Assignment is below at the bottom

Video 13.1 [https://www.youtube.com/watch?v=kIGHE7Cfe1s (https://www.youtube.com/watch?v=kIGHE7Cfe1s)](https://www.youtube.com/watch?v=kIGHE7Cfe1s)

Video 13.2 [https://www.youtube.com/watch?v=Rm9bJcDd1KU (https://www.youtube.com/watch?v=Rm9bJcDd1KU)](https://www.youtube.com/watch?v=Rm9bJcDd1KU)

Video 13.3 [https://youtu.be/6HjZk-3LsjE (https://youtu.be/6HjZk-3LsjE)](https://youtu.be/6HjZk-3LsjE)

In [11]:
```
from keras.callbacks import TensorBoard
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Demo data set up

(xtrain, ytrain), (xtest, ytest) = mnist.load_data()

xtrain = xtrain.astype('float32') / 255.
xtest = xtest.astype('float32') / 255.
xtrain = xtrain.reshape((len(xtrain), np.prod(xtrain.shape[1:])))
xtest = xtest.reshape((len(xtest), np.prod(xtest.shape[1:])))
xtrain.shape, xtest.shape
```

Out[2]:
```
((60000, 784), (10000, 784))
```

In [28]:
```
# this is the size of our encoded representations
encoding_dim = 4  # 32 floats -> compression of factor 24.5, assuming

# this is our input placeholder
x = input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
encoded = Dense(encoding_dim, activation='relu')(x)


# "decoded" is the lossy reconstruction of the input
x = Dense(128, activation='relu')(encoded)
x = Dense(256, activation='relu')(x)
decoded = Dense(784, activation='sigmoid')(x)

# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)

encoder = Model(input_img, encoded)

# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
dcd1 = autoencoder.layers[-1]
dcd2 = autoencoder.layers[-2]
dcd3 = autoencoder.layers[-3]

# create the decoder model
decoder = Model(encoded_input, dcd1(dcd2(dcd3(encoded_input))))
```

In [29]:
```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

In [ ]:
```
autoencoder.fit(xtrain, xtrain,
                epochs=100,
                batch_size=256,
                shuffle=True,
                validation_data=(xtest, xtest),
                #callbacks=[TensorBoard(log_dir='/tmp/autoencoder')])
```

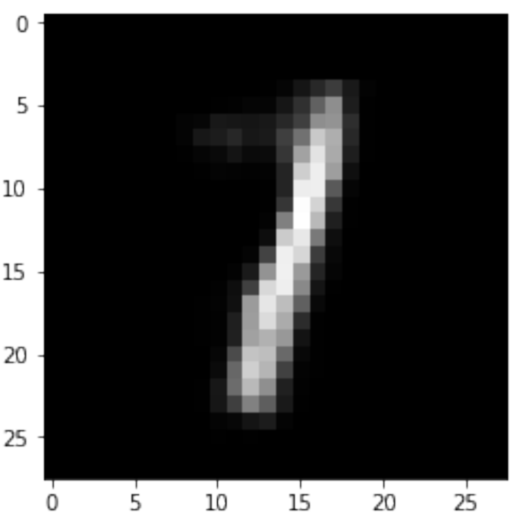
In [38]:
```
encoded_imgs
```

Out[38]:
```
array([[11.943697 ,  9.005527 , 12.027234 , 32.89881  ],
       [23.76052  , 13.926956 ,  5.6552634,  8.942506 ],
       [35.62965  , 34.729908 , 24.666973 , 41.5047   ],
       ...,
       [ 5.3135986, 11.108302 , 14.398285 , 17.106884 ],
       [ 4.376413 , 19.419018 , 15.854642 , 11.992302 ],
       [ 7.41167  , 18.699078 , 30.420742 , 11.0364065]], dtype=float
32)
```

In [52]:
```
noise = np.random.normal(20,4, (4,4))
noise_preds = decoder.predict(noise)
```

In [55]:
```
plt.imshow(noise_preds[1].reshape(28,28))
```

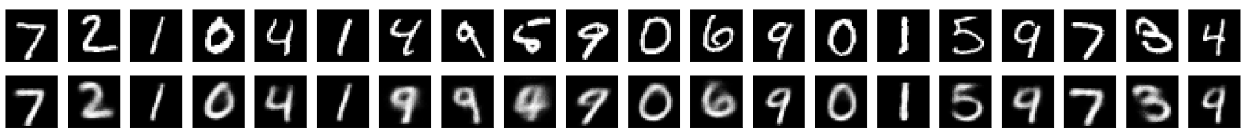
Out[55]:
```
<matplotlib.image.AxesImage at 0x13bf35780>
```



In [41]:
```
np.max(encoded_imgs)
```

Out[41]:
```
54.59457
```

In [32]:
```
encoded_imgs = encoder.predict(xtest)
decoded_imgs = decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt

n = 20  # how many digits we will display
plt.figure(figsize=(40, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(xtest[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



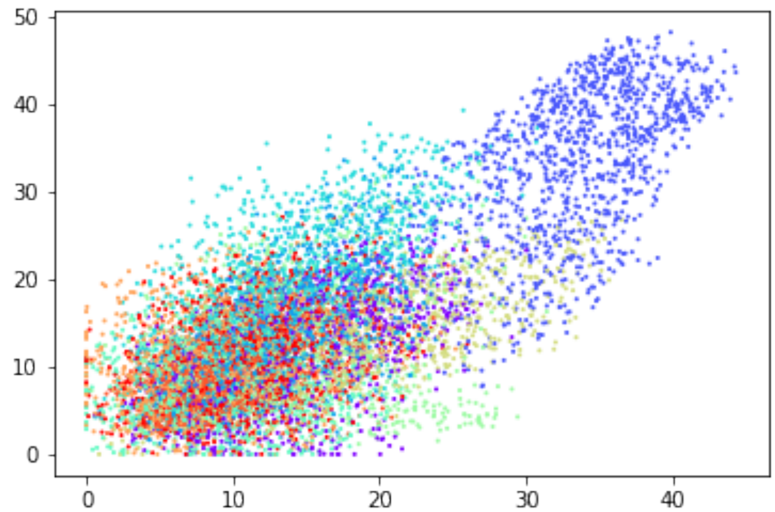
In [33]:
```
encoded_imgs
```

Out[33]:
```
array([[11.943697 ,  9.005527 , 12.027234 , 32.89881  ],
       [23.76052  , 13.926956 ,  5.6552634,  8.942506 ],
       [35.62965  , 34.729908 , 24.666973 , 41.5047   ],
       ...,
       [ 5.3135986, 11.108302 , 14.398285 , 17.106884 ],
       [ 4.376413 , 19.419018 , 15.854642 , 11.992302 ],
       [ 7.41167  , 18.699078 , 30.420742 , 11.0364065]], dtype=float
32)
```

In [26]:
```
%matplotlib inline
```

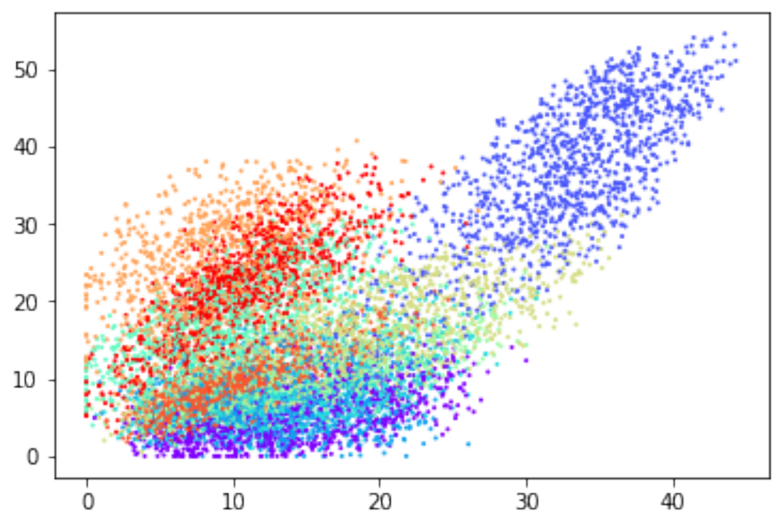
In [34]:
```
plt.scatter(encoded_imgs[:,1], encoded_imgs[:,0], s=1, c=ytest, cmap='
# plt.show()
```

Out[34]: <matplotlib.collections.PathCollection at 0x13c081978>



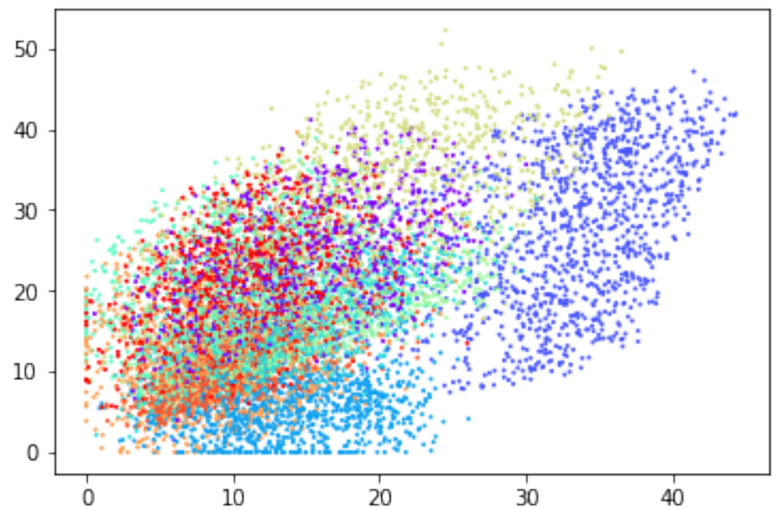
In [35]:
```
plt.scatter(encoded_imgs[:,1], encoded_imgs[:,3], s=1, c=ytest, cmap='
# plt.show()
```

Out[35]: <matplotlib.collections.PathCollection at 0x13b695e10>



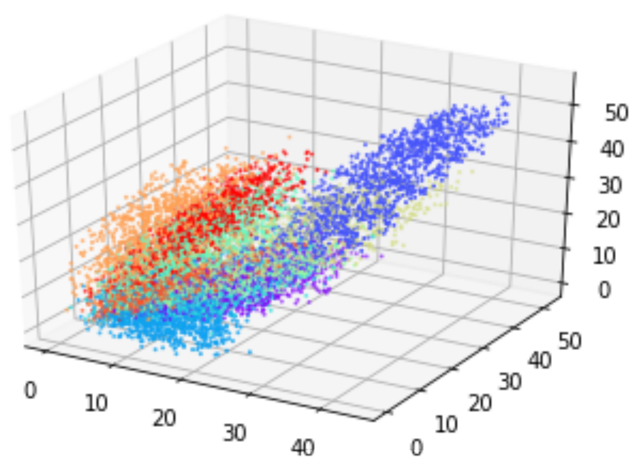
In [36]:
```
plt.scatter(encoded_imgs[:,1], encoded_imgs[:,2], s=1, c=ytest, cmap='
# plt.show()
```

Out[36]: <matplotlib.collections.PathCollection at 0x13b6eaf60>



In [37]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(encoded_imgs[:,1], encoded_imgs[:,2], encoded_imgs[:,3], c=
```

Out[37]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x13c0e7da0>



Assignment

1. change the `encoding_dim` through various values (`range(2,18,2)` and save the loss you can get. Plot the 8 pairs of dimensions vs loss on a scatter plot

In [7]:
```
# Assignment Data Setup

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
x_train.shape, x_test.shape
```

Out[7]:
```
((60000, 784), (10000, 784))
```

In [12]:
```
# Loop Setup

dimensions = range(2,18,2)
losses = []


for encoding_dim in dimensions:

    # Layer Setup  this seems inefficient to remake the layer for each
    x = input_img = Input(shape=(784,))
    # "encoded" is the encoded representation of the input
    x = Dense(256, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    encoded = Dense(encoding_dim, activation='relu')(x)


    # "decoded" is the lossy reconstruction of the input
    x = Dense(128, activation='relu')(encoded)
    x = Dense(256, activation='relu')(x)
    decoded = Dense(784, activation='sigmoid')(x)

    # this model maps an input to its reconstruction
    autoencoder = Model(input_img, decoded)

    encoder = Model(input_img, encoded)

    # create a placeholder for an encoded (32-dimensional) input
    encoded_input = Input(shape=(encoding_dim,))
    # retrieve the last layer of the autoencoder model
    dcd1 = autoencoder.layers[-1]
    dcd2 = autoencoder.layers[-2]
    dcd3 = autoencoder.layers[-3]

    # create the decoder model
    decoder = Model(encoded_input, dcd1(dcd2(dcd3(encoded_input))))

    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
    history = autoencoder.fit(x_train, x_train,
                    epochs=100,
                    batch_size=256,
                    shuffle=True,
                    verbose=0, # processes faster when not set to verbose
                    validation_data=(x_test, x_test))

    plt.plot(history.history['loss'], label=f'{encoding_dim} dimensior

    autoencoder.summary()

    loss = autoencoder.evaluate(x_train, x_train, verbose=0)

    print(loss)

    losses.append(loss)


plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs for Different Encoded Dimensions')
plt.legend()
plt.show()
printout = [item for sublist in zip(dimensions, losses) for item in su
print("Dimensions, Loss, Accuracy:", printout)
```

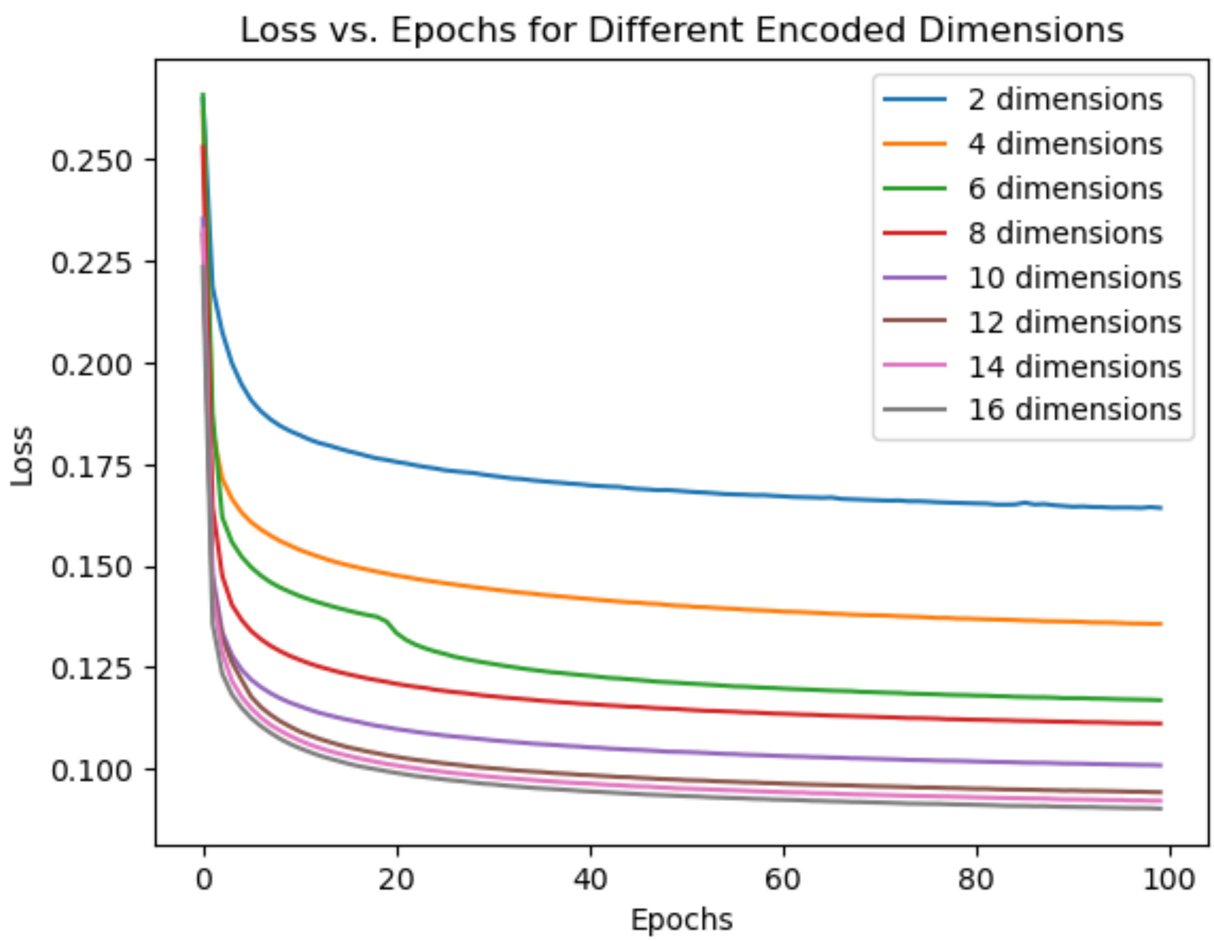
```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 784)]             0

 dense_12 (Dense)            (None, 256)               200960

 dense_13 (Dense)            (None, 128)               32896

 dense_14 (Dense)            (None, 2)                 258

 dense_15 (Dense)            (None, 128)               384

 dense_16 (Dense)            (None, 256)               33024

 dense_17 (Dense)            (None, 784)               201488

=================================================================
Total params: 469010 (1.79 MB)
Trainable params: 469010 (1.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
0.16411425173282623
```

```
Model: "model_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 784)]             0

 dense_18 (Dense)            (None, 256)               200960

 dense_19 (Dense)            (None, 128)               32896

 dense_20 (Dense)            (None, 4)                 516

 dense_21 (Dense)            (None, 128)               640

 dense_22 (Dense)            (None, 256)               33024

 dense_23 (Dense)            (None, 784)               201488

=================================================================
Total params: 469524 (1.79 MB)
Trainable params: 469524 (1.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
0.13520190119743347
Model: "model_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_9 (InputLayer)        [(None, 784)]             0

 dense_24 (Dense)            (None, 256)               200960

 dense_25 (Dense)            (None, 128)               32896

 dense_26 (Dense)            (None, 6)                 774

 dense_27 (Dense)            (None, 128)               896

 dense_28 (Dense)            (None, 256)               33024

 dense_29 (Dense)            (None, 784)               201488

=================================================================
Total params: 470038 (1.79 MB)
Trainable params: 470038 (1.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
0.11663084477186203
Model: "model_15"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_11 (InputLayer)       [(None, 784)]             0

 dense_30 (Dense)            (None, 256)               200960

 dense_31 (Dense)            (None, 128)               32896

 dense_32 (Dense)            (None, 8)                 1032

 dense_33 (Dense)            (None, 128)               1152

 dense_34 (Dense)            (None, 256)               33024

 dense_35 (Dense)            (None, 784)               201488

=================================================================
Total params: 470552 (1.80 MB)
Trainable params: 470552 (1.80 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
0.11094623804092407
Model: "model_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_13 (InputLayer)       [(None, 784)]             0

 dense_36 (Dense)            (None, 256)               200960

 dense_37 (Dense)            (None, 128)               32896

 dense_38 (Dense)            (None, 10)                1290

 dense_39 (Dense)            (None, 128)               1408

 dense_40 (Dense)            (None, 256)               33024

 dense_41 (Dense)            (None, 784)               201488

=================================================================
Total params: 471066 (1.80 MB)
Trainable params: 471066 (1.80 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
0.1008804589509964
Model: "model_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 input_15 (InputLayer)       [(None, 784)]             0

 dense_42 (Dense)            (None, 256)               200960

 dense_43 (Dense)            (None, 128)               32896

 dense_44 (Dense)            (None, 12)                1548

 dense_45 (Dense)            (None, 128)               1664

 dense_46 (Dense)            (None, 256)               33024

 dense_47 (Dense)            (None, 784)               201488

=================================================================
Total params: 471580 (1.80 MB)
Trainable params: 471580 (1.80 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
0.09382828325033188
Model: "model_24"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_17 (InputLayer)       [(None, 784)]             0

 dense_48 (Dense)            (None, 256)               200960

 dense_49 (Dense)            (None, 128)               32896

 dense_50 (Dense)            (None, 14)                1806

 dense_51 (Dense)            (None, 128)               1920

 dense_52 (Dense)            (None, 256)               33024

 dense_53 (Dense)            (None, 784)               201488

=================================================================
Total params: 472094 (1.80 MB)
Trainable params: 472094 (1.80 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
0.0918399766087532
Model: "model_27"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_19 (InputLayer)       [(None, 784)]             0
```

```
 dense_54 (Dense)            (None, 256)                200960

 dense_55 (Dense)            (None, 128)                32896

 dense_56 (Dense)            (None, 16)                 2064

 dense_57 (Dense)            (None, 128)                2176

 dense_58 (Dense)            (None, 256)                33024

 dense_59 (Dense)            (None, 784)                201488

=================================================================
Total params: 472608 (1.80 MB)
Trainable params: 472608 (1.80 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
0.09008263051509857
```

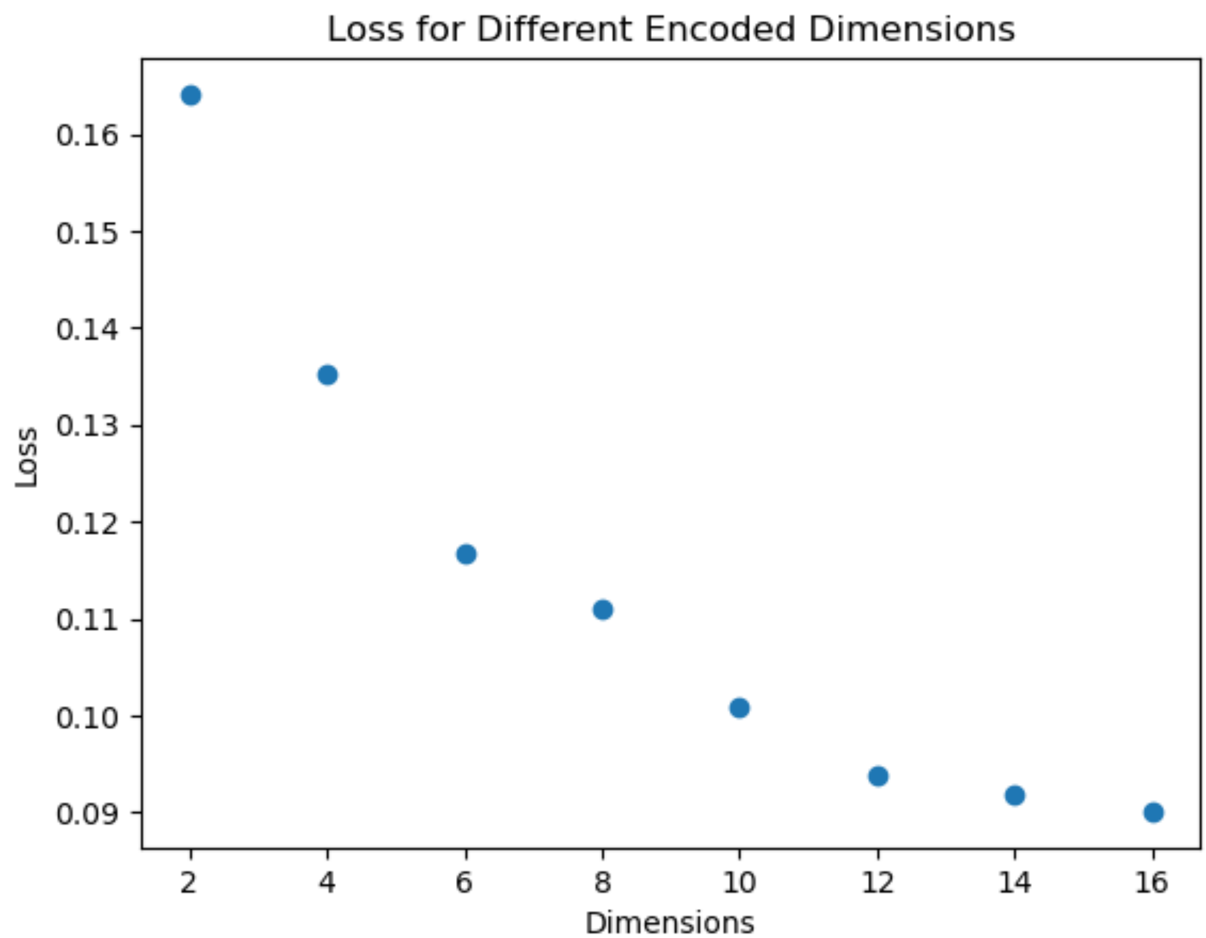



```
Dimensions, Loss, Accuracy: [2, 0.16411425173282623, 4, 0.13520190119
743347, 6, 0.11663084477186203, 8, 0.11094623804092407, 10, 0.1008804
589509964, 12, 0.09382828325033188, 14, 0.0918399766087532, 16, 0.090
```

```
08263051509857]
```

In [57]:
```
# Scatter Plot

plt.scatter(dimensions, losses)
plt.xlabel('Dimensions')
plt.ylabel('Loss')
plt.title('Loss for Different Encoded Dimensions')
```

Out[57]:
```
Text(0.5, 1.0, 'Loss for Different Encoded Dimensions')
```



2. ***After*** training an autoencoder with `encoding_dim=8` , apply noise (like the previous assignment) to *only* the input of the trained autoencoder (not the output). The output images should be without noise.

Print a few noisy images along with the output images to show they don't have noise.

In [42]:
```
# Building the model

encoding_dim = 8

x = input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
# input_img_noise = input_img + np.random.normal(scale = 1.0, size = (
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
encoded = Dense(encoding_dim, activation='relu')(x)


# "decoded" is the lossy reconstruction of the input
x = Dense(128, activation='relu')(encoded)
x = Dense(256, activation='relu')(x)
decoded = Dense(784, activation='sigmoid')(x)

# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)

encoder = Model(input_img, encoded)

# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
dcd1 = autoencoder.layers[-1]
dcd2 = autoencoder.layers[-2]
dcd3 = autoencoder.layers[-3]

# create the decoder model
decoder = Model(encoded_input, dcd1(dcd2(dcd3(encoded_input))))
```

In [43]:
```
np.shape(x_train),np.shape(x_test)
```

Out[43]:
```
((60000, 784), (10000, 784))
```

In [50]:
```
# Noise and modeling

x_train_noise = x_train + np.random.normal(scale = 0.5, size = (60000,
x_test_noise = x_test + np.random.normal(scale = 0.5, size = (10000, 7

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train_noise, x_train,
                epochs=100,
                batch_size=256,
                shuffle=True,
                verbose=0,
                validation_data=(x_test_noise, x_test))
```

Out[50]:
```
<keras.src.callbacks.History at 0x7f9267bcc460>
```

In [51]:
```
# Displaying images

encoded_imgs = encoder.predict(x_test_noise)
decoded_imgs = decoder.predict(encoded_imgs)

n = 20  # how many digits we will display
plt.figure(figsize=(40, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test_noise[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

```
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 1ms/step
```