In [1]:
```
""" ECGR 5105 Intro to Machine Learning
    Homework 1
    Phillip Harmon """;
```

In [2]:
```
""" Part 1a : numeric data raw input """
%reset -f
```

In [3]:
```
#Module Inclusions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [4]:
```
#Function to calculate the loss of a model
def get_loss(x, y, theta):
    """x is input data (m x n)
       y is ground truths (m x 1)
       theta is model params (n x 1)"""
    h = x.dot(theta)
    error = np.subtract(h, y)
    sqError = np.square(error)
    sumSqError = np.sum(sqError)
    avgSqError = sumSqError /(2 * len(y))
    return avgSqError
```

In [5]:
```
#Function to run the gradient descent algorithm
def gradient_descent(xT, yT, theta, a, iterations, xV, yV):
    """xT is input training data (m x n)
       yT is training ground truths (m x 1)
       theta is model params (n x 1)
       a is learn rate (scalar)
       iterations. duh. (scalar)
       xV is input vaidatation data
       yV is validation ground truths """
    tLoss = np.zeros(iterations)
    vLoss = np.zeros(iterations)
    for i in range(iterations):
        h = xT.dot(theta)
        error = np.subtract(h, yT)
        delta = xT.transpose().dot(error)
        grad = delta / len(yT)
        theta = theta - (a * grad)
        tLoss[i] = get_loss(xT, yT, theta)
        vLoss[i] = get_loss(xV, yV, theta)
    return theta, tLoss, vLoss
```

In [6]:
```
#Read in the CSV into a dataframe
csvData = pd.read_csv("./Housing.csv")

csvCols = len(csvData.columns)
csvRows = len(csvData)

csvData.head(5)
```

Out[6]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterhea
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

In [7]:
```
#Collect Data
dataLabels = ['price','area','bedrooms','bathrooms','stories','parking']
data = csvData[dataLabels]
data.head(5)
```

Out[7]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

In [8]:
```
#Split T-set and V-set
#and
#Prepare the training set for Gradient Descent
np.random.seed(1337)
frameT, frameV = train_test_split(data, train_size = 0.8, test_size = 0.2)
n = len(frameT.columns)
m = len(frameT)
nV = len(frameV.columns)
mV = len(frameV)
Yv = frameV.pop('price')
Xv = frameV
Y = frameT.pop('price')
X = frameT
print(X.head(5))
Y.head(5)
```

```
      area  bedrooms  bathrooms  stories  parking
536   3420         5          1        2        0
420   4120         2          1        2        0
63    6360         4          2        3        2
465   3800         2          1        1        0
277  10360         2          1        1        1
```

Out[8]:
```
536    1960000
420    3360000
63     7035000
465    3045000
277    4305000
Name: price, dtype: int64
```

In [9]:
```
Yv = Yv.values[:].reshape([mV,1])
Xv = np.hstack((np.ones((mV,1)), Xv.values[:,:].reshape([mV,nV-1])))
Y = Y.values[:].reshape([m,1])
X = np.hstack((np.ones((m,1)), X.values[:,:].reshape([m,n-1])))
print(X[0:5])
Y[0:5]
```

```
[[1.000e+00 3.420e+03 5.000e+00 1.000e+00 2.000e+00 0.000e+00]
 [1.000e+00 4.120e+03 2.000e+00 1.000e+00 2.000e+00 0.000e+00]
 [1.000e+00 6.360e+03 4.000e+00 2.000e+00 3.000e+00 2.000e+00]
 [1.000e+00 3.800e+03 2.000e+00 1.000e+00 1.000e+00 0.000e+00]
 [1.000e+00 1.036e+04 2.000e+00 1.000e+00 1.000e+00 1.000e+00]]
```

Out[9]:
```
array([[1960000],
       [3360000],
       [7035000],
       [3045000],
       [4305000]], dtype=int64)
```

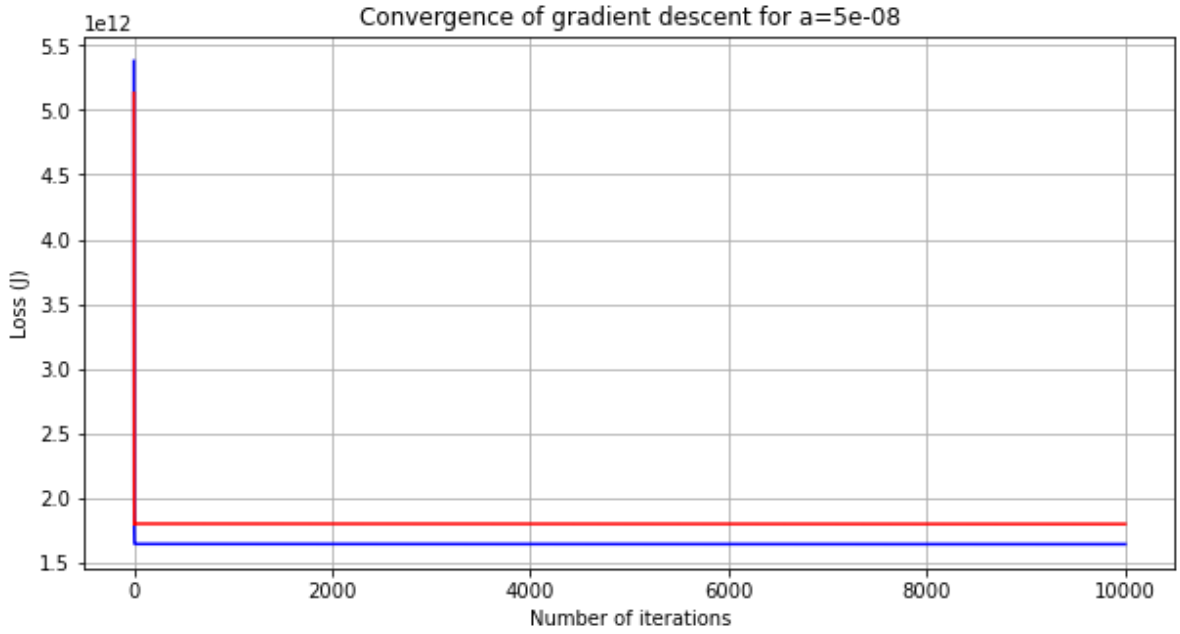
In [10]:
```
#set up gradient descent
LEARN_RATE = 0.00000005
ITERATIONS = 10000
#init theta
theta = np.zeros((n,1))
```

In [11]:
```
#Perform the Gradient Descent on the data
theta, lossT, lossV = gradient_descent(X,Y,theta,LEARN_RATE,ITERATIONS,Xv,Yv)
print("Converged Training Loss J = {:.3f}".format(lossT[-1]))
print("Final Validation Loss J = {:.3f}".format(lossV[-1]))
```

```
Converged Training Loss J = 1640371583531.789
Final Validation Loss J = 1796490856056.696
```

In [12]:
```
#Plot loss over training interval
plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent for a={}'.format(LEARN_RATE))
plt.plot(range(1,ITERATIONS+1),lossT, color='blue')
plt.plot(range(1,ITERATIONS+1),lossV, color='red')
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x1bb59e68790>]
```



In [13]:
```
print("Theta = {}".format(theta))
```

```
Theta = [[177.1673769 ]
 [851.43845112]
 [656.80845063]
 [369.30754091]
 [595.56440216]
 [133.30470252]]
```

In [14]:
```
""" Part 1b : numeric and boolean data with raw input """
%reset -f
```

In [15]:
```
#Module Inclusions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [16]:
```
#Function to calculate the loss of a model
def get_loss(x, y, theta):
    """x is input data (m x n)
       y is ground truths (m x 1)
       theta is model params (n x 1)"""
    h = x.dot(theta)
    error = np.subtract(h, y)
    sqError = np.square(error)
    sumSqError = np.sum(sqError)
    avgSqError = sumSqError /(2 * len(y))
    return avgSqError
```

In [17]:
```
#Function to run the gradient descent algorithm
def gradient_descent(xT, yT, theta, a, iterations, xV, yV):
    """xT is input training data (m x n)
       yT is training ground truths (m x 1)
       theta is model params (n x 1)
       a is learn rate (scalar)
       iterations. duh. (scalar)
       xV is input vaidatation data
       yV is validation ground truths """
    tLoss = np.zeros(iterations)
    vLoss = np.zeros(iterations)
    for i in range(iterations):
        h = xT.dot(theta)
        error = np.subtract(h, yT)
        delta = xT.transpose().dot(error)
        grad = delta / len(yT)
        theta = theta - (a * grad)
        tLoss[i] = get_loss(xT, yT, theta)
        vLoss[i] = get_loss(xV, yV, theta)
    return theta, tLoss, vLoss
```

In [18]:
```
#Function to convert Y/N booleans into 1/0 booleans
def yn_bool_convert(val):
    return val.map({ "yes" : 1 , "no" : 0 })
```

In [19]:
```
#Read in the CSV into a dataframe
csvData = pd.read_csv("./Housing.csv")

csvCols = len(csvData.columns)
csvRows = len(csvData)

csvData.head(5)
```

Out[19]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterhea
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

In [20]:
```
#Collect numeric data
dataLabels = ['price','area','bedrooms','bathrooms','stories','parking']
data = csvData[dataLabels]
data
```

Out[20]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2
...	...	...	...	...	...	...
540	1820000	3000	2	1	1	2
541	1767150	2400	3	1	1	0
542	1750000	3620	2	1	1	0
543	1750000	2910	3	1	1	0
544	1750000	3850	3	1	2	0

545 rows × 6 columns

In [21]:
```
#Collect boolean data
boolLabels = ['mainroad','guestroom','basement','hotwaterheating','airconditio
boolData = csvData[boolLabels]
boolData
```

Out[21]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea
0	yes	no	no	no	yes	yes
1	yes	no	no	no	yes	no
2	yes	no	yes	no	no	yes
3	yes	no	yes	no	yes	yes
4	yes	yes	yes	no	yes	no
...	...	...	...	...	...	...
540	yes	no	yes	no	no	no
541	no	no	no	no	no	no
542	yes	no	no	no	no	no
543	no	no	no	no	no	no
544	yes	no	no	no	no	no

545 rows × 6 columns

In [22]:
```
#Convert Y/N booleans to numerical booleans
boolData = boolData.apply(yn_bool_convert)
boolData
```

Out[22]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea
0	1	0	0	0	1	1
1	1	0	0	0	1	0
2	1	0	1	0	0	1
3	1	0	1	0	1	1
4	1	1	1	0	1	0
...	...	...	...	...	...	...
540	1	0	1	0	0	0
541	0	0	0	0	0	0
542	1	0	0	0	0	0
543	0	0	0	0	0	0
544	1	0	0	0	0	0

545 rows × 6 columns

In [23]:
```
#Join datasets
newdata = pd.concat([data,boolData],axis=1,join='outer')
newdata
```

Out[23]:

	price	area	bedrooms	bathrooms	stories	parking	mainroad	guestroom	basement
0	13300000	7420	4	2	3	2	1	0	0
1	12250000	8960	4	4	4	3	1	0	0
2	12250000	9960	3	2	2	2	1	0	1
3	12215000	7500	4	2	2	3	1	0	1
4	11410000	7420	4	1	2	2	1	1	1
...	...	...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	2	1	0	1
541	1767150	2400	3	1	1	0	0	0	0
542	1750000	3620	2	1	1	0	1	0	0
543	1750000	2910	3	1	1	0	0	0	0
544	1750000	3850	3	1	2	0	1	0	0

545 rows × 12 columns

In [24]:
```
#Split T-set and V-set
#and
#Prepare the training set for Gradient Descent
np.random.seed(1337)
frameT, frameV = train_test_split(newdata, train_size = 0.8, test_size = 0.2)
n = len(frameT.columns)
m = len(frameT)
nV = len(frameV.columns)
mV = len(frameV)
Yv = frameV.pop('price')
Xv = frameV
Y = frameT.pop('price')
X = frameT
print(Y.head(5))
X.head(5)
```

```
536    1960000
420    3360000
63     7035000
465    3045000
277    4305000
Name: price, dtype: int64
```

Out[24]:

	area	bedrooms	bathrooms	stories	parking	mainroad	guestroom	basement	hotwaterh
536	3420	5	1	2	0	0	0	0	
420	4120	2	1	2	0	1	0	0	
63	6360	4	2	3	2	1	0	0	
465	3800	2	1	1	0	1	0	0	
277	10360	2	1	1	1	1	0	0	

In [25]:
```
Yv = Yv.values[:].reshape([mV,1])
Xv = np.hstack((np.ones((mV,1)), Xv.values[:,:].reshape([mV,nV-1])))
Y = Y.values[:].reshape([m,1])
X = np.hstack((np.ones((m,1)), X.values[:,:].reshape([m,n-1])))
print(X[0:5])
Y[0:5]
```

```
[[1.000e+00 3.420e+03 5.000e+00 1.000e+00 2.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00]
 [1.000e+00 4.120e+03 2.000e+00 1.000e+00 2.000e+00 0.000e+00 1.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00]
 [1.000e+00 6.360e+03 4.000e+00 2.000e+00 3.000e+00 2.000e+00 1.000e+00
  0.000e+00 0.000e+00 0.000e+00 1.000e+00 1.000e+00 0.000e+00]
 [1.000e+00 3.800e+03 2.000e+00 1.000e+00 1.000e+00 0.000e+00 1.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00]
 [1.000e+00 1.036e+04 2.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00]]
```

Out[25]:
```
array([[1960000],
       [3360000],
       [7035000],
       [3045000],
       [4305000]], dtype=int64)
```

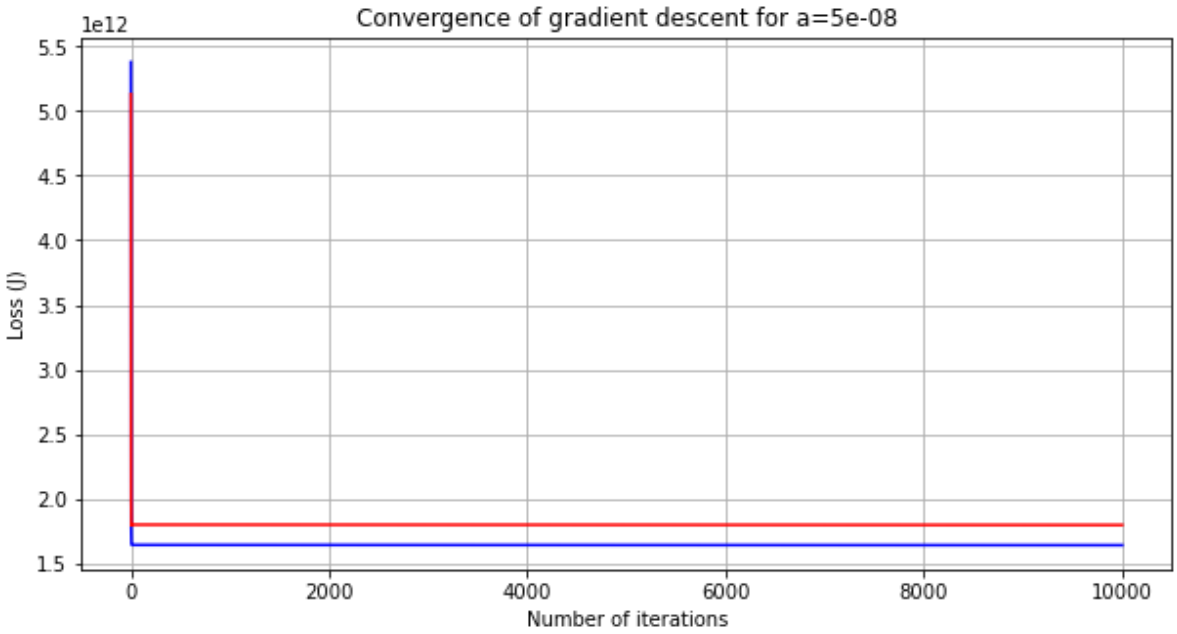
In [26]:
```
#set up gradient descent
LEARN_RATE = 0.00000005
ITERATIONS = 10000
#init theta
theta = np.zeros((n,1))
```

In [27]:
```
#Perform the Gradient Descent on the data
theta, lossT, lossV = gradient_descent(X,Y,theta,LEARN_RATE,ITERATIONS,Xv,Yv)
print("Converged Training Loss J = {:.3f}".format(lossT[-1]))
print("Final Validation Loss J = {:.3f}".format(lossV[-1]))
```

```
Converged Training Loss J = 1640219523234.121
Final Validation Loss J = 1796331915056.757
```

In [28]:
```
#Plot loss over training interval
plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent for a={}'.format(LEARN_RATE))
plt.plot(range(1,ITERATIONS+1),lossT, color='blue')
plt.plot(range(1,ITERATIONS+1),lossV, color='red')
```

Out[28]:
```
[<matplotlib.lines.Line2D at 0x1bb59deceb0>]
```



In [29]:
```
print("Theta = {}".format(theta))
```

```
Theta = [[177.16196901]
 [851.39254663]
 [656.79149008]
 [369.29942061]
 [595.55147017]
 [133.30172726]
 [157.29239896]
 [ 74.84861923]
 [112.37594565]
 [ 17.07433362]
 [152.94483886]
 [ 97.26423902]]
```

In [30]:
```
""" Part 2a : numeric data with standardization and normalization """
%reset -f
```

In [31]:
```
#Module Inclusions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn_pandas import DataFrameMapper
```

In [32]:
```
#Function to calculate the loss of a model
def get_loss(x, y, theta):
    """x is input data (m x n)
       y is ground truths (m x 1)
       theta is model params (n x 1)"""
    h = x.dot(theta)
    error = np.subtract(h, y)
    sqError = np.square(error)
    sumSqError = np.sum(sqError)
    avgSqError = sumSqError /(2 * len(y))
    return avgSqError
```

In [33]:
```
#Function to run the gradient descent algorithm
def gradient_descent(xT, yT, theta, a, iterations, xV, yV):
    """xT is input training data (m x n)
       yT is training ground truths (m x 1)
       theta is model params (n x 1)
       a is learn rate (scalar)
       iterations. duh. (scalar)
       xV is input vaidatation data
       yV is validation ground truths """
    tLoss = np.zeros(iterations)
    vLoss = np.zeros(iterations)
    for i in range(iterations):
        h = xT.dot(theta)
        error = np.subtract(h, yT)
        delta = xT.transpose().dot(error)
        grad = delta / len(yT)
        theta = theta - (a * grad)
        tLoss[i] = get_loss(xT, yT, theta)
        vLoss[i] = get_loss(xV, yV, theta)
    return theta, tLoss, vLoss
```

In [34]:
```
#Read in the CSV into a dataframe
csvData = pd.read_csv("./Housing.csv")

csvCols = len(csvData.columns)
csvRows = len(csvData)

csvData.head(5)
```

Out[34]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterhea
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

In [35]:
```
#Collect Data
dataLabels = ['price','area','bedrooms','bathrooms','stories','parking']
data = csvData[dataLabels]
data.head(5)
```

Out[35]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

In [36]:
```
#Split T-set and V-set
np.random.seed(1337)
frameT, frameV = train_test_split(data, train_size = 0.8, test_size = 0.2)
n = len(frameT.columns)
m = len(frameT)
mV = len(frameV)
frameT.head(5)
```

Out[36]:

	price	area	bedrooms	bathrooms	stories	parking
536	1960000	3420	5	1	2	0
420	3360000	4120	2	1	2	0
63	7035000	6360	4	2	3	2
465	3045000	3800	2	1	1	0
277	4305000	10360	2	1	1	1

In [37]:
```
#Preprocessing
TNScaler = DataFrameMapper([(frameT.columns,MinMaxScaler())])
frameTN = TNScaler.fit_transform(frameT,n)
frameTN = pd.DataFrame(frameTN, index=frameT.index, columns=frameT.columns)
VNScaler = DataFrameMapper([(frameV.columns,MinMaxScaler())])
frameVN = VNScaler.fit_transform(frameV,n)
frameVN = pd.DataFrame(frameVN, index=frameV.index, columns=frameV.columns)
TSScaler = DataFrameMapper([(frameT.columns,StandardScaler())])
frameTS = TSScaler.fit_transform(frameT,n)
frameTS = pd.DataFrame(frameTS, index=frameT.index, columns=frameT.columns)
VSScaler = DataFrameMapper([(frameV.columns,StandardScaler())])
frameVS = VSScaler.fit_transform(frameV,n)
frameVS = pd.DataFrame(frameVS, index=frameV.index, columns=frameV.columns)
print("Normalied Training Set:")
print(frameTN.head(5))
print("Normalized Validation Set:")
print(frameVN.head(5))
print("Standardized Training Set:")
print(frameTS.head(5))
print("Standardized Validation Set:")
print(frameVS.head(5))
```

```
Normalied Training Set:
        price      area  bedrooms  bathrooms   stories   parking
536  0.018182  0.118621       0.8   0.000000  0.333333  0.000000
420  0.139394  0.166897       0.2   0.000000  0.333333  0.000000
63   0.457576  0.321379       0.6   0.333333  0.666667  0.666667
465  0.112121  0.144828       0.2   0.000000  0.000000  0.000000
277  0.221212  0.597241       0.2   0.000000  0.000000  0.333333
Normalized Validation Set:
        price      area  bedrooms  bathrooms   stories   parking
326  0.214047  0.216929       0.6        0.0  0.000000  0.000000
449  0.133779  0.000000       0.4        0.0  0.333333  0.000000
224  0.287625  0.760581       0.2        0.0  0.000000  0.666667
140  0.381271  0.367452       0.4        0.5  1.000000  0.000000
13   0.715719  0.163804       0.6        0.5  0.333333  0.666667
Standardized Training Set:
        price      area  bedrooms  bathrooms   stories   parking
536 -1.487025 -0.795113  2.857902  -0.555867  0.236492 -0.810745
420 -0.741523 -0.475576 -1.304833  -0.555867  0.236492 -0.810745
63   1.215421  0.546940  1.470324   1.414524  1.395039  1.553702
465 -0.909261 -0.621650 -1.304833  -0.555867 -0.922054 -0.810745
277 -0.238309  2.372862 -1.304833  -0.555867 -0.922054  0.371478
Standardized Validation Set:
        price      area  bedrooms  bathrooms   stories  parking
326 -0.455403 -0.484521  1.179999  -0.632049 -0.959479    -0.79
449 -0.914334 -1.665249 -0.080980  -0.632049  0.177295    -0.79
224 -0.034715  2.474528 -1.341960  -0.632049 -0.959479     1.39
140  0.500705  0.334760 -0.080980   1.455629  2.450844    -0.79
13   2.412920 -0.773679  1.179999   1.455629  0.177295     1.39
```

In [38]:
```
#Break data into X,Y sets
YvN = frameVN.pop('price')
XvN = frameVN
YN = frameTN.pop('price')
XN = frameTN
YvS = frameVS.pop('price')
XvS = frameVS
YS = frameTS.pop('price')
XS = frameTS
```

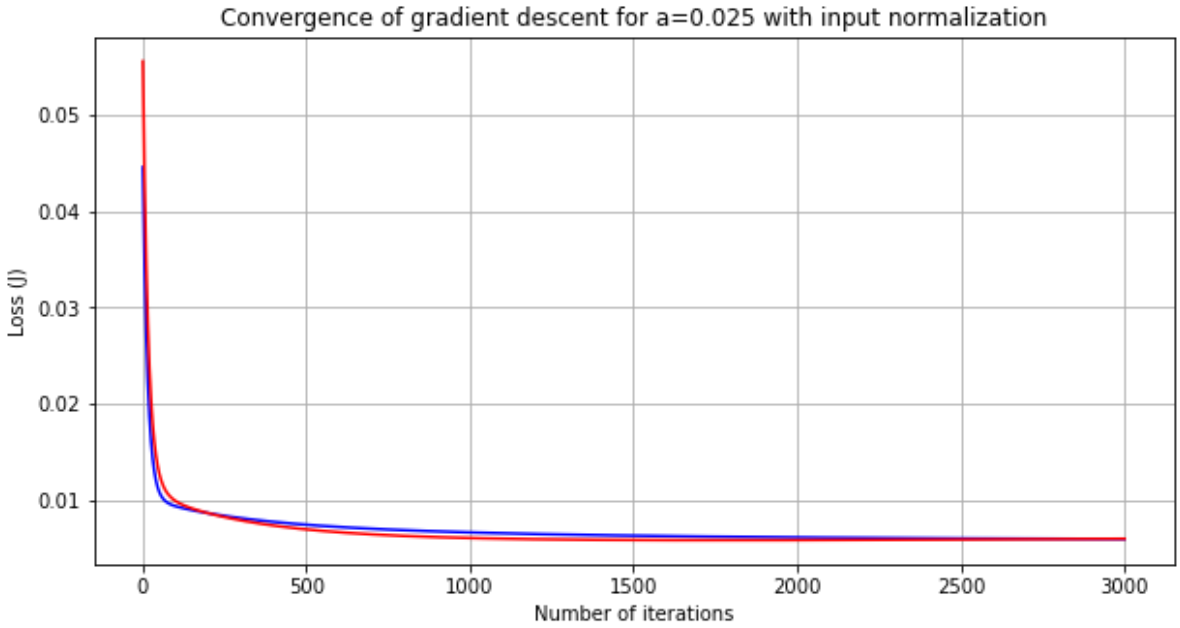
In [39]:
```
#Format Data For Gradient Descent
YvN = YvN.values[:].reshape([mV,1])
XvN = np.hstack((np.ones((mV,1)), XvN.values[:,:].reshape([mV,n-1])))
YN = YN.values[:].reshape([m,1])
XN = np.hstack((np.ones((m,1)), XN.values[:,:].reshape([m,n-1])))
YvS = YvS.values[:].reshape([mV,1])
XvS = np.hstack((np.ones((mV,1)), XvS.values[:,:].reshape([mV,n-1])))
YS = YS.values[:].reshape([m,1])
XS = np.hstack((np.ones((m,1)), XS.values[:,:].reshape([m,n-1])))
```

In [40]:
```
#set up gradient descent
LEARN_RATE_N = 0.025
ITERATIONS_N = 3000
LEARN_RATE_S = 0.025
ITERATIONS_S = 400
#init theta
theta = np.zeros((n,1))
```

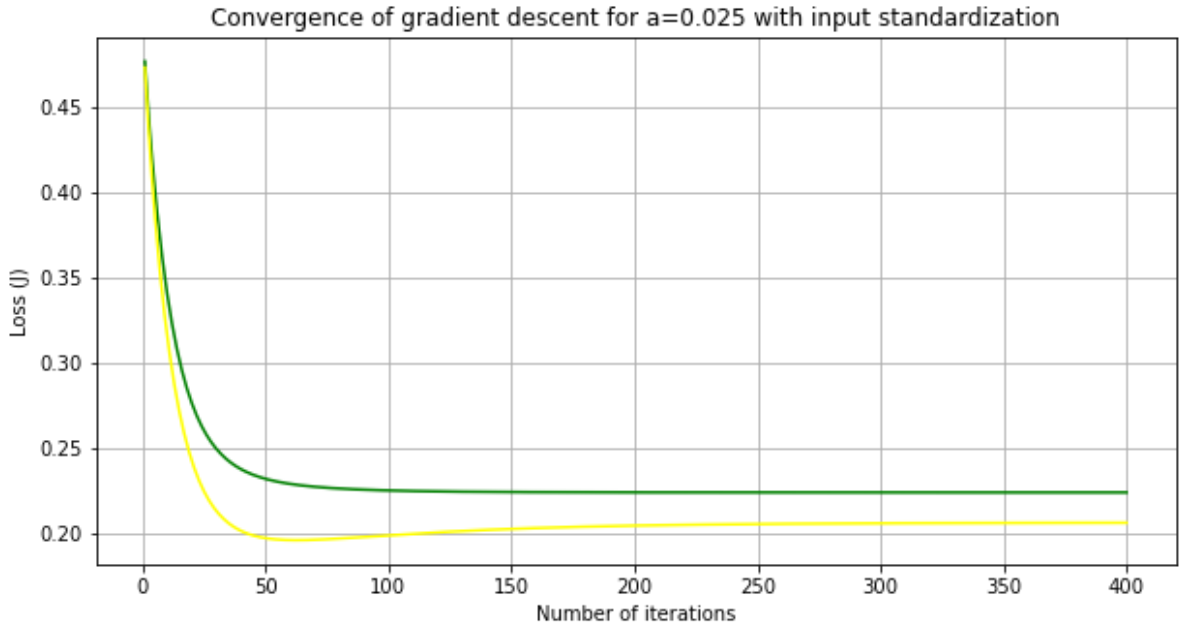
In [41]:
```
#Perform the Gradient Descent on the data
thetaN, lossTN, lossVN = gradient_descent(XN,YN,theta,LEARN_RATE_N,ITERATIONS_
print("Converged Training Loss J for normalized set = {:.3f}".format(lossTN[-1
print("Final Validation Loss J for normalized set = {:.3f}".format(lossVN[-1])

thetaS, lossTS, lossVS = gradient_descent(XS,YS,theta,LEARN_RATE_S,ITERATIONS_
print("Converged Training Loss J for standardized set = {:.3f}".format(lossTS[
print("Final Validation Loss J for standardized set = {:.3f}".format(lossVS[-1
```

```
Converged Training Loss J for normalized set = 0.006
Final Validation Loss J for normalized set = 0.006
Converged Training Loss J for standardized set = 0.224
Final Validation Loss J for standardized set = 0.206
```

In [42]:
```
#Plot loss over training interval with Input Normalization
plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent for a={} with input normalization'.
plt.plot(range(1,ITERATIONS_N+1),lossTN, color='blue')
plt.plot(range(1,ITERATIONS_N+1),lossVN, color='red');
```



In [43]:
```
#Plot loss over training interval
plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent for a={} with input standardization
plt.plot(range(1,ITERATIONS_S+1),lossTS, color='green')
plt.plot(range(1,ITERATIONS_S+1),lossVS, color='yellow');
```



In [44]:
```
print("Normalized Theta = {}".format(thetaN))
print("Standardized Theta = {}".format(thetaS))
```

```
Normalized Theta = [[0.05832319]
 [0.35839946]
 [0.06342379]
 [0.25440021]
 [0.15871885]
 [0.11216531]]
Standardized Theta = [[4.78974658e-17]
 [4.14055189e-01]
 [3.33386959e-02]
 [2.90975856e-01]
 [2.75804041e-01]
 [1.57998917e-01]]
```

In [45]:
```
""" Part 2b : numeric and boolean data with normalization and standardization
%reset -f
```

In [46]:
```
#Module Inclusions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn_pandas import DataFrameMapper
```

In [47]:
```
#Function to calculate the loss of a model
def get_loss(x, y, theta):
    """x is input data (m x n)
       y is ground truths (m x 1)
       theta is model params (n x 1)"""
    h = x.dot(theta)
    error = np.subtract(h, y)
    sqError = np.square(error)
    sumSqError = np.sum(sqError)
    avgSqError = sumSqError /(2 * len(y))
    return avgSqError
```

In [48]:
```
#Function to run the gradient descent algorithm
def gradient_descent(xT, yT, theta, a, iterations, xV, yV):
    """xT is input training data (m x n)
       yT is training ground truths (m x 1)
       theta is model params (n x 1)
       a is learn rate (scalar)
       iterations. duh. (scalar)
       xV is input vaidatation data
       yV is validation ground truths """
    tLoss = np.zeros(iterations)
    vLoss = np.zeros(iterations)
    for i in range(iterations):
        h = xT.dot(theta)
        error = np.subtract(h, yT)
        delta = xT.transpose().dot(error)
        grad = delta / len(yT)
        theta = theta - (a * grad)
        tLoss[i] = get_loss(xT, yT, theta)
        vLoss[i] = get_loss(xV, yV, theta)
    return theta, tLoss, vLoss
```

In [49]:
```
#Function to convert Y/N booleans into 1/0 booleans
def yn_bool_convert(val):
    return val.map({ "yes" : 1 , "no" : 0 })
```

In [50]:
```
#Read in the CSV into a dataframe
csvData = pd.read_csv("./Housing.csv")

csvCols = len(csvData.columns)
csvRows = len(csvData)

csvData.head(5)
```

Out[50]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterhea
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

In [51]:
```
#Collect numeric data
dataLabels = ['price','area','bedrooms','bathrooms','stories','parking']
data = csvData[dataLabels]
data
```

Out[51]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2
...	...	...	...	...	...	...
540	1820000	3000	2	1	1	2
541	1767150	2400	3	1	1	0
542	1750000	3620	2	1	1	0
543	1750000	2910	3	1	1	0
544	1750000	3850	3	1	2	0

545 rows × 6 columns

In [52]:
```
#Collect boolean data
boolLabels = ['mainroad','guestroom','basement','hotwaterheating','airconditio
boolData = csvData[boolLabels]
boolData
```

Out[52]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea
0	yes	no	no	no	yes	yes
1	yes	no	no	no	yes	no
2	yes	no	yes	no	no	yes
3	yes	no	yes	no	yes	yes
4	yes	yes	yes	no	yes	no
...	...	...	...	...	...	...
540	yes	no	yes	no	no	no
541	no	no	no	no	no	no
542	yes	no	no	no	no	no
543	no	no	no	no	no	no
544	yes	no	no	no	no	no

545 rows × 6 columns

In [53]:
```
#Convert Y/N booleans to numerical booleans
boolData = boolData.apply(yn_bool_convert)
boolData
```

Out[53]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea
0	1	0	0	0	1	1
1	1	0	0	0	1	0
2	1	0	1	0	0	1
3	1	0	1	0	1	1
4	1	1	1	0	1	0
...	...	...	...	...	...	...
540	1	0	1	0	0	0
541	0	0	0	0	0	0
542	1	0	0	0	0	0
543	0	0	0	0	0	0
544	1	0	0	0	0	0

545 rows × 6 columns

In [54]:
```
#Join datasets
newdata = pd.concat([data,boolData],axis=1,join='outer')
newdata
```

Out[54]:

	price	area	bedrooms	bathrooms	stories	parking	mainroad	guestroom	basement
0	13300000	7420	4	2	3	2	1	0	0
1	12250000	8960	4	4	4	3	1	0	0
2	12250000	9960	3	2	2	2	1	0	1
3	12215000	7500	4	2	2	3	1	0	1
4	11410000	7420	4	1	2	2	1	1	1
...	...	...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	2	1	0	1
541	1767150	2400	3	1	1	0	0	0	0
542	1750000	3620	2	1	1	0	1	0	0
543	1750000	2910	3	1	1	0	0	0	0
544	1750000	3850	3	1	2	0	1	0	0

545 rows × 12 columns

In [55]:
```
#Split T-set and V-set
np.random.seed(1337)
frameT, frameV = train_test_split(newdata, train_size = 0.8, test_size = 0.2)
n = len(frameT.columns)
m = len(frameT)
mV = len(frameV)
frameT.head(5)
```

Out[55]:

	price	area	bedrooms	bathrooms	stories	parking	mainroad	guestroom	basement
536	1960000	3420	5	1	2	0	0	0	0
420	3360000	4120	2	1	2	0	1	0	0
63	7035000	6360	4	2	3	2	1	0	0
465	3045000	3800	2	1	1	0	1	0	0
277	4305000	10360	2	1	1	1	1	0	0

In [56]:
```
#Preprocessing
TNScaler = DataFrameMapper([(frameT.columns,MinMaxScaler())])
frameTN = TNScaler.fit_transform(frameT,n)
frameTN = pd.DataFrame(frameTN, index=frameT.index, columns=frameT.columns)
VNScaler = DataFrameMapper([(frameV.columns,MinMaxScaler())])
frameVN = VNScaler.fit_transform(frameV,n)
frameVN = pd.DataFrame(frameVN, index=frameV.index, columns=frameV.columns)
TSScaler = DataFrameMapper([(frameT.columns,StandardScaler())])
frameTS = TSScaler.fit_transform(frameT,n)
frameTS = pd.DataFrame(frameTS, index=frameT.index, columns=frameT.columns)
VSScaler = DataFrameMapper([(frameV.columns,StandardScaler())])
frameVS = VSScaler.fit_transform(frameV,n)
frameVS = pd.DataFrame(frameVS, index=frameV.index, columns=frameV.columns)
print("Normalied Training Set:")
print(frameTN.head(5))
print("Normalized Validation Set:")
print(frameVN.head(5))
print("Standardized Training Set:")
print(frameTS.head(5))
print("Standardized Validation Set:")
print(frameVS.head(5))
```

```
Normalied Training Set:
        price      area  bedrooms  bathrooms   stories   parking  mainroad  \
536  0.018182  0.118621       0.8   0.000000  0.333333  0.000000       0.0
420  0.139394  0.166897       0.2   0.000000  0.333333  0.000000       1.0
63   0.457576  0.321379       0.6   0.333333  0.666667  0.666667       1.0
465  0.112121  0.144828       0.2   0.000000  0.000000  0.000000       1.0
277  0.221212  0.597241       0.2   0.000000  0.000000  0.333333       1.0

     guestroom  basement  hotwaterheating  airconditioning  prefarea
536        0.0       0.0              0.0              0.0       0.0
420        0.0       0.0              0.0              0.0       0.0
63         0.0       0.0              0.0              1.0       1.0
465        0.0       0.0              0.0              0.0       0.0
277        0.0       0.0              0.0              0.0       1.0
Normalized Validation Set:
        price      area  bedrooms  bathrooms   stories   parking  mainroad  \
326  0.214047  0.216929       0.6        0.0  0.000000  0.000000       0.0
449  0.133779  0.000000       0.4        0.0  0.333333  0.000000       0.0
224  0.287625  0.760581       0.2        0.0  0.000000  0.666667       1.0
140  0.381271  0.367452       0.4        0.5  1.000000  0.000000       1.0
13   0.715719  0.163804       0.6        0.5  0.333333  0.666667       1.0

     guestroom  basement  hotwaterheating  airconditioning  prefarea
326        0.0       1.0              0.0              0.0       0.0
449        0.0       1.0              0.0              0.0       0.0
224        0.0       0.0              0.0              1.0       1.0
140        0.0       0.0              0.0              1.0       0.0
13         0.0       0.0              1.0              0.0       0.0
Standardized Training Set:
        price      area  bedrooms  bathrooms   stories   parking  mainroad  \
536 -1.487025 -0.795113  2.857902  -0.555867  0.236492 -0.810745 -2.410913
420 -0.741523 -0.475576 -1.304833  -0.555867  0.236492 -0.810745  0.414781
63   1.215421  0.546940  1.470324   1.414524  1.395039  1.553702  0.414781
465 -0.909261 -0.621650 -1.304833  -0.555867 -0.922054 -0.810745  0.414781
```

```
277 -0.238309  2.372862 -1.304833  -0.555867 -0.922054  0.371478  0.414781

     guestroom  basement  hotwaterheating  airconditioning  prefarea
536  -0.474045  -0.72053        -0.219265        -0.684115 -0.552620
420  -0.474045  -0.72053        -0.219265        -0.684115 -0.552620
63   -0.474045  -0.72053        -0.219265         1.461742  1.809561
465  -0.474045  -0.72053        -0.219265        -0.684115 -0.552620
277  -0.474045  -0.72053        -0.219265        -0.684115  1.809561
Standardized Validation Set:
        price      area  bedrooms  bathrooms   stories  parking  mainroad  \
326 -0.455403 -0.484521  1.179999  -0.632049 -0.959479    -0.79 -2.717465
449 -0.914334 -1.665249 -0.080980  -0.632049  0.177295    -0.79 -2.717465
224 -0.034715  2.474528 -1.341960  -0.632049 -0.959479     1.39  0.367990
140  0.500705  0.334760 -0.080980   1.455629  2.450844    -0.79  0.367990
13   2.412920 -0.773679  1.179999   1.455629  0.177295     1.39  0.367990

     guestroom  basement  hotwaterheating  airconditioning  prefarea
326  -0.429863  1.263027        -0.219265        -0.658947 -0.559690
449  -0.429863  1.263027        -0.219265        -0.658947 -0.559690
224  -0.429863 -0.791748        -0.219265         1.517574  1.786703
140  -0.429863 -0.791748        -0.219265         1.517574 -0.559690
13   -0.429863 -0.791748         4.560702        -0.658947 -0.559690
```

In [57]:
```
#Break data into X,Y sets
YvN = frameVN.pop('price')
XvN = frameVN
YN = frameTN.pop('price')
XN = frameTN
YvS = frameVS.pop('price')
XvS = frameVS
YS = frameTS.pop('price')
XS = frameTS
```

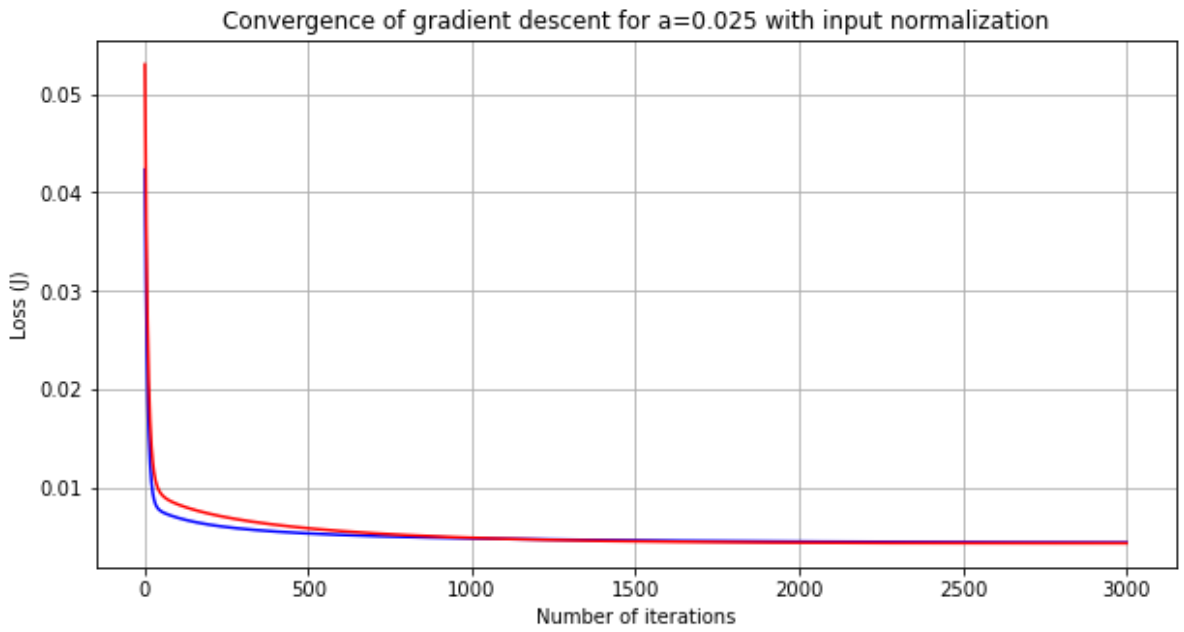
In [58]:
```
#Format Data For Gradient Descent
YvN = YvN.values[:].reshape([mV,1])
XvN = np.hstack((np.ones((mV,1)), XvN.values[:,:].reshape([mV,n-1])))
YN = YN.values[:].reshape([m,1])
XN = np.hstack((np.ones((m,1)), XN.values[:,:].reshape([m,n-1])))
YvS = YvS.values[:].reshape([mV,1])
XvS = np.hstack((np.ones((mV,1)), XvS.values[:,:].reshape([mV,n-1])))
YS = YS.values[:].reshape([m,1])
XS = np.hstack((np.ones((m,1)), XS.values[:,:].reshape([m,n-1])))
```

In [59]:
```
#set up gradient descent
LEARN_RATE_N = 0.025
ITERATIONS_N = 3000
LEARN_RATE_S = 0.025
ITERATIONS_S = 400
#init theta
theta = np.zeros((n,1))
```

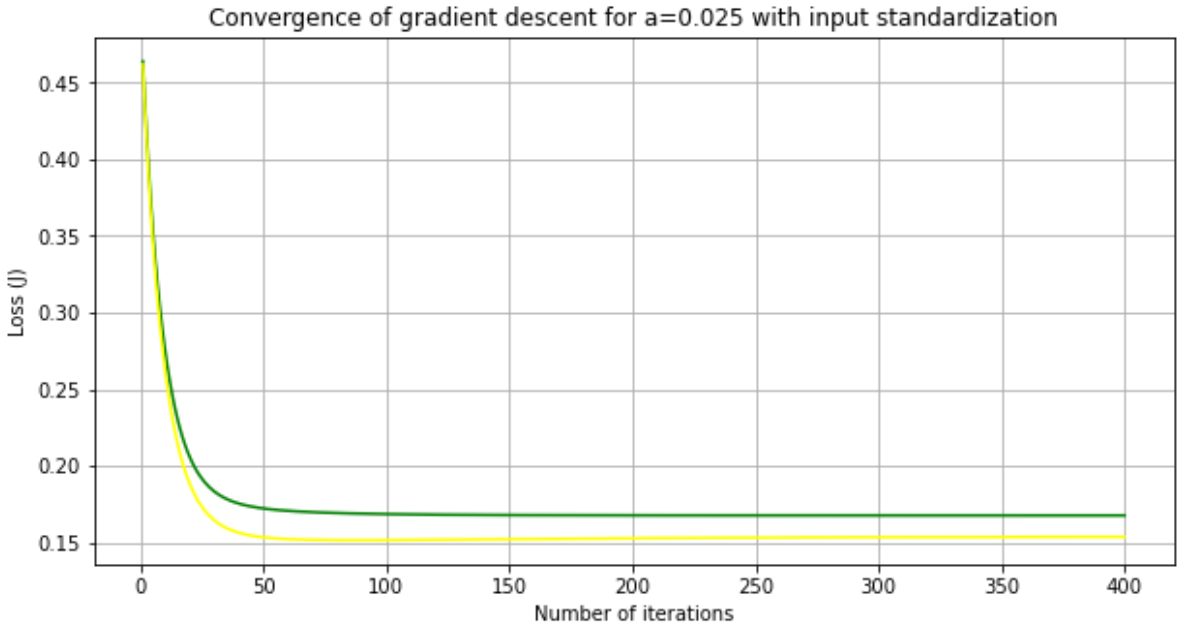
In [60]:
```
#Perform the Gradient Descent on the data
thetaN, lossTN, lossVN = gradient_descent(XN,YN,theta,LEARN_RATE_N,ITERATIONS_
print("Converged Training Loss J for normalized set = {:.3f}".format(lossTN[-1
print("Final Validation Loss J for normalized set = {:.3f}".format(lossVN[-1])

thetaS, lossTS, lossVS = gradient_descent(XS,YS,theta,LEARN_RATE_S,ITERATIONS_
print("Converged Training Loss J for standardized set = {:.3f}".format(lossTS[
print("Final Validation Loss J for standardized set = {:.3f}".format(lossVS[-1
```

```
Converged Training Loss J for normalized set = 0.004
Final Validation Loss J for normalized set = 0.004
Converged Training Loss J for standardized set = 0.168
Final Validation Loss J for standardized set = 0.154
```

In [61]:
```
#Plot loss over training interval with Input Normalization
plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent for a={} with input normalization'.
plt.plot(range(1,ITERATIONS_N+1),lossTN, color='blue')
plt.plot(range(1,ITERATIONS_N+1),lossVN, color='red');
```



In [62]:
```
#Plot loss over training interval
plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent for a={} with input standardization
plt.plot(range(1,ITERATIONS_S+1),lossTS, color='green')
plt.plot(range(1,ITERATIONS_S+1),lossVS, color='yellow');
```



In [63]:
```
print("Normalized Theta = {}".format(thetaN))
print("Standardized Theta = {}".format(thetaS))
```

```
Normalized Theta = [[0.00571338]
 [0.25459249]
 [0.04973833]
 [0.21951471]
 [0.12779848]
 [0.08839834]
 [0.0458586 ]
 [0.02674523]
 [0.03364802]
 [0.05127891]
 [0.08031061]
 [0.06858081]]
Standardized Theta = [[5.95344365e-17]
 [3.07148779e-01]
 [2.59633038e-02]
 [2.55753688e-01]
 [2.25697778e-01]
 [1.26377042e-01]
 [8.98721270e-02]
 [5.39972666e-02]
 [9.99097249e-02]
 [6.72282523e-02]
 [2.19594016e-01]
 [1.67027160e-01]]
```

In [64]:
```
""" Part 3a : numeric data with input cleaning and parameter penalization """
%reset -f
```

In [65]:
```
#Module Inclusions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn_pandas import DataFrameMapper
```

In [66]:
```
#Function to calculate the loss of a model
def get_loss(x, y, theta):
    """x is input data (m x n)
       y is ground truths (m x 1)
       theta is model params (n x 1)"""
    h = x.dot(theta)
    error = np.subtract(h, y)
    sqError = np.square(error)
    sumSqError = np.sum(sqError)
    avgSqError = sumSqError /(2 * len(y))
    return avgSqError
```

In [67]:
```
#Function to run the gradient descent algorithm
def gradient_descent(xT, yT, theta, a, lam, iterations, xV, yV):
    """xT is input training data (m x n)
       yT is training ground truths (m x 1)
       theta is model params (n x 1)
       a is learn rate (scalar)
       lam is parameter penalty (scalar)
       iterations. duh. (scalar)
       xV is input vaidatation data
       yV is validation ground truths """
    tLoss = np.zeros(iterations)
    vLoss = np.zeros(iterations)
    m = len(yT)
    for i in range(iterations):
        h = xT.dot(theta)
        error = np.subtract(h, yT)
        delta = xT.transpose().dot(error)
        grad = delta / m
        theta = (theta * (1 - (a * lam / m))) - (a * grad)
        tLoss[i] = get_loss(xT, yT, theta)
        vLoss[i] = get_loss(xV, yV, theta)
    return theta, tLoss, vLoss
```

In [68]:
```
#Read in the CSV into a dataframe
csvData = pd.read_csv("./Housing.csv")

csvCols = len(csvData.columns)
csvRows = len(csvData)

csvData.head(5)
```

Out[68]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterhea
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

In [69]:
```
#Collect Data
dataLabels = ['price','area','bedrooms','bathrooms','stories','parking']
data = csvData[dataLabels]
data.head(5)
```

Out[69]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

In [70]:
```
#Split T-set and V-set
np.random.seed(1337)
frameT, frameV = train_test_split(data, train_size = 0.8, test_size = 0.2)
n = len(frameT.columns)
m = len(frameT)
mV = len(frameV)
frameT.head(5)
```

Out[70]:

	price	area	bedrooms	bathrooms	stories	parking
536	1960000	3420	5	1	2	0
420	3360000	4120	2	1	2	0
63	7035000	6360	4	2	3	2
465	3045000	3800	2	1	1	0
277	4305000	10360	2	1	1	1

In [71]:
```
#Preprocessing
TNScaler = DataFrameMapper([(frameT.columns,MinMaxScaler())])
frameTN = TNScaler.fit_transform(frameT,n)
frameTN = pd.DataFrame(frameTN, index=frameT.index, columns=frameT.columns)
VNScaler = DataFrameMapper([(frameV.columns,MinMaxScaler())])
frameVN = VNScaler.fit_transform(frameV,n)
frameVN = pd.DataFrame(frameVN, index=frameV.index, columns=frameV.columns)
TSScaler = DataFrameMapper([(frameT.columns,StandardScaler())])
frameTS = TSScaler.fit_transform(frameT,n)
frameTS = pd.DataFrame(frameTS, index=frameT.index, columns=frameT.columns)
VSScaler = DataFrameMapper([(frameV.columns,StandardScaler())])
frameVS = VSScaler.fit_transform(frameV,n)
frameVS = pd.DataFrame(frameVS, index=frameV.index, columns=frameV.columns)
print("Normalied Training Set:")
print(frameTN.head(5))
print("Normalized Validation Set:")
print(frameVN.head(5))
print("Standardized Training Set:")
print(frameTS.head(5))
print("Standardized Validation Set:")
print(frameVS.head(5))
```

```
Normalied Training Set:
        price      area  bedrooms  bathrooms   stories   parking
536  0.018182  0.118621       0.8   0.000000  0.333333  0.000000
420  0.139394  0.166897       0.2   0.000000  0.333333  0.000000
63   0.457576  0.321379       0.6   0.333333  0.666667  0.666667
465  0.112121  0.144828       0.2   0.000000  0.000000  0.000000
277  0.221212  0.597241       0.2   0.000000  0.000000  0.333333
Normalized Validation Set:
        price      area  bedrooms  bathrooms   stories   parking
326  0.214047  0.216929       0.6        0.0  0.000000  0.000000
449  0.133779  0.000000       0.4        0.0  0.333333  0.000000
224  0.287625  0.760581       0.2        0.0  0.000000  0.666667
140  0.381271  0.367452       0.4        0.5  1.000000  0.000000
13   0.715719  0.163804       0.6        0.5  0.333333  0.666667
Standardized Training Set:
        price      area  bedrooms  bathrooms   stories   parking
536 -1.487025 -0.795113  2.857902  -0.555867  0.236492 -0.810745
420 -0.741523 -0.475576 -1.304833  -0.555867  0.236492 -0.810745
63   1.215421  0.546940  1.470324   1.414524  1.395039  1.553702
465 -0.909261 -0.621650 -1.304833  -0.555867 -0.922054 -0.810745
277 -0.238309  2.372862 -1.304833  -0.555867 -0.922054  0.371478
Standardized Validation Set:
        price      area  bedrooms  bathrooms   stories  parking
326 -0.455403 -0.484521  1.179999  -0.632049 -0.959479    -0.79
449 -0.914334 -1.665249 -0.080980  -0.632049  0.177295    -0.79
224 -0.034715  2.474528 -1.341960  -0.632049 -0.959479     1.39
140  0.500705  0.334760 -0.080980   1.455629  2.450844    -0.79
13   2.412920 -0.773679  1.179999   1.455629  0.177295     1.39
```

In [72]:
```
#Break data into X,Y sets
YvN = frameVN.pop('price')
XvN = frameVN
YN = frameTN.pop('price')
XN = frameTN
YvS = frameVS.pop('price')
XvS = frameVS
YS = frameTS.pop('price')
XS = frameTS
```

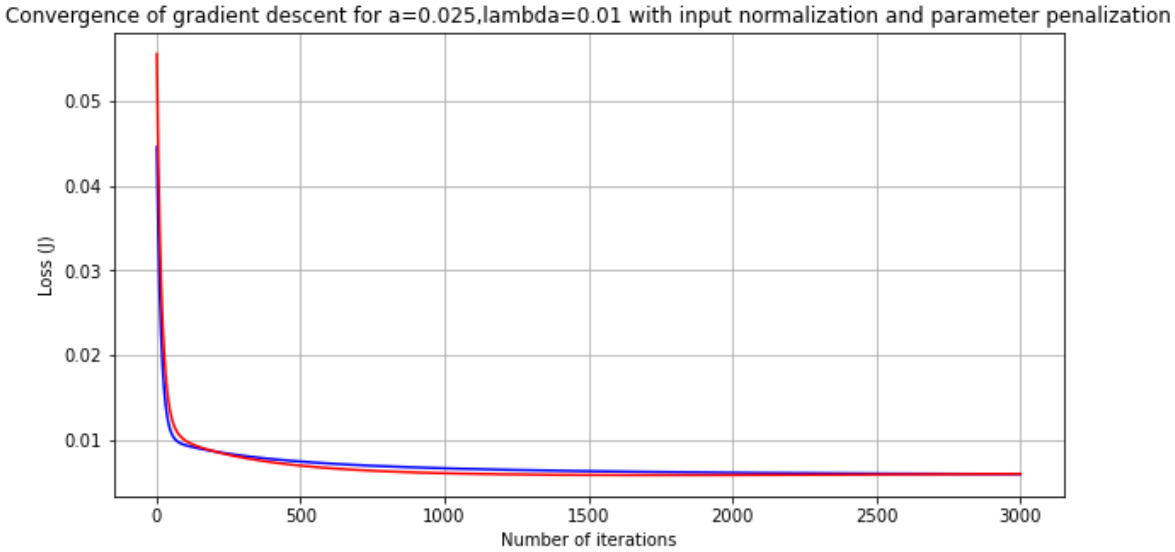
In [73]:
```
#Format Data For Gradient Descent
YvN = YvN.values[:].reshape([mV,1])
XvN = np.hstack((np.ones((mV,1)), XvN.values[:,:].reshape([mV,n-1])))
YN = YN.values[:].reshape([m,1])
XN = np.hstack((np.ones((m,1)), XN.values[:,:].reshape([m,n-1])))
YvS = YvS.values[:].reshape([mV,1])
XvS = np.hstack((np.ones((mV,1)), XvS.values[:,:].reshape([mV,n-1])))
YS = YS.values[:].reshape([m,1])
XS = np.hstack((np.ones((m,1)), XS.values[:,:].reshape([m,n-1])))
```

In [74]:
```
#set up gradient descent
LEARN_RATE_N = 0.025
ITERATIONS_N = 3000
PENALTY_N = 0.01
LEARN_RATE_S = 0.025
ITERATIONS_S = 400
PENALTY_S = 0.01
#init theta
theta = np.zeros((n,1))
```

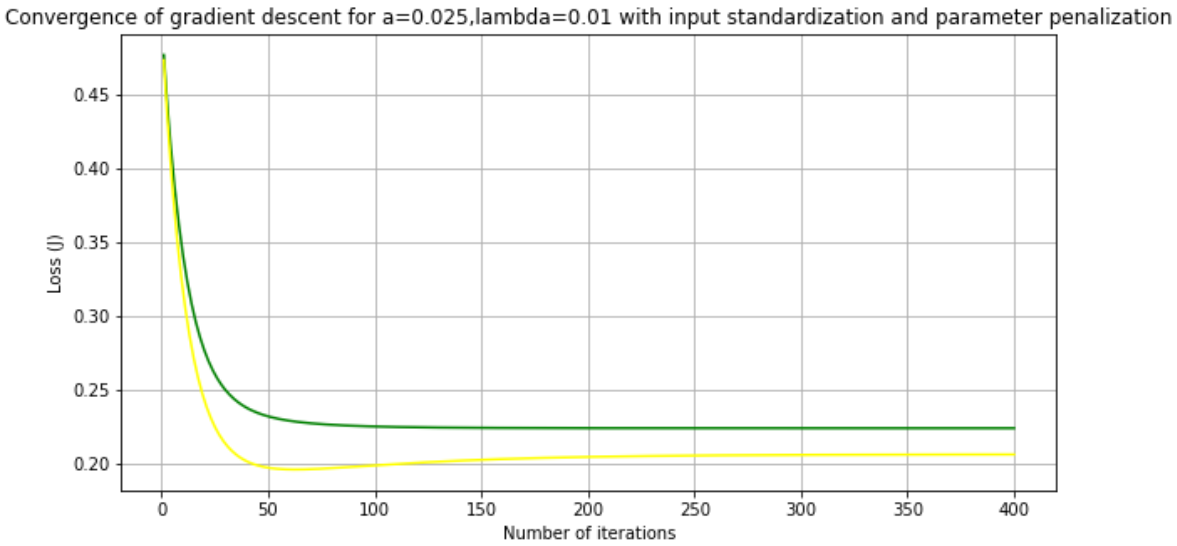
In [75]:
```
#Perform the Gradient Descent on the data
thetaN, lossTN, lossVN = gradient_descent(XN,YN,theta,LEARN_RATE_N,PENALTY_N,I
print("Converged Training Loss J for normalized set = {:.3f}".format(lossTN[-1
print("Final Validation Loss J for normalized set = {:.3f}".format(lossVN[-1])

thetaS, lossTS, lossVS = gradient_descent(XS,YS,theta,LEARN_RATE_S,PENALTY_S,I
print("Converged Training Loss J for standardized set = {:.3f}".format(lossTS[
print("Final Validation Loss J for standardized set = {:.3f}".format(lossVS[-1
```

```
Converged Training Loss J for normalized set = 0.006
Final Validation Loss J for normalized set = 0.006
Converged Training Loss J for standardized set = 0.224
Final Validation Loss J for standardized set = 0.206
```

In [76]:
```
#Plot loss over training interval with Input Normalization
plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title(
    'Convergence of gradient descent for a={},lambda={} with input normalizati
    .format(LEARN_RATE_N,PENALTY_N))
plt.plot(range(1,ITERATIONS_N+1),lossTN, color='blue')
plt.plot(range(1,ITERATIONS_N+1),lossVN, color='red');
```



In [77]:
```
#Plot loss over training interval
plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title(
    'Convergence of gradient descent for a={},lambda={} with input standardiza
    .format(LEARN_RATE_S,PENALTY_S))
plt.plot(range(1,ITERATIONS_S+1),lossTS, color='green')
plt.plot(range(1,ITERATIONS_S+1),lossVS, color='yellow');
```



In [78]:
```
print("Normalized Theta = {}".format(thetaN))
print("Standardized Theta = {}".format(thetaS))
```

```
Normalized Theta = [[0.05837351]
 [0.35820404]
 [0.063445  ]
 [0.25428011]
 [0.158702  ]
 [0.11217477]]
Standardized Theta = [[5.03801906e-17]
 [4.14046633e-01]
 [3.33433655e-02]
 [2.90971291e-01]
 [2.75797770e-01]
 [1.57998359e-01]]
```

In [79]:
```
""" Part 3b : numeric and boolean data with input cleaning and parameter penal
%reset -f
```

In [80]:
```
#Module Inclusions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn_pandas import DataFrameMapper
```

In [81]:
```
#Function to calculate the loss of a model
def get_loss(x, y, theta):
    """x is input data (m x n)
       y is ground truths (m x 1)
       theta is model params (n x 1)"""
    h = x.dot(theta)
    error = np.subtract(h, y)
    sqError = np.square(error)
    sumSqError = np.sum(sqError)
    avgSqError = sumSqError /(2 * len(y))
    return avgSqError
```

In [82]:
```
#Function to run the gradient descent algorithm
def gradient_descent(xT, yT, theta, a, lam, iterations, xV, yV):
    """xT is input training data (m x n)
       yT is training ground truths (m x 1)
       theta is model params (n x 1)
       a is learn rate (scalar)
       lam is parameter penalty (scalar)
       iterations. duh. (scalar)
       xV is input vaidatation data
       yV is validation ground truths """
    tLoss = np.zeros(iterations)
    vLoss = np.zeros(iterations)
    m = len(yT)
    for i in range(iterations):
        h = xT.dot(theta)
        error = np.subtract(h, yT)
        delta = xT.transpose().dot(error)
        grad = delta / m
        theta = (theta * (1 - (a * lam / m))) - (a * grad)
        tLoss[i] = get_loss(xT, yT, theta)
        vLoss[i] = get_loss(xV, yV, theta)
    return theta, tLoss, vLoss
```

In [83]:
```
#Function to convert Y/N booleans into 1/0 booleans
def yn_bool_convert(val):
    return val.map({ "yes" : 1 , "no" : 0 })
```

In [84]:
```
#Read in the CSV into a dataframe
csvData = pd.read_csv("./Housing.csv")

csvCols = len(csvData.columns)
csvRows = len(csvData)

csvData.head(5)
```

Out[84]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterhea
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

In [85]:
```
#Collect Data
dataLabels = ['price','area','bedrooms','bathrooms','stories','parking']
data = csvData[dataLabels]
data.head(5)
```

Out[85]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

In [86]:
```
#Collect boolean data
boolLabels = ['mainroad','guestroom','basement','hotwaterheating','airconditio
boolData = csvData[boolLabels]
boolData
```

Out[86]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea
0	yes	no	no	no	yes	yes
1	yes	no	no	no	yes	no
2	yes	no	yes	no	no	yes
3	yes	no	yes	no	yes	yes
4	yes	yes	yes	no	yes	no
...	...	...	...	...	...	...
540	yes	no	yes	no	no	no
541	no	no	no	no	no	no
542	yes	no	no	no	no	no
543	no	no	no	no	no	no
544	yes	no	no	no	no	no

545 rows × 6 columns

In [87]:
```
#Convert Y/N booleans to numerical booleans
boolData = boolData.apply(yn_bool_convert)
boolData
```

Out[87]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea
0	1	0	0	0	1	1
1	1	0	0	0	1	0
2	1	0	1	0	0	1
3	1	0	1	0	1	1
4	1	1	1	0	1	0
...	...	...	...	...	...	...
540	1	0	1	0	0	0
541	0	0	0	0	0	0
542	1	0	0	0	0	0
543	0	0	0	0	0	0
544	1	0	0	0	0	0

545 rows × 6 columns

In [88]:
```
#Join datasets
newdata = pd.concat([data,boolData],axis=1,join='outer')
newdata
```

Out[88]:

	price	area	bedrooms	bathrooms	stories	parking	mainroad	guestroom	basement
0	13300000	7420	4	2	3	2	1	0	0
1	12250000	8960	4	4	4	3	1	0	0
2	12250000	9960	3	2	2	2	1	0	1
3	12215000	7500	4	2	2	3	1	0	1
4	11410000	7420	4	1	2	2	1	1	1
...	...	...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	2	1	0	1
541	1767150	2400	3	1	1	0	0	0	0
542	1750000	3620	2	1	1	0	1	0	0
543	1750000	2910	3	1	1	0	0	0	0
544	1750000	3850	3	1	2	0	1	0	0

545 rows × 12 columns

In [89]:
```
#Split T-set and V-set
np.random.seed(1337)
frameT, frameV = train_test_split(newdata, train_size = 0.8, test_size = 0.2)
n = len(frameT.columns)
m = len(frameT)
mV = len(frameV)
frameT.head(5)
```

Out[89]:

	price	area	bedrooms	bathrooms	stories	parking	mainroad	guestroom	basement
536	1960000	3420	5	1	2	0	0	0	0
420	3360000	4120	2	1	2	0	1	0	0
63	7035000	6360	4	2	3	2	1	0	0
465	3045000	3800	2	1	1	0	1	0	0
277	4305000	10360	2	1	1	1	1	0	0

In [90]:
```
#Preprocessing
TNScaler = DataFrameMapper([(frameT.columns,MinMaxScaler())])
frameTN = TNScaler.fit_transform(frameT,n)
frameTN = pd.DataFrame(frameTN, index=frameT.index, columns=frameT.columns)
VNScaler = DataFrameMapper([(frameV.columns,MinMaxScaler())])
frameVN = VNScaler.fit_transform(frameV,n)
frameVN = pd.DataFrame(frameVN, index=frameV.index, columns=frameV.columns)
TSScaler = DataFrameMapper([(frameT.columns,StandardScaler())])
frameTS = TSScaler.fit_transform(frameT,n)
frameTS = pd.DataFrame(frameTS, index=frameT.index, columns=frameT.columns)
VSScaler = DataFrameMapper([(frameV.columns,StandardScaler())])
frameVS = VSScaler.fit_transform(frameV,n)
frameVS = pd.DataFrame(frameVS, index=frameV.index, columns=frameV.columns)
print("Normalied Training Set:")
print(frameTN.head(5))
print("Normalized Validation Set:")
print(frameVN.head(5))
print("Standardized Training Set:")
print(frameTS.head(5))
print("Standardized Validation Set:")
print(frameVS.head(5))
```

```
Normalied Training Set:
        price      area  bedrooms  bathrooms   stories   parking  mainroad  \
536  0.018182  0.118621       0.8   0.000000  0.333333  0.000000       0.0
420  0.139394  0.166897       0.2   0.000000  0.333333  0.000000       1.0
63   0.457576  0.321379       0.6   0.333333  0.666667  0.666667       1.0
465  0.112121  0.144828       0.2   0.000000  0.000000  0.000000       1.0
277  0.221212  0.597241       0.2   0.000000  0.000000  0.333333       1.0

     guestroom  basement  hotwaterheating  airconditioning  prefarea
536        0.0       0.0              0.0              0.0       0.0
420        0.0       0.0              0.0              0.0       0.0
63         0.0       0.0              0.0              1.0       1.0
465        0.0       0.0              0.0              0.0       0.0
277        0.0       0.0              0.0              0.0       1.0
Normalized Validation Set:
        price      area  bedrooms  bathrooms   stories   parking  mainroad  \
326  0.214047  0.216929       0.6        0.0  0.000000  0.000000       0.0
449  0.133779  0.000000       0.4        0.0  0.333333  0.000000       0.0
224  0.287625  0.760581       0.2        0.0  0.000000  0.666667       1.0
140  0.381271  0.367452       0.4        0.5  1.000000  0.000000       1.0
13   0.715719  0.163804       0.6        0.5  0.333333  0.666667       1.0

     guestroom  basement  hotwaterheating  airconditioning  prefarea
326        0.0       1.0              0.0              0.0       0.0
449        0.0       1.0              0.0              0.0       0.0
224        0.0       0.0              0.0              1.0       1.0
140        0.0       0.0              0.0              1.0       0.0
13         0.0       0.0              1.0              0.0       0.0
Standardized Training Set:
        price      area  bedrooms  bathrooms   stories   parking  mainroad  \
536 -1.487025 -0.795113  2.857902  -0.555867  0.236492 -0.810745 -2.410913
420 -0.741523 -0.475576 -1.304833  -0.555867  0.236492 -0.810745  0.414781
63   1.215421  0.546940  1.470324   1.414524  1.395039  1.553702  0.414781
465 -0.909261 -0.621650 -1.304833  -0.555867 -0.922054 -0.810745  0.414781
```

```
277 -0.238309  2.372862 -1.304833  -0.555867 -0.922054  0.371478  0.414781

     guestroom  basement  hotwaterheating  airconditioning  prefarea
536  -0.474045  -0.72053        -0.219265        -0.684115 -0.552620
420  -0.474045  -0.72053        -0.219265        -0.684115 -0.552620
63   -0.474045  -0.72053        -0.219265         1.461742  1.809561
465  -0.474045  -0.72053        -0.219265        -0.684115 -0.552620
277  -0.474045  -0.72053        -0.219265        -0.684115  1.809561
Standardized Validation Set:
        price      area  bedrooms  bathrooms   stories  parking  mainroad  \
326 -0.455403 -0.484521  1.179999  -0.632049 -0.959479    -0.79 -2.717465
449 -0.914334 -1.665249 -0.080980  -0.632049  0.177295    -0.79 -2.717465
224 -0.034715  2.474528 -1.341960  -0.632049 -0.959479     1.39  0.367990
140  0.500705  0.334760 -0.080980   1.455629  2.450844    -0.79  0.367990
13   2.412920 -0.773679  1.179999   1.455629  0.177295     1.39  0.367990

     guestroom  basement  hotwaterheating  airconditioning  prefarea
326  -0.429863  1.263027        -0.219265        -0.658947 -0.559690
449  -0.429863  1.263027        -0.219265        -0.658947 -0.559690
224  -0.429863 -0.791748        -0.219265         1.517574  1.786703
140  -0.429863 -0.791748        -0.219265         1.517574 -0.559690
13   -0.429863 -0.791748         4.560702        -0.658947 -0.559690
```

In [91]:
```
#Break data into X,Y sets
YvN = frameVN.pop('price')
XvN = frameVN
YN = frameTN.pop('price')
XN = frameTN
YvS = frameVS.pop('price')
XvS = frameVS
YS = frameTS.pop('price')
XS = frameTS
```

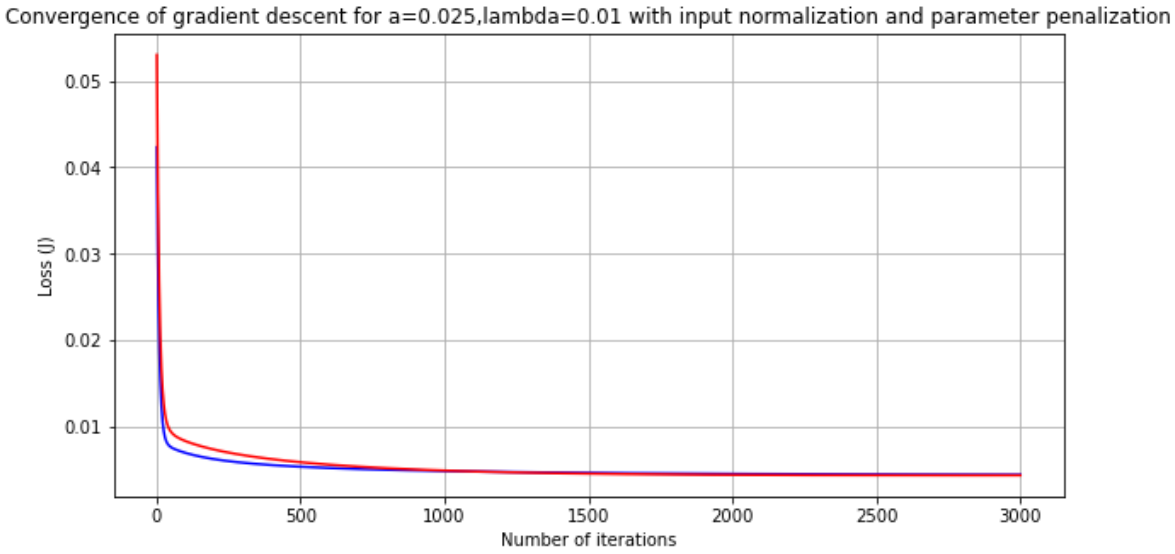
In [92]:
```
#Format Data For Gradient Descent
YvN = YvN.values[:].reshape([mV,1])
XvN = np.hstack((np.ones((mV,1)), XvN.values[:,:].reshape([mV,n-1])))
YN = YN.values[:].reshape([m,1])
XN = np.hstack((np.ones((m,1)), XN.values[:,:].reshape([m,n-1])))
YvS = YvS.values[:].reshape([mV,1])
XvS = np.hstack((np.ones((mV,1)), XvS.values[:,:].reshape([mV,n-1])))
YS = YS.values[:].reshape([m,1])
XS = np.hstack((np.ones((m,1)), XS.values[:,:].reshape([m,n-1])))
```

In [93]:
```
#set up gradient descent
LEARN_RATE_N = 0.025
ITERATIONS_N = 3000
PENALTY_N = 0.01
LEARN_RATE_S = 0.025
ITERATIONS_S = 400
PENALTY_S = 0.01
#init theta
theta = np.zeros((n,1))
```

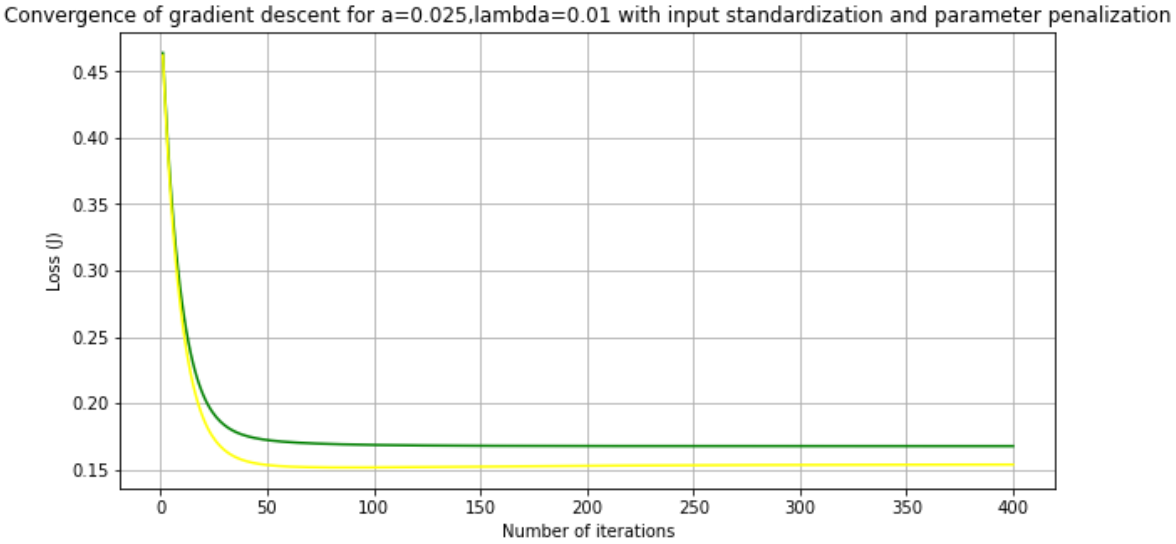
In [94]:
```
#Perform the Gradient Descent on the data
thetaN, lossTN, lossVN = gradient_descent(XN,YN,theta,LEARN_RATE_N,PENALTY_N,I
print("Converged Training Loss J for normalized set = {:.3f}".format(lossTN[-1
print("Final Validation Loss J for normalized set = {:.3f}".format(lossVN[-1])

thetaS, lossTS, lossVS = gradient_descent(XS,YS,theta,LEARN_RATE_S,PENALTY_S,I
print("Converged Training Loss J for standardized set = {:.3f}".format(lossTS[
print("Final Validation Loss J for standardized set = {:.3f}".format(lossVS[-1
```

```
Converged Training Loss J for normalized set = 0.004
Final Validation Loss J for normalized set = 0.004
Converged Training Loss J for standardized set = 0.168
Final Validation Loss J for standardized set = 0.154
```

In [95]:
```
#Plot loss over training interval with Input Normalization
plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title(
    'Convergence of gradient descent for a={},lambda={} with input normalizati
    .format(LEARN_RATE_N,PENALTY_N))
plt.plot(range(1,ITERATIONS_N+1),lossTN, color='blue')
plt.plot(range(1,ITERATIONS_N+1),lossVN, color='red');
```



In [96]:
```
#Plot loss over training interval
plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title(
    'Convergence of gradient descent for a={},lambda={} with input standardiza
    .format(LEARN_RATE_S,PENALTY_S))
plt.plot(range(1,ITERATIONS_S+1),lossTS, color='green')
plt.plot(range(1,ITERATIONS_S+1),lossVS, color='yellow');
```



In [97]:
```
print("Normalized Theta = {}".format(thetaN))
print("Standardized Theta = {}".format(thetaS))
```

```
Normalized Theta = [[0.00574689]
 [0.25444651]
 [0.04974596]
 [0.21940653]
 [0.1277795 ]
 [0.0883993 ]
 [0.04586924]
 [0.02675469]
 [0.03364668]
 [0.05125839]
 [0.08031928]
 [0.06858441]]
Standardized Theta = [[6.31488829e-17]
 [3.07143161e-01]
 [2.59676880e-02]
 [2.55749705e-01]
 [2.25692689e-01]
 [1.26376641e-01]
 [8.98731313e-02]
 [5.39988944e-02]
 [9.99070723e-02]
 [6.72264200e-02]
 [2.19591420e-01]
 [1.67025372e-01]]
```