In [1]:
```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

In [2]:
```
import cvxopt
from cvxopt import solvers, matrix
```

In [3]:
```
# Set the seed to be reproductive
np.random.seed(8675309)
```

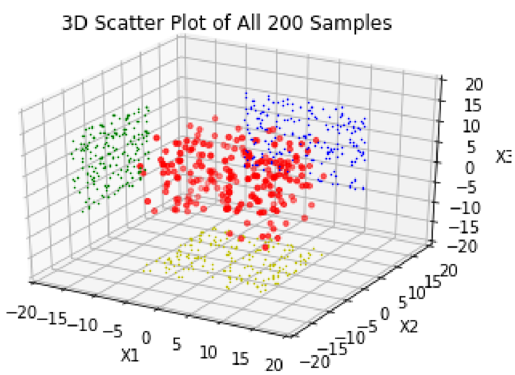
Part1: Generate Training and Testing Data

In [4]:
```
# Initialize the matrix and predefine the size of the input dataset
n = 100
d = 3
syntheticData21 = np.zeros((n,d))
syntheticData22 = np.zeros((n,d))

# Generate 200 random data points varying from -10 to 10
for ii in range(0,d):
    syntheticData21[:,ii] = np.random.uniform(-10,10,n)
    syntheticData22[:,ii] = np.random.uniform(-10,10,n)
```

In [5]:
```
# Assign required values for ground-true weight vector and bias
w = np.array([-0.8,2.1,1.5],ndmin=1)
b = 10
# Assign values for noise which follows required distribution
eps_mean = 0.0
eps_std = np.sqrt(10)
eps = np.random.normal(eps_mean,eps_std,size=(n,))
# Generate ground-true for y_training
X_training = syntheticData21
y_training = X_training.dot(w) + b + eps
# Generate ground-true for y_testing
X_testing = syntheticData22
y_testing = X_testing.dot(w) + b + eps
```

In [6]:

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot both the training and testing data points
ax.scatter(X_training[:,0], X_training[:,1], X_training[:,2], c='r', marker='o',s=10)
ax.scatter(X_testing[:,0], X_testing[:,1], X_testing[:,2], c='r', marker='o',s=10)
# Set labels for each dimension of X
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('X3')
# Plot the projections of X_training into 3 different plans
ax.plot(X_training[:,0], X_training[:,2], 'b*', zdir='y', zs=20,markersize=0.9)
ax.plot(X_training[:,1], X_training[:,2], 'g*', zdir='x', zs=-20,markersize=0.9)
ax.plot(X_training[:,0], X_training[:,1], 'y*', zdir='z', zs=-20,markersize=0.9)
# Plot the projections of X_testing into 3 different plans
ax.plot(X_testing[:,0], X_testing[:,2], 'b*', zdir='y', zs=20,markersize=0.9)
ax.plot(X_testing[:,1], X_testing[:,2], 'g*', zdir='x', zs=-20,markersize=0.9)
ax.plot(X_testing[:,0], X_testing[:,1], 'y*', zdir='z', zs=-20,markersize=0.9)
# Set limits for each axis
ax.set_xlim([-20, 20])
ax.set_ylim([-20, 20])
ax.set_zlim([-20, 20])
# Create title
plt.title('3D Scatter Plot of All 200 Samples')
plt.show()
```



Part 2: Fit and Evaluate a Ridge Regression Model

In [7]:

```
# Initialization
X_training_sd = np.zeros((n,d))
X_testing_sd = np.zeros((n,d))
y_training_centered = np.zeros((n,d))

# Standardization on X
for i in range(0,d):
    X_training_sd[:,i] = (X_training[:,i]-X_training[:,i].mean())/X_training[:,i].std()
    X_testing_sd[:,i] = (X_testing[:,i]-X_testing[:,i].mean())/X_testing[:,i].std()

# Centeralization on yTrain
y_training_centered = y_training-y_training.mean()
```

In [8]:

```
# Define Lambda values
lbd = np.arange(0.0, 10, 0.1)
```

In [9]:

```
# Initialization on weight, yPred, SSE
weight_R = np.zeros((d,len(lbd)))
y_pred_R = np.zeros((n,len(lbd)))
SSE_R = np.zeros((len(lbd),1))
```

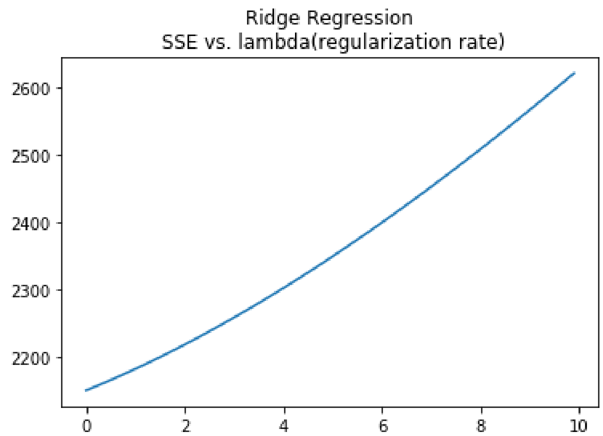
In [10]:

```
# learned bias term from training set
bias = y_training.mean()
print('learned bias term:', bias)
```

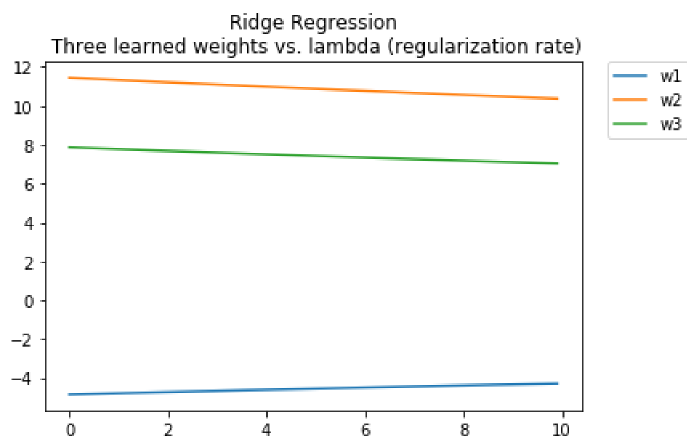
```
('learned bias term:', 13.353027832357101)
```

In [11]:
```
for i,l in enumerate(lbd):
    # Ridge Regression:
    weight_R[:,i] = np.linalg.pinv(X_training_sd.T.dot(X_training_sd)+l*np.identity(d)).dot
    # Evaluate model performance for each Ridge Regression Model
    y_pred_R[:,i] = X_testing_sd.dot(weight_R[:,i])
    SSE_R[i] = sum((y_testing - y_pred_R[:,i] - bias)**2)
```

In [12]:
```
plt.plot(lbd, SSE_R)
plt.title('Ridge Regression\n SSE vs. lambda(regularization rate)')
plt.show()
```



In [13]:
```
plt.plot(lbd, weight_R[0, :], label = 'w1')
plt.plot(lbd, weight_R[1, :], label = 'w2')
plt.plot(lbd, weight_R[2, :], label = 'w3')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title('Ridge Regression\n Three learned weights vs. lambda (regularization rate)')
plt.show()
```



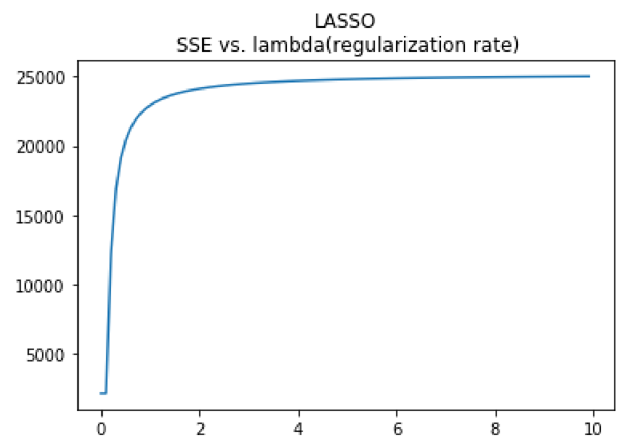
2 required plots for Ridge Regression are shown above.

The value of learned bias is 13.353027832357101

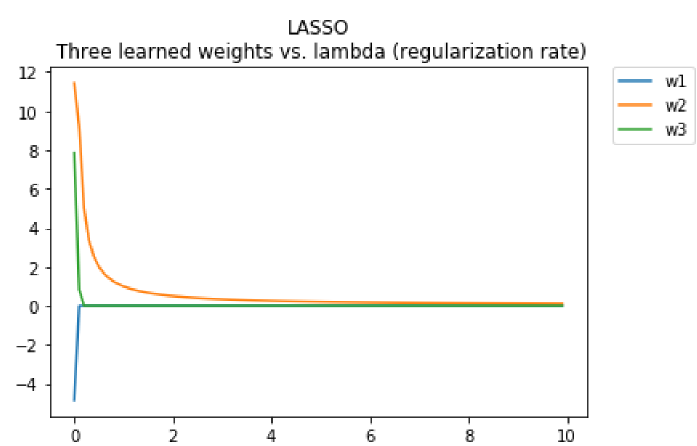
Part 3: Fit and Evaluate a LASSO Regression Model

In [14]:
```
weight_zero = np.linalg.inv(X_training_sd.T.dot(X_training_sd)).dot(X_training_sd.T).dot(y_
t_zero = abs(weight_zero).sum()

weight_L =weight_zero.reshape(d,-1).dot( np.ones((1,len(lbd))))
y_pred_L = np.zeros((n, len(lbd)))
SSE_L = np.zeros((len(lbd),1))
SSE_L[0] = sum((y_testing - X_testing_sd.dot(weight_zero) - bias)**2)
GE = np.array([[]])
H = matrix(np.array(X_training_sd.T.dot(X_training_sd)))
f = matrix((-2*y_training_centered.T.dot(X_training_sd)).reshape(d,-1))

solvers.options['show_progress'] = False

for i,l in enumerate(lbd):
    GE = np.array([[]]).reshape(-1,d)
    if l==0: continue
    t = 1/l
    while abs(weight_L[:,i]).sum() >= t:
        diag_sign = np.diag(np.sign(np.linalg.pinv(np.identity(d)*weight_L[:,i])))
        GE = np.append(GE,diag_sign).reshape(-1,d)
        A = matrix(GE)
        b = matrix(t*np.ones((len(GE),1)))
        weight_L[:,i] = np.array(solvers.qp(H, f, A, b)['x']).reshape(d,)
        y_pred_L[:,i] = X_testing_sd.dot(weight_L[:,i-1])
        SSE_L[i] = sum((y_testing - y_pred_L[:,i] - bias)**2)
```

In [15]:
```
plt.plot(lbd, SSE_L)
plt.title('LASSO\n SSE vs. lambda(regularization rate)')
plt.show()
```



In [16]:
```
plt.plot(lbd, weight_L[0, :], label = 'w1')
plt.plot(lbd, weight_L[1, :], label = 'w2')
plt.plot(lbd, weight_L[2, :], label = 'w3')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title('LASSO\n Three learned weights vs. lambda (regularization rate)')
plt.show()
```



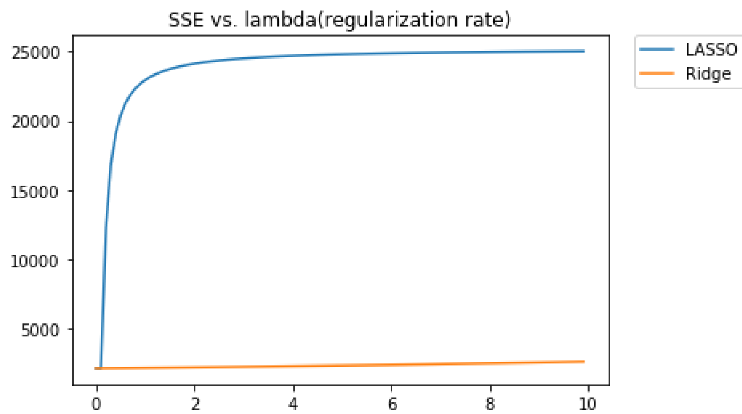
2 required plots for LASSO are shown above.

The value of learned bias is 13.353027832357101

The plot below shows the difference between LASSO and Ridge regression in terms of SSE vs. lambda. It can be observed that when lambda equals to zero, their SSE value is matched since when lambda equals to zero, both LASSO and Ridge regression is converged to least squared linear regression problem.

In [17]:

```
plt.plot(lbd, SSE_L, label = 'LASSO')
plt.plot(lbd, SSE_R, label = 'Ridge')
plt.title('SSE vs. lambda(regularization rate)')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



The plot below shows the difference between LASSO and Ridge regression in terms of three learned weights vs. lambda. It can be observed that when lambda equals to zero, their weights from LASSO and Ridge are matched since when lambda equals to zero, both LASSO and Ridge regression is converged to least squared linear regression problem. Furthermore, the plot shows that the LASSO has a functionality of feature selection. w1 and w3 shrinks to zero to meet the constraint.

In [18]:

```
plt.plot(lbd, weight_R[0, :], label = 'w1--Ridge')
plt.plot(lbd, weight_R[1, :], label = 'w2--Ridge')
plt.plot(lbd, weight_R[2, :], label = 'w3--Ridge')

plt.plot(lbd, weight_L[0, :], label = 'w1--LASSO')
plt.plot(lbd, weight_L[1, :], label = 'w2--LASSO')
plt.plot(lbd, weight_L[2, :], label = 'w3--LASSO')

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title('Three learned weights vs. lambda (regularization rate)')

plt.show()
```