Hussein Hamie 500876254 AER 850 Machine Learning Assignment 2

We begin by importing the initial libraries for plotting and data preprocessing

In [1]:
```python
# Importing of libraries
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import scale
from sklearn.linear_model import SGDRegressor
```

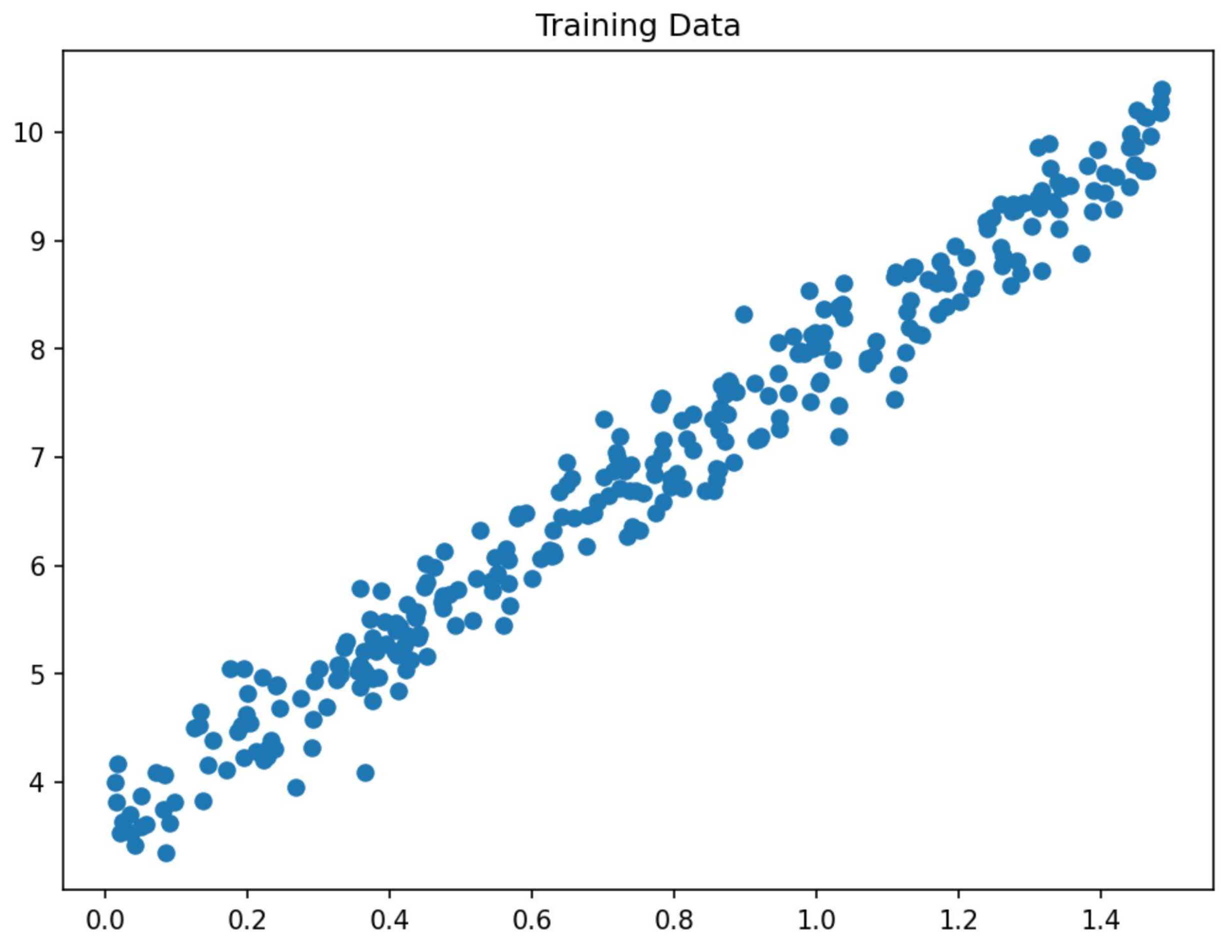
Function used to import datasets. The function saves X and Y Columns as Numpy arrays and reshapes them in 2D form.

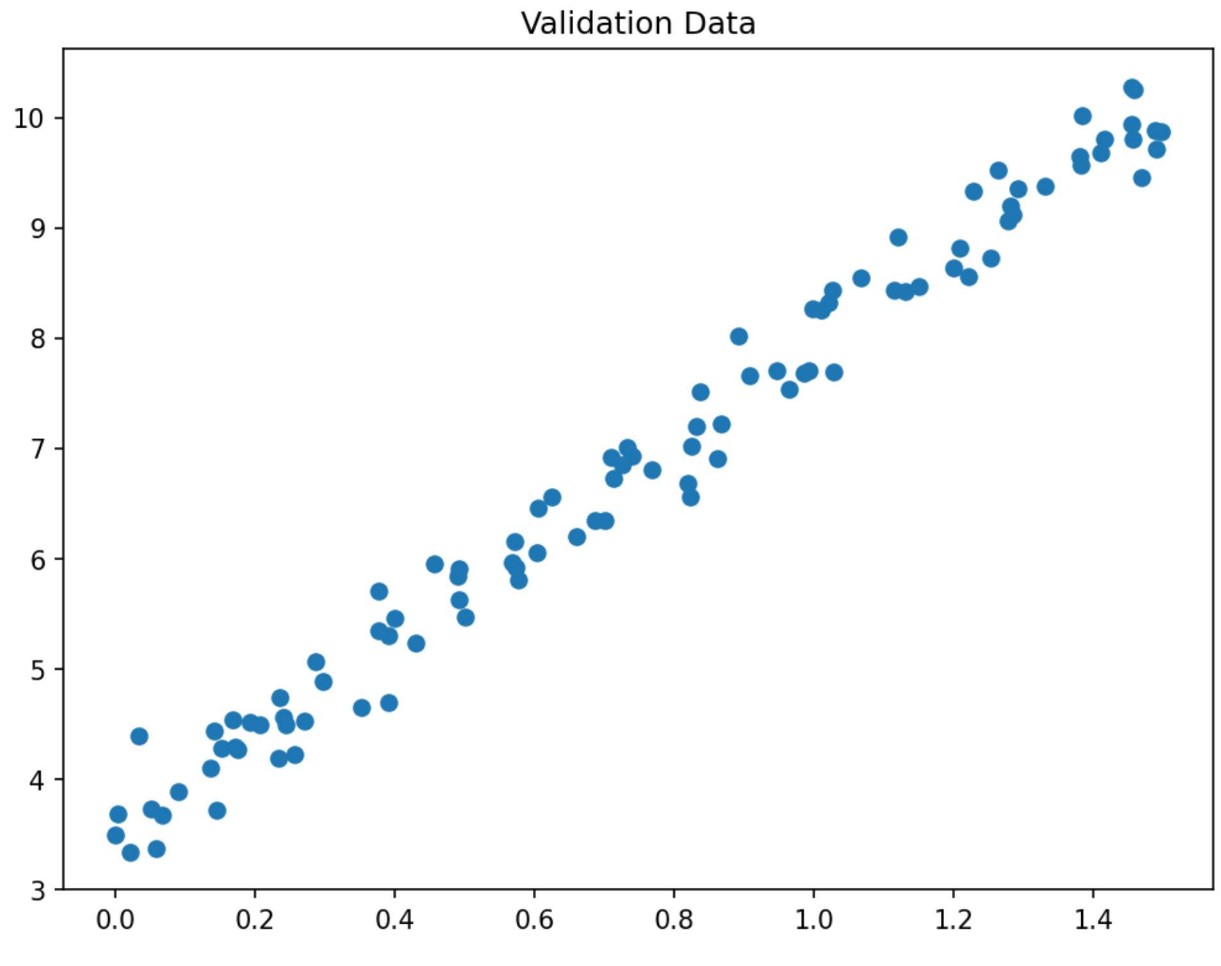
In [3]:
```python
def getdata(filename):
    with open (filename,'r') as csvfile:
        csvData = pd.read_csv(csvfile,header=None)
        #Ordering the data properly
        csvData.sort_values(csvData.columns[0],axis=0,inplace=True)
        print(csvData)
        X = csvData.iloc[:,[0]].to_numpy()
        Y = csvData.iloc[:,[1]].to_numpy()
        print(X)
    return X , Y
```

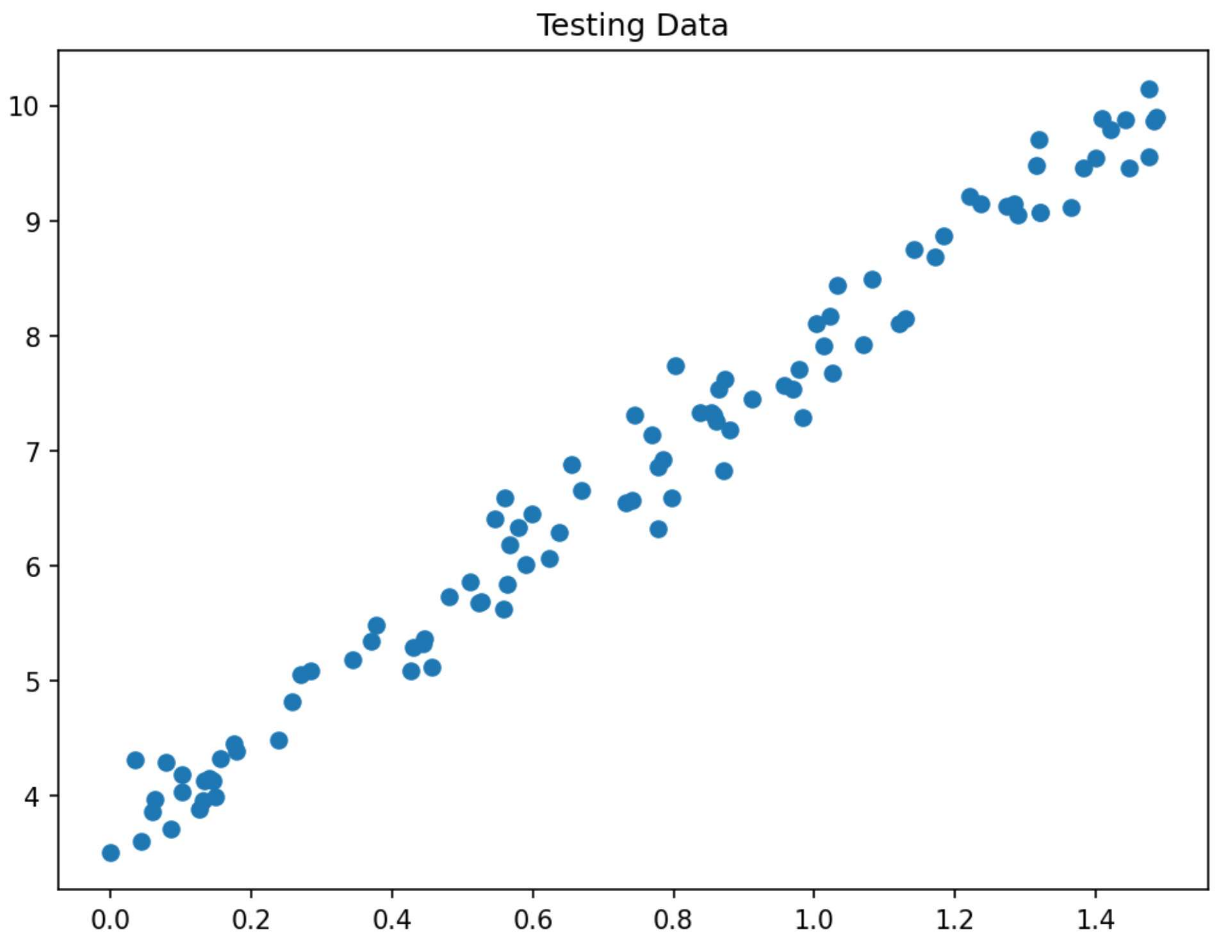
Importing all datasets using get data function and assigning X and Y respectively

In [4]:
```python
%%capture
X_test, Y_test = getdata("Dataset2/Dataset_2_test.csv")
X_train, Y_train = getdata("Dataset2/Dataset_2_train.csv")
X_valid, Y_valid = getdata("Dataset2/Dataset_2_valid.csv")
```

In [7]:
```python
plt.figure(figsize=(8, 6), dpi=150)
plt.scatter(X_train,Y_train)
plt.title('Training Data')
plt.show()

plt.figure(figsize=(8, 6), dpi=150)
plt.scatter(X_valid,Y_valid)
plt.title('Validation Data')
plt.show()

plt.figure(figsize=(8, 6), dpi=150)
plt.scatter(X_test,Y_test)
plt.title('Testing Data')
plt.show()
```


Training Data
10
9
8
7
6
5
4
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4

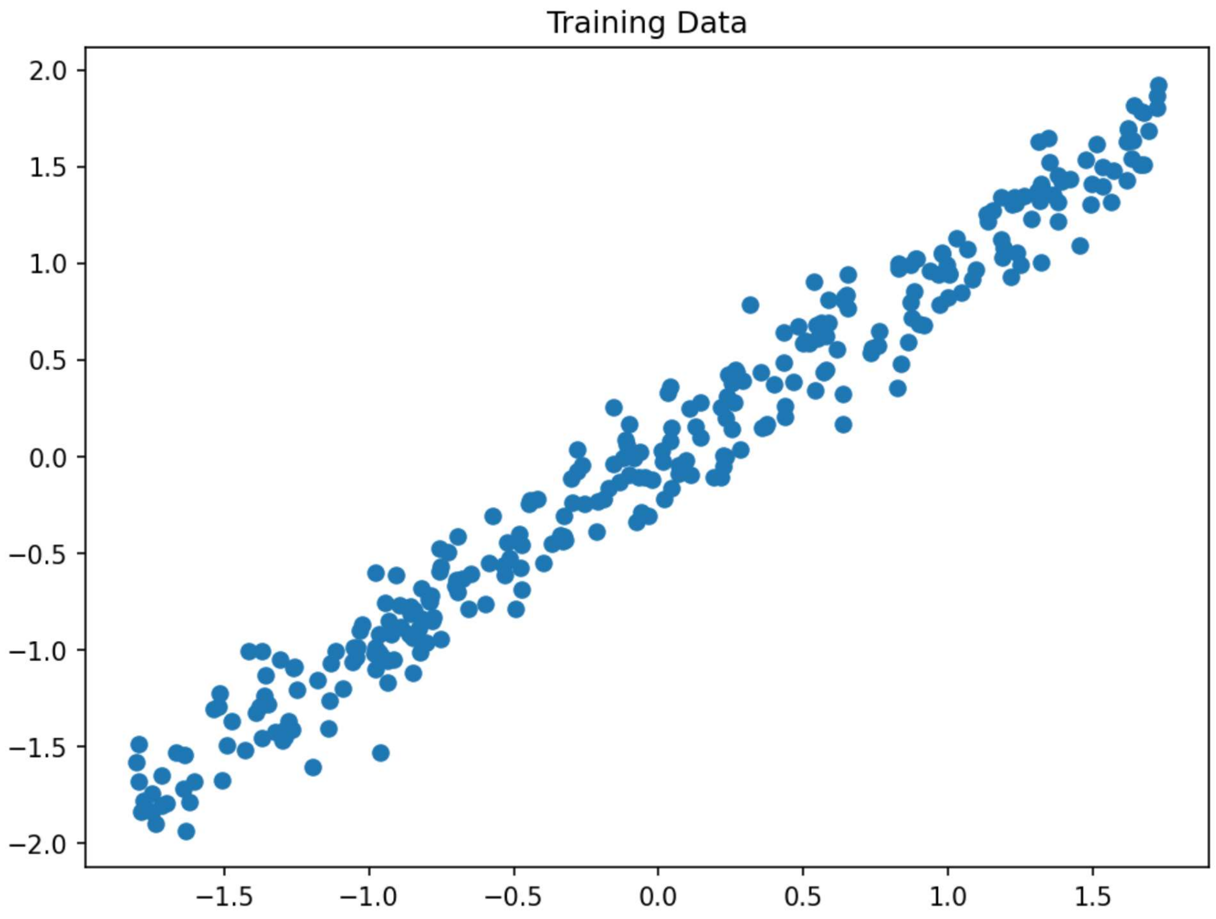

Validation Data
10
9
8
7
6
5
4
3
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4



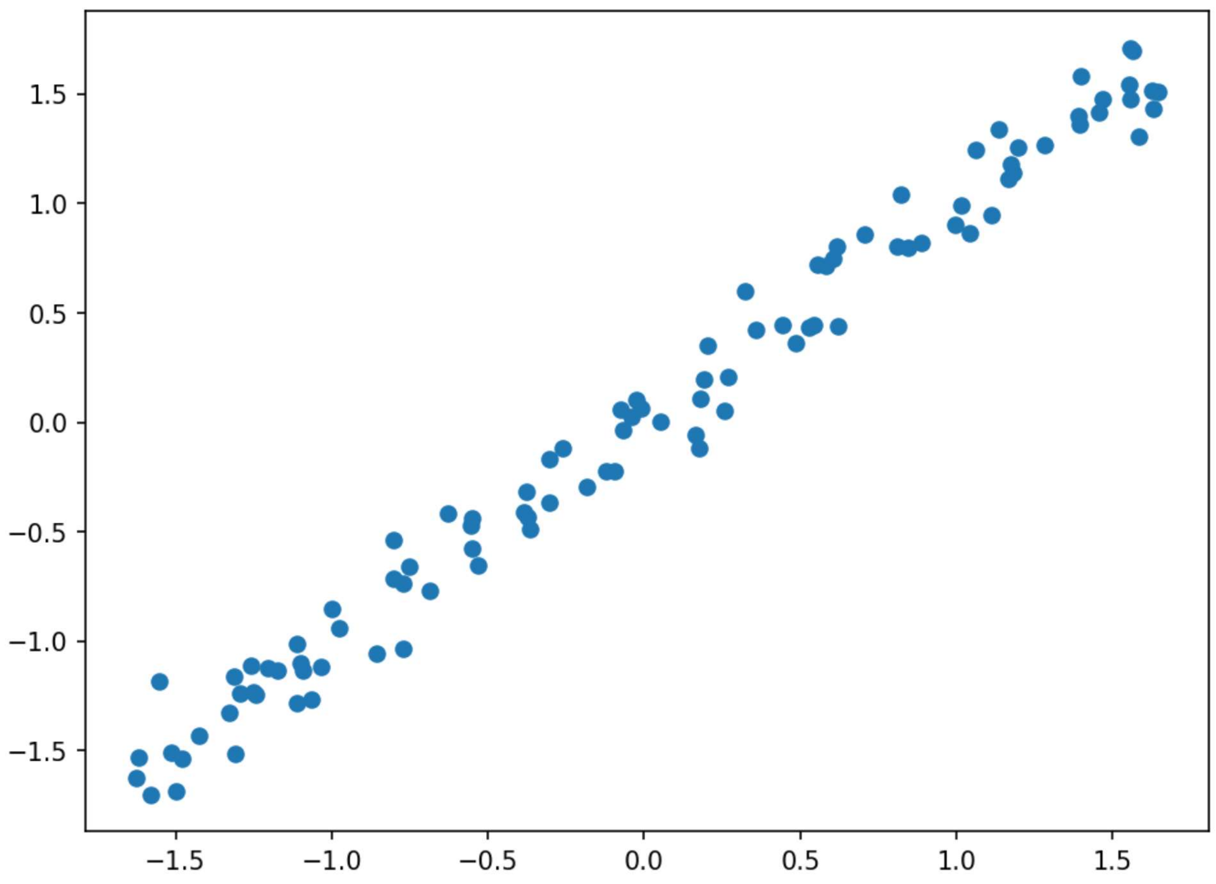
Scaling the data is generally a good idea when using stochastic gradient descent (SGD) for linear regression. This is because SGD is sensitive to the scale of the input features, and features with larger scales can dominate the learning process and make it difficult for the algorithm to converge.

In particular, it's recommended to standardize the input features so that they have zero mean and unit variance. This can be achieved using the StandardScaler class from scikit-learn.

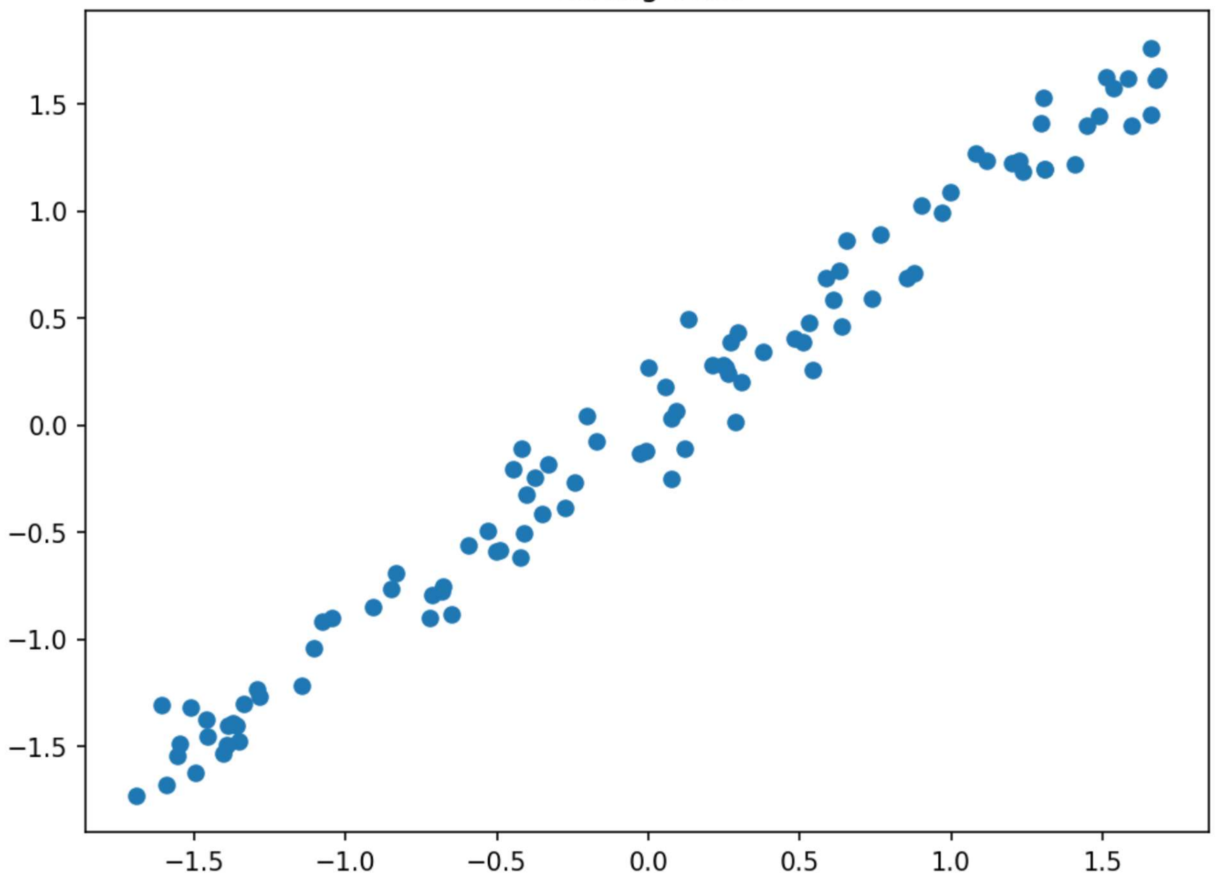
In [8]:
```
X_train = scale(X_train)
Y_train = scale(Y_train)
plt.figure(figsize=(8, 6), dpi=150)
plt.scatter(X_train,Y_train)
plt.title('Training Data')
plt.show()


X_valid = scale(X_valid)
Y_valid = scale(Y_valid)
plt.figure(figsize=(8, 6), dpi=150)
plt.scatter(X_valid,Y_valid)
plt.title('Validation Data')
plt.show()

X_test = scale(X_test)
Y_test = scale(Y_test)
plt.figure(figsize=(8, 6), dpi=150)
plt.scatter(X_test,Y_test)
plt.title('Testing Data')
plt.show()
```



Validation Data



Testing Data



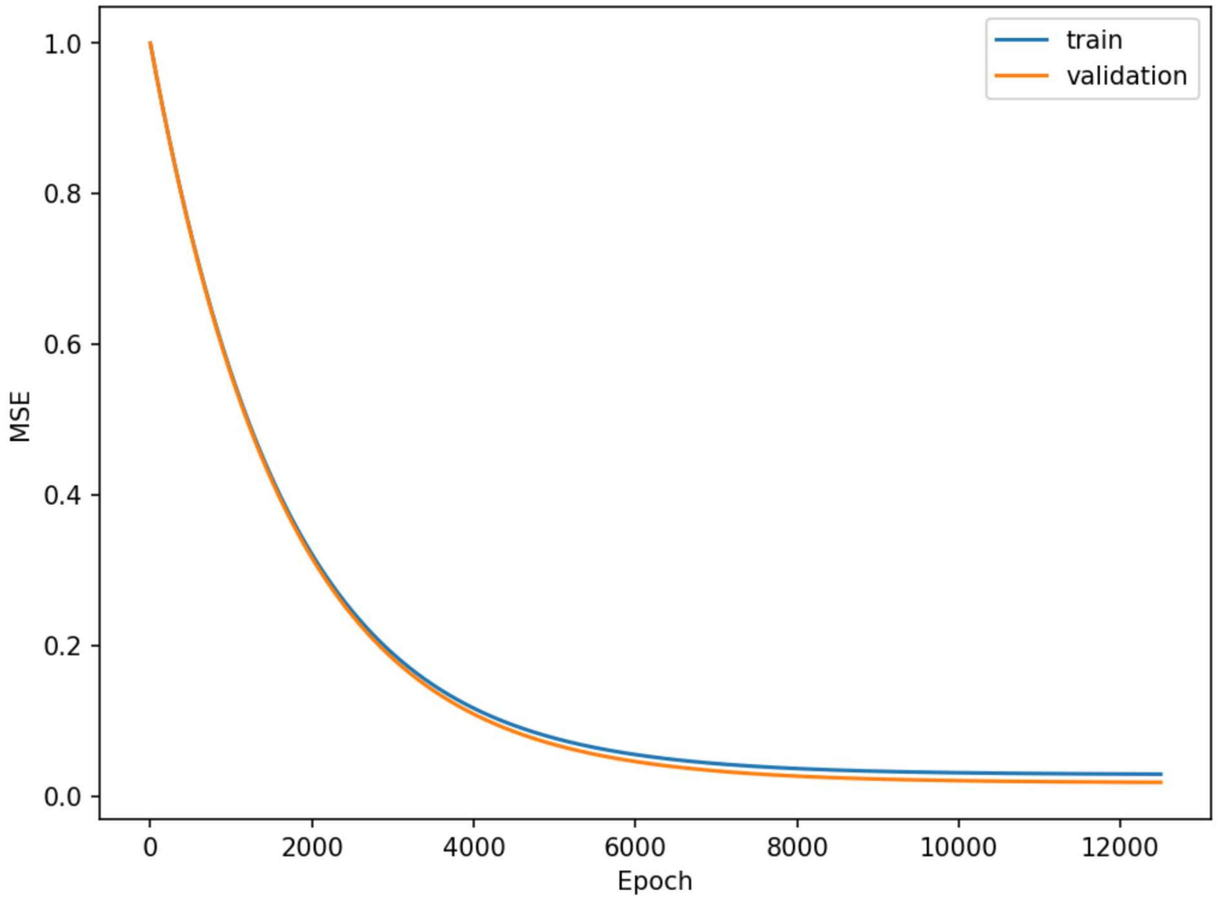
In scikit-learn's SGDRegressor class, the step size used in stochastic gradient descent is controlled by the eta0 parameter. The eta0 parameter is the initial step size for the updates, and it is multiplied by the learning rate schedule (controlled by the learning_rate parameter) at each iteration to determine the actual step size.

The alpha parameter, on the other hand, controls the L2 regularization strength in the model. In the example code I provided in my previous answer, the alpha parameter is set to 0.0 to perform linear regression without regularization.

So in the context of the SGDRegressor class, the step size is controlled by the eta0 parameter, not the alpha parameter.

In [12]:
```
model = SGDRegressor(alpha=0.0, learning_rate='constant', eta0=1e-6, random_state=42)
train_errors = []
val_errors = []
for epoch in range(12500):
    model.partial_fit(X_train, Y_train)
    y_train_pred = model.predict(X_train)
    train_mse = mean_squared_error(Y_train, y_train_pred)
    train_errors.append(train_mse)
    y_val_pred = model.predict(X_valid)
    val_mse = mean_squared_error(Y_valid, y_val_pred)
    val_errors.append(val_mse)
print(f"Epoch {epoch+1}: train MSE = {train_mse:.4f}, val MSE = {val_mse:.4f}")
# Plot the training and validation MSE for every epoch
plt.figure(figsize=(8, 6), dpi=150)
plt.plot(train_errors, label='train')
plt.plot(val_errors, label='validation')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.show()
```

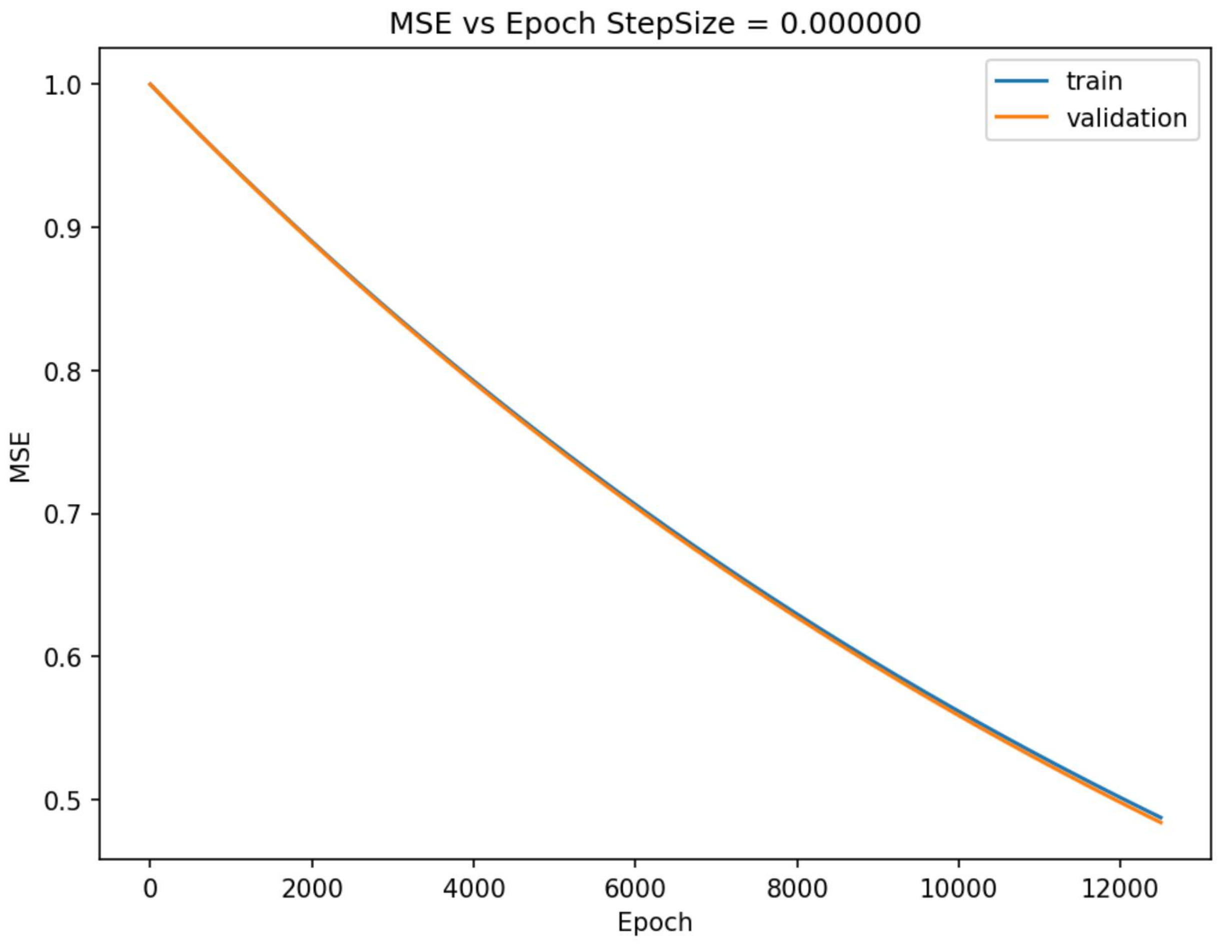
```
Epoch 12500: train MSE = 0.0291, val MSE = 0.0183
```

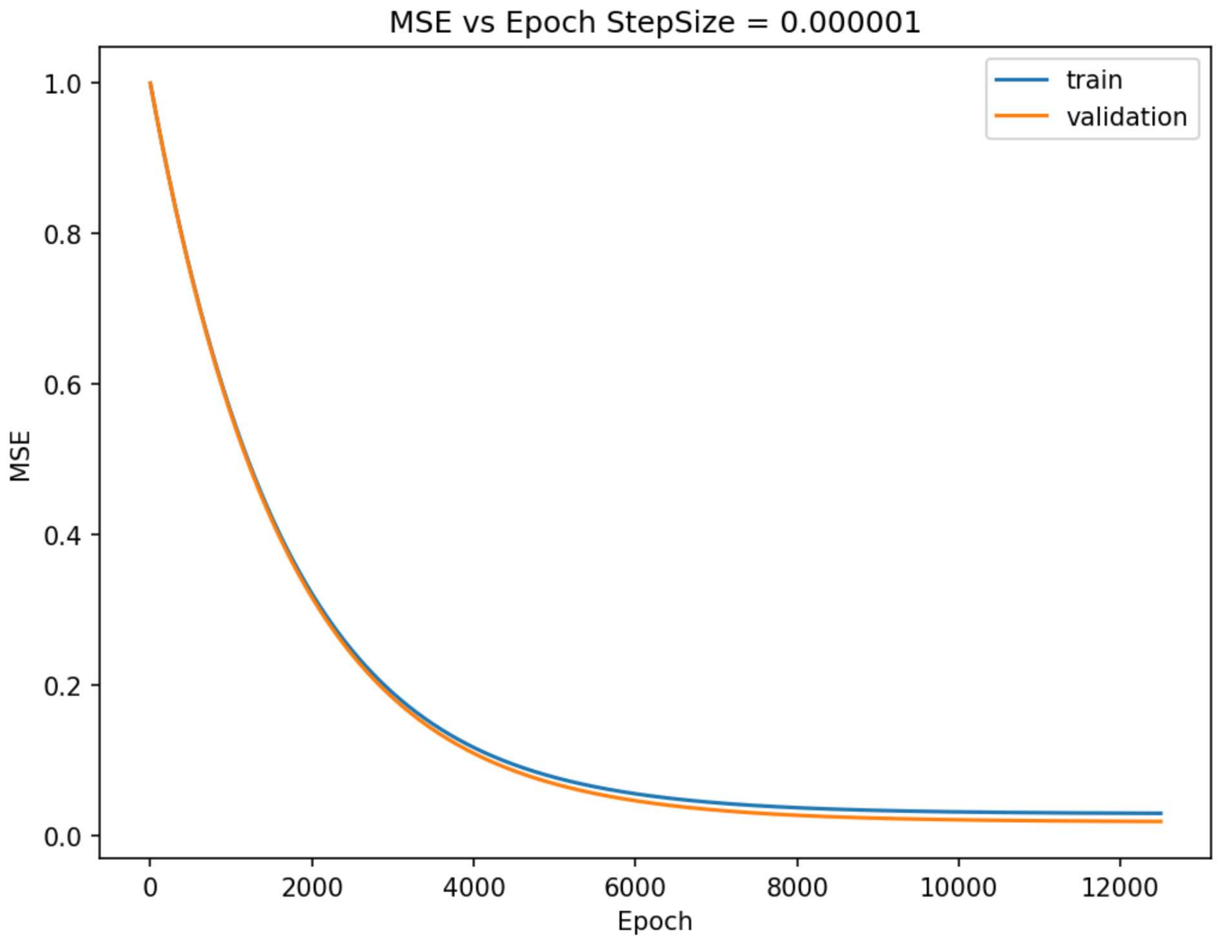


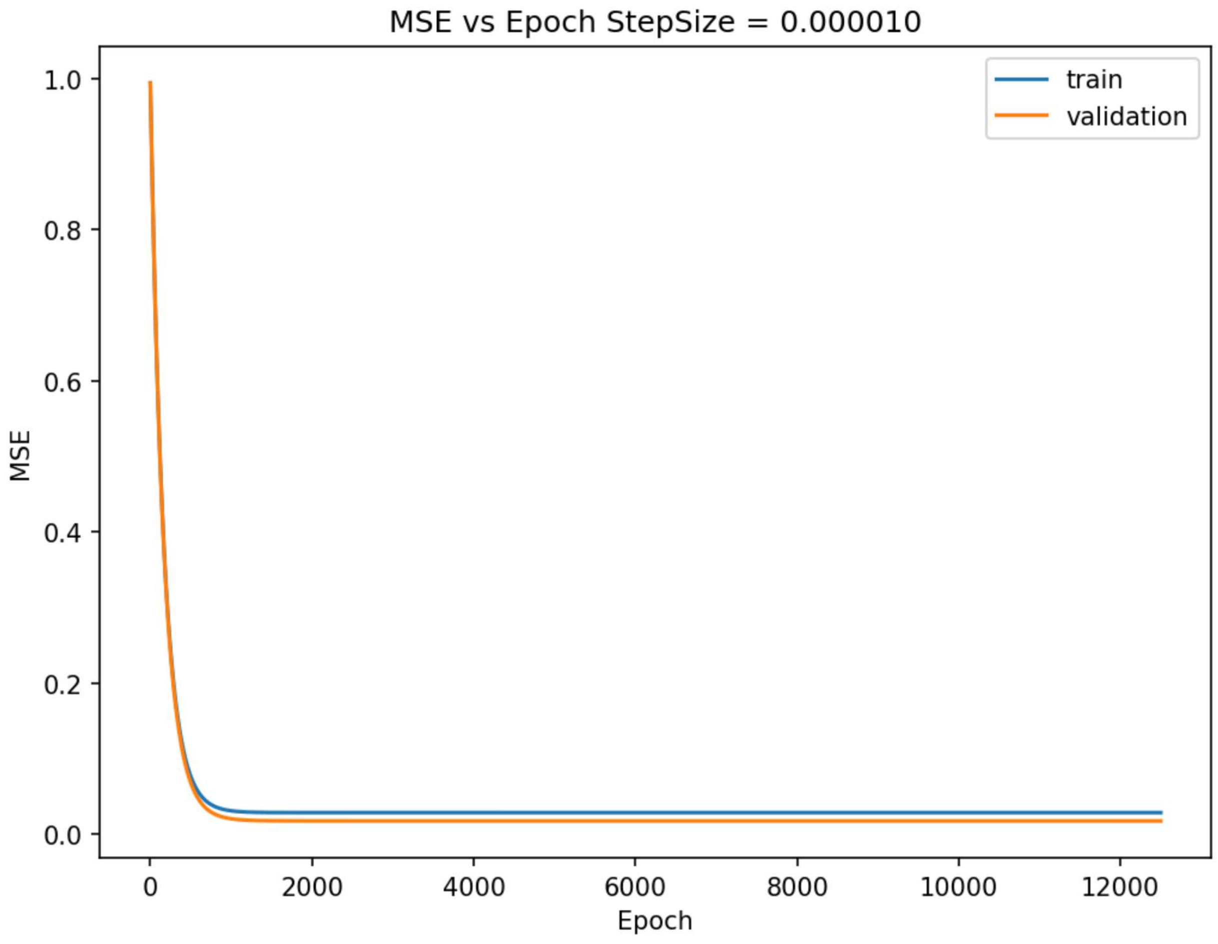
PART B

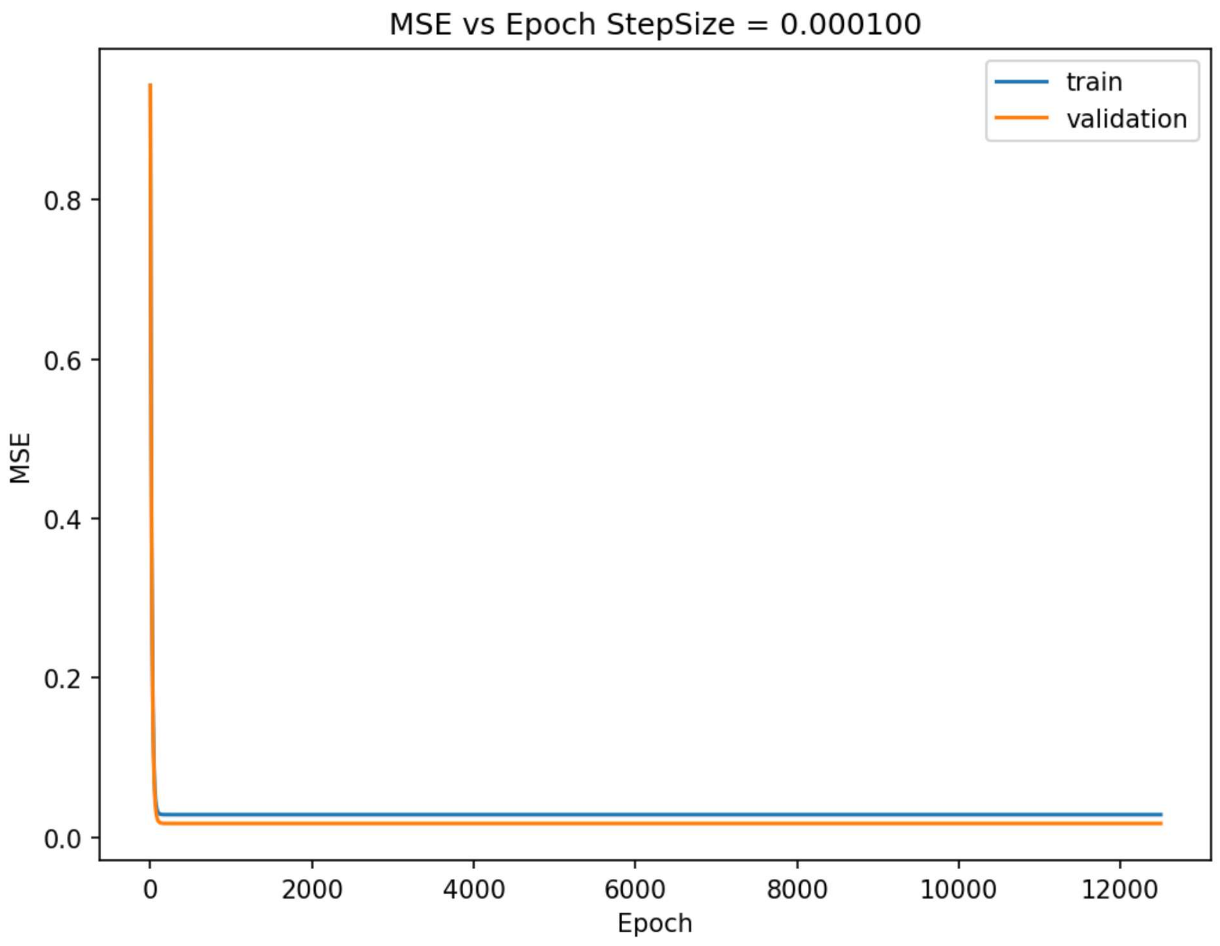
To try different step sizes and choose the best one for our linear regression model trained with stochastic gradient descent, we can use a nested cross-validation approach. Here's an example of how to do this using scikit-learn:

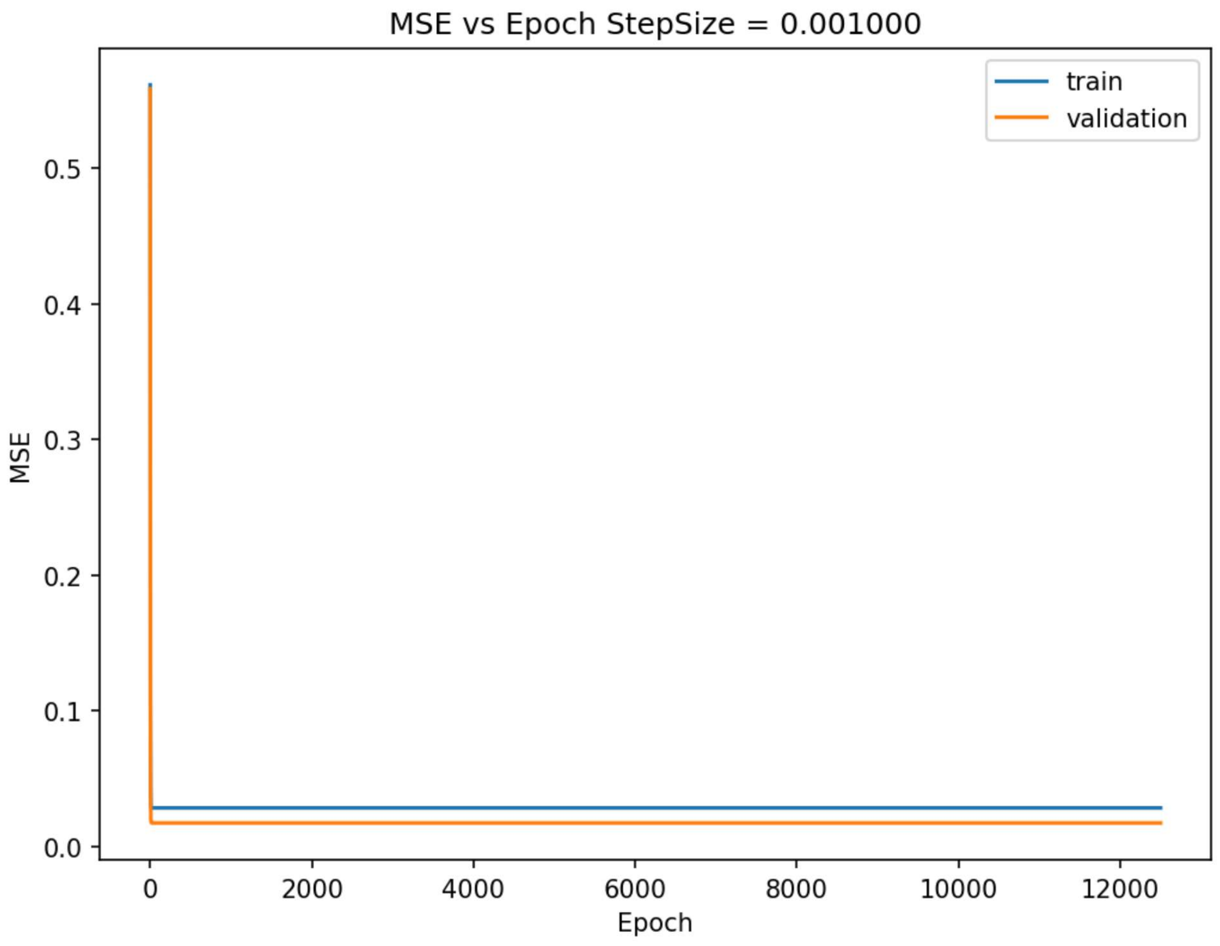
In [13]:
```
etagrid =[1e-7, 1e-6,1e-5, 1e-4, 1e-3, 1e-2, 1e-1]


for eta in etagrid:

    model = SGDRegressor(alpha=0.0, learning_rate='constant', eta0=eta, random_state=4
    train_errors = []
    val_errors = []
    for epoch in range(12500):
        model.partial_fit(X_train, Y_train)
        y_train_pred = model.predict(X_train)
        train_mse = mean_squared_error(Y_train, y_train_pred)
        train_errors.append(train_mse)
        y_val_pred = model.predict(X_valid)
        val_mse = mean_squared_error(Y_valid, y_val_pred)
        val_errors.append(val_mse)
        #print(f"Epoch {epoch+1}: train MSE = {train_mse:.4f}, val MSE = {val_mse:.4f}

    # Plot the training and validation MSE for every epoch
    plt.figure(figsize=(8, 6), dpi=150)
    plt.plot(train_errors, label='train')
    plt.plot(val_errors, label='validation')
    plt.legend()
    plt.title("MSE vs Epoch StepSize = %f" % eta)
    plt.xlabel('Epoch')
    plt.ylabel('MSE')
    plt.show()
```

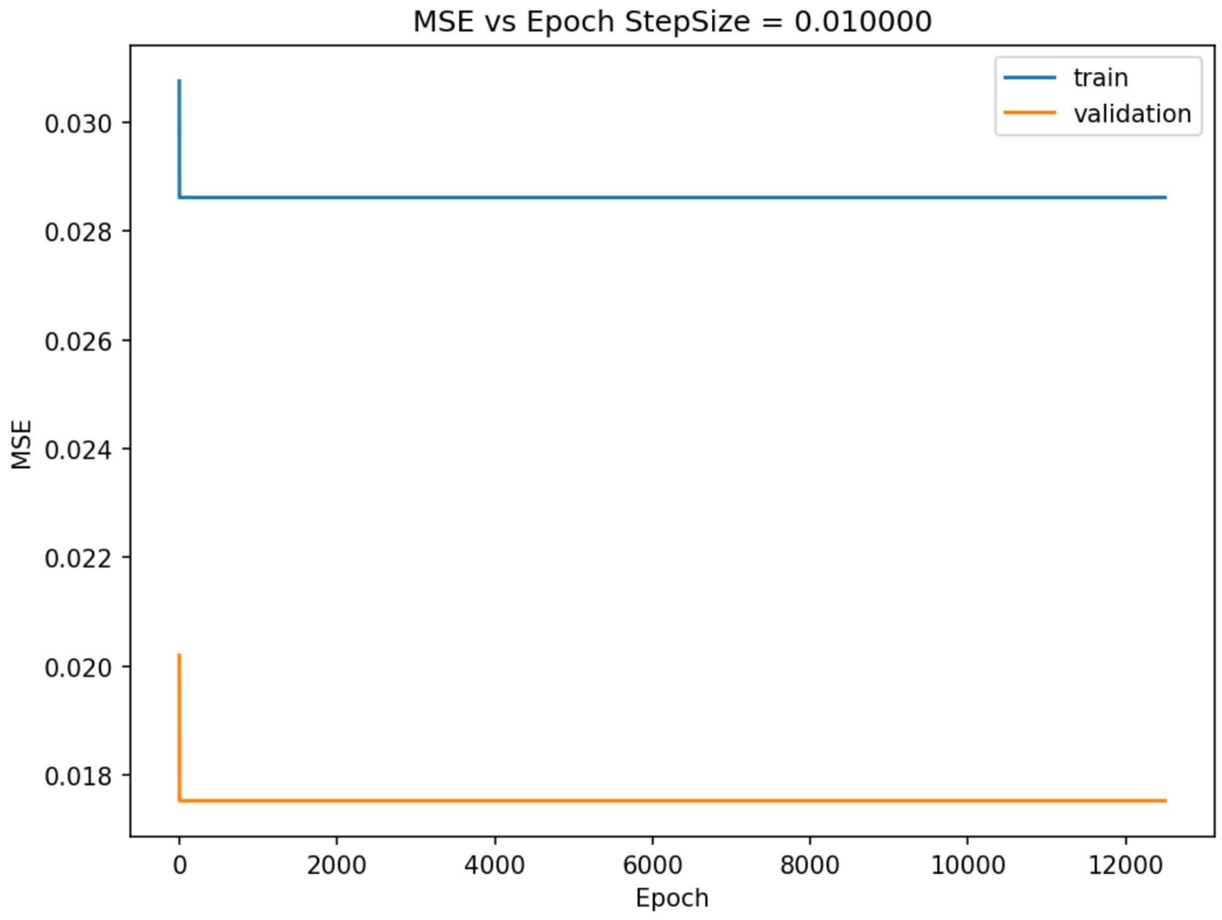


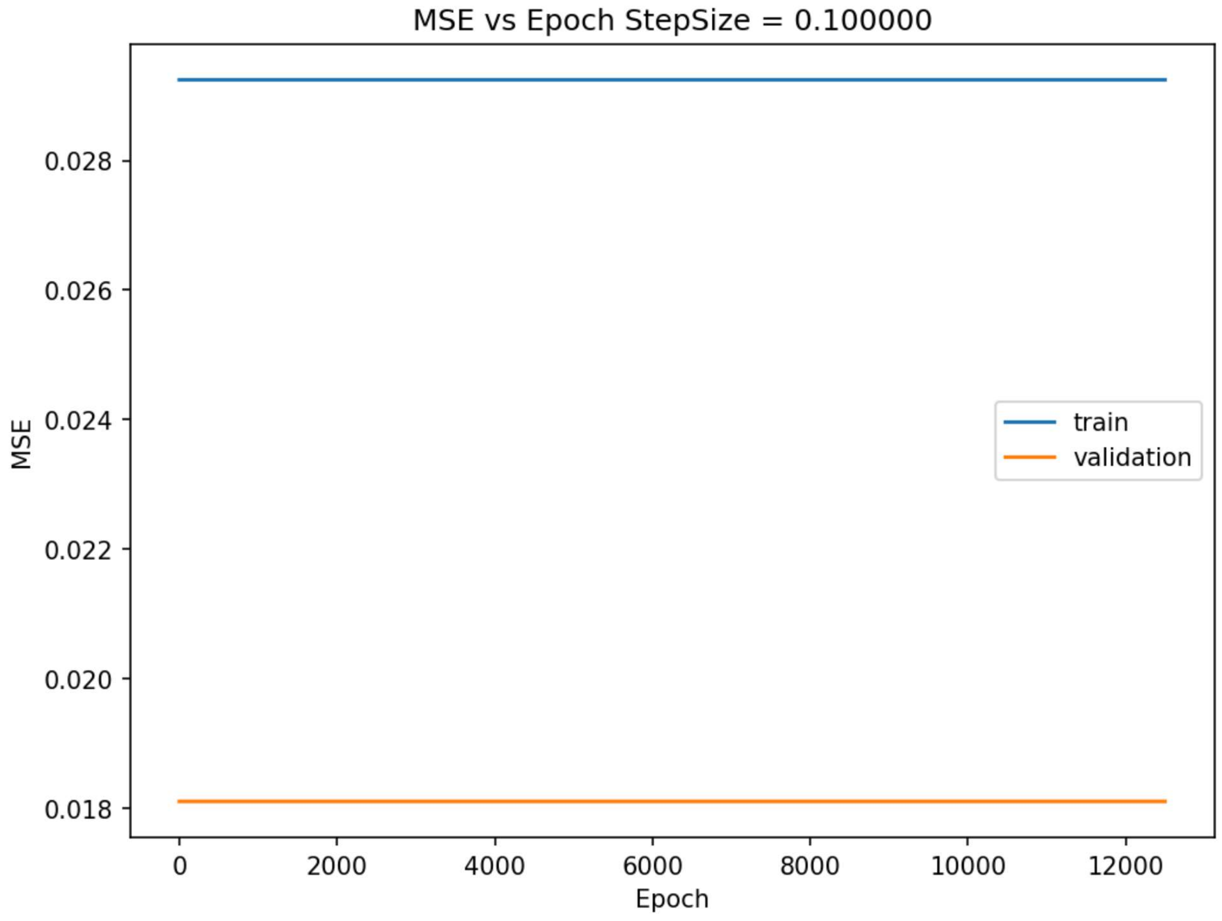

MSE vs Epoch StepSize = 0.000001
train
validation
MSE
Epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
12000


MSE vs Epoch StepSize = 0.000010
train
validation
MSE
Epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
12000


MSE vs Epoch StepSize = 0.000100
train
validation
MSE
Epoch
0.0
0.2
0.4
0.6
0.8
0
2000
4000
6000
8000
10000
12000


MSE vs Epoch StepSize = 0.001000
train
validation
MSE
Epoch
0.0
0.1
0.2
0.3
0.4
0.5
0
2000
4000
6000
8000
10000
12000


MSE vs Epoch StepSize = 0.010000
train
validation
MSE
0.030
0.028
0.026
0.024
0.022
0.020
0.018
Epoch
0
2000
4000
6000
8000
10000
12000


MSE vs Epoch StepSize = 0.100000
train
validation
MSE
0.028
0.026
0.024
0.022
0.020
0.018
Epoch
0
2000
4000
6000
8000
10000
12000

We can see from the above plots that a step size of 0.0001 appears to be the best option for the stochastic gradient descent

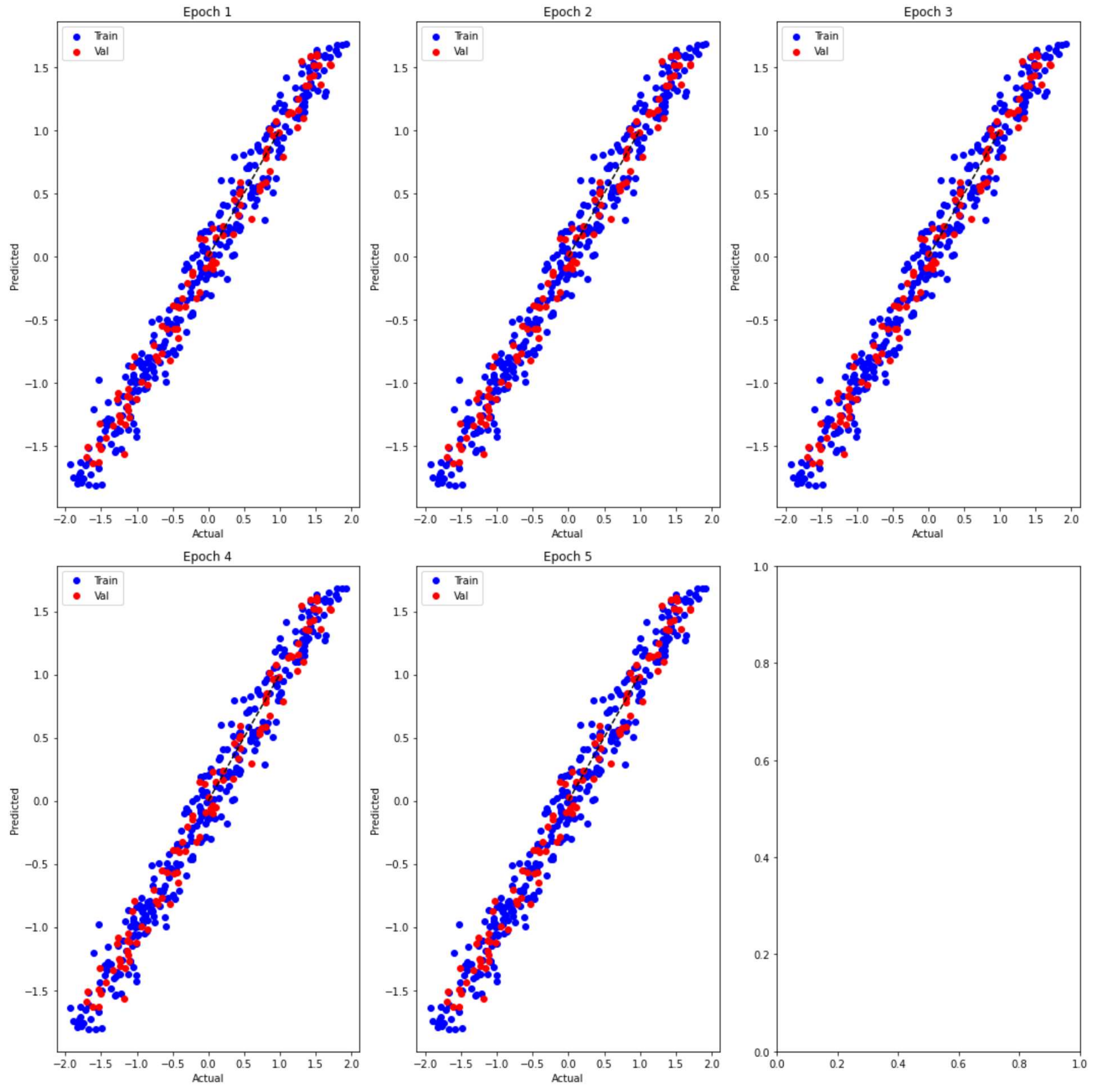
In [14]:
```
model = SGDRegressor(alpha=0.0, learning_rate='constant', eta0=0.0001, random_state=42
mean_squared_error(Y_test, y_val_pred)
```

Out[14]:
```
0.026103544813848734
```

Using the step size found earlier we run the model with our Test data getting an MSE of 0.026

PART C

In [33]:
```
model = SGDRegressor(alpha=0.0, learning_rate='constant', eta0=eta, random_state=42, w
train_mse = []
val_mse = []
train_pred_history = []
val_pred_history = []

for epoch in range(12500):
    model.partial_fit(X_train, Y_train)
    train_preds = model.predict(X_train)
    train_mse.append(mean_squared_error(Y_train, train_preds))
    val_preds = model.predict(X_valid)
    val_mse.append(mean_squared_error(Y_valid, val_preds))
    train_pred_history.append(train_preds)
    val_pred_history.append(val_preds)


fig, axes = plt.subplots(2,3,figsize=(15, 15))
for i in range(5):
    row, col = divmod(i, 3)
    ax = axes[row, col]
    ax.scatter(Y_train, train_pred_history[i], color='b', label='Train')
    ax.scatter(Y_valid, val_pred_history[i], color='r', label='Val')
    ax.plot([0, 1], [0, 1], color='k', linestyle='--')
    ax.set_xlabel('Actual')
    ax.set_ylabel('Predicted')
    ax.set_title(f'Epoch {i+1}')
    ax.legend()
plt.tight_layout()
plt.show()
```



In the code above, we train the model for 12500 epochs and record the predicted values on the training set and the validation set for each epoch. We then plot the regression fit for the first five epochs by scatter plotting the predicted values against the actual values for both the training and validation sets, and adding a diagonal line for reference.

The resulting plot shows how the regression fit evolves during the training process. The first epoch has a relatively poor fit, with a lot of scatter around the diagonal line. As the training progresses, the fit gradually improves, with the scatter decreasing and the points clustering more tightly around the diagonal line. By the fifth epoch, the fit is quite good, with most of the points falling very close to the diagonal line.

Here are five visualizations which show how the regression fit evolves during the training process:

```
In [ ]:
```