In [41]:

```
#import files
import pandas as pd
import seaborn as sb
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression

import pdb;

from IPython.core.debugger import set_trace
import random
import matplotlib.pyplot as plt
```

In [42]:

```
#Read csv
electricity_data = pd.read_csv("energydata_complete.csv")
```

In [43]:

```
electricity_data.shape
```

Out[43]:

```
(19735, 29)
```

In [44]:

```
for col in electricity_data.columns:
    print(col)
```

```
date
Appliances
lights
T1
RH_1
T2
RH_2
T3
RH_3
T4
RH_4
T5
RH_5
T6
RH_6
T7
RH_7
T8
RH_8
T9
RH_9
T_out
Press_mm_hg
RH_out
Windspeed
Visibility
Tdewpoint
rv1
rv2
```

In [45]:

```
electricity_data.isnull().sum()
```

Out[45]:

```
date           0
Appliances     0
lights         0
T1             0
RH_1           0
T2             0
RH_2           0
T3             0
RH_3           0
T4             0
RH_4           0
T5             0
RH_5           0
T6             0
RH_6           0
T7             0
RH_7           0
T8             0
RH_8           0
T9             0
RH_9           0
T_out          0
Press_mm_hg    0
RH_out         0
Windspeed      0
Visibility     0
Tdewpoint      0
rv1            0
rv2            0
dtype: int64
```

In [46]:

```
electricity_data = electricity_data.drop(['date','lights'] , axis = 1)
```

In [47]:

```
electricity_data.shape
```

Out[47]:

```
(19735, 27)
```

In [48]:

```
electricity_data.head()
```

Out[48]:

	Appliances	T1	RH_1	T2	RH_2	T3
0	60	19.89	47.596667	19.2	44.790000	19.79
1	60	19.89	46.693333	19.2	44.722500	19.79
2	50	19.89	46.300000	19.2	44.626667	19.79
3	50	19.89	46.066667	19.2	44.590000	19.79
4	60	19.89	46.333333	19.2	44.530000	19.79

5 rows × 27 columns

In [49]:

```
electricity_data_appliance =  electricity_data
```

In [50]:

```
electricity_data_appliance.describe()
```

Out[50]:

	Appliances	T1	RH_1	
count	19735.000000	19735.000000	19735.000000	19735
mean	97.694958	21.686571	40.259739	20
std	102.524891	1.606066	3.979299	2
min	10.000000	16.790000	27.023333	16
25%	50.000000	20.760000	37.333333	18
50%	60.000000	21.600000	39.656667	20
75%	100.000000	22.600000	43.066667	21
max	1080.000000	26.260000	63.360000	29

8 rows × 27 columns

In [51]:

```
x_appliances = electricity_data_appliance.drop(labels = [
'Appliances','T6','RH_6','T7','RH_7','T8','RH_8',

'T9','RH_9','rv1','rv2'],axis = 1)
```

In [52]:

```
x_appliances.columns
```

Out[52]:

```
Index(['T1', 'RH_1', 'T2', 'RH_2', 'T3', 'RH
_3', 'T4', 'RH_4', 'T5', 'RH_5',
       'T_out', 'Press_mm_hg', 'RH_out', 'Wi
ndspeed', 'Visibility',
       'Tdewpoint'],
      dtype='object')
```

In [53]:

```
y_appliances = electricity_data_appliance[['Appliances']]
```

In [54]:

```
random.seed(1)
```

In [55]:

```
scaler = MinMaxScaler()
x_appliances_scaled = x_appliances
y_appliances_scaled = y_appliances

x_appliances_scaled = scaler.fit_transform(x_appliances)
y_appliances_scaled = scaler.fit_transform(y_appliances)
```

In [56]:

```
x_appliances_scaled = np.array(x_appliances_scaled)
```

In [63]:

```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_app
liances_scaled, y_appliances_scaled, test_size=0.3,
                                                    random
_state=1)
```

In [74]:

```
from sklearn import datasets, linear_model
reg = linear_model.LinearRegression()
reg.fit(X_Train, Y_Train)

coef = reg.coef_
intercept = reg.intercept_
print('Coefficients: \n', coef)
print('Intercept: \n', intercept)
```

```
Coefficients:
 [[ 9.26409170e-02  6.46469736e-01 -2.167601
58e-01 -5.04050671e-01
   2.10412855e-01 -1.14714311e-02 -6.2376363
6e-02 -6.67453765e-02
  -7.17091939e-02  2.05145323e-02 -1.8462283
8e-01  1.44464273e-05
  -1.22493914e-01  2.06152925e-02  1.0519223
1e-02  1.07238390e-01]]
Intercept:
 [0.25349343]
```

In [65]:

```
main_beta = np.array(np.repeat(0, 16)).reshape(1,16)
beta_0 = 0
totalRows = len(X_Train) #Total number of rows in the dat
afram
learningRate = 0.6
CostValues_6_training = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Train * (X_Train @ main_beta.T - Y_Train + bet
a_0), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Train @ main_beta.T - Y_Train + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Train @ main
_beta.T + beta_0 - Y_Train) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_6_training.append(cost)
    print(beta_0)
    print(main_beta)

print(CostValues_6_training)

main_beta = np.array(np.repeat(0, 16)).reshape(1,16)
beta_0 = 0
totalRows = len(X_Train) #Total number of rows in the dat
afram
learningRate = 0.7
CostValues_7_training = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Train * (X_Train @ main_beta.T - Y_Train + bet
a_0), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Train @ main_beta.T - Y_Train + beta_0), axis=0)
```

```
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Train @ main
_beta.T + beta_0 - Y_Train) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_7_training.append(cost)
    print(beta_0)
    print(main_beta)

print(CostValues_7_training)

main_beta = np.array(np.repeat(0, 16)).reshape(1,16)
beta_0 = 0
totalRows = len(X_Train) #Total number of rows in the dat
afram
learningRate = 0.8
CostValues_8_training = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Train * (X_Train @ main_beta.T - Y_Train + bet
a_0), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Train @ main_beta.T - Y_Train + beta_0), axis=0)
    cost = (0.5 * (1/totalRows))* (np.sum(X_Train @ main_
beta.T + beta_0 - Y_Train) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_8_training.append(cost)
    print(beta_0)
    print(main_beta)

print(CostValues_8_training)

main_beta = np.array(np.repeat(0, 16)).reshape(1,16)
beta_0 = 0
totalRows = len(X_Train) #Total number of rows in the dat
afram
learningRate = 0.9
CostValues_9_training = []
```

```
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Train * (X_Train @ main_beta.T - Y_Train + bet
a_0), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Train @ main_beta.T - Y_Train + beta_0), axis=0)
    cost = (0.5 * (1/totalRows))* (np.sum(X_Train @ main_
beta.T + beta_0 - Y_Train) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_9_training.append(cost)
    print(beta_0)
    print(main_beta)

print(CostValues_9_training)
```

```
Cost =  3.6400796742223706e-07 when steps =
5000
[0.15439116]
[[ 0.01949751  0.48021288 -0.06135644 -0.286
94813  0.21163903 -0.01040788
  -0.07043403 -0.08922617 -0.08526411  0.022
61276 -0.0574259   0.00154629
  -0.05979125  0.02609363  0.01052023 -0.005
03149]]
Cost =  3.519677226186959e-08 when steps =
10000
[0.21785583]
[[ 0.08044307  0.62147143 -0.19028135 -0.464
7919   0.21087455 -0.01329604
  -0.06341534 -0.07119753 -0.07494581  0.021
19149 -0.12298918  0.00066072
  -0.09103235  0.02243259  0.01033023  0.057
90813]]
Cost =  1.9501242213130992e-09 when steps =
15000
[0.24408605]
[[ 9.16966744e-02  6.45433467e-01 -2.1459669
7e-01 -5.00020940e-01
   2.10561649e-01 -1.22988829e-02 -6.2309753
2e-02 -6.72614324e-02
  -7.23033579e-02  2.06804464e-02 -1.6573208
9e-01  1.95201224e-04
  -1.12730425e-01  2.10652851e-02  1.0439197
5e-02  9.27296838e-02]]
Cost =  6.621785406564736e-11 when steps =
20000
[0.2516985]
[[ 9.25830968e-02  6.46571068e-01 -2.1660683
6e-01 -5.03622297e-01
   2.10442694e-01 -1.16479013e-02 -6.2345326
5e-02 -6.68083975e-02
  -7.18085717e-02  2.05455015e-02 -1.8087754
```

```
8e-01  4.94804462e-05
  -1.20552620e-01  2.06995532e-02  1.0502333
9e-02  1.04390178e-01]]
Cost =  1.1119413295288403e-12 when steps =
25000
[0.25325963]
[[ 9.26357090e-02  6.46488615e-01 -2.1674511
4e-01 -5.04001339e-01
   2.10416769e-01 -1.14947708e-02 -6.2371972
7e-02 -6.67529123e-02
  -7.17218730e-02  2.05185532e-02 -1.8413230
9e-01  1.90202426e-05
  -1.22239556e-01  2.06262378e-02  1.0516992
4e-02  1.06865870e-01]]
Cost =  8.419908329836574e-15 when steps =
30000
[0.25347308]
[[ 9.26404805e-02  6.46471421e-01 -2.1675888
4e-01 -5.04046423e-01
   2.10413196e-01 -1.14734655e-02 -6.2375978
8e-02 -6.67460277e-02
  -7.17102958e-02  2.05148823e-02 -1.8458011
5e-01  1.48446827e-05
  -1.22471760e-01  2.06162452e-02  1.0519028
7e-02  1.07205949e-01]]
[3.6400796742223706e-07, 3.519677226186959e-
08, 1.9501242213130992e-09, 6.62178540656473
6e-11, 1.1119413295288403e-12, 8.41990832983
6574e-15]
Cost =  1.6876717472445035e-07 when steps =
5000
[0.16399425]
[[ 0.02977581  0.50479312 -0.08285142 -0.316
75244  0.21145036 -0.0112062
  -0.06946547 -0.08647918 -0.08351939  0.022
37285 -0.06592966  0.00138975
  -0.06371497  0.02551338  0.01044447  0.003
```

```
63443]]
Cost =  1.1392067346684683e-08 when steps =
10000
[0.22606937]
[[ 8.48351130e-02  6.31140770e-01 -1.9978758
3e-01 -4.78311131e-01
   2.10795242e-01 -1.31406104e-02 -6.2939748
6e-02 -6.96750479e-02
  -7.40107241e-02  2.10209106e-02 -1.3524233
9e-01  5.02867070e-04
  -9.72143119e-02  2.19810375e-02  1.0344925
8e-02  6.81453507e-02]]
Cost =  4.27069775491913e-10 when steps =  1
5000
[0.24760736]
[[ 9.22231312e-02  6.46252653e-01 -2.1578315
8e-01 -5.02024477e-01
   2.10509580e-01 -1.20201926e-02 -6.2308935
2e-02 -6.70108464e-02
  -7.20595749e-02  2.06162771e-02 -1.7257868
2e-01  1.25238007e-04
  -1.16263394e-01  2.08922638e-02  1.0465328
2e-02  9.80283741e-02]]
Cost =  8.407388761124096e-12 when steps =
20000
[0.252648]
[[ 9.26194356e-02  6.46532697e-01 -2.1670149
8e-01 -5.03866815e-01
   2.10427197e-01 -1.15560871e-02 -6.2360908
5e-02 -6.67723757e-02
  -7.17551208e-02  2.05289335e-02 -1.8284862
3e-01  3.05422863e-05
  -1.21574357e-01  2.06545795e-02  1.0510977
0e-02  1.05890431e-01]]
Cost =  6.844842649996859e-14 when steps =
25000
[0.25341696]
```

```
[[ 9.26393348e-02  6.46476313e-01 -2.1675561
7e-01 -5.04035044e-01
   2.10414156e-01 -1.14791430e-02 -6.2374911
6e-02 -6.67477144e-02
  -7.17133069e-02  2.05158329e-02 -1.8446194
2e-01  1.59040395e-05
  -1.22410507e-01  2.06188415e-02  1.0518472
5e-02  1.07116229e-01]]
Cost =  2.1511039447337323e-16 when steps =
30000
[0.25348914]
[[ 9.26408294e-02  6.46470108e-01 -2.1675990
6e-01 -5.04049798e-01
   2.10412928e-01 -1.14718636e-02 -6.2376282
0e-02 -6.67455073e-02
  -7.17094244e-02  2.05146052e-02 -1.8461381
6e-01  1.45281567e-05
  -1.22489237e-01  2.06154915e-02  1.0519181
0e-02  1.07231540e-01]]
[1.6876717472445035e-07, 1.1392067346684683e
-08, 4.27069775491913e-10, 8.407388761124096
e-12, 6.844842649996859e-14, 2.1511039447337
323e-16]
Cost =  6.13142399903917e-08 when steps =  5
000
[0.1728293]
[[ 0.03863335  0.52558292 -0.10158656 -0.342
61975  0.21132528 -0.01175986
  -0.06851241 -0.08387936 -0.08200709  0.022
15533 -0.0743312   0.00122786
  -0.06769343  0.02497027  0.01038755  0.011
9255 ]]
Cost =  2.9299427466886107e-09 when steps =
10000
[0.23242013]
[[ 8.76453211e-02  6.37174196e-01 -2.0588263
0e-01 -4.87115849e-01
```

```
   2.10724848e-01 -1.29250842e-02 -6.2659603
5e-02 -6.86773182e-02
  -7.33568318e-02  2.08937442e-02 -1.4542436
7e-01  3.82231797e-04
  -1.02383817e-01  2.16425939e-02  1.0366043
4e-02  7.64686659e-02]]
Cost =  7.365868988480825e-11 when steps =
15000
[0.24983592]
[[ 9.24510429e-02  6.46510523e-01 -2.1630700
9e-01 -5.02994194e-01
   2.10474767e-01 -1.18259778e-02 -6.2324126
4e-02 -6.68855872e-02
  -7.19179461e-02  2.05766806e-02 -1.7704275
4e-01  8.16813188e-05
  -1.18570968e-01  2.07850373e-02  1.0483853
2e-02  1.01457972e-01]]
Cost =  8.184309307929531e-13 when steps =
20000
[0.25310185]
[[ 9.26324138e-02  6.46502988e-01 -2.1673669
8e-01 -5.03970440e-01
   2.10419619e-01 -1.15111751e-02 -6.2369000
3e-02 -6.67569808e-02
  -7.17301950e-02  2.05211220e-02 -1.8379773
5e-01  2.17041265e-05
  -1.22066361e-01  2.06333163e-02  1.0515282
9e-02  1.06611844e-01]]
Cost =  3.1594110732897567e-15 when steps =
25000
[0.25346907]
[[ 9.26404428e-02  6.46471938e-01 -2.1675881
5e-01 -5.04045832e-01
   2.10413276e-01 -1.14739116e-02 -6.2375897
4e-02 -6.67460826e-02
  -7.17104940e-02  2.05149419e-02 -1.8457145
0e-01  1.48981612e-05
```

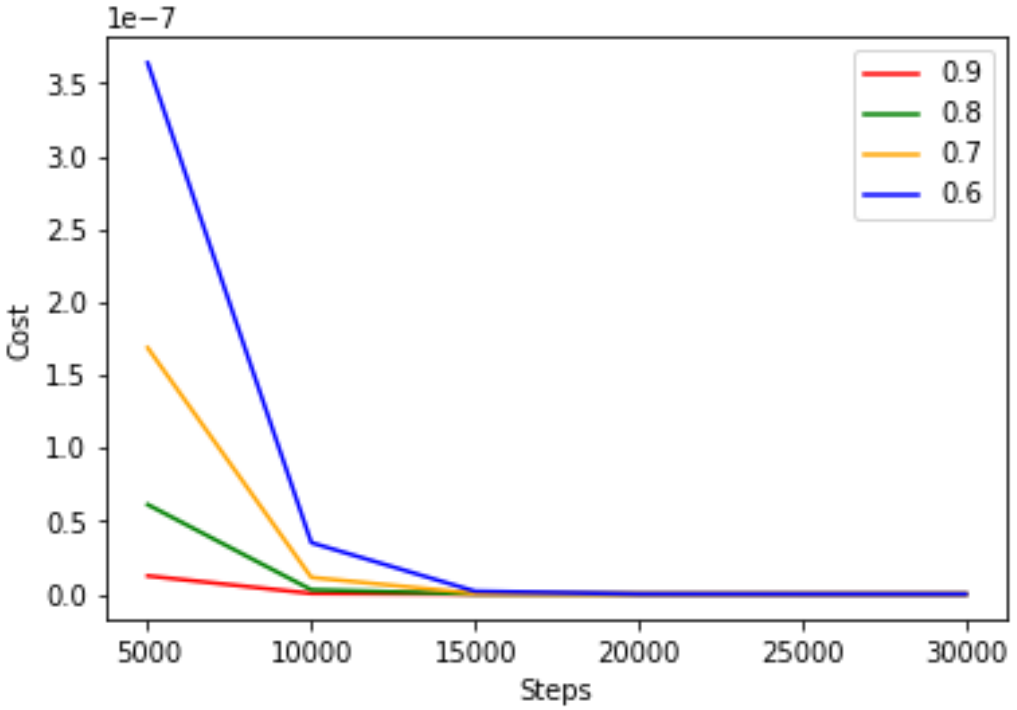
```
  -1.22467284e-01  2.06164130e-02  1.0518977
3e-02  1.07199381e-01]]
Cost =  4.033905212638956e-18 when steps =
30000
[0.25349256]
[[ 9.26409001e-02  6.46469815e-01 -2.1676011
0e-01 -5.04050498e-01
   2.10412870e-01 -1.14715197e-02 -6.2376347
0e-02 -6.67454017e-02
  -7.17092404e-02  2.05145469e-02 -1.8462100
2e-01  1.44625698e-05
  -1.22492962e-01  2.06153326e-02  1.0519214
3e-02  1.07236996e-01]]
[6.13142399903917e-08, 2.9299427466886107e-0
9, 7.365868988480825e-11, 8.184309307929531e
-13, 3.1594110732897567e-15, 4.0339052126389
56e-18]
Cost =  1.2459268512622227e-08 when steps =
5000
[0.18093492]
[[ 0.04625438  0.54328814 -0.11782695 -0.365
03158  0.21124114 -0.01215399
  -0.0676383  -0.08152387 -0.08068897  0.021
96024 -0.08257524  0.00107353
  -0.0716575   0.02447365  0.01034595  0.019
83176]]
Cost =  4.2493144570932596e-10 when steps =
10000
[0.23734071]
[[ 8.94443294e-02  6.40915817e-01 -2.0979303
2e-01 -4.92871241e-01
   2.10664233e-01 -1.26959117e-02 -6.2499126
6e-02 -6.80211528e-02
  -7.28950315e-02  2.07984059e-02 -1.5377385
0e-01  2.90067339e-04
  -1.06642181e-01  2.13875734e-02  1.0388901
5e-02  8.31834645e-02]]
```

```
Cost =  7.0690189748241 47e-12 when steps =
15000
[0.25124074]
[[ 9.25522522e-02  6.46567255e-01 -2.1654482
5e-01 -5.03483999e-01
   2.10451936e-01 -1.16959470e-02 -6.2340021
1e-02 -6.68206963e-02
  -7.18339094e-02  2.05522241e-02 -1.7991180
5e-01  5.47820593e-05
  -1.20055423e-01  2.07185634e-02  1.0496465
5e-02  1.03652268e-01]]
Cost =  4.329898696382461e-14 when steps =
20000
[0.2533153]
[[ 9.26374631e-02  6.46486133e-01 -2.1675066
4e-01 -5.04015747e-01
   2.10415985e-01 -1.14897196e-02 -6.2372957
4e-02 -6.67503104e-02
  -7.17186505e-02  2.05174938e-02 -1.8424625
1e-01  1.76547810e-05
  -1.22298827e-01  2.06234143e-02  1.0517377
9e-02  1.06952520e-01]]
Cost =  7.767959252795762e-17 when steps =
25000
[0.25348588]
[[ 9.26407788e-02  6.46470451e-01 -2.1675977
3e-01 -5.04049213e-01
   2.10412988e-01 -1.14722074e-02 -6.2376218
0e-02 -6.67455833e-02
  -7.17095938e-02  2.05146578e-02 -1.8460686
9e-01  1.45824397e-05
  -1.22485641e-01  2.06156366e-02  1.0519144
8e-02  1.07226270e-01]]
Cost =  3.944175410795833e-20 when steps =
30000
[0.25349326]
[[ 9.26409139e-02  6.46469752e-01 -2.1676014
```

```
9e-01 -5.04050638e-01
   2.10412858e-01 -1.14714486e-02 -6.2376360
3e-02 -6.67453812e-02
  -7.17092029e-02  2.05145351e-02 -1.8462247
8e-01  1.44494922e-05
  -1.22493728e-01  2.06153003e-02  1.0519221
3e-02  1.07238117e-01]]
[1.2459268512622227e-08, 4.2493144570932596e
-10, 7.0690189748241 47e-12, 4.32989869638246
1e-14, 7.767959252795762e-17, 3.944175410795
833e-20]
```

In [69]:

```
steps = np.arange(5000,30001,5000)



plt.plot(steps, CostValues_9_training, color='red', label
='0.9')
plt.plot(steps, CostValues_8_training, color='green', lab
el='0.8')
plt.plot(steps, CostValues_7_training, color='orange', la
bel='0.7')
plt.plot(steps, CostValues_6_training, color='blue', labe
l='0.6')


plt.xlabel('Steps')
plt.ylabel('Cost')
plt.legend()
plt.title('Linear Regression Gradient Descend for trainin
g data set', y=-0.25)
plt.show()
```



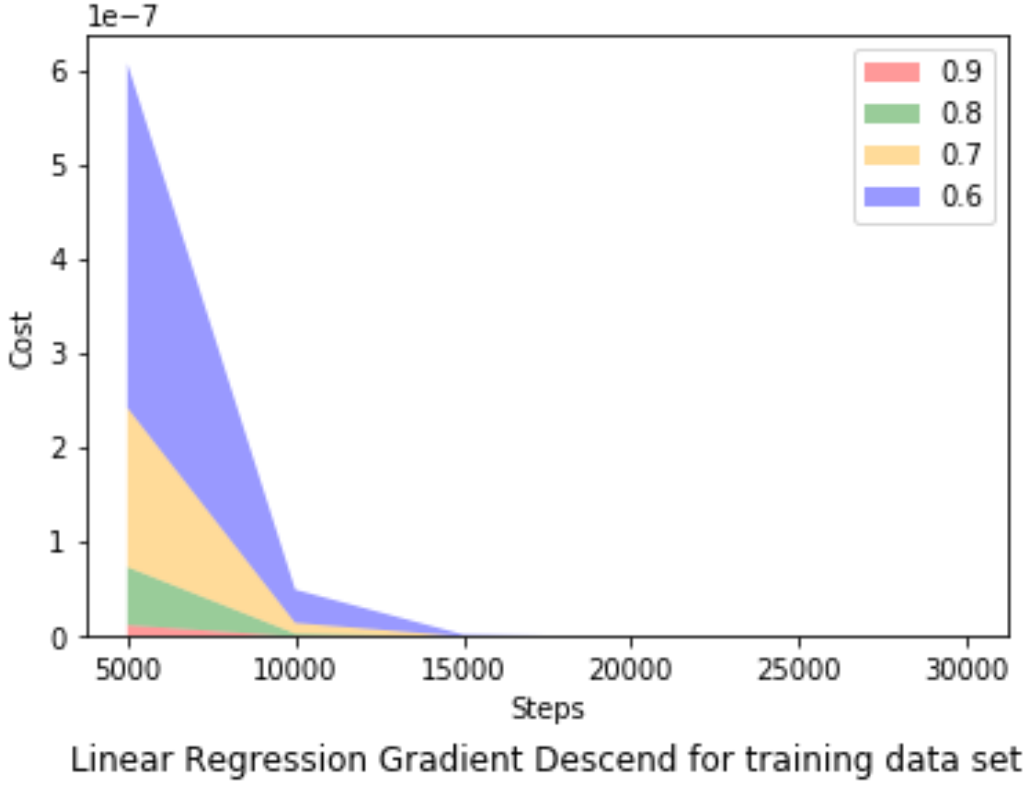
Linear Regression Gradient Descend for training data set

In [70]:

```
steps = np.arange(5000,30001,5000)
y = [CostValues_9_training,CostValues_8_training,CostValu
es_7_training,CostValues_6_training]

pal = ["red", "green", "orange", "blue"]
plt.stackplot(steps,y, labels=['0.9','0.8','0.7','0.6'],
colors=pal, alpha=0.4 )
plt.legend(loc='upper right')
plt.xlabel('Steps')
plt.ylabel('Cost')
plt.title('Linear Regression Gradient Descend for trainin
g data set', y=-0.25)
```

Out[70]:

```
Text(0.5, -0.25, 'Linear Regression Gradient
Descend for training data set')
```



In [68]:

```
main_beta = np.array(np.repeat(0, 16)).reshape(1,16)
beta_0 = 0
totalRows = len(X_Test) #Total number of rows in the data
fram
learningRate = 0.9
CostValues_9_validation = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Test * (X_Test @ main_beta.T - Y_Test + beta_0
), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Test @ main_beta.T - Y_Test + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Test @ main_
beta.T + beta_0 - Y_Test) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_9_validation.append(cost)
    print(beta_0)
    print(main_beta)
print(CostValues_9_validation)


main_beta = np.array(np.repeat(0, 16)).reshape(1,16)
beta_0 = 0
learningRate = 0.8
CostValues_8_validation = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Test * (X_Test @ main_beta.T - Y_Test + beta_0
), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Test @ main_beta.T - Y_Test + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Test @ main_
beta.T + beta_0 - Y_Test) ** 2)
```

```
    print("Cost = ",cost, "when steps = ", j)
    CostValues_8_validation.append(cost)
    print(beta_0)
    print(main_beta)
print(CostValues_8_validation)


main_beta = np.array(np.repeat(0, 16)).reshape(1,16)
beta_0 = 0
learningRate = 0.7
CostValues_7_validation = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Test * (X_Test @ main_beta.T - Y_Test + beta_0
), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Test @ main_beta.T - Y_Test + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Test @ main_
beta.T + beta_0 - Y_Test) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_7_validation.append(cost)
    print(beta_0)
    print(main_beta)
print(CostValues_7_validation)


main_beta = np.array(np.repeat(0, 16)).reshape(1,16)
beta_0 = 0
learningRate = 0.6
CostValues_6_validation = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Test * (X_Test @ main_beta.T - Y_Test + beta_0
), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
```

```
um((X_Test @ main_beta.T - Y_Test + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Test @ main_
beta.T + beta_0 - Y_Test) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_6_validation.append(cost)
    print(beta_0)
    print(main_beta)
print(CostValues_6_validation)
```

```
Cost =  5.635200902373342e-09 when steps =
5000
[0.19245097]
[[ 0.06590958  0.57368779 -0.14474731 -0.396
09369  0.19693672 -0.02004382
  -0.05110178 -0.05848366 -0.09790213  0.017
02862 -0.07156067 -0.00152628
  -0.07921944  0.01992595  0.00986567  0.014
42965]]
Cost =  1.6647621887560253e-10 when steps =
10000
[0.2489024]
[[ 0.10992952  0.67430961 -0.24159393 -0.531
47102  0.19706837 -0.02086022
  -0.0465955  -0.04361906 -0.08786523  0.016
09185 -0.13857775 -0.00234831
  -0.111901    0.01702626  0.00998446  0.076
22633]]
Cost =  2.197260278135993e-12 when steps =
15000
[0.26169237]
[[ 0.11297588  0.6800125  -0.24842081 -0.542
1213   0.19695321 -0.01993345
  -0.04650376 -0.04241631 -0.08665347  0.015
88551 -0.16212269 -0.00256703
  -0.12401561  0.01642138  0.01009741  0.094
79752]]
Cost =  9.812075620656433e-15 when steps =
20000
[0.26337662]
[[ 0.11306646  0.6799976  -0.24864114 -0.542
62527  0.19693115 -0.01976166
  -0.04653384 -0.04235586 -0.08654138  0.015
8586  -0.16559872 -0.00259723
  -0.12582233  0.01634412  0.01011613  0.097
44845]]
Cost =  1.1852945617677017e-17 when steps =
```

```
25000
[0.26349427]
[[ 0.11307018  0.67999011 -0.24865081 -0.542
65291  0.19692955 -0.01974924
  -0.04653629 -0.04235246 -0.08653395  0.015
85672 -0.1658447  -0.00259935
  -0.12595032  0.01633874  0.01011747  0.097
63537]]
Cost =  3.7441457354055025e-21 when steps =
30000
[0.26349844]
[[ 0.1130703   0.67998983 -0.24865114 -0.542
65388  0.19692949 -0.0197488
  -0.04653638 -0.04235234 -0.08653368  0.015
85665 -0.16585341 -0.00259942
  -0.12595485  0.01633855  0.01011751  0.097
64198]]
[5.635200902373342e-09, 1.6647621887560253e-
10, 2.197260278135993e-12, 9.812075620656433
e-15, 1.1852945617677017e-17, 3.744145735405
5025e-21]
Cost =  2.8047553670287754e-08 when steps =
5000
[0.18406991]
[[ 0.05801016  0.55513408 -0.12728253 -0.371
83881  0.19688284 -0.01958867
  -0.05190579 -0.06118446 -0.09964341  0.017
18638 -0.06367226 -0.00136202
  -0.07548231  0.02040283  0.00989335  0.006
46989]]
Cost =  1.1816680949131O2e-09 when steps =
10000
[0.24414974]
[[ 0.10814614  0.6705513  -0.23759599 -0.525
57603  0.19708956 -0.02108203
  -0.04671696 -0.04429894 -0.08842072  0.016
17266 -0.13071006 -0.00225866
```

```
  -0.10788309  0.01726444  0.00995612  0.069
82656]]
Cost =  2.4335517333262187e-11 when steps =
15000
[0.2604735]
[[ 0.11287368  0.67992827 -0.24817596 -0.541
6431   0.19696737 -0.02004933
  -0.04648679 -0.0424755  -0.086741    0.015
90553 -0.15966483 -0.00254373
  -0.12273882  0.01647879  0.01008512  0.092
91335]]
Cost =  2.052012271269615e-13 when steps =
20000
[0.26321659]
[[ 0.11306077  0.68000577 -0.24862608 -0.542
58513  0.19693322 -0.01977821
  -0.04653057 -0.04236104 -0.08655166  0.015
8612  -0.16526614 -0.0025942
  -0.12564921  0.01635155  0.01011439  0.097
19557]]
Cost =  5.603734771832336e-16 when steps =
25000
[0.26348376]
[[ 0.11306984  0.67999075 -0.24864991 -0.542
6504   0.19692968 -0.01975034
  -0.04653607 -0.04235278 -0.08653461  0.015
85689 -0.16582279 -0.00259915
  -0.12593892  0.01633922  0.01011735  0.097
61872]]
Cost =  4.722646679923605e-19 when steps =
30000
[0.26349809]
[[ 0.11307029  0.67998985 -0.24865111 -0.542
65379  0.19692949 -0.01974884
  -0.04653637 -0.04235235 -0.08653371  0.015
85666 -0.16585267 -0.00259941
  -0.12595447  0.01633856  0.01011751  0.097
```

```
64143]]
[2.8047553670287754e-08, 1.181668094913102e-
09, 2.43355517333262187e-11, 2.05201227126961
58e-13, 5.603734771832336e-16, 4.72264667992
3605e-19]
Cost =  7.820348161821728e-08 when steps =
5000
[0.17488302]
[[ 0.04882406  0.53331601 -0.10704016 -0.343
71266  0.19683574 -0.01894499
  -0.05280514 -0.06424967 -0.10165524  0.017
36101 -0.05568019 -0.00118628
  -0.07176913  0.0209277   0.00993516 -0.001
88605]]
Cost =  4.721601399448468e-09 when steps =
10000
[0.23793044]
[[ 0.10534146  0.66445988 -0.23132142 -0.516
48009  0.19710419 -0.02129023
  -0.04693951 -0.04534717 -0.08922168  0.016
28007 -0.12101764 -0.00213942
  -0.10296041  0.01758274  0.00992683  0.061
78863]]
Cost =  1.495912254345756e-10 when steps =
15000
[0.258482]
[[ 0.11264748  0.67964149 -0.24764293 -0.540
68541  0.19698851 -0.02022743
  -0.04646699 -0.04259349 -0.0868936   0.015
93864 -0.15572956 -0.00250484
  -0.12069685  0.01657384  0.0100663   0.089
88072]]
Cost =  2.324614055708201e-12 when steps =
20000
[0.26285913]
[[ 0.11304631  0.68001921 -0.24858804 -0.542
48958  0.19693769 -0.01981461
```

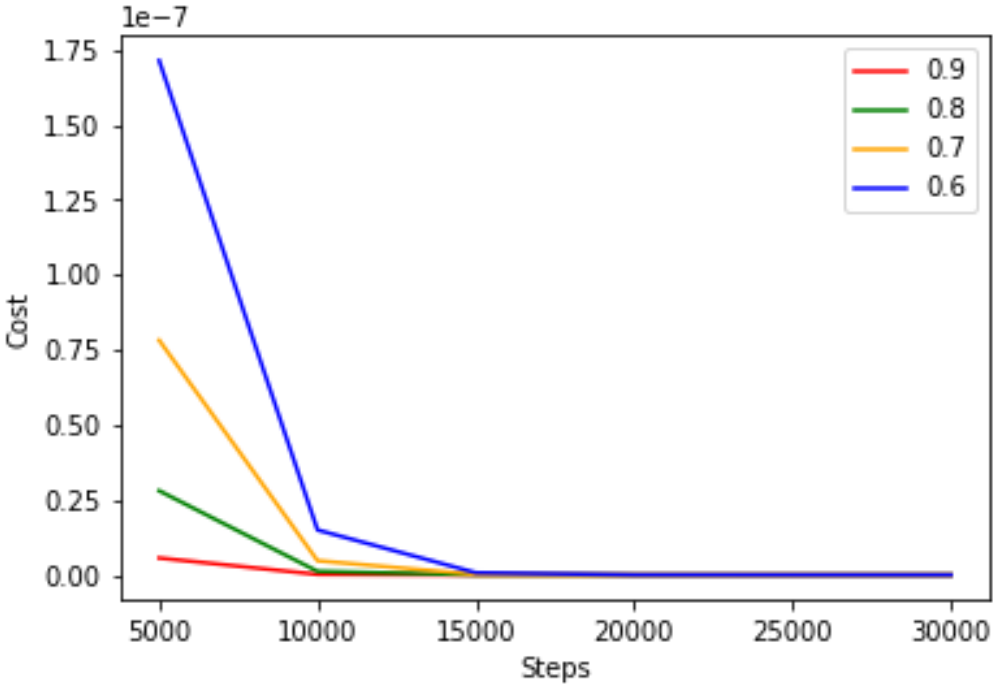
```
  -0.04652348 -0.0423736  -0.08657497  0.015
86707 -0.16452693 -0.00258725
  -0.12526436  0.01636829  0.01011061  0.096
63301]]
Cost =  1.4046447740718253e-14 when steps =
25000
[0.2634487]
[[ 0.11306868  0.67999272 -0.24864676 -0.542
64179  0.19693013 -0.01975396
  -0.04653534 -0.04235392 -0.08653686  0.015
85746 -0.16574991 -0.00259847
  -0.12590096  0.01634086  0.01011697  0.097
56332]]
Cost =  3.085991554178671e-17 when steps =
30000
[0.26349618]
[[ 0.11307023  0.67998997 -0.24865095 -0.542
65333  0.19692952 -0.01974903
  -0.04653633 -0.04235241 -0.08653383  0.015
85669 -0.16584871 -0.00259938
  -0.1259524   0.01633865  0.01011749  0.097
63841]]
[7.820348161821728e-08, 4.721601399448468e-0
9, 1.495912254345756e-10, 2.324614055708201e
-12, 1.4046447740718253e-14, 3.0859915541786
71e-17]
Cost =  1.7138489230002227e-07 when steps =
5000
[0.16483037]
[[ 0.03817557  0.50751221 -0.08368715 -0.311
13266  0.19680531 -0.01800152
  -0.05375939 -0.06765389 -0.10399429  0.017
55067 -0.04766035 -0.00101017
  -0.06816166  0.02149509  0.00999479 -0.010
61224]]
Cost =  1.4965837253300542e-08 when steps =
10000
```

```
[0.22978185]
[[ 0.10092413  0.65463109 -0.22145868 -0.502
38747  0.19710322 -0.02143491
  -0.04733149 -0.04696865 -0.09038744  0.016
42319 -0.10926296 -0.00198095
  -0.09703622  0.01800964  0.00990038  0.051
78331]]
Cost =  7.222094996089207e-10 when steps =
15000
[0.25524512]
[[ 0.11212958  0.67879958 -0.2464422  -0.538
68568  0.1970184  -0.02048992
  -0.04645502 -0.04283705 -0.0871655   0.015
99315 -0.14953009 -0.00244049
  -0.11748669  0.01673083  0.01003847  0.085
06229]]
Cost =  2.012485785926386e-11 when steps =
20000
[0.26207477]
[[ 0.11300712  0.68002911 -0.24848692 -0.542
25644  0.19694709 -0.01989265
  -0.04650889 -0.04240449 -0.08662748  0.015
88009 -0.16291725 -0.00257164
  -0.1244264   0.0164054   0.01010262  0.095
40602]]
Cost =  2.628279429285786e-13 when steps =
25000
[0.26333501]
[[ 0.11306472  0.67999845 -0.24863594 -0.542
61308  0.19693155 -0.01976558
  -0.046533   -0.0423578  -0.0865442   0.015
85934 -0.16551433 -0.00259623
  -0.12577826  0.01634621  0.01011578  0.097
3842 ]]
Cost =  1.47110883146531O7e-15 when steps =
30000
[0.26348628]
```

```
[[ 0.1130699   0.6799905  -0.24865003 -0.542
65087  0.19692964 -0.01975005
  -0.04653613 -0.04235274 -0.08653446  0.015
85685 -0.16582817 -0.00259918
  -0.1259417   0.01633912  0.01011739  0.097
62279]]
[1.7138489230002227e-07, 1.4965837253300542e
-08, 7.222094996089207e-10, 2.01248578592638
6e-11, 2.628279429285786e-13, 1.471108831465
3107e-15]
```

In [71]:

```
steps = np.arange(5000,30001,5000)



plt.plot(steps, CostValues_9_validation, color='red', lab
el='0.9')
plt.plot(steps, CostValues_8_validation, color='green', l
abel='0.8')
plt.plot(steps, CostValues_7_validation, color='orange',
label='0.7')
plt.plot(steps, CostValues_6_validation, color='blue', la
bel='0.6')


plt.xlabel('Steps')
plt.ylabel('Cost')
plt.legend()
plt.title('Linear Regression Gradient Descend for validat
ion data set', y=-0.25)
plt.show()
```



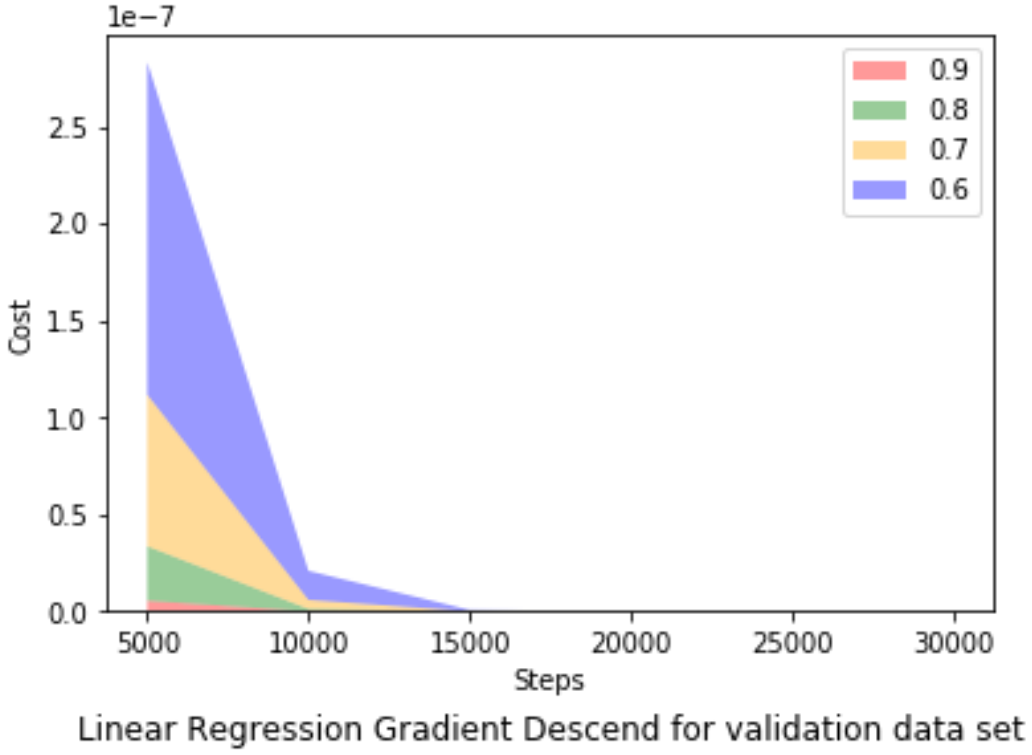
Linear Regression Gradient Descend for validation data set

In [72]:

```
steps = np.arange(5000,30001,5000)
y = [CostValues_9_validation,CostValues_8_validation,Cost
Values_7_validation,CostValues_6_validation]

pal = ["red","green","orange","blue"]
plt.stackplot(steps,y, labels=['0.9','0.8','0.7','0.6'],
colors=pal, alpha=0.4 )
plt.legend(loc='upper right')
plt.xlabel('Steps')
plt.ylabel('Cost')
plt.title('Linear Regression Gradient Descend for validat
ion data set', y=-0.25)
```

Out[72]:

```
Text(0.5, -0.25, 'Linear Regression Gradient
Descend for validation data set')
```



Linear Regression Gradient Descend for validation data set

In [73]:

```
beta_0 = 0.25349326
beta_coefficients = [9.26409139e-02,  6.46469752e-01, -2.
16760149e-01, -5.04050638e-01,
   2.10412858e-01, -1.14714486e-02, -6.23763603e-02, -6.6
7453812e-02,
  -7.17092029e-02,  2.05145351e-02, -1.84622478e-01,  1.4
4494922e-05,
  -1.22493728e-01,  2.06153003e-02, 1.05192213e-02,  1.07
238117e-01]

mse = (1/len(X_Train)) * np.sum( (X_Test * beta_coefficie
nts + beta_0 ) -  Y_Test) ** 2
print(mse)
```

```
16557.31436129327
```

In [75]:

```
#setting all betas to be 0.1
main_beta = np.array(np.repeat(0.1, 16)).reshape(1,16)
beta_0 = 0.1
totalRows = len(X_Train) #Total number of rows in the dat
afram
learningRate = 0.6
CostValues_6_training = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Train * (X_Train @ main_beta.T - Y_Train + bet
a_0), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Train @ main_beta.T - Y_Train + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Train @ main
_beta.T + beta_0 - Y_Train) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_6_training.append(cost)
    print(beta_0)
    print(main_beta)

print(CostValues_6_training)

main_beta = np.array(np.repeat(0.1, 16)).reshape(1,16)
beta_0 = 0.1
totalRows = len(X_Train) #Total number of rows in the dat
afram
learningRate = 0.7
CostValues_7_training = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Train * (X_Train @ main_beta.T - Y_Train + bet
a_0), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
```

```
um((X_Train @ main_beta.T - Y_Train + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Train @ main
_beta.T + beta_0 - Y_Train) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_7_training.append(cost)
    print(beta_0)
    print(main_beta)

print(CostValues_7_training)

main_beta = np.array(np.repeat(0.1, 16)).reshape(1,16)
beta_0 = 0.1
totalRows = len(X_Train) #Total number of rows in the dat
afram
learningRate = 0.8
CostValues_8_training = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Train * (X_Train @ main_beta.T - Y_Train + bet
a_0), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Train @ main_beta.T - Y_Train + beta_0), axis=0)
    cost = (0.5 * (1/totalRows))* (np.sum(X_Train @ main_
beta.T + beta_0 - Y_Train) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_8_training.append(cost)
    print(beta_0)
    print(main_beta)

print(CostValues_8_training)

main_beta = np.array(np.repeat(0.1, 16)).reshape(1,16)
beta_0 = 0.1
totalRows = len(X_Train) #Total number of rows in the dat
afram
learningRate = 0.9
```

```
CostValues_9_training = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Train * (X_Train @ main_beta.T - Y_Train + bet
a_0), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Train @ main_beta.T - Y_Train + beta_0), axis=0)
    cost = (0.5 * (1/totalRows))* (np.sum(X_Train @ main_
beta.T + beta_0 - Y_Train) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_9_training.append(cost)
    print(beta_0)
    print(main_beta)

print(CostValues_9_training)
```

```
Cost =  2.1118442222131228e-07 when steps =
5000
[0.18390075]
[[ 0.02316922  0.48548785 -0.07004888 -0.302
10887  0.21132301 -0.00810985
  -0.07037335 -0.08705004 -0.08333698  0.022
09082 -0.1155483   0.00096881
  -0.08979124  0.02466269  0.0107644   0.039
72843]]
Cost =  1.671212082362786e-08 when steps =
10000
[0.23052415]
[[ 8.12328270e-02  6.21689216e-01 -1.9217299
2e-01 -4.68877614e-01
   2.10668500e-01 -1.21085674e-02 -6.3577189
2e-02 -7.06421141e-02
  -7.42007812e-02  2.09707732e-02 -1.4898306
3e-01  4.15152350e-04
  -1.04488998e-01  2.18329084e-02  1.0444380
7e-02  7.77608444e-02]]
Cost =  7.537384711069361e-10 when steps =
15000
[0.24777422]
[[ 9.18093081e-02  6.45210165e-01 -2.1489864
5e-01 -5.00883962e-01
   2.10500264e-01 -1.19353412e-02 -6.2374451
6e-02 -6.71337280e-02
  -7.20998664e-02  2.06168468e-02 -1.7343486
5e-01  1.23187241e-04
  -1.16723271e-01  2.08922303e-02  1.0473982
4e-02  9.85861033e-02]]
Cost =  2.3810698975839886e-11 when steps =
20000
[0.2524207]
[[ 9.25994345e-02  6.46513365e-01 -2.1665383
6e-01 -5.03775378e-01
   2.10430606e-01 -1.15758450e-02 -6.2358852
```

```
1e-02 -6.67850473e-02
  -7.17693785e-02  2.05330798e-02 -1.8239246
4e-01  3.53534166e-05
  -1.21338150e-01  2.06657410e-02  1.0509221
1e-02  1.05540697e-01]]
Cost =  3.9538402406464404e-13 when steps =
25000
[0.25335405]
[[ 9.26377393e-02  6.46480812e-01 -2.1675103
5e-01 -5.04021058e-01
   2.10415188e-01 -1.14853338e-02 -6.2373757
0e-02 -6.67498901e-02
  -7.17167608e-02  2.05169298e-02 -1.8433049
7e-01  1.71727449e-05
  -1.22342328e-01  2.06218184e-02  1.0517894
2e-02  1.07016363e-01]]
Cost =  2.991132599176649e-15 when steps =
30000
[0.2534813]
[[ 9.26406565e-02  6.46470739e-01 -2.1675939
8e-01 -5.04048138e-01
   2.10413058e-01 -1.14726436e-02 -6.2376134
3e-02 -6.67457647e-02
  -7.17098507e-02  2.05147409e-02 -1.8459737
5e-01  1.46837928e-05
  -1.22480710e-01  2.06158603e-02  1.0519107
2e-02  1.07219055e-01]]
[2.1118442222131228e-07, 1.671212082362786e-
08, 7.537384711069361e-10, 2.381069897583988
6e-11, 3.9538402406464404e-13, 2.99113259917
6649e-15]
Cost =  8.40446301233546e-08 when steps =  5
000
[0.19717604]
[[ 0.03372362  0.51003721 -0.09197047 -0.332
90834  0.21100704 -0.00845468
  -0.06950452 -0.08430785 -0.0813776   0.021
```

```
79219 -0.1318135   0.00076369
  -0.09772469  0.02392435  0.01072917  0.054
35191]]
Cost =  4.086954008981725e-09 when steps =
10000
[0.23837375]
[[ 8.54865813e-02  6.31052387e-01 -2.0135958
7e-01 -4.81929518e-01
   2.10590463e-01 -1.19598580e-02 -6.3113929
8e-02 -6.91841998e-02
  -7.33036162e-02  2.08094454e-02 -1.6067475
4e-01  2.70213708e-04
  -1.10380710e-01  2.14049014e-02  1.0460497
4e-02  8.75427457e-02]]
Cost =  1.1629990231723983e-10 when steps =
15000
[0.25048744]
[[ 9.22983818e-02  6.46043228e-01 -2.1598738
3e-01 -5.02656568e-01
   2.10460747e-01 -1.17321136e-02 -6.2361274
9e-02 -6.69182713e-02
  -7.19028820e-02  2.05672058e-02 -1.7862041
5e-01  7.04149880e-05
  -1.19394670e-01  2.07584007e-02  1.0493392
5e-02  1.02619039e-01]]
Cost =  2.1286481798665043e-12 when steps =
20000
[0.253069]
[[ 9.26282658e-02  6.46496782e-01 -2.1672675
1e-01 -5.03953176e-01
   2.10420033e-01 -1.15136481e-02 -6.2368884
3e-02 -6.67594703e-02
  -7.17324637e-02  2.05217724e-02 -1.8373427
6e-01  2.25181728e-05
  -1.22033464e-01  2.06350397e-02  1.0515107
8e-02  1.06562891e-01]]
Cost =  1.719396040208547e-14 when steps =
```

```
25000
[0.25345511]
[[ 9.26401149e-02  6.46473009e-01 -2.1675786
3e-01 -5.04042814e-01
   2.10413507e-01 -1.14752944e-02 -6.2375637
4e-02 -6.67465508e-02
  -7.17112561e-02  2.05151841e-02 -1.8454221
8e-01  1.51768402e-05
  -1.22452122e-01  2.06170711e-02  1.0518847
0e-02  1.07177177e-01]]
Cost =  5.401104493321829e-17 when steps =
30000
[0.25349128]
[[ 9.26408731e-02  6.46469922e-01 -2.1676003
1e-01 -5.04050233e-01
   2.10412892e-01 -1.14716478e-02 -6.2376322
7e-02 -6.67454421e-02
  -7.17093094e-02  2.05145688e-02 -1.8461831
7e-01  1.44873804e-05
  -1.22491570e-01  2.06153922e-02  1.0519202
0e-02  1.07234958e-01]]
[8.40446301233546e-08, 4.086954008981725e-0
9, 1.1629990231723983e-10, 2.128648179866504
3e-12, 1.719396040208547e-14, 5.401104493321
829e-17]
Cost =  2.5429112133742706e-08 when steps =
5000
[0.20901791]
[[ 0.04271215  0.53061193 -0.11086648 -0.359
35757  0.21078547 -0.00863692
  -0.06863894 -0.08173311 -0.07970087  0.021
52954 -0.14670471  0.00056819
  -0.10505092  0.02325847  0.01071008  0.067
59715]]
Cost =  7.352461766456169e-10 when steps =
10000
[0.24397081]
```

```
[[ 8.81694973e-02  6.36861062e-01 -2.0715382
5e-01 -4.90242476e-01
   2.10528462e-01 -1.17940071e-02 -6.2835870
7e-02 -6.82562625e-02
  -7.27061821e-02  2.06978842e-02 -1.6944936
0e-01  1.69520159e-04
  -1.14820963e-01  2.11074811e-02  1.0478722
8e-02  9.47726628e-02]]
Cost =  1.2844495248280417e-11 when steps =
15000
[0.25199808]
[[ 9.25007389e-02  6.46333622e-01 -2.1644236
3e-01 -5.03444828e-01
   2.10437450e-01 -1.16069411e-02 -6.2364372
6e-02 -6.68207221e-02
  -7.18016733e-02  2.05402802e-02 -1.8159550
6e-01  4.16208789e-05
  -1.20929997e-01  2.06854820e-02  1.0505573
6e-02  1.04915736e-01]]
Cost =  1.3257161631081417e-13 when steps =
20000
[0.25333608]
[[ 9.26370135e-02  6.46481909e-01 -2.1674970
0e-01 -5.04017079e-01
   2.10415567e-01 -1.14873278e-02 -6.2373477
7e-02 -6.67501796e-02
  -7.17176882e-02  2.05171828e-02 -1.8429184
9e-01  1.73603107e-05
  -1.22322423e-01  2.06225411e-02  1.0517646
1e-02  1.06986944e-01]]
Cost =  5.0886409850188885e-16 when steps =
25000
[0.25348366]
[[ 9.26407256e-02  6.46470617e-01 -2.1675961
6e-01 -5.04048726e-01
   2.10413024e-01 -1.14724264e-02 -6.2376176
7e-02 -6.67456602e-02
```

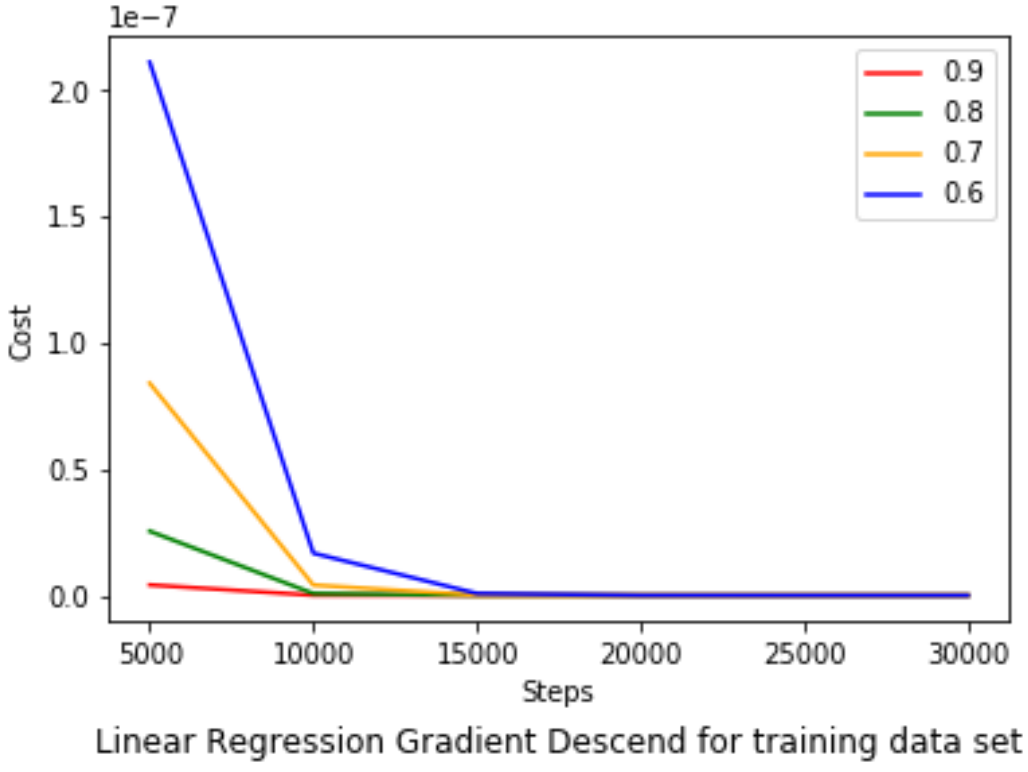
```
  -7.17097158e-02  2.05146967e-02 -1.8460221
7e-01  1.46277039e-05
  -1.22483228e-01  2.06157422e-02  1.0519124
4e-02  1.07222736e-01]]
Cost =  6.495704088374019e-19 when steps =
30000
[0.25349308]
[[ 9.26409102e-02  6.46469768e-01 -2.1676013
9e-01 -5.04050601e-01
   2.10412861e-01 -1.14714667e-02 -6.2376356
9e-02 -6.67453866e-02
  -7.17092126e-02  2.05145382e-02 -1.8462210
1e-01  1.44529051e-05
  -1.22493532e-01  2.06153086e-02  1.0519219
6e-02  1.07237831e-01]]
[2.5429112133742706e-08, 7.352461766456169e-
10, 1.2844495248280417e-11, 1.32571616310814
17e-13, 5.0886409850188885e-16, 6.4957040883
74019e-19]
Cost =  4.142276843868136e-09 when steps =
5000
[0.21951211]
[[ 5.03569200e-02  5.47955116e-01 -1.2706251
7e-01 -3.82008494e-01
   2.10627512e-01 -8.72547920e-03 -6.7838998
2e-02 -7.94210762e-02
  -7.82650200e-02  2.13016443e-02 -1.6023704
1e-01  3.93368760e-04
  -1.11740312e-01  2.26704263e-02  1.0703176
0e-02  7.95191960e-02]]
Cost =  6.64183865808826e-11 when steps =  1
0000
[0.24790215]
[[ 8.98581256e-02  6.40453981e-01 -2.1079857
6e-01 -4.95522734e-01
   2.10480983e-01 -1.16430124e-02 -6.2669797
6e-02 -6.76685706e-02
```

```
  -7.23105350e-02  2.06217026e-02 -1.7586309
0e-01  1.00948856e-04
  -1.18075941e-01  2.09034247e-02  1.0495621
9e-02  9.99977620e-02]]
Cost =  6.738329208707139e-13 when steps =
15000
[0.25281307]
[[ 9.25846100e-02  6.46427085e-01 -2.1663259
0e-01 -5.03797468e-01
   2.10424326e-01 -1.15347962e-02 -6.2369807
7e-02 -6.67766080e-02
  -7.17502229e-02  2.05260730e-02 -1.8323379
6e-01  2.64718666e-05
  -1.21776267e-01  2.06468492e-02  1.0512728
7e-02  1.06174429e-01]]
Cost =  3.814845821846937e-15 when steps =
20000
[0.25344062]
[[ 9.26397693e-02  6.46474295e-01 -2.1675708
1e-01 -5.04039973e-01
   2.10413782e-01 -1.14768343e-02 -6.2375372
3e-02 -6.67468748e-02
  -7.17120116e-02  2.05154109e-02 -1.8451133
5e-01  1.53969328e-05
  -1.22436156e-01  2.06177019e-02  1.0518677
6e-02  1.07153720e-01]]
Cost =  6.813940246204127e-18 when steps =
25000
[0.2534912]
[[ 9.26408759e-02  6.46469947e-01 -2.1676004
4e-01 -5.04050239e-01
   2.10412894e-01 -1.14716610e-02 -6.2376320
5e-02 -6.67454378e-02
  -7.17093124e-02  2.05145695e-02 -1.8461810
9e-01  1.44867086e-05
  -1.22491464e-01  2.06153944e-02  1.0519199
9e-02  1.07234800e-01]]
```

```
Cost =  3.4595286589349745e-21 when steps =
30000
[0.25349338]
[[ 9.26409161e-02  6.46469741e-01 -2.1676015
5e-01 -5.04050661e-01
   2.10412856e-01 -1.14714363e-02 -6.2376362
7e-02 -6.67453779e-02
  -7.17091966e-02  2.05145331e-02 -1.8462273
1e-01  1.44473350e-05
  -1.22493859e-01  2.06152948e-02  1.0519222
6e-02  1.07238309e-01]]
[4.1422768438681 36e-09, 6.64183865808826e-1
1, 6.7383292087071 39e-13, 3.814845821846937e
-15, 6.813940246204127e-18, 3.45952865893497
45e-21]
```

In [77]:

```
steps = np.arange(5000,30001,5000)

plt.plot(steps, CostValues_9_training, color='red', label
='0.9')
plt.plot(steps, CostValues_8_training, color='green', lab
el='0.8')
plt.plot(steps, CostValues_7_training, color='orange', la
bel='0.7')
plt.plot(steps, CostValues_6_training, color='blue', labe
l='0.6')


plt.xlabel('Steps')
plt.ylabel('Cost')
plt.legend()
plt.title('Linear Regression Gradient Descend for trainin
g data set', y=-0.25)
plt.show()
```



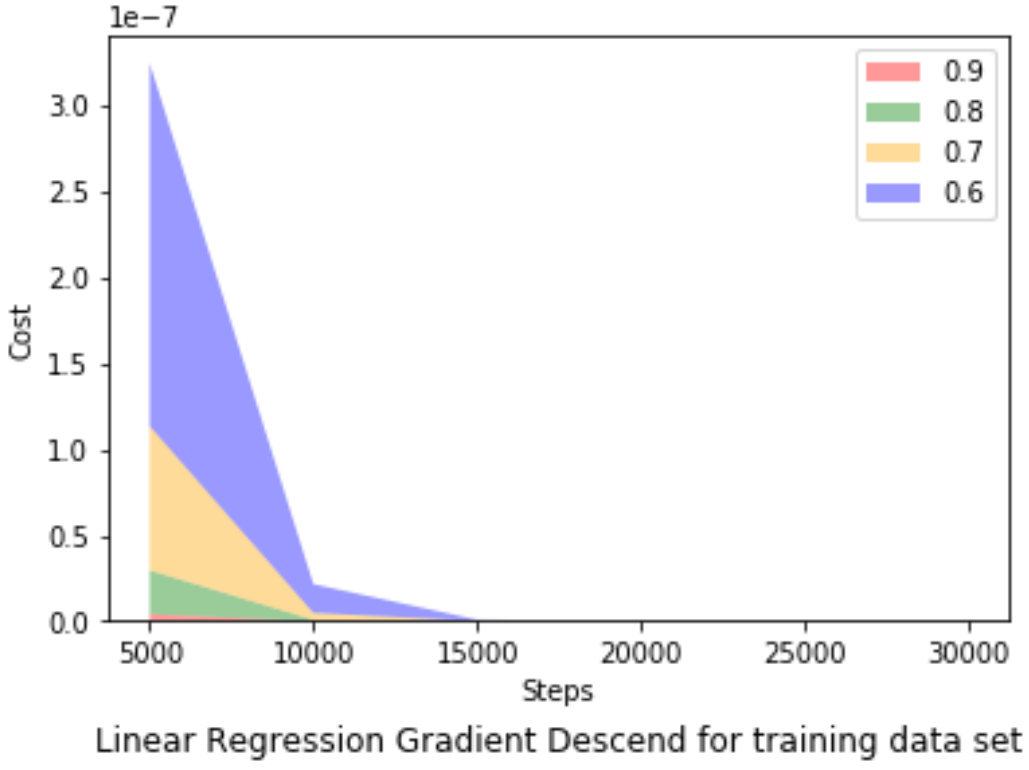
Linear Regression Gradient Descend for training data set

In [78]:

```
steps = np.arange(5000,30001,5000)
y = [CostValues_9_training,CostValues_8_training,CostValu
es_7_training,CostValues_6_training]

pal = ["red","green","orange","blue"]
plt.stackplot(steps,y, labels=['0.9','0.8','0.7','0.6'],
colors=pal, alpha=0.4 )
plt.legend(loc='upper right')
plt.xlabel('Steps')
plt.ylabel('Cost')
plt.title('Linear Regression Gradient Descend for trainin
g data set', y=-0.25)
```

Out[78]:

```
Text(0.5, -0.25, 'Linear Regression Gradient
Descend for training data set')
```



In [76]:

```
main_beta = np.array(np.repeat(0.1, 16)).reshape(1,16)
beta_0 = 0.1
totalRows = len(X_Test) #Total number of rows in the data
fram
learningRate = 0.9
CostValues_9_validation = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Test * (X_Test @ main_beta.T - Y_Test + beta_0
), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Test @ main_beta.T - Y_Test + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Test @ main_
beta.T + beta_0 - Y_Test) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_9_validation.append(cost)
    print(beta_0)
    print(main_beta)
print(CostValues_9_validation)


main_beta = np.array(np.repeat(0.1, 16)).reshape(1,16)
beta_0 = 0.1
learningRate = 0.8
CostValues_8_validation = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Test * (X_Test @ main_beta.T - Y_Test + beta_0
), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Test @ main_beta.T - Y_Test + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Test @ main_
beta.T + beta_0 - Y_Test) ** 2)
```

```
    print("Cost = ",cost, "when steps = ", j)
    CostValues_8_validation.append(cost)
    print(beta_0)
    print(main_beta)
print(CostValues_8_validation)


main_beta = np.array(np.repeat(0.1, 16)).reshape(1,16)
beta_0 = 0.1
learningRate = 0.7
CostValues_7_validation = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Test * (X_Test @ main_beta.T - Y_Test + beta_0
), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
um((X_Test @ main_beta.T - Y_Test + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Test @ main_
beta.T + beta_0 - Y_Test) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_7_validation.append(cost)
    print(beta_0)
    print(main_beta)
print(CostValues_7_validation)


main_beta = np.array(np.repeat(0.1, 16)).reshape(1,16)
beta_0 = 0.1
learningRate = 0.6
CostValues_6_validation = []
for j in range(5000,30001,5000):
    for i in range(0,j) :
        main_beta = main_beta - (learningRate/totalRows)
* np.sum(X_Test * (X_Test @ main_beta.T - Y_Test + beta_0
), axis=0)
        beta_0 = beta_0 - (learningRate/totalRows) * np.s
```

```
um((X_Test @ main_beta.T - Y_Test + beta_0), axis=0)
    cost = (0.5 * (1/totalRows)) * (np.sum(X_Test @ main_
beta.T + beta_0 - Y_Test) ** 2)
    print("Cost = ",cost, "when steps = ", j)
    CostValues_6_validation.append(cost)
    print(beta_0)
    print(main_beta)
print(CostValues_6_validation)
```

```
Cost =  1.8723675973008893e-09 when steps =
5000
[0.23005582]
[[ 0.07007188  0.57866091 -0.15440097 -0.413
60959  0.19651771 -0.0165801
  -0.05145554 -0.05640127 -0.09508217  0.016
42744 -0.14654111 -0.00219327
  -0.11807996  0.01818271  0.01025814  0.072
16238]]
Cost =  2.580776433939215e-11 when steps =
10000
[0.25844647]
[[ 0.11040163  0.67412351 -0.24275238 -0.534
20761  0.19694243 -0.01988013
  -0.04677154 -0.04328946 -0.08723633  0.015
93937 -0.158325   -0.00251958
  -0.12216709  0.01658872  0.01009103  0.091
27547]]
Cost =  2.075042274927082e-13 when steps =
15000
[0.26295531]
[[ 0.11301688  0.67993493 -0.24852719 -0.542
42132  0.19693607 -0.01980032
  -0.04652997 -0.04237947 -0.08657353  0.015
86535 -0.16476176 -0.00258979
  -0.12538872  0.01636365  0.01011178  0.096
80314]]
Cost =  8.557609589629237e-16 when steps =
20000
[0.26346256]
[[ 0.11306908  0.6799919  -0.248648   -0.542
64519  0.19692998 -0.01975257
  -0.04653564 -0.0423534  -0.08653596  0.015
85722 -0.16577851 -0.00259877
  -0.12591588  0.01634019  0.01011711  0.097
58504]]
Cost =  1.02910839646087e-18 when steps =  2
```

```
5000
[0.26349726]
[[ 0.11307027  0.67998991 -0.24865105 -0.542
6536   0.19692951 -0.01974892
  -0.04653636 -0.04235237 -0.08653376  0.015
85667 -0.16585096 -0.0025994
  -0.12595357  0.0163386   0.0101175   0.097
64012]]
Cost =  3.2508798486743966e-22 when steps =
30000
[0.26349849]
[[ 0.11307031  0.67998983 -0.24865115 -0.542
65389  0.19692949 -0.01974879
  -0.04653638 -0.04235234 -0.08653368  0.015
85665 -0.16585352 -0.00259942
  -0.12595491  0.01633854  0.01011751  0.097
64207]]
[1.8723675973008893e-09, 2.580776433939215e-
11, 2.0750422749270826e-13, 8.55760958962923
7e-16, 1.02910839646087e-18, 3.2508798486743
966e-22]
Cost =  1.1662970073173089e-08 when steps =
5000
[0.21950004]
[[ 0.0621353   0.56041372 -0.13693946 -0.389
09073  0.19652762 -0.01641707
  -0.05219088 -0.05906209 -0.0969529   0.016
61495 -0.13386975 -0.00201081
  -0.11186614  0.01874324  0.01025188  0.060
57136]]
Cost =  2.954350404014773e-10 when steps =
10000
[0.25473855]
[[ 0.10872953  0.67050801 -0.23903001 -0.528
81063  0.19695462 -0.02001318
  -0.04690479 -0.04390097 -0.08771108  0.016
00164 -0.15250026 -0.00245405
```

```
  -0.11921234  0.01677436  0.01007083  0.086
4487 ]]
Cost =  4.236658699107406e-12 when steps =
15000
[0.26226192]
[[ 0.11293573  0.67983096 -0.24833841 -0.542
08381  0.19694376 -0.01986294
  -0.04652347 -0.04241981 -0.08662672  0.015
87668 -0.16338933 -0.00257689
  -0.12467716  0.0163963   0.01010495  0.095
74497]]
Cost =  3.31924060248372e-14 when steps =  2
0000
[0.2633853]
[[ 0.11306611  0.67999532 -0.24864027 -0.542
62521  0.19693098 -0.01976055
  -0.0465341  -0.04235595 -0.08654095  0.015
85848 -0.16561813 -0.00259732
  -0.12583242  0.01634377  0.01011626  0.097
46304]]
Cost =  9.0125553859819e-17 when steps =  25
000
[0.2634926]
[[ 0.11307012  0.6799902  -0.24865065 -0.542
65249  0.19692957 -0.01974941
  -0.04653626 -0.04235252 -0.08653405  0.015
85675 -0.16584123 -0.00259931
  -0.12594851  0.01633882  0.01011745  0.097
63273]]
Cost =  7.593920048179569e-20 when steps =
30000
[0.26349834]
[[ 0.1130703   0.67998984 -0.24865114 -0.542
65385  0.19692949 -0.01974881
  -0.04653638 -0.04235234 -0.08653369  0.015
85665 -0.16585321 -0.00259942
  -0.12595475  0.01633855  0.01011751  0.097
```

```
64183]]
[1.1662970073173089e-08, 2.954350404014773e-
10, 4.236658699107406e-12, 3.31924060248372e
-14, 9.0125553859819e-17, 7.593920048179569e
-20]
Cost =  3.916591327823445e-08 when steps =
5000
[0.20750492]
[[ 0.052806    0.53876297 -0.11648689 -0.360
34486  0.19656472 -0.01613585
  -0.05300668 -0.0621102  -0.09915096  0.016
83145 -0.11987253 -0.0018032
  -0.10503974  0.0193836   0.01025571  0.047
62813]]
Cost =  1.6909121805333585e-09 when steps =
10000
[0.24936786]
[[ 0.10604992  0.66462345 -0.23306117 -0.520
22811  0.19696385 -0.02015779
  -0.04713243 -0.0448783  -0.08844022  0.016
09339 -0.14441001 -0.00235608
  -0.11512304  0.01704835  0.01004681  0.079
65407]]
Cost =  4.0779217197417213e-11 when steps =
15000
[0.26093303]
[[ 0.11273959  0.67952918 -0.24788515 -0.541
31568  0.19695708 -0.01997512
  -0.04651644 -0.04251193 -0.08673528  0.015
89869 -0.160815   -0.00255151
  -0.12334406  0.0164598   0.01009277  0.093
74897]]
Cost =  5.897072095111102e-13 when steps =
20000
[0.26317719]
[[ 0.11305677  0.68000094 -0.2486162  -0.542
56703  0.19693358 -0.01978163
```

```
  -0.0465301  -0.04236349 -0.08655466  0.015
86189 -0.16518865 -0.00259331
  -0.12560889  0.01635351  0.01011406  0.097
13596]]
Cost =  3.5354612568729343e-15 when steps =
25000
[0.26347352]
[[ 0.11306948  0.67999126 -0.24864894 -0.542
64781  0.19692981 -0.01975138
  -0.04653586 -0.04235313 -0.08653528  0.015
85706 -0.16580158 -0.00259895
  -0.12592786  0.01633971  0.01011724  0.097
60259]]
Cost =  7.763959682601583e-18 when steps =
30000
[0.26349734]
[[ 0.11307027  0.6799899  -0.24865105 -0.542
65361  0.1969295  -0.01974891
  -0.04653636 -0.04235237 -0.08653375  0.015
85667 -0.16585113 -0.0025994
  -0.12595366  0.0163386   0.0101175   0.097
64025]]
[3.916591327823445e-08, 1.6909121805333585e-
09, 4.0779217197417213e-11, 5.89707209511110
2e-13, 3.5354612568729343e-15, 7.76395968260
1583e-18]
Cost =  1.0030654674507757e-07 when steps =
5000
[0.19395743]
[[ 0.04186989  0.51295294 -0.09264874 -0.326
71824  0.19664742 -0.01564155
  -0.05386302 -0.06552339 -0.10173776  0.017
0766  -0.10454432 -0.0015792
  -0.09763677  0.02010254  0.01027375  0.033
28499]]
Cost =  7.099461513762172e-09 when steps =
10000
```

```
[0.24171696]
[[ 0.10176477  0.65506935 -0.22351581 -0.506
62014  0.1969628  -0.02027921
  -0.04751971 -0.0464327  -0.08955345  0.016
22635 -0.13349988 -0.0022127
  -0.10963726  0.01744666  0.0100213   0.070
32103]]
Cost =  2.7957914300073496e-10 when steps =
15000
[0.25847179]
[[ 0.11226379  0.67868835 -0.24679471 -0.539
56037  0.19697831 -0.02016244
  -0.04651855 -0.04272171 -0.08695429  0.015
94004 -0.1561957  -0.00250349
  -0.12095701  0.01657939  0.01007235  0.090
13599]]
Cost =  7.257577667673512e-12 when steps =
20000
[0.26264626]
[[ 0.11302678  0.67999927 -0.24854037 -0.542
39946  0.19693989 -0.01983398
  -0.0465207  -0.04238537 -0.0865907   0.015
87069 -0.16410281 -0.00258281
  -0.12504385  0.01637861  0.01010867  0.096
30738]]
Cost =  9.374619662158974e-14 when steps =
25000
[0.26340089]
[[ 0.11306691  0.67999485 -0.24864195 -0.542
62937  0.19693072 -0.01975881
  -0.04653437 -0.04235561 -0.08653997  0.015
85826 -0.16565108 -0.00259751
  -0.12584948  0.01634312  0.01011648  0.097
48815]]
Cost =  5.242315972992695e-16 when steps =
30000
[0.26349121]
```

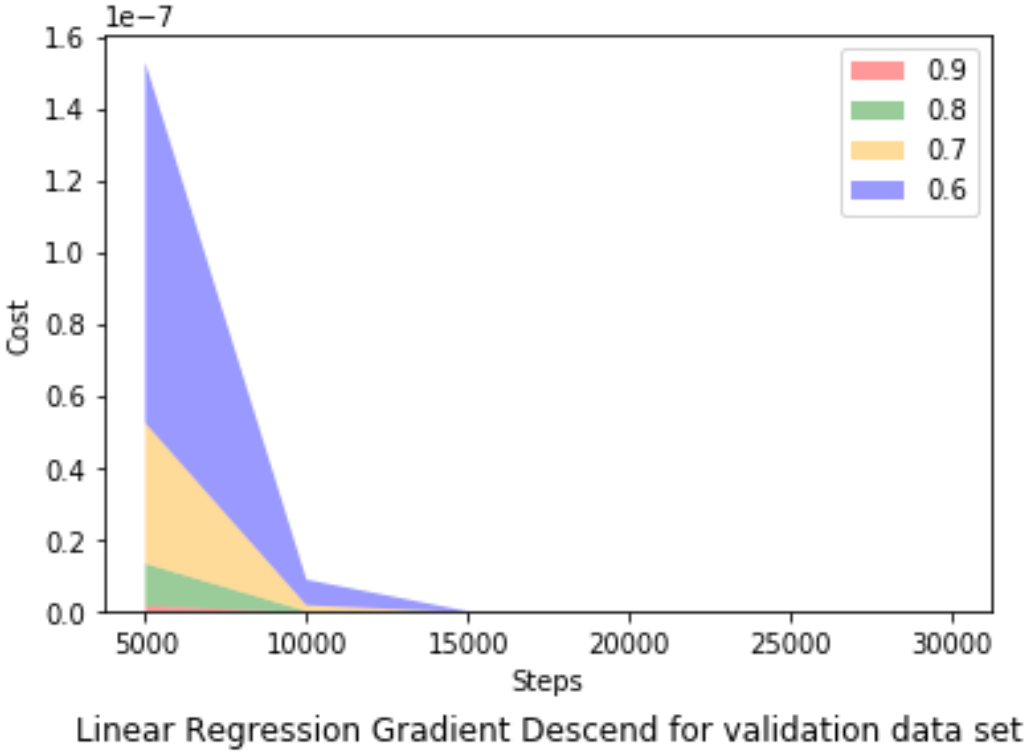
```
[[ 0.11307006  0.67999023 -0.24865048 -0.542
65209  0.19692958 -0.01974954
  -0.04653623 -0.04235258 -0.08653415  0.015
85677 -0.1658384  -0.00259928
  -0.12594704  0.01633889  0.01011744  0.097
63058]]
[1.0030654674507757e-07, 7.099461513762172e-
09, 2.7957914300073496e-10, 7.25757766767351
2e-12, 9.374619662158974e-14, 5.242315972992
695e-16]
```

In [79]:

```
steps = np.arange(5000,30001,5000)
y = [CostValues_9_validation,CostValues_8_validation,Cost
Values_7_validation,CostValues_6_validation]

pal = ["red","green","orange","blue"]
plt.stackplot(steps,y, labels=['0.9','0.8','0.7','0.6'],
colors=pal, alpha=0.4 )
plt.legend(loc='upper right')
plt.xlabel('Steps')
plt.ylabel('Cost')
plt.title('Linear Regression Gradient Descend for validat
ion data set', y=-0.25)
```

Out[79]:

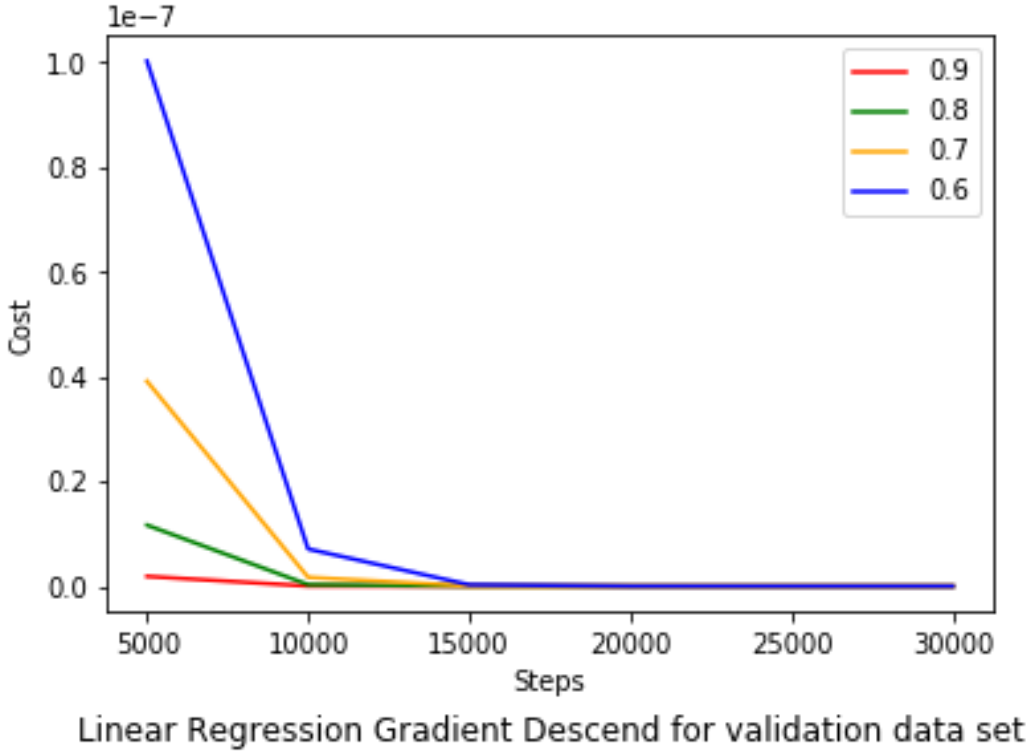
```
Text(0.5, -0.25, 'Linear Regression Gradient
Descend for validation data set')
```



Linear Regression Gradient Descend for validation data set

In [80]:

```
steps = np.arange(5000,30001,5000)

plt.plot(steps, CostValues_9_validation, color='red', lab
el='0.9')
plt.plot(steps, CostValues_8_validation, color='green', l
abel='0.8')
plt.plot(steps, CostValues_7_validation, color='orange',
label='0.7')
plt.plot(steps, CostValues_6_validation, color='blue', la
bel='0.6')


plt.xlabel('Steps')
plt.ylabel('Cost')
plt.legend()
plt.title('Linear Regression Gradient Descend for validat
ion data set', y=-0.25)
plt.show()
```



Linear Regression Gradient Descend for validation data set

In [81]:

```
beta_0 = 0.25349338
beta_coefficients = [ 9.26409161e-02,  6.46469741e-01, -
2.16760155e-01, -5.04050661e-01,
   2.10412856e-01, -1.14714363e-02, -6.23763627e-02, -6.6
7453779e-02,
  -7.17091966e-02,  2.05145331e-02, -1.84622731e-01,  1.4
4473350e-05,
  -1.22493859e-01,  2.06152948e-02,  1.05192226e-02,  1.0
7238309e-01]

mse = (1/len(X_Train)) * np.sum( (X_Test * beta_coefficie
nts + beta_0 ) -  Y_Test) ** 2
print(mse)
```

```
16557.3377051091
```

In [82]:

```
x_appliances
```

Out[82]:

	T1	RH_1	T2	RH_2	
0	19.890000	47.596667	19.200000	44.790000	19.7
1	19.890000	46.693333	19.200000	44.722500	19.7
2	19.890000	46.300000	19.200000	44.626667	19.7
3	19.890000	46.066667	19.200000	44.590000	19.7
4	19.890000	46.333333	19.200000	44.530000	19.7
5	19.890000	46.026667	19.200000	44.500000	19.7
6	19.890000	45.766667	19.200000	44.500000	19.7
7	19.856667	45.560000	19.200000	44.500000	19.7
8	19.790000	45.597500	19.200000	44.433333	19.7
9	19.856667	46.090000	19.230000	44.400000	19.7
10	19.926667	45.863333	19.356667	44.400000	19.7
11	20.066667	46.396667	19.426667	44.400000	19.7
12	20.133333	48.000000	19.566667	44.400000	19.8
13	20.260000	52.726667	19.730000	45.100000	19.8
14	20.426667	55.893333	19.856667	45.833333	20.0
15	20.566667	53.893333	20.033333	46.756667	20.1
16	20.730000	52.660000	20.166667	47.223333	20.2
17	20.856667	53.660000	20.200000	47.056667	20.2
18	20.890000	51.193333	20.200000	46.330000	20.2
19	20.890000	49.800000	20.200000	46.026667	20.1
20	20.890000	48.433333	20.200000	45.722500	20.1

	T1	RH_1	T2	RH_2	
21	20.963333	47.633333	20.260000	45.530000	20.2
22	21.033333	47.063333	20.290000	45.223333	20.2
23	21.100000	46.596667	20.356667	44.963333	20.2
24	21.133333	46.060000	20.426667	44.760000	20.2
25	21.200000	45.800000	20.500000	44.760000	20.3
26	21.290000	45.900000	20.533333	45.090000	20.3
27	21.356667	45.826667	20.666667	45.163333	20.3
28	21.390000	45.690000	20.700000	45.060000	20.3
29	21.500000	45.333333	20.700000	44.933333	20.3
...	...	...	...	...	
19705	25.033333	48.363333	26.528571	40.595714	28.4
19706	25.166667	48.156667	26.600000	40.940000	28.2
19707	25.323333	47.930000	26.600000	41.012857	28.2
19708	25.390000	47.656667	26.600000	41.036000	28.2
19709	25.500000	47.133333	26.600000	41.000000	28.2
19710	25.500000	47.060000	26.540000	41.000000	28.2
19711	25.600000	46.990000	26.512500	41.203750	28.2
19712	25.600000	46.730000	26.437143	41.384286	28.4
19713	25.566667	46.633333	26.370000	41.378000	28.5
19714	25.500000	46.360000	26.318571	41.264286	28.5
19715	25.500000	46.060000	26.350000	41.000000	28.4
19716	25.500000	45.933333	26.277143	41.000000	28.3
19717	25.500000	45.760000	26.200000	41.000000	28.2

	T1	RH_1	T2	RH_2	
19718	25.500000	45.626667	26.171429	41.000000	28.2
19719	25.500000	45.590000	26.100000	41.000000	28.2
19720	25.500000	45.522500	26.100000	41.051429	28.2
19721	25.500000	45.633333	26.080000	41.196000	28.1
19722	25.500000	45.730000	26.000000	41.428571	28.1
19723	25.500000	45.790000	26.000000	41.590000	28.1
19724	25.500000	45.933333	26.000000	41.652857	28.0
19725	25.426667	46.060000	26.000000	41.700000	28.0
19726	25.500000	46.530000	26.000000	41.725714	27.8
19727	25.500000	47.456667	26.000000	42.320000	27.6
19728	25.600000	47.193333	25.968571	42.528571	27.3
19729	25.533333	46.860000	25.978000	42.534000	27.3
19730	25.566667	46.560000	25.890000	42.025714	27.2
19731	25.500000	46.500000	25.754000	42.080000	27.1
19732	25.500000	46.596667	25.628571	42.768571	27.0
19733	25.500000	46.990000	25.414000	43.036000	26.8
19734	25.500000	46.600000	25.264286	42.971429	26.8

19735 rows × 16 columns

In [83]:

```
#Logistic regression
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
scaler = MinMaxScaler()


x_appliances_logistic = x_appliances
y_appliances_logistic = y_appliances

x_appliances_scaled_logistic = scaler.fit_transform(x_app
liances_logistic)
y_appliances_scaled_logistic = scaler.fit_transform(y_app
liances_logistic)

pd.DataFrame(y_appliances_scaled_logistic).median()

#x_appliances_logistic = np.array(x_appliances)
y_appliances_scaled_logistic = np.where(y_appliances_scal
ed_logistic<0.04,0,1)
```

In [84]:

```
from sklearn.linear_model import LogisticRegression
random.seed(1)
X_Train_logistic, X_Test_logistic, Y_Train_logistic, Y_Te
st_logistic = train_test_split(x_appliances_scaled_logist
ic, y_appliances_scaled_logistic, test_size=0.3, random_s
tate=1)
```

In [85]:

```
#For cross validation
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
model = LogisticRegression(max_iter = 40)

model.fit(X_Train_logistic, Y_Train_logistic)
predicted_classes = model.predict(X_Test_logistic)
accuracy = accuracy_score(Y_Test_logistic.flatten(),predi
cted_classes)
parameters = model.coef_
print(model.coef_)
print(accuracy)
```

```
[[ 2.78419467  9.22171651  0.55770816 -3.188
82771  0.16389383 -4.74210771
  -0.4300216  -0.49701037 -0.71114709  1.352
71357 -0.2412339  -0.80274879
  -1.36159686  0.41937844  0.15751819 -0.393
52681]]
0.6914372572200642

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\l
inear_model\logistic.py:432: FutureWarning:
Default solver will be changed to 'lbfgs' in
0.22. Specify a solver to silence this warni
ng.
  FutureWarning)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
```

In [111]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

logi_model = SGDClassifier(loss='log', alpha = 0.01 , max
_iter=10, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy0 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy0)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)


accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.01 , max
_iter=100, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy1 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy1)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)


accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.01 , max
_iter=1000, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy2 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
```

```
accuracy_Values.append(accuracy2)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)

accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.01 , max
_iter=10000, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy3 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy3)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)

accuracy_Values_0_01 = [accuracy0,accuracy1,accuracy2,acc
uracy3]
print(accuracy_Values_0_01)
```

```
[0.6740415470359736, 0.6740415470359736, 0.6
740415470359736, 0.6740415470359736]

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
```

In [112]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

logi_model = SGDClassifier(loss='log', alpha = 0.001 , ma
x_iter=10, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy0 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy0)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)


accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.001 , ma
x_iter=100, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy1 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy1)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)


accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.001 , ma
x_iter=1000, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy2 = accuracy_score(Y_Test_logistic.flatten(),vali
```

```
dation_data)
accuracy_Values.append(accuracy2)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)

accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.001 , ma
x_iter=10000, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy3 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy3)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)

accuracy_Values_0_001 = [accuracy0,accuracy1,accuracy2,ac
curacy3]
print(accuracy_Values_0_001)
```

```
[0.6877216686370545, 0.6836682992737714, 0.6
836682992737714, 0.6836682992737714]
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\l
inear_model\stochastic_gradient.py:561: Conv
ergenceWarning: Maximum number of iteration
reached before convergence. Consider increas
ing max_iter to improve the fit.
  ConvergenceWarning)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
```

```
().
  y = column_or_1d(y, warn=True)
```

In [113]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

logi_model = SGDClassifier(loss='log', alpha = 0.0001 , m
ax_iter=10, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy0 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)


accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.0001 , m
ax_iter=100, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy1 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy1)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)


accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.0001 , m
ax_iter=1000, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy2 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy2)
```

```
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)

accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.0001 , m
ax_iter=10000, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy3 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy3)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)

accuracy_Values_0_0001 = [accuracy0,accuracy1,accuracy2,a
ccuracy3]
print(accuracy_Values_0_0001)
```

```
[0.6900861340989698, 0.6870461070765074, 0.6
870461070765074, 0.6870461070765074]
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\l
inear_model\stochastic_gradient.py:561: Conv
ergenceWarning: Maximum number of iteration
reached before convergence. Consider increas
ing max_iter to improve the fit.
  ConvergenceWarning)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
```

```
().
  y = column_or_1d(y, warn=True)
```

In [114]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

logi_model = SGDClassifier(loss='log', alpha = 0.00001 ,
max_iter=10, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy0 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy0)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)


accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.00001 ,
max_iter=100, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy1 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy1)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)


accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.00001 ,
max_iter=1000, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy2 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
```

```
accuracy_Values.append(accuracy2)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)

accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.00001 ,
max_iter=10000, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy3 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy3)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)

accuracy_Values_0_00001 = [accuracy0,accuracy1,accuracy2,
accuracy3]
print(accuracy_Values_0_00001)
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\l
inear_model\stochastic_gradient.py:561: Conv
ergenceWarning: Maximum number of iteration
reached before convergence. Consider increas
ing max_iter to improve the fit.
  ConvergenceWarning)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)

[0.6828238473230873, 0.6843438608343185, 0.6
843438608343185, 0.6843438608343185]
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
```

In [115]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

logi_model = SGDClassifier(loss='log', alpha = 0.000001 ,
max_iter=10, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy0 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy0)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)


accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.000001 ,
max_iter=100, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy1 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy1)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)


accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.000001 ,
max_iter=1000, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy2 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
```

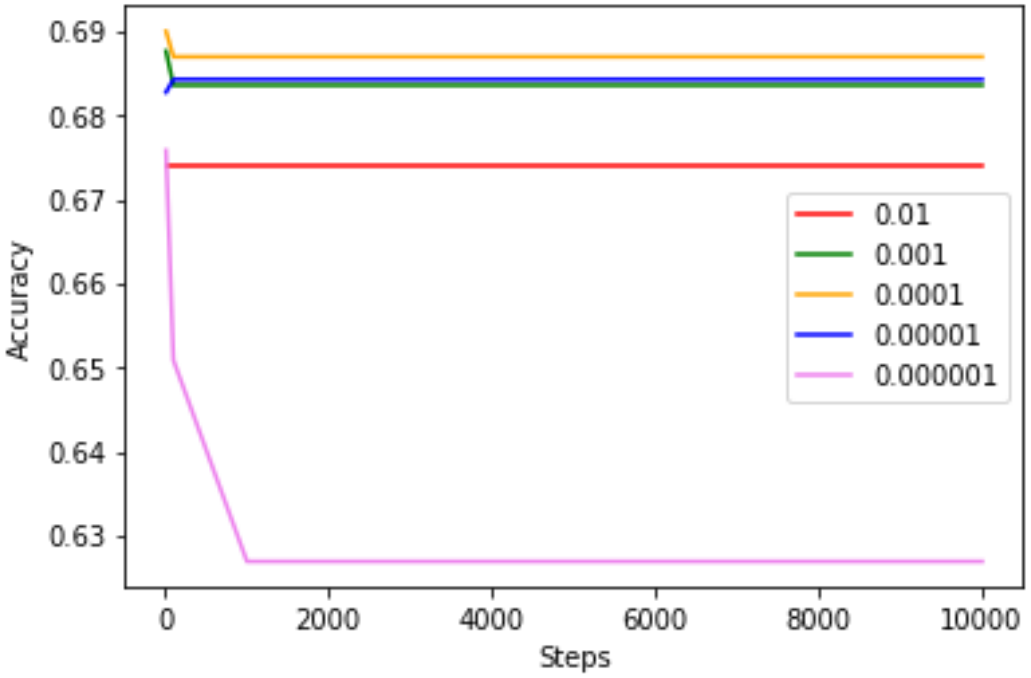
```
accuracy_Values.append(accuracy2)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)

accuracy_Values = []
logi_model = SGDClassifier(loss='log', alpha = 0.000001 ,
max_iter=10000, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
accuracy3 = accuracy_score(Y_Test_logistic.flatten(),vali
dation_data)
accuracy_Values.append(accuracy3)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)

accuracy_Values_0_000001 = [accuracy0,accuracy1,accuracy2
,accuracy3]
print(accuracy_Values_0_000001)
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\l
inear_model\stochastic_gradient.py:561: Conv
ergenceWarning: Maximum number of iteration
reached before convergence. Consider increas
ing max_iter to improve the fit.
  ConvergenceWarning)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\l
inear_model\stochastic_gradient.py:561: Conv
ergenceWarning: Maximum number of iteration
reached before convergence. Consider increas
ing max_iter to improve the fit.
  ConvergenceWarning)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
```

```
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)



[0.6758993413274784, 0.6509035635872319, 0.6
269211281878061, 0.6269211281878061]
```

In [116]:

```
steps = [10, 100, 1000, 10000]

plt.plot(steps, accuracy_Values_0_01, color='red', label=
'0.01')
plt.plot(steps, accuracy_Values_0_001, color='green', lab
el='0.001')
plt.plot(steps, accuracy_Values_0_0001, color='orange', l
abel='0.0001')
plt.plot(steps, accuracy_Values_0_00001, color='blue', la
bel='0.00001')
plt.plot(steps, accuracy_Values_0_000001, color='violet',
label='0.000001')

plt.xlabel('Steps')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Logistic Regression Steps vs Accuracy', y=-0.2
5)
plt.show()
```



Logistic Regression Steps vs Accuracy

In [129]:

```
logi_model = SGDClassifier(loss='log', alpha = 0.0001 , m
ax_iter=10, random_state = 1)
logi_model.fit(X_Train_logistic, Y_Train_logistic)
validation_data = logi_model.predict(X_Test_logistic)
#confusion_matrix()
roc_auc_score(Y_Test_logistic, validation_data)
print(logi_model.coef_)
print(logi_model.intercept_)
```

```
[[ 2.52754395  8.34048411  1.04910764 -2.706
34032  0.25027259 -4.45253276
  -0.30169553 -0.54235959 -0.62798051  1.548
83637 -0.62292934 -1.04076414
  -1.48840341  0.52930166  0.02767399 -0.231
02268]]
[1.74381151]

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\u
tils\validation.py:724: DataConversionWarnin
g: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of
y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\l
inear_model\stochastic_gradient.py:561: Conv
ergenceWarning: Maximum number of iteration
reached before convergence. Consider increas
ing max_iter to improve the fit.
  ConvergenceWarning)
```

In [120]:

```
confusion_matrix(Y_Test_logistic, validation_data)
```

Out[120]:

```
array([[ 812, 1399],
       [ 436, 3274]], dtype=int64)
```

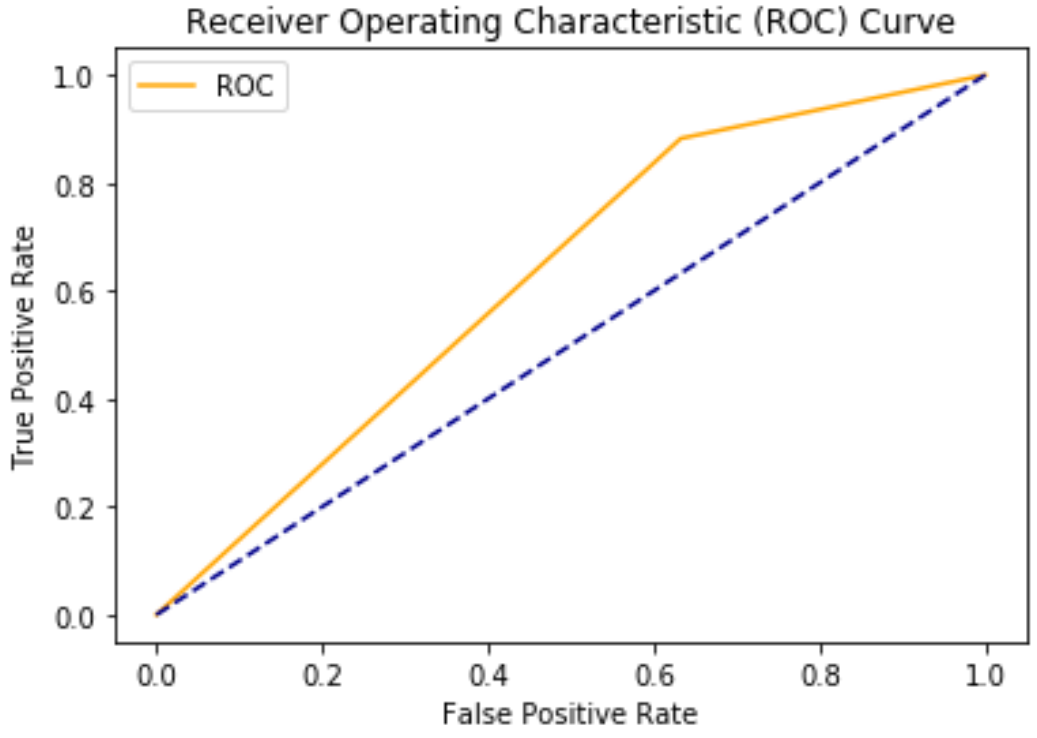
In [128]:

```
# calculate AUC
auc = roc_auc_score(Y_Test_logistic,validation_data)
print('AUC: %.3f' % auc)
```

```
AUC: 0.625
```

In [122]:

```
from sklearn import metrics
from sklearn.metrics import classification_report

fpr, tpr, _ = metrics.roc_curve(Y_Test_logistic,validatio
n_data)
plt.plot(fpr,  tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--'
)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve'
)
plt.legend()
plt.show()
```



In [127]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_Test_logistic,validation_da
ta, target_names=['0','1']))
```

```
              precision    recall  f1-score
support

           0       0.65      0.37      0.47
2211
           1       0.70      0.88      0.78
3710

    accuracy                           0.69
5921
   macro avg       0.68      0.62      0.63
5921
weighted avg       0.68      0.69      0.66
5921
```

In [ ]:

```

```