Module 12 Assignment: Autoencoders

Video 13.1
https://www.youtube.com/watch?v=kIGHE7Cfe1s

Video 13.2
https://www.youtube.com/watch?v=Rm9bJcDd1KU

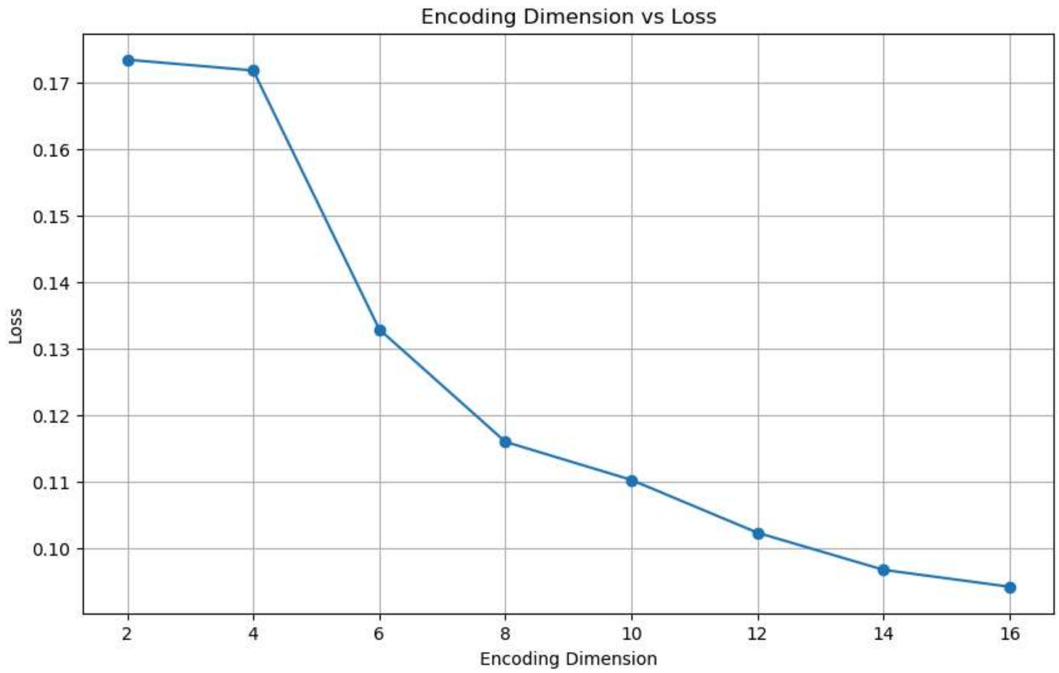
Video 13.3
https://youtu.be/6HjZk-3LsjE

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 [16]:

```
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

# Loading MNIST dataset
(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

# Creating function to train the autoencoder and return the validation
def train_autoencoder(encoding_dim):
    input_img = Input(shape=(784,))
    x = Dense(256, activation='relu')(input_img)
    x = Dense(128, activation='relu')(x)
    encoded = Dense(encoding_dim, activation='relu')(x)
    x = Dense(128, activation='relu')(encoded)
    x = Dense(256, activation='relu')(x)
    decoded = Dense(784, activation='sigmoid')(x)

    autoencoder = Model(input_img, decoded)
    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
    history = autoencoder.fit(xtrain, xtrain,
                              epochs=50,
                              batch_size=256,
                              shuffle=True,
                              validation_data=(xtest, xtest),
                              verbose=0)
    val_loss = history.history['val_loss'][-1]
    return val_loss

# Training autoencoders with various encoding dimensions and saving th
dimensions = range(2, 18, 2)
losses = []

for encoding_dim in dimensions:
    loss = train_autoencoder(encoding_dim)
    losses.append(loss)
    print(f'Encoding Dim: {encoding_dim}, Loss: {loss}')

# Plotting the losses vs encoding dimensions
plt.figure(figsize=(10, 6))
plt.plot(dimensions, losses, marker='o')
plt.title('Encoding Dimension vs Loss')
plt.xlabel('Encoding Dimension')
plt.ylabel('Loss')
plt.grid(True)
plt.show()
```

```
Encoding Dim: 2, Loss: 0.1735038012266159
Encoding Dim: 4, Loss: 0.1718769073486328
Encoding Dim: 6, Loss: 0.13293473422527313
Encoding Dim: 8, Loss: 0.11601827293634415
Encoding Dim: 10, Loss: 0.11028366535902023
Encoding Dim: 12, Loss: 0.10233931243419647
Encoding Dim: 14, Loss: 0.09675975143909454
Encoding Dim: 16, Loss: 0.09421230852603912
```



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 [15]:

```
# Training an autoencoder with encoding_dim=8
encoding_dim = 8

input_img = Input(shape=(784,))
x = Dense(256, activation='relu')(input_img)
x = Dense(128, activation='relu')(x)
encoded = Dense(encoding_dim, activation='relu')(x)
x = Dense(128, activation='relu')(encoded)
x = Dense(256, activation='relu')(x)
decoded = Dense(784, activation='sigmoid')(x)

autoencoder = Model(input_img, decoded)
encoder = Model(input_img, encoded)

encoded_input = Input(shape=(encoding_dim,))
dcd1 = autoencoder.layers[-1]
dcd2 = autoencoder.layers[-2]
dcd3 = autoencoder.layers[-3]
decoder = Model(encoded_input, dcd1(dcd2(dcd3(encoded_input))))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(xtrain, xtrain,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(xtest, xtest),
                verbose=0)

# Adding noise to the input
noise_factor = 0.5
xtest_noisy = xtest + noise_factor * np.random.normal(loc=0.0, scale=1
xtest_noisy = np.clip(xtest_noisy, 0., 1.)

# Getting the denoised images
decoded_imgs = autoencoder.predict(xtest_noisy)

# Displaying the noisy and denoised images
n = 10  # number of digits to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original + noise
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(xtest_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display denoised image
    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 ━━━━━━━━━━━━━━━━━━━━ **2s** 5ms/step