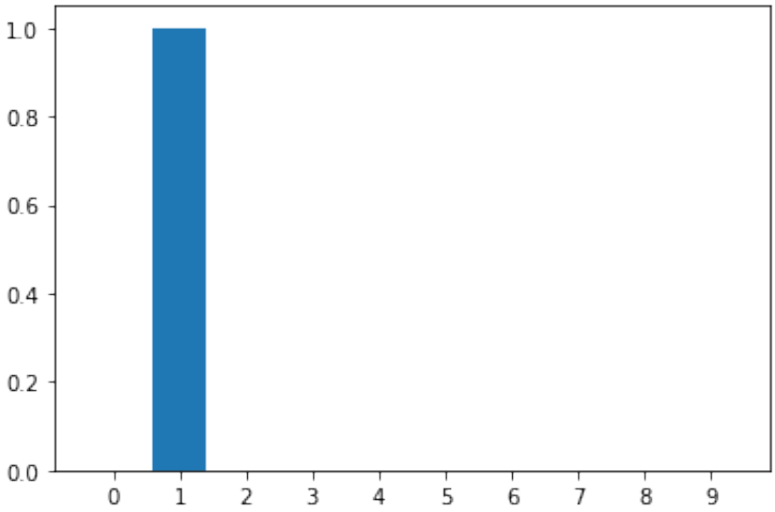





0
5
10
15
20
25
0
5
10
15
20
25
250
200
150
100
50
0


1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
9


0
5
10
15
20
25
0
5
10
15
20
25
250
200
150
100
50
0



In [99]: