AML_Hw2_aaj2146

February 7, 2018

1 1.1

```
In [79]: %matplotlib inline
         import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
         #import mglearn
         from IPython.core.display import display
         from IPython.display import display
         from sklearn.datasets import load_iris
         #iris_dataset = load_iris()

In [80]: from sklearn.datasets import fetch_california_housing
         import plotly.plotly as py
         from sklearn.model_selection import train_test_split

In [100]: cal_housing = fetch_california_housing()
          housing__data_df = pd.DataFrame(cal_housing.data, columns=cal_housing.feature_names)
          housing_target_df = pd.DataFrame(cal_housing.target,columns=["ln(Median House value)"]

In [101]: print(cal_housing.DESCR)

California housing dataset.

The original database is available from StatLib

    http://lib.stat.cmu.edu/datasets/

The data contains 20,640 observations on 9 variables.

This dataset contains the average house value as target variable
and the following input variables (features): average income,
housing average age, average rooms, average bedrooms, population,
average occupation, latitude, and longitude in that order.

References
----------
```

```
Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
Statistics and Probability Letters, 33 (1997) 291-297.
```

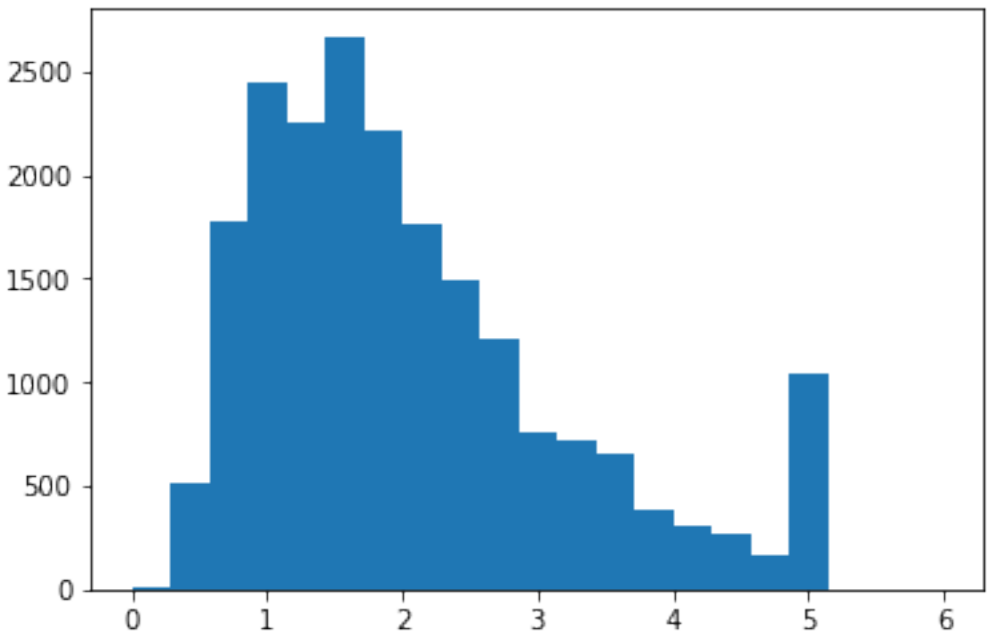
```
In [102]: cal_housing.feature_names
```

```
Out[102]: ['MedInc',
           'HouseAge',
           'AveRooms',
           'AveBedrms',
           'Population',
           'AveOccup',
           'Latitude',
           'Longitude']
```

```
In [103]: cal_housing.target
```

```
Out[103]: array([4.526, 3.585, 3.521, ..., 0.923, 0.847, 0.894])
```

```
In [104]: plt.hist(cal_housing.target, range=(0,6), bins = 21)
          plt.show()
```

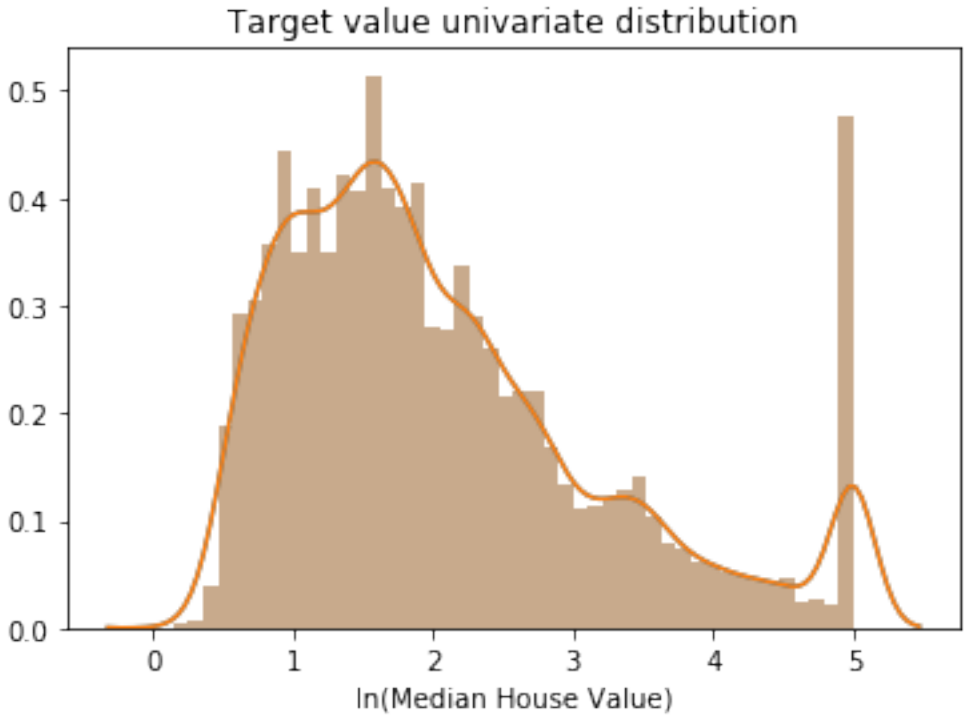



```
In [105]: import seaborn as sns
          print(min(cal_housing.target))
          print(max(cal_housing.target))
```

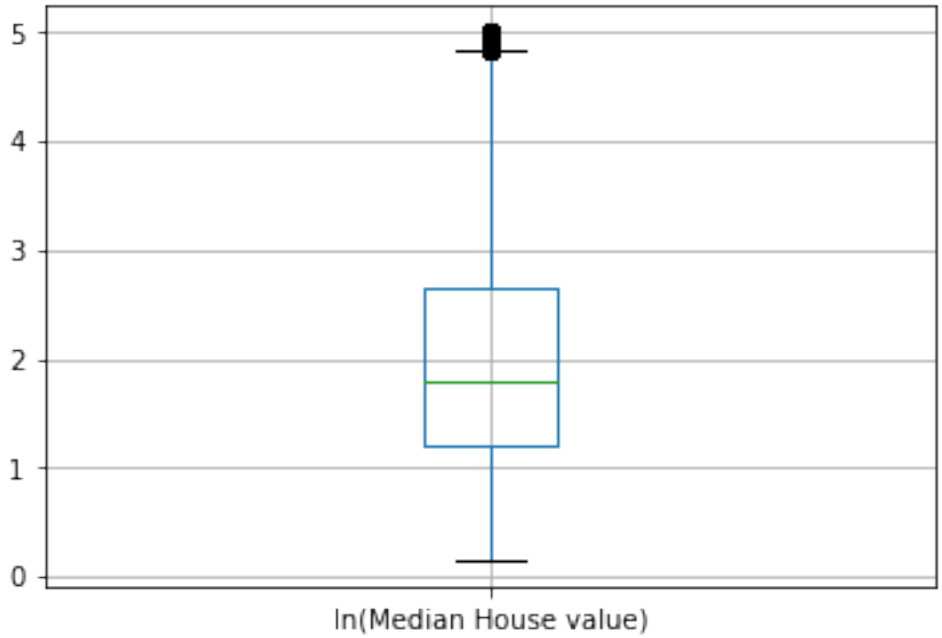
```
0.14999
5.00001
```

In [109]:
```
# Approximate probabilty distribution overlayed on a histogram of the target variable
sns.distplot(cal_housing.target)

ax = sns.distplot(cal_housing.target)
ax.set(xlabel='ln(Median House Value)')
ax.set_title('Target value univariate distribution')
```

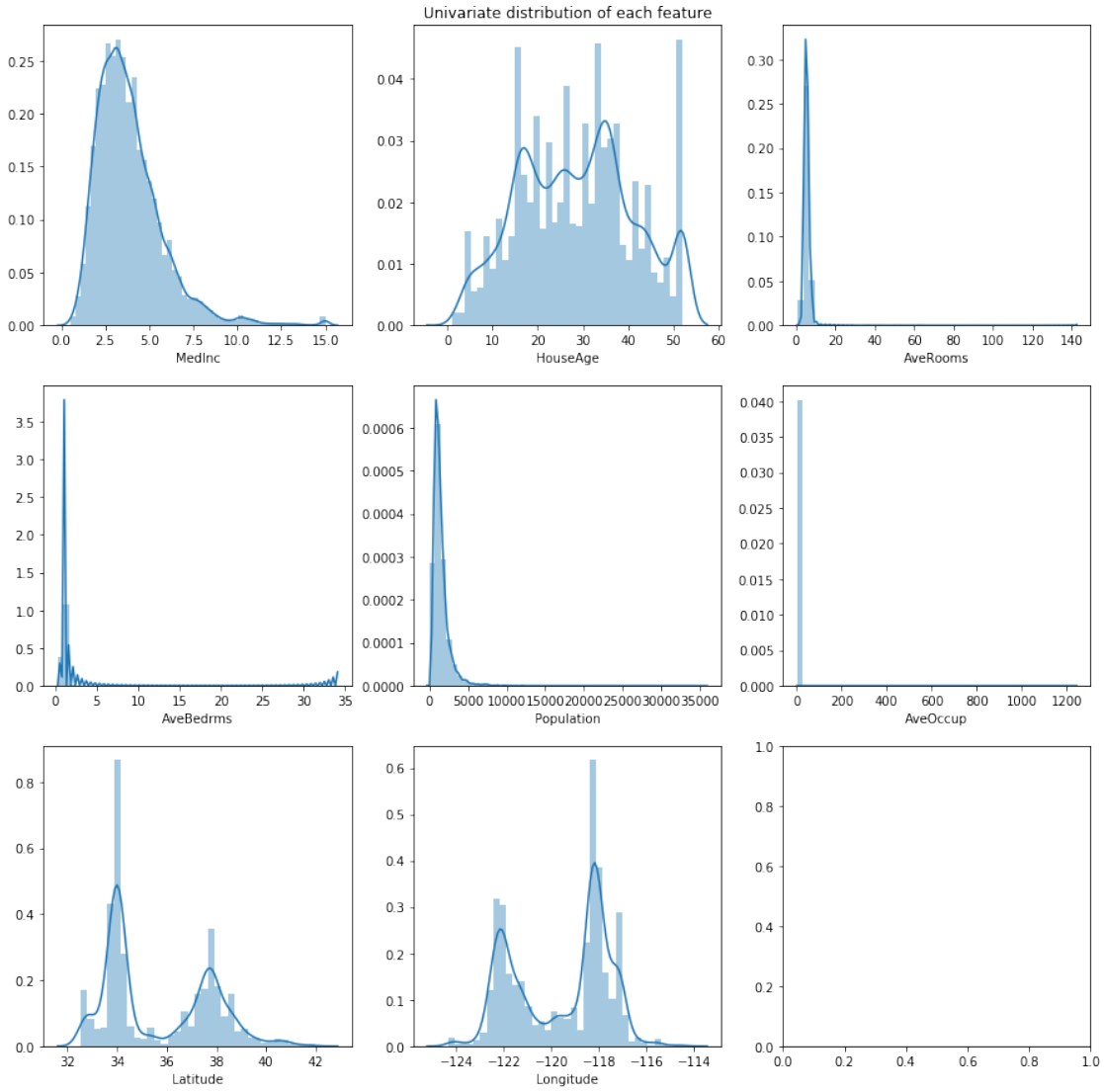
Out[109]:
```
Text(0.5,1,'Target value univariate distribution')
```



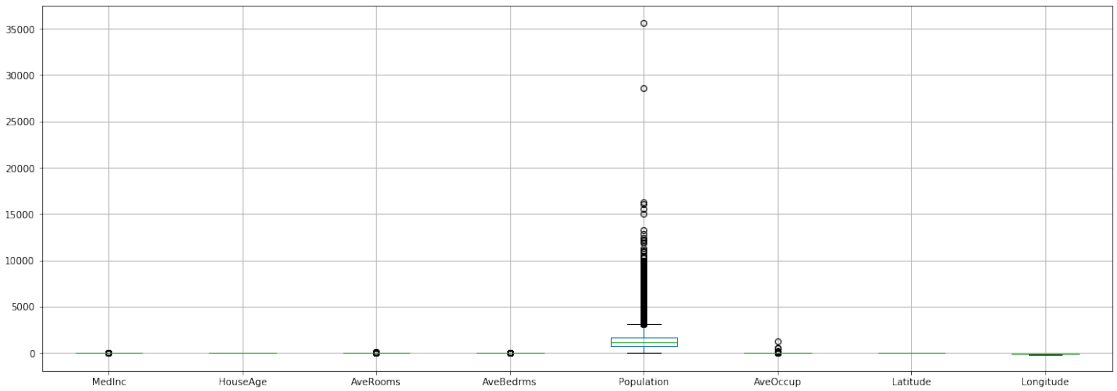
In [118]:
```
#box plot of the target variable
ax = housing_target_df.boxplot()
```




```
In [113]: fig,ax = plt.subplots(3,3, figsize=(15,15))
          k = 0
          ax[0,1].set_title("Univariate distribution of each feature")
          for i in range(3):
              for j in range(3):
                  if i == 2 and j == 2:
                      #sns.distplot(housing_target_df.iloc[:,0],ax = ax[i,j])
                      break
                  sns.distplot(housing__data_df.iloc[:,k],ax = ax[i,j])
                  k +=1
```



In [123]:
```
plt.figure(figsize=(20,7))
ax = housing__data_df.boxplot()
```



```
In [124]: print(min(cal_housing.data[:,2]))
          print(max(cal_housing.data[:,2]))
```

```
0.8461538461538461
141.9090909090909
```

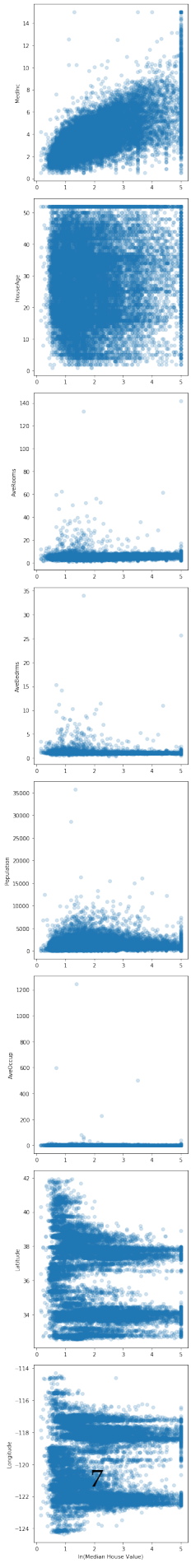
2 1.1 Inferences

1. The target variable is ln(median house value). From the distribution of the target we see that there is a spike at 5 which means that there is a disproportionately large number of housing units with value of the order of $100,000. We will have to be careful about this as the model might get biased towards this value.
2. There are several outliers in some features. For example in the average number of rooms, average number of bedrooms, average number of occupants seem to bunch up between certain values. But then there are massive outliers. Because of their large magnitude, these outliers could also skew the model's output
3. Also, the lattitde and longitude plots of the centroids of the blocks is bimodal. This means that housing units in California are predominantly clustered at 2 locations in the state

3 1.2

Scatter plot of all features vs target variable (ln(Median House Value))

```
In [127]: fig,ax = plt.subplots(8,1, figsize=(5,40))

          for i in range(8):
              ax[i].scatter(cal_housing.target,cal_housing.data[:,i], alpha = 0.2)
              ax[i].set_ylabel(cal_housing.feature_names[i])

          ax[7].set_xlabel('ln(Median House Value)')

          plt.tight_layout()

          plt.show()
```


MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
ln(Median House Value)

4 1.3

In [128]:
```
#1.3
#Never call fit on anything test
X_train, X_test, y_train, y_test = train_test_split(cal_housing.data, cal_housing.targ
```

In [129]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import cross_val_score
```

In [130]:
```
lr = LinearRegression()
rr = Ridge()
ls = Lasso()
en = ElasticNet()
```

In [131]:
```
lr.fit(X_train, y_train)
rr.fit(X_train, y_train)
ls.fit(X_train, y_train)
en.fit(X_train, y_train)
```

Out[131]:
```
ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

In [132]:
```
print("Linear Regression mean cross val score is:{:.3f}".format(np.mean(cross_val_scor
print("Ridge Regression mean cross val score is:{:.3f}".format(np.mean(cross_val_score
print("Lasso Regression mean cross val score is:{:.3f}".format(np.mean(cross_val_score
print("Elastic Net mean cross val score is:{:.3f}".format(np.mean(cross_val_score(en,
```

```
Linear Regression mean cross val score is:0.608
Ridge Regression mean cross val score is:0.608
Lasso Regression mean cross val score is:0.287
Elastic Net mean cross val score is:0.424
```

In [28]:
```
from sklearn.preprocessing import StandardScaler
```

In [39]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

ridge_scaled = Ridge().fit(X_train_scaled, y_train)
linear_scaled = LinearRegression().fit(X_train_scaled, y_train)
```

```
elastic_scaled = ElasticNet().fit(X_train_scaled, y_train)
lasso_scaled = Lasso().fit(X_train_scaled, y_train)
```

```
print("Scaled Linear Regression score is:{:.3f}".format(linear_scaled.score(X_test_scal
print("Scaled Ridge Regression score is:{:.3f}".format(ridge_scaled.score(X_test_scaled
print("Scaled Lasso Regression score is:{:.3f}".format(lasso_scaled.score(X_test_scaled
print("Scaled ElasticNet Regression score is:{:.3f}".format(elastic_scaled.score(X_test
```

```
Scaled Linear Regression score is:0.591
Scaled Ridge Regression score is:0.591
Scaled Lasso Regression score is:-0.000
Scaled ElasticNet Regression score is:0.204
```

```
In [40]: print("Scaled Linear Regression mean cross val score is:{:.3f}".format(np.mean(cross_va
         print("Scaled Ridge Regression mean cross val score is:{:.3f}".format(np.mean(cross_val
         print("Scaled Lasso Regression mean cross val score is:{:.3f}".format(np.mean(cross_val
         print("Scaled Elastic Regression mean cross val score is:{:.3f}".format(np.mean(cross_v
```

```
Scaled Linear Regression mean cross val score is:0.608
Scaled Ridge Regression mean cross val score is:0.608
Scaled Lasso Regression mean cross val score is:-0.000
Scaled Elastic Regression mean cross val score is:0.206
```

5 1.3 Inferences

There is no improvement in Linear Regression and Ridge. However, for the case of Lasso and Elastic Net the mean cross val score goes down. Thus scaling in this case does not help.

6 1.4

```
In [133]: from sklearn.model_selection import GridSearchCV
```

```
In [142]: #For Lasso
          param_grid_lasso = {'alpha': np.logspace(-8, 2, 30)}
          print(param_grid_lasso)

          grid_lasso = GridSearchCV(Lasso(), param_grid_lasso, return_train_score=True)
          grid_lasso.fit(X_train, y_train)
          print(grid_lasso.best_params_)
          print(grid_lasso.best_score_)
          print("test-set score: {:.3f}".format(grid_lasso.score(X_test, y_test)))
          pd.DataFrame(grid_lasso.cv_results_)
```

```
{'alpha': array([1.00000000e-08, 2.21221629e-08, 4.89390092e-08, 1.08263673e-07,
       2.39502662e-07, 5.29831691e-07, 1.17210230e-06, 2.59294380e-06,
       5.73615251e-06, 1.26896100e-05, 2.80721620e-05, 6.21016942e-05,
       1.37382380e-04, 3.03919538e-04, 6.72335754e-04, 1.48735211e-03,
       3.29034456e-03, 7.27895384e-03, 1.61026203e-02, 3.56224789e-02,
       7.88046282e-02, 1.74332882e-01, 3.85662042e-01, 8.53167852e-01,
       1.88739182e+00, 4.17531894e+00, 9.23670857e+00, 2.04335972e+01,
       4.52035366e+01, 1.00000000e+02])}
{'alpha': 0.00013738237958832637}
0.6081916093875163
test-set score: 0.591
```

Out[142]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score
0	0.081641	0.002720	0.608191	0.610151
1	0.094287	0.000884	0.608191	0.610151
2	0.118962	0.000962	0.608191	0.610151
3	0.080485	0.001157	0.608191	0.610151
4	0.080049	0.001716	0.608191	0.610151
5	0.085413	0.007360	0.608191	0.610151
6	0.091002	0.003409	0.608191	0.610151
7	0.070520	0.000890	0.608191	0.610151
8	0.065768	0.000882	0.608191	0.610151
9	0.061042	0.000889	0.608191	0.610151
10	0.064737	0.000885	0.608191	0.610151
11	0.057702	0.000879	0.608191	0.610151
12	0.053805	0.000915	0.608192	0.610150
13	0.048200	0.000889	0.608190	0.610147
14	0.045960	0.001460	0.608180	0.610132
15	0.041874	0.000878	0.608114	0.610057
16	0.035635	0.000883	0.607765	0.609691
17	0.033076	0.000873	0.605996	0.607897
18	0.029632	0.000906	0.598019	0.599655
19	0.023083	0.000870	0.588525	0.589701
20	0.016648	0.000851	0.564371	0.564583
21	0.004000	0.000800	0.507332	0.508237
22	0.003560	0.001499	0.477297	0.478007
23	0.003235	0.001466	0.347405	0.347538
24	0.002832	0.000810	0.000506	0.000678
25	0.004124	0.000827	0.000500	0.000670
26	0.002802	0.000804	0.000466	0.000631
27	0.003467	0.000799	0.000284	0.000440
28	0.002734	0.000801	-0.000140	0.000000
29	0.003450	0.000834	-0.000140	0.000000

	param_alpha	params	rank_test_score
0	1e-08	{'alpha': 1e-08}	13
1	2.21222e-08	{'alpha': 2.2122162910704503e-08}	12

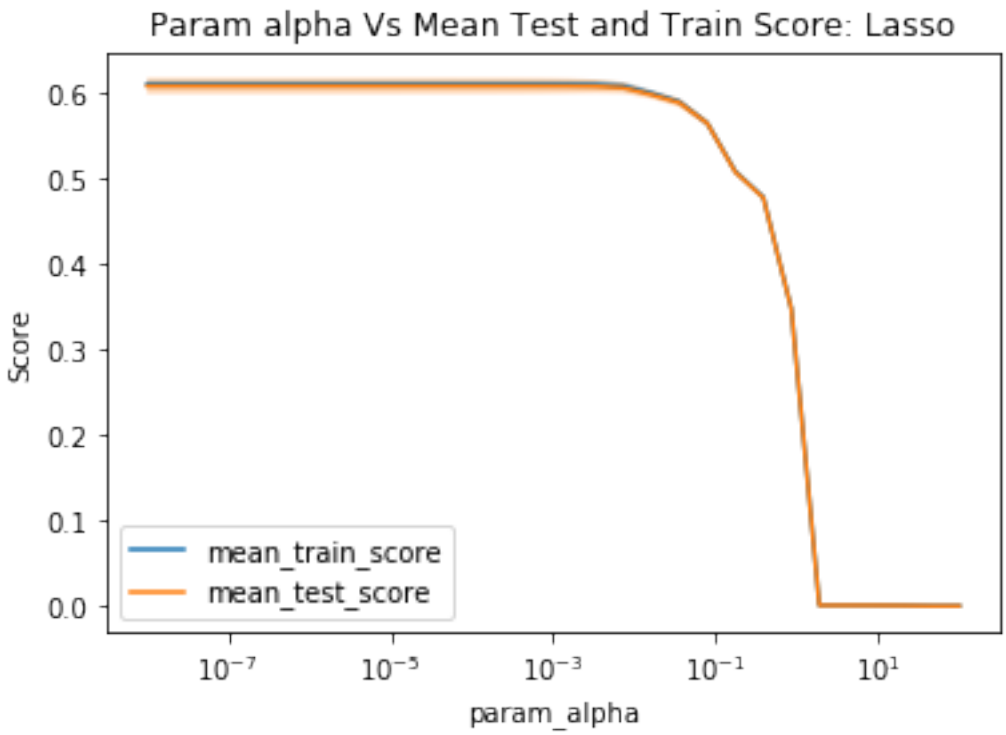
2	4.8939e-08	{'alpha': 4.893900918477499e-08}	11
3	1.08264e-07	{'alpha': 1.0826367338740541e-07}	10
4	2.39503e-07	{'alpha': 2.395026619987486e-07}	9
5	5.29832e-07	{'alpha': 5.298316906283712e-07}	8
6	1.1721e-06	{'alpha': 1.1721022975334793e-06}	7
7	2.59294e-06	{'alpha': 2.592943797404667e-06}	6
8	5.73615e-06	{'alpha': 5.736152510448681e-06}	5
9	1.26896e-05	{'alpha': 1.2689610031679234e-05}	4
10	2.80722e-05	{'alpha': 2.8072162039411814e-05}	3
11	6.21017e-05	{'alpha': 6.210169418915616e-05}	2
12	0.000137382	{'alpha': 0.00013738237958832637}	1
13	0.00030392	{'alpha': 0.0003039195382313201}	14
14	0.000672336	{'alpha': 0.0006723357536499335}	15
15	0.00148735	{'alpha': 0.0014873521072935117}	16
16	0.00329034	{'alpha': 0.003290344562312671}	17
17	0.00727895	{'alpha': 0.0072789538439831615}	18
18	0.0161026	{'alpha': 0.016102620275609426}	19
19	0.0356225	{'alpha': 0.0356224789026244}	20
20	0.0788046	{'alpha': 0.0788046281566992}	21
21	0.174333	{'alpha': 0.1743328822199991}	22
22	0.385662	{'alpha': 0.38566204211634725}	23
23	0.853168	{'alpha': 0.8531678524172814}	24
24	1.88739	{'alpha': 1.8873918221350996}	25
25	4.17532	{'alpha': 4.175318936560409}	26
26	9.23671	{'alpha': 9.236708571873883}	27
27	20.4336	{'alpha': 20.433597178569478}	28
28	45.2035	{'alpha': 45.203536563602405}	29
29	100	{'alpha': 100.0}	29

	split0_test_score	split0_train_score	split1_test_score	\
0	0.607549	0.610921	0.619933	
1	0.607549	0.610921	0.619933	
2	0.607549	0.610921	0.619933	
3	0.607549	0.610921	0.619933	
4	0.607549	0.610921	0.619933	
5	0.607549	0.610921	0.619933	
6	0.607548	0.610921	0.619932	
7	0.607548	0.610921	0.619932	
8	0.607548	0.610921	0.619931	
9	0.607547	0.610921	0.619928	
10	0.607546	0.610921	0.619922	
11	0.607542	0.610921	0.619909	
12	0.607533	0.610920	0.619879	
13	0.607513	0.610917	0.619811	
14	0.607461	0.610903	0.619652	
15	0.607311	0.610831	0.619258	
16	0.606810	0.610481	0.618175	
17	0.604855	0.608763	0.614746	

```
18           0.596398            0.600349           0.604559
19           0.587398            0.590569           0.595487
20           0.563169            0.565741           0.569857
21           0.506692            0.508684           0.510649
22           0.475966            0.478658           0.480155
23           0.346758            0.349496           0.350052
24           0.000401            0.000730           0.000607
25           0.000406            0.000722           0.000571
26           0.000396            0.000683           0.000474
27           0.000261            0.000490           0.000179
28          -0.000161            0.000000          -0.000249
29          -0.000161            0.000000          -0.000249

    split1_train_score  split2_test_score  split2_train_score  std_fit_time  \
0             0.604033           0.597091            0.615499      0.009390
1             0.604033           0.597091            0.615499      0.013639
2             0.604033           0.597091            0.615499      0.019644
3             0.604033           0.597091            0.615499      0.003972
4             0.604033           0.597091            0.615499      0.004778
5             0.604033           0.597091            0.615499      0.005925
6             0.604033           0.597091            0.615499      0.020156
7             0.604033           0.597092            0.615499      0.006103
8             0.604033           0.597094            0.615499      0.004247
9             0.604033           0.597097            0.615499      0.006816
10            0.604033           0.597106            0.615499      0.011126
11            0.604033           0.597123            0.615498      0.007946
12            0.604032           0.597163            0.615498      0.008159
13            0.604029           0.597247            0.615495      0.005847
14            0.604014           0.597425            0.615479      0.008516
15            0.603939           0.597772            0.615401      0.008119
16            0.603573           0.598311            0.615019      0.006471
17            0.601778           0.598389            0.613151      0.007894
18            0.594607           0.593100            0.604008      0.007573
19            0.585474           0.582689            0.593061      0.003773
20            0.560064           0.560087            0.567943      0.001043
21            0.505455           0.504654            0.510573      0.000081
22            0.475488           0.475770            0.479877      0.000168
23            0.346573           0.345404            0.346546      0.000096
24            0.000553           0.000510            0.000751      0.000083
25            0.000545           0.000524            0.000743      0.000970
26            0.000509           0.000528            0.000702      0.000035
27            0.000330           0.000411            0.000499      0.000962
28            0.000000          -0.000009            0.000000      0.000097
29            0.000000          -0.000009            0.000000      0.000960

    std_score_time  std_test_score  std_train_score
0    2.594336e-03        0.009336         0.004712
1    6.453470e-06        0.009336         0.004712
```

```
2     1.047712e-04        0.009336        0.004712
3     3.854676e-04        0.009336        0.004712
4     1.158085e-03        0.009336        0.004712
5     8.870706e-03        0.009336        0.004712
6     3.540131e-03        0.009336        0.004712
7     1.129969e-05        0.009335        0.004712
8     4.566751e-06        0.009334        0.004712
9     9.898121e-06        0.009332        0.004712
10    4.913197e-06        0.009326        0.004712
11    3.781439e-06        0.009313        0.004712
12    5.783932e-05        0.009285        0.004712
13    7.512603e-06        0.009224        0.004712
14    8.147552e-04        0.009088        0.004712
15    8.485379e-07        0.008790        0.004711
16    1.000475e-05        0.008137        0.004706
17    3.170957e-06        0.006726        0.004683
18    7.212292e-05        0.004816        0.003869
19    1.755613e-06        0.005285        0.003158
20    1.753417e-05        0.004078        0.003319
21    9.199649e-07        0.002489        0.002113
22    9.599254e-04        0.002022        0.001850
23    9.074602e-04        0.001952        0.001384
24    1.148596e-05        0.000084        0.000089
25    3.304217e-05        0.000069        0.000089
26    1.227541e-05        0.000054        0.000087
27    5.072585e-06        0.000096        0.000078
28    5.589152e-06        0.000099        0.000000
29    2.526962e-05        0.000099        0.000000
```

In [143]:
```
resultsLasso = pd.DataFrame(grid_lasso.cv_results_)

#Plotting Param_alpha vs Mean Test and Mean Train Score
resultsLasso.plot('param_alpha', 'mean_train_score')
resultsLasso.plot('param_alpha', 'mean_test_score', ax=plt.gca())
plt.fill_between(resultsLasso.param_alpha.astype(np.float),
                 resultsLasso['mean_train_score'] + resultsLasso['std_train_score'],
                 resultsLasso['mean_train_score'] - resultsLasso['std_train_score'], a
plt.fill_between(resultsLasso.param_alpha.astype(np.float),
                 resultsLasso['mean_test_score'] + resultsLasso['std_test_score'],
                 resultsLasso['mean_test_score'] - resultsLasso['std_test_score'], alp
plt.legend()
plt.title("Param alpha Vs Mean Test and Train Score: Lasso")
plt.ylabel("Score")
plt.xscale("log")
plt.savefig("Lasso_1.png")
plt.show()
```




```
In [144]: # For ElasticNet
          param_grid_en = {'alpha': np.logspace(-7, -1, 10),
                           'l1_ratio': [ 1.2, 1.5, 3, 5, 100]}
          grid_en = GridSearchCV(ElasticNet(), param_grid_en, cv=10, return_train_score=True)
          grid_en.fit(X_train, y_train)
          print(grid_en.best_params_)
          print(grid_en.best_score_)
          print("test-set score: {:.3f}".format(grid_en.score(X_test, y_test)))
          pd.DataFrame(grid_en.cv_results_)

          # import pandas as pd
          # res = pd.pivot_table(pd.DataFrame(grid.cv_results_),
          #     values='mean_test_score', index='param_alpha', columns='param_l1_ratio')

{'alpha': 1e-05, 'l1_ratio': 100}
0.6062663507014436
test-set score: 0.591
```

Out[144]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	\
0	0.110256	0.000973	0.606259	0.609977	
1	0.107550	0.000806	0.606259	0.609977	
2	0.103927	0.001526	0.606259	0.609977	

3	0.100268	0.000853	0.606259	0.609977
4	0.082261	0.000795	0.606259	0.609977
5	0.099651	0.000788	0.606259	0.609977
6	0.098435	0.000809	0.606259	0.609977
7	0.094226	0.000790	0.606259	0.609977
8	0.092383	0.000791	0.606259	0.609977
9	0.073196	0.000787	0.606259	0.609977
10	0.090453	0.000797	0.606259	0.609977
11	0.089009	0.000785	0.606259	0.609977
12	0.084943	0.000785	0.606259	0.609977
13	0.081797	0.000782	0.606259	0.609977
14	0.063964	0.000780	0.606261	0.609977
15	0.081129	0.000787	0.606259	0.609977
16	0.079902	0.001149	0.606259	0.609977
17	0.075613	0.000791	0.606259	0.609977
18	0.072614	0.000779	0.606259	0.609977
19	0.053380	0.000786	0.606266	0.609972
20	0.072049	0.000790	0.606259	0.609977
21	0.077226	0.000809	0.606259	0.609977
22	0.071017	0.000888	0.606260	0.609977
23	0.065612	0.001577	0.606261	0.609977
24	0.045754	0.000825	0.606153	0.609825
25	0.063315	0.000996	0.606258	0.609975
26	0.061755	0.000837	0.606258	0.609975
27	0.055695	0.002702	0.606260	0.609972
28	0.061867	0.002661	0.606260	0.609966
29	0.035191	0.001012	0.595147	0.598560
30	0.058203	0.000921	0.606217	0.609928
31	0.058631	0.001257	0.606210	0.609917
32	0.057358	0.001655	0.606149	0.609840
33	0.050487	0.000942	0.605993	0.609667
34	0.027051	0.004889	0.558755	0.559354
35	0.045678	0.000856	0.605168	0.608868
36	0.041324	0.001045	0.604818	0.608513
37	0.047465	0.001112	0.600838	0.604604
38	0.031403	0.002126	0.594082	0.597547
39	0.004323	0.002340	0.488103	0.488834
40	0.029778	0.000796	0.590979	0.594345
41	0.026406	0.001425	0.589031	0.591892
42	0.023846	0.001396	0.579955	0.581186
43	0.023668	0.000800	0.546300	0.546836
44	0.003356	0.000751	0.000062	0.000673
45	0.015894	0.000777	0.524479	0.524973
46	0.004908	0.000754	0.511823	0.512386
47	0.004301	0.000750	0.498594	0.499721
48	0.004338	0.001124	0.478429	0.479023
49	0.003346	0.000749	0.000008	0.000618

	param_alpha	param_l1_ratio	\
0	1e-07	1.2	
1	1e-07	1.5	
2	1e-07	3	
3	1e-07	5	
4	1e-07	100	
5	4.64159e-07	1.2	
6	4.64159e-07	1.5	
7	4.64159e-07	3	
8	4.64159e-07	5	
9	4.64159e-07	100	
10	2.15443e-06	1.2	
11	2.15443e-06	1.5	
12	2.15443e-06	3	
13	2.15443e-06	5	
14	2.15443e-06	100	
15	1e-05	1.2	
16	1e-05	1.5	
17	1e-05	3	
18	1e-05	5	
19	1e-05	100	
20	4.64159e-05	1.2	
21	4.64159e-05	1.5	
22	4.64159e-05	3	
23	4.64159e-05	5	
24	4.64159e-05	100	
25	0.000215443	1.2	
26	0.000215443	1.5	
27	0.000215443	3	
28	0.000215443	5	
29	0.000215443	100	
30	0.001	1.2	
31	0.001	1.5	
32	0.001	3	
33	0.001	5	
34	0.001	100	
35	0.00464159	1.2	
36	0.00464159	1.5	
37	0.00464159	3	
38	0.00464159	5	
39	0.00464159	100	
40	0.0215443	1.2	
41	0.0215443	1.5	
42	0.0215443	3	
43	0.0215443	5	
44	0.0215443	100	
45	0.1	1.2	
46	0.1	1.5	

```
47          0.1              3
48          0.1              5
49          0.1            100

                                                  params  rank_test_score  \
0                      {'alpha': 1e-07, 'l1_ratio': 1.2}               26
1                      {'alpha': 1e-07, 'l1_ratio': 1.5}               25
2                        {'alpha': 1e-07, 'l1_ratio': 3}               24
3                        {'alpha': 1e-07, 'l1_ratio': 5}               21
4                      {'alpha': 1e-07, 'l1_ratio': 100}               12
5   {'alpha': 4.641588833612782e-07, 'l1_ratio': 1.2}               23
6   {'alpha': 4.641588833612782e-07, 'l1_ratio': 1.5}               22
7     {'alpha': 4.641588833612782e-07, 'l1_ratio': 3}               20
8     {'alpha': 4.641588833612782e-07, 'l1_ratio': 5}               17
9   {'alpha': 4.641588833612782e-07, 'l1_ratio': 100}                7
10  {'alpha': 2.1544346900318822e-06, 'l1_ratio': ...               19
11  {'alpha': 2.1544346900318822e-06, 'l1_ratio': ...               18
12   {'alpha': 2.1544346900318822e-06, 'l1_ratio': 3}               16
13   {'alpha': 2.1544346900318822e-06, 'l1_ratio': 5}               13
14  {'alpha': 2.1544346900318822e-06, 'l1_ratio': ...                2
15                     {'alpha': 1e-05, 'l1_ratio': 1.2}               15
16                     {'alpha': 1e-05, 'l1_ratio': 1.5}               14
17                       {'alpha': 1e-05, 'l1_ratio': 3}               10
18                       {'alpha': 1e-05, 'l1_ratio': 5}                8
19                     {'alpha': 1e-05, 'l1_ratio': 100}                1
20  {'alpha': 4.641588833612772e-05, 'l1_ratio': 1.2}               11
21  {'alpha': 4.641588833612772e-05, 'l1_ratio': 1.5}                9
22    {'alpha': 4.641588833612772e-05, 'l1_ratio': 3}                5
23    {'alpha': 4.641588833612772e-05, 'l1_ratio': 5}                3
24  {'alpha': 4.641588833612772e-05, 'l1_ratio': 100}               31
25  {'alpha': 0.00021544346900318823, 'l1_ratio': ...               28
26  {'alpha': 0.00021544346900318823, 'l1_ratio': ...               27
27   {'alpha': 0.00021544346900318823, 'l1_ratio': 3}                4
28   {'alpha': 0.00021544346900318823, 'l1_ratio': 5}                6
29  {'alpha': 0.00021544346900318823, 'l1_ratio': ...               37
30                     {'alpha': 0.001, 'l1_ratio': 1.2}               29
31                     {'alpha': 0.001, 'l1_ratio': 1.5}               30
32                       {'alpha': 0.001, 'l1_ratio': 3}               32
33                       {'alpha': 0.001, 'l1_ratio': 5}               33
34                     {'alpha': 0.001, 'l1_ratio': 100}               42
35   {'alpha': 0.004641588833612773, 'l1_ratio': 1.2}               34
36   {'alpha': 0.004641588833612773, 'l1_ratio': 1.5}               35
37     {'alpha': 0.004641588833612773, 'l1_ratio': 3}               36
38     {'alpha': 0.004641588833612773, 'l1_ratio': 5}               38
39   {'alpha': 0.004641588833612773, 'l1_ratio': 100}               47
40   {'alpha': 0.021544346900318822, 'l1_ratio': 1.2}               39
41   {'alpha': 0.021544346900318822, 'l1_ratio': 1.5}               40
42     {'alpha': 0.021544346900318822, 'l1_ratio': 3}               41
```

43	{'alpha': 0.021544346900318822, 'l1_ratio': 5}	43	
44	{'alpha': 0.021544346900318822, 'l1_ratio': 100}	49	
45	{'alpha': 0.1, 'l1_ratio': 1.2}	44	
46	{'alpha': 0.1, 'l1_ratio': 1.5}	45	
47	{'alpha': 0.1, 'l1_ratio': 3}	46	
48	{'alpha': 0.1, 'l1_ratio': 5}	48	
49	{'alpha': 0.1, 'l1_ratio': 100}	50	

	split0_test_score	split0_train_score	...	split7_test_score	\
0	0.610474	0.609773	...	0.571816	
1	0.610474	0.609773	...	0.571816	
2	0.610474	0.609773	...	0.571816	
3	0.610474	0.609773	...	0.571816	
4	0.610473	0.609773	...	0.571819	
5	0.610474	0.609773	...	0.571816	
6	0.610474	0.609773	...	0.571816	
7	0.610473	0.609773	...	0.571816	
8	0.610473	0.609773	...	0.571817	
9	0.610469	0.609773	...	0.571832	
10	0.610473	0.609773	...	0.571818	
11	0.610473	0.609773	...	0.571818	
12	0.610473	0.609773	...	0.571819	
13	0.610472	0.609773	...	0.571821	
14	0.610449	0.609772	...	0.571893	
15	0.610470	0.609773	...	0.571827	
16	0.610470	0.609773	...	0.571828	
17	0.610468	0.609773	...	0.571833	
18	0.610466	0.609773	...	0.571841	
19	0.610355	0.609767	...	0.572180	
20	0.610457	0.609773	...	0.571869	
21	0.610456	0.609773	...	0.571874	
22	0.610448	0.609772	...	0.571899	
23	0.610438	0.609772	...	0.571932	
24	0.609728	0.609612	...	0.573600	
25	0.610396	0.609770	...	0.572063	
26	0.610389	0.609770	...	0.572086	
27	0.610350	0.609767	...	0.572201	
28	0.610295	0.609761	...	0.572356	
29	0.596116	0.598762	...	0.568935	
30	0.610083	0.609724	...	0.572912	
31	0.610037	0.609713	...	0.573019	
32	0.609774	0.609634	...	0.573565	
33	0.609323	0.609454	...	0.574319	
34	0.556504	0.559496	...	0.527294	
35	0.607896	0.608678	...	0.575734	
36	0.607345	0.608320	...	0.576070	
37	0.602295	0.604392	...	0.576770	
38	0.595102	0.597764	...	0.564907	

```
39          0.483889           0.489049       ...                0.460295
40          0.591982           0.594686       ...                0.560680
41          0.589217           0.591971       ...                0.559711
42          0.578399           0.581312       ...                0.550078
43          0.543935           0.546989       ...                0.514125
44          0.000860           0.000599       ...               -0.000511
45          0.521811           0.525136       ...                0.491543
46          0.508815           0.512422       ...                0.479174
47          0.494919           0.500014       ...                0.468660
48          0.473737           0.479239       ...                0.452004
49          0.000634           0.000544       ...               -0.000336

    split7_train_score  split8_test_score  split8_train_score  \
0             0.613336           0.601821            0.610724
1             0.613336           0.601821            0.610724
2             0.613336           0.601821            0.610724
3             0.613336           0.601821            0.610724
4             0.613336           0.601822            0.610724
5             0.613336           0.601821            0.610724
6             0.613336           0.601821            0.610724
7             0.613336           0.601821            0.610724
8             0.613336           0.601821            0.610724
9             0.613336           0.601827            0.610724
10            0.613336           0.601822            0.610724
11            0.613336           0.601822            0.610724
12            0.613336           0.601822            0.610724
13            0.613336           0.601823            0.610724
14            0.613336           0.601852            0.610724
15            0.613336           0.601825            0.610724
16            0.613336           0.601825            0.610724
17            0.613336           0.601827            0.610724
18            0.613336           0.601830            0.610724
19            0.613331           0.601962            0.610719
20            0.613336           0.601839            0.610724
21            0.613336           0.601841            0.610724
22            0.613336           0.601851            0.610724
23            0.613335           0.601864            0.610724
24            0.613209           0.602432            0.610582
25            0.613333           0.601905            0.610722
26            0.613333           0.601914            0.610721
27            0.613330           0.601958            0.610719
28            0.613325           0.602016            0.610713
29            0.599178           0.597563            0.599027
30            0.613286           0.602178            0.610676
31            0.613275           0.602214            0.610665
32            0.613206           0.602387            0.610592
33            0.613052           0.602586            0.610427
34            0.562526           0.564739            0.559092
```

35	0.612197	0.602739	0.609627
36	0.611859	0.602716	0.609286
37	0.608103	0.601308	0.605595
38	0.597985	0.597127	0.598041
39	0.492952	0.502195	0.487713
40	0.595711	0.595435	0.594977
41	0.594596	0.593496	0.591859
42	0.584569	0.584982	0.581030
43	0.549924	0.552839	0.546442
44	0.000777	0.001113	0.000605
45	0.528112	0.532135	0.524371
46	0.516210	0.520729	0.511949
47	0.503761	0.511294	0.498872
48	0.483198	0.493049	0.477714
49	0.000723	0.000950	0.000547

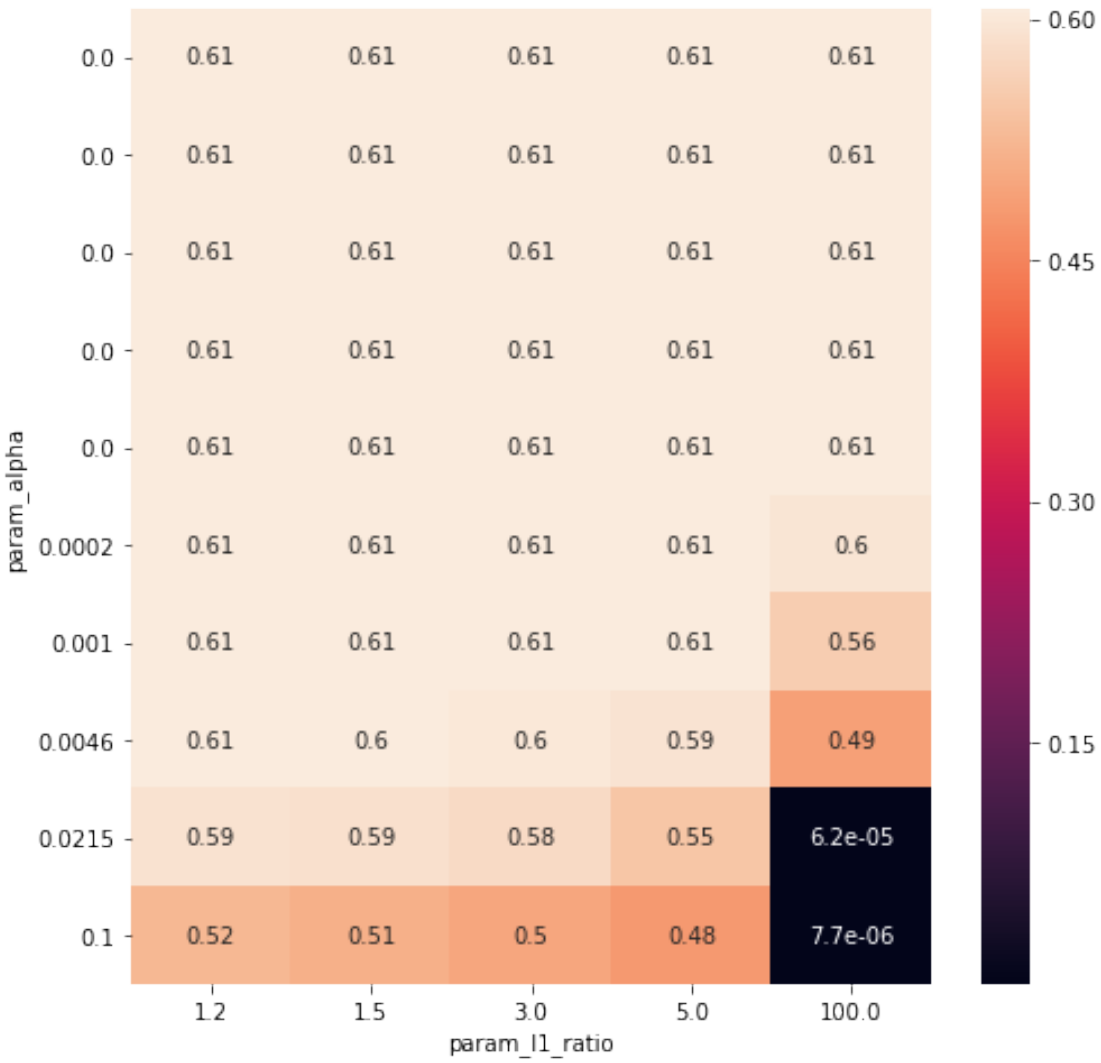
	split9_test_score	split9_train_score	std_fit_time	std_score_time	\
0	0.599496	0.611014	0.006248	0.000454	
1	0.599496	0.611014	0.000410	0.000031	
2	0.599496	0.611014	0.001546	0.002153	
3	0.599496	0.611014	0.000429	0.000199	
4	0.599495	0.611014	0.000581	0.000022	
5	0.599496	0.611014	0.000476	0.000010	
6	0.599496	0.611014	0.000458	0.000039	
7	0.599496	0.611014	0.000376	0.000014	
8	0.599496	0.611014	0.004192	0.000010	
9	0.599493	0.611014	0.000425	0.000009	
10	0.599496	0.611014	0.000436	0.000008	
11	0.599496	0.611014	0.000291	0.000008	
12	0.599495	0.611014	0.000327	0.000006	
13	0.599495	0.611014	0.000449	0.000007	
14	0.599483	0.611014	0.000336	0.000005	
15	0.599495	0.611014	0.000547	0.000015	
16	0.599494	0.611014	0.000521	0.001049	
17	0.599493	0.611014	0.000323	0.000034	
18	0.599492	0.611014	0.000320	0.000006	
19	0.599430	0.611008	0.006494	0.000013	
20	0.599490	0.611014	0.000353	0.000012	
21	0.599489	0.611014	0.009323	0.000035	
22	0.599484	0.611014	0.005943	0.000137	
23	0.599479	0.611013	0.001375	0.001177	
24	0.599038	0.610857	0.008485	0.000060	
25	0.599465	0.611012	0.000332	0.000350	
26	0.599460	0.611011	0.000341	0.000104	
27	0.599438	0.611008	0.006332	0.004651	
28	0.599406	0.611002	0.009606	0.004342	
29	0.587107	0.600115	0.006658	0.000424	
30	0.599312	0.610965	0.008935	0.000069	

31	0.599282	0.610953	0.008614	0.000663
32	0.599106	0.610875	0.013905	0.002415
33	0.598789	0.610696	0.010941	0.000146
34	0.550109	0.560463	0.000773	0.006383
35	0.597753	0.609898	0.011146	0.000173
36	0.597283	0.609537	0.007686	0.000484
37	0.592581	0.605538	0.005957	0.000623
38	0.586406	0.599115	0.005840	0.003968
39	0.480307	0.489534	0.000051	0.004736
40	0.583365	0.595570	0.006070	0.000014
41	0.581121	0.593128	0.003699	0.001918
42	0.571262	0.582202	0.000538	0.001326
43	0.537991	0.547973	0.000360	0.000019
44	0.000461	0.000689	0.000018	0.000008
45	0.516650	0.526074	0.000206	0.000016
46	0.504141	0.513199	0.000099	0.000014
47	0.491214	0.500519	0.000069	0.000009
48	0.470433	0.479689	0.000101	0.001113
49	0.000430	0.000635	0.000019	0.000011

	std_test_score	std_train_score
0	0.019700	0.002101
1	0.019700	0.002101
2	0.019700	0.002101
3	0.019700	0.002101
4	0.019698	0.002101
5	0.019700	0.002101
6	0.019700	0.002101
7	0.019699	0.002101
8	0.019699	0.002101
9	0.019694	0.002101
10	0.019699	0.002101
11	0.019699	0.002101
12	0.019698	0.002101
13	0.019698	0.002101
14	0.019673	0.002101
15	0.019696	0.002101
16	0.019695	0.002101
17	0.019693	0.002101
18	0.019691	0.002101
19	0.019571	0.002102
20	0.019680	0.002101
21	0.019679	0.002101
22	0.019670	0.002101
23	0.019659	0.002101
24	0.019026	0.002113
25	0.019611	0.002101
26	0.019603	0.002101

```
27        0.019562         0.002101
28        0.019506         0.002102
29        0.016888         0.001473
30        0.019297         0.002101
31        0.019257         0.002101
32        0.019045         0.002104
33        0.018732         0.002114
34        0.018468         0.001834
35        0.018043         0.002088
36        0.017836         0.002095
37        0.016745         0.002174
38        0.017443         0.001463
39        0.015975         0.002218
40        0.018041         0.001567
41        0.017626         0.001694
42        0.017758         0.001835
43        0.018610         0.001885
44        0.000927         0.000065
45        0.018505         0.001999
46        0.018093         0.002196
47        0.016997         0.002163
48        0.015345         0.002245
49        0.000811         0.000066

[50 rows x 32 columns]
```

```
In [150]: res = pd.pivot_table(pd.DataFrame(grid_en.cv_results_),
                              values='mean_test_score', index='param_alpha', columns='param_l1_

          a4_dims = (8, 8)
          fig, ax = plt.subplots(figsize=a4_dims)
          sns.heatmap(ax = ax, data = res, annot=True)
          ax.set_yticklabels(np.round(param_grid_en['alpha'],4))
          plt.show()
```




```
In [173]: # For Ridge
          param_grid_rr = {'alpha': np.logspace(-2, 3, 30)}
          print(param_grid_rr)

          grid_rr = GridSearchCV(Ridge(), param_grid_rr, return_train_score=True)
          grid_rr.fit(X_train, y_train)

          print(grid_rr.best_params_)
          print(grid_rr.best_score_)
          print("test-set score: {:.3f}".format(grid_rr.score(X_test, y_test)))
          pd.DataFrame(grid_rr.cv_results_)
```

```
{'alpha': array([1.00000000e-02, 1.48735211e-02, 2.21221629e-02, 3.29034456e-02,
       4.89390092e-02, 7.27895384e-02, 1.08263673e-01, 1.61026203e-01,
```

```
       2.39502662e-01, 3.56224789e-01, 5.29831691e-01, 7.88046282e-01,
       1.17210230e+00, 1.74332882e+00, 2.59294380e+00, 3.85662042e+00,
       5.73615251e+00, 8.53167852e+00, 1.26896100e+01, 1.88739182e+01,
       2.80721620e+01, 4.17531894e+01, 6.21016942e+01, 9.23670857e+01,
       1.37382380e+02, 2.04335972e+02, 3.03919538e+02, 4.52035366e+02,
       6.72335754e+02, 1.00000000e+03])}
{'alpha': 2.592943797404667}
0.6081924379983338
test-set score: 0.591
```

Out[173]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score
0	0.007420	0.001136	0.608191	0.610151
1	0.003237	0.000863	0.608191	0.610151
2	0.003141	0.000838	0.608191	0.610151
3	0.003463	0.000947	0.608191	0.610151
4	0.003400	0.000965	0.608191	0.610151
5	0.003237	0.000978	0.608191	0.610151
6	0.003168	0.000841	0.608191	0.610151
7	0.003176	0.000868	0.608191	0.610151
8	0.003190	0.000834	0.608191	0.610151
9	0.008133	0.001429	0.608191	0.610151
10	0.007244	0.001647	0.608191	0.610151
11	0.003182	0.000760	0.608192	0.610151
12	0.002913	0.000750	0.608192	0.610151
13	0.002893	0.000757	0.608192	0.610150
14	0.003111	0.000835	0.608192	0.610150
15	0.003023	0.000822	0.608192	0.610148
16	0.004458	0.000821	0.608191	0.610145
17	0.005807	0.003722	0.608187	0.610138
18	0.003465	0.000810	0.608177	0.610124
19	0.003017	0.000767	0.608152	0.610094
20	0.003002	0.000823	0.608097	0.610033
21	0.002998	0.000775	0.607981	0.609910
22	0.002913	0.000763	0.607754	0.609676
23	0.002952	0.000757	0.607335	0.609253
24	0.002905	0.000756	0.606619	0.608539
25	0.002957	0.000762	0.605488	0.607420
26	0.002968	0.000790	0.603830	0.605785
27	0.002910	0.000753	0.601533	0.603523
28	0.002893	0.000770	0.598431	0.600465
29	0.002962	0.000768	0.594247	0.596324

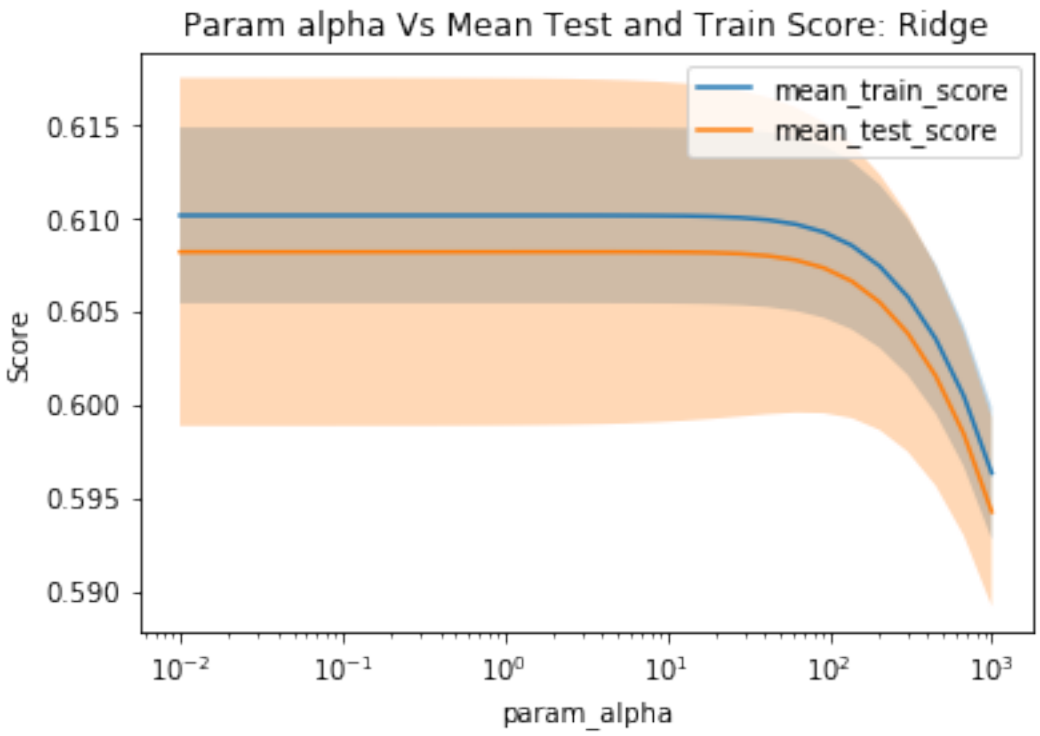
	param_alpha	params	rank_test_score
0	0.01	{'alpha': 0.01}	17
1	0.0148735	{'alpha': 0.014873521072935119}	16
2	0.0221222	{'alpha': 0.022122162910704492}	15
3	0.0329034	{'alpha': 0.03290344562312668}	14

```
4     0.048939   {'alpha': 0.04893900918477494}              13
5    0.0727895   {'alpha': 0.07278953843983153}              12
6     0.108264   {'alpha': 0.10826367338740546}              11
7     0.161026   {'alpha': 0.16102620275609392}              10
8     0.239503    {'alpha': 0.2395026619987486}               9
9     0.356225   {'alpha': 0.35622478902624444}               8
10    0.529832     {'alpha': 0.529831690628371}               6
11    0.788046    {'alpha': 0.7880462815669912}               5
12      1.1721    {'alpha': 1.1721022975334805}               4
13     1.74333     {'alpha': 1.743328822199989}               3
14     2.59294     {'alpha': 2.592943797404667}               1
15     3.85662     {'alpha': 3.856620421163472}               2
16     5.73615     {'alpha': 5.736152510448681}               7
17     8.53168     {'alpha': 8.531678524172815}              18
18     12.6896    {'alpha': 12.689610031679234}              19
19     18.8739    {'alpha': 18.873918221350976}              20
20     28.0722    {'alpha': 28.072162039411786}              21
21     41.7532    {'alpha': 41.753189365604044}              22
22     62.1017     {'alpha': 62.10169418915616}              23
23     92.3671     {'alpha': 92.36708571873865}              24
24     137.382     {'alpha': 137.3823795883264}              25
25     204.336    {'alpha': 204.33597178569437}              26
26      303.92    {'alpha': 303.91953823132013}              27
27     452.035      {'alpha': 452.035365636025}              28
28     672.336     {'alpha': 672.3357536499335}              29
29        1000                {'alpha': 1000.0}              30

    split0_test_score  split0_train_score  split1_test_score  \
0            0.607549            0.610921           0.619933
1            0.607549            0.610921           0.619932
2            0.607548            0.610921           0.619932
3            0.607548            0.610921           0.619932
4            0.607548            0.610921           0.619932
5            0.607548            0.610921           0.619931
6            0.607548            0.610921           0.619930
7            0.607547            0.610921           0.619929
8            0.607547            0.610921           0.619927
9            0.607546            0.610921           0.619924
10           0.607545            0.610921           0.619920
11           0.607543            0.610921           0.619913
12           0.607540            0.610921           0.619904
13           0.607536            0.610921           0.619889
14           0.607530            0.610920           0.619868
15           0.607520            0.610919           0.619836
16           0.607505            0.610916           0.619787
17           0.607482            0.610910           0.619715
18           0.607444            0.610897           0.619605
19           0.607382            0.610870           0.619438
```

```
20           0.607280           0.610815           0.619184
21           0.607107           0.610703           0.618796
22           0.606819           0.610489           0.618210
23           0.606347           0.610098           0.617337
24           0.605600           0.609429           0.616079
25           0.604478           0.608366           0.614341
26           0.602876           0.606788           0.612046
27           0.600680           0.604570           0.609121
28           0.597710           0.601535           0.605432
29           0.593664           0.597390           0.600706

    split1_train_score  split2_test_score  split2_train_score  std_fit_time  \
0             0.604033           0.597091            0.615499      0.003117
1             0.604033           0.597091            0.615499      0.000049
2             0.604033           0.597092            0.615499      0.000040
3             0.604033           0.597092            0.615499      0.000165
4             0.604033           0.597092            0.615499      0.000143
5             0.604033           0.597093            0.615499      0.000088
6             0.604033           0.597095            0.615499      0.000025
7             0.604033           0.597096            0.615499      0.000028
8             0.604033           0.597099            0.615499      0.000029
9             0.604033           0.597103            0.615499      0.000583
10            0.604033           0.597109            0.615499      0.000872
11            0.604033           0.597118            0.615499      0.000365
12            0.604033           0.597132            0.615498      0.000024
13            0.604032           0.597151            0.615498      0.000013
14            0.604032           0.597180            0.615497      0.000206
15            0.604031           0.597221            0.615495      0.000074
16            0.604028           0.597281            0.615492      0.002171
17            0.604022           0.597365            0.615483      0.004094
18            0.604010           0.597481            0.615466      0.000788
19            0.603984           0.597635            0.615429      0.000169
20            0.603930           0.597826            0.615353      0.000066
21            0.603824           0.598039            0.615202      0.000082
22            0.603621           0.598232            0.614917      0.000013
23            0.603254           0.598321            0.614406      0.000063
24            0.602633           0.598178            0.613554      0.000018
25            0.601655           0.597646            0.612238      0.000048
26            0.600215           0.596569            0.610353      0.000074
27            0.598196           0.594797            0.607803      0.000003
28            0.595414           0.592152            0.604445      0.000005
29            0.591564           0.588371            0.600017      0.000050

    std_score_time  std_test_score  std_train_score
0         0.000402        0.009336         0.004712
1         0.000038        0.009336         0.004712
2         0.000010        0.009336         0.004712
3         0.000074        0.009335         0.004712
```

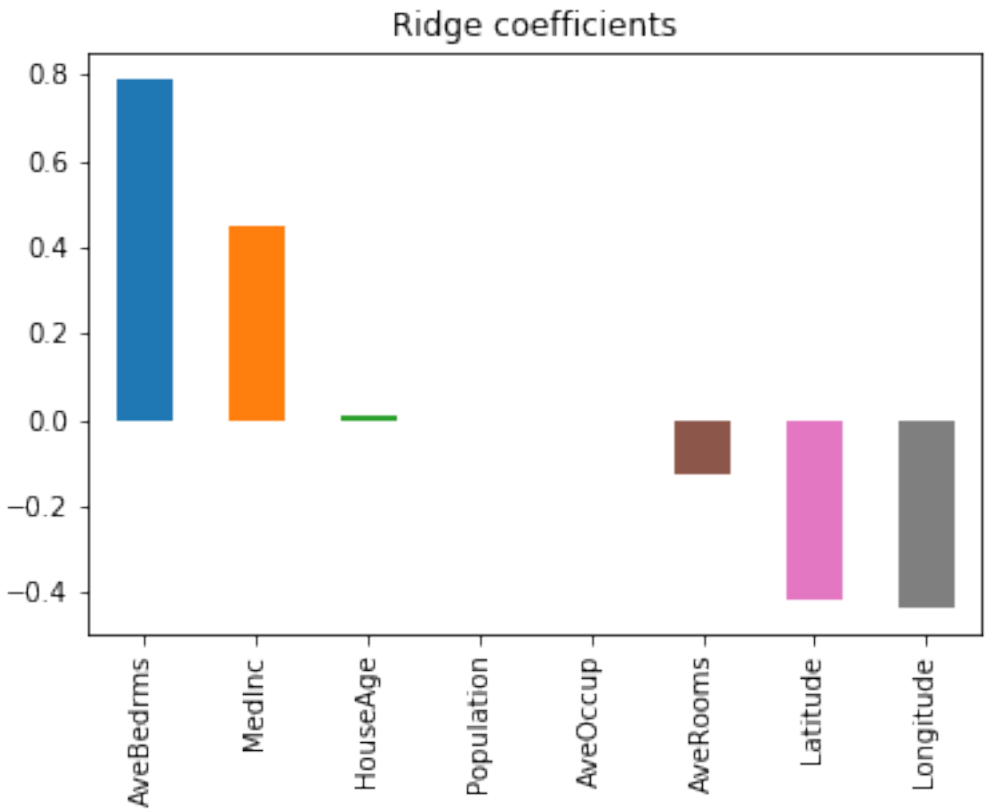
```
4         0.000079        0.009335        0.004712
5         0.000178        0.009335        0.004712
6         0.000010        0.009334        0.004712
7         0.000047        0.009332        0.004712
8         0.000011        0.009330        0.004712
9         0.000241        0.009328        0.004712
10        0.000084        0.009323        0.004712
11        0.000010        0.009317        0.004712
12        0.000008        0.009308        0.004712
13        0.000004        0.009294        0.004712
14        0.000095        0.009274        0.004712
15        0.000071        0.009245        0.004712
16        0.000045        0.009201        0.004712
17        0.004043        0.009138        0.004711
18        0.000069        0.009047        0.004709
19        0.000005        0.008918        0.004704
20        0.000072        0.008738        0.004696
21        0.000012        0.008496        0.004679
22        0.000013        0.008183        0.004647
23        0.000005        0.007795        0.004592
24        0.000003        0.007344        0.004503
25        0.000005        0.006853        0.004372
26        0.000055        0.006354        0.004199
27        0.000001        0.005879        0.003991
28        0.000013        0.005446        0.003764
29        0.000023        0.005053        0.003532
```

In [174]:
```
resultsRidge = pd.DataFrame(grid_rr.cv_results_)

#Plotting Param_alpha vs Mean Test and Mean Train Score
resultsRidge.plot('param_alpha', 'mean_train_score')
resultsRidge.plot('param_alpha', 'mean_test_score', ax=plt.gca())
plt.fill_between(resultsRidge.param_alpha.astype(np.float),
                 resultsRidge['mean_train_score'] + resultsRidge['std_train_score'],
                 resultsRidge['mean_train_score'] - resultsRidge['std_train_score'], a
plt.fill_between(resultsRidge.param_alpha.astype(np.float),
                 resultsRidge['mean_test_score'] + resultsRidge['std_test_score'],
                 resultsRidge['mean_test_score'] - resultsRidge['std_test_score'], alp
plt.legend()
plt.title("Param alpha Vs Mean Test and Train Score: Ridge")
plt.ylabel("Score")
plt.xscale("log")
plt.savefig("Ridge_1.png")
plt.show()
```



7 1.5

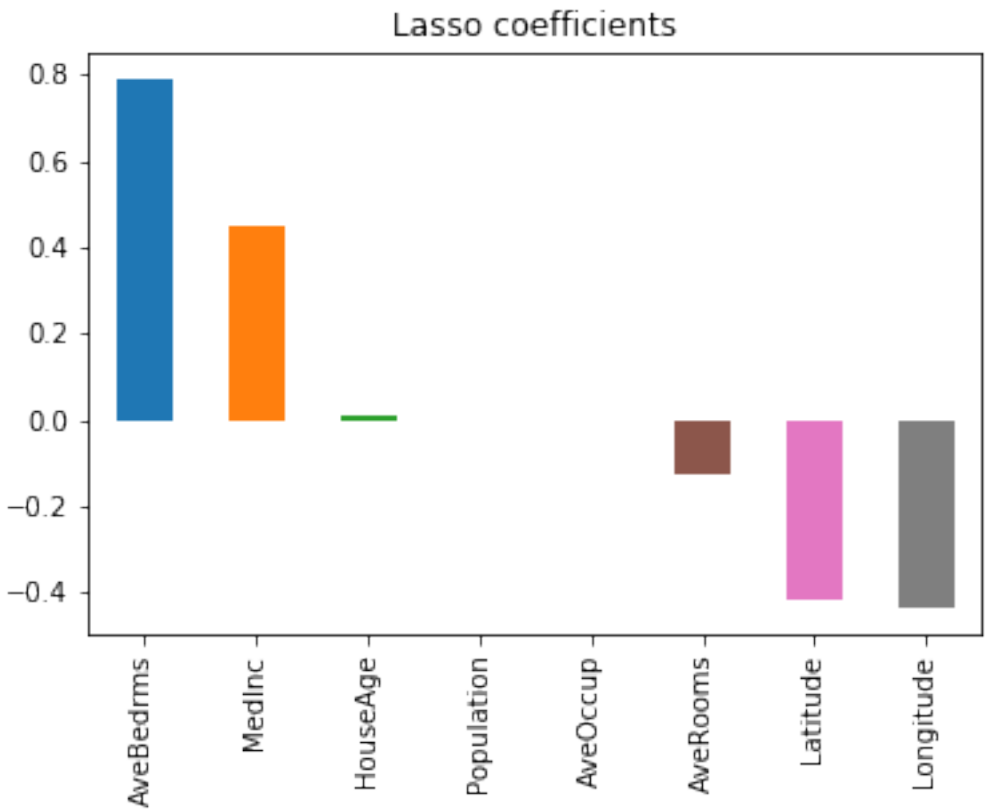
```
In [175]: grid_rr.best_estimator_.coef_

Out[175]: array([ 4.47141672e-01,  9.57239325e-03, -1.23889506e-01,  7.89627880e-01,
                 -1.42768907e-06, -3.44174434e-03, -4.18546175e-01, -4.33334838e-01])

In [182]: ridge = grid_rr.best_estimator_
          ridge_coef = pd.Series(ridge.coef_,housing__data_df.columns).sort_values(ascending=Fal
          ridge_coef.plot(kind='bar')
          plt.xticks(rotation='vertical')
          plt.title("Ridge coefficients")
          plt.show()
          print (ridge_coef)
```




```
AveBedrms     0.789628
MedInc        0.447142
HouseAge      0.009572
Population   -0.000001
AveOccup     -0.003442
AveRooms     -0.123890
Latitude     -0.418546
Longitude    -0.433335
dtype: float64
```

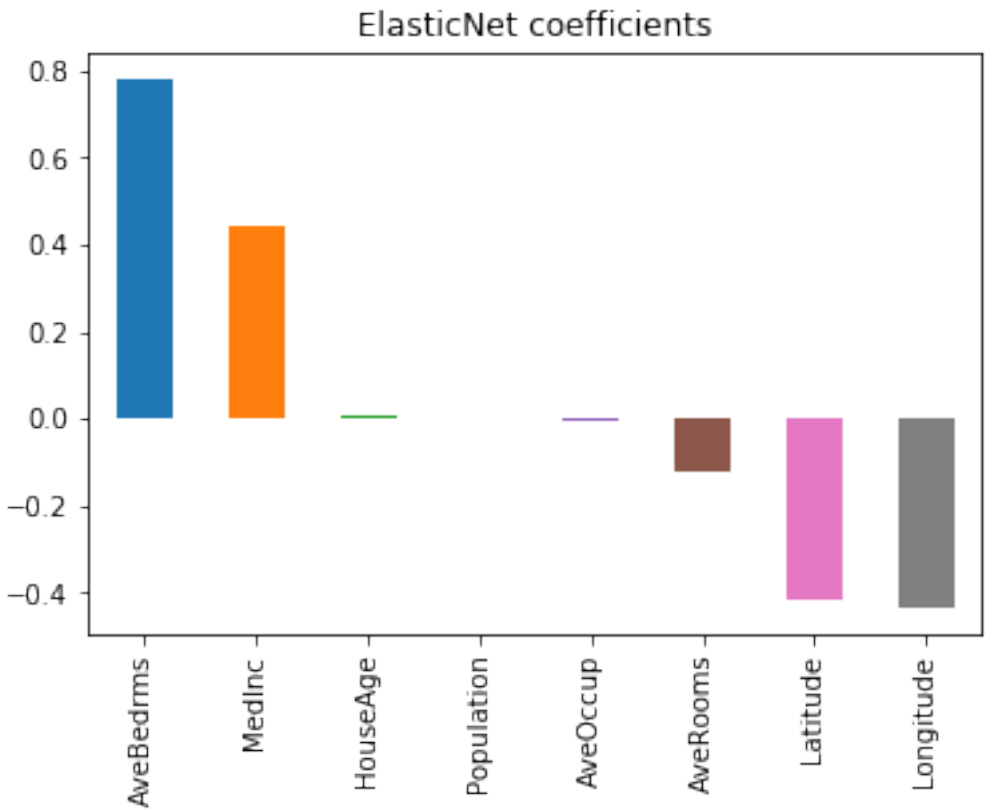
In [ ]:

In [181]:
```
lasso = grid_lasso.best_estimator_
lasso_coef = pd.Series(lasso.coef_,housing__data_df.columns).sort_values(ascending=Fal
lasso_coef.plot(kind='bar')
plt.xticks(rotation='vertical')
plt.title("Lasso coefficients")
plt.show()
print (lasso_coef)
```




```
AveBedrms     0.788690
MedInc        0.447054
HouseAge      0.009576
Population   -0.000001
AveOccup     -0.003441
AveRooms     -0.123712
Latitude     -0.418385
Longitude    -0.433150
dtype: float64
```

```
In [183]: elastic_net =grid_en.best_estimator_
          elastic_net_coef = pd.Series(elastic_net.coef_,housing__data_df.columns).sort_values(a
          elastic_net_coef.plot(kind='bar')
          plt.xticks(rotation='vertical')
          plt.title("ElasticNet coefficients")
          plt.show()
          print (elastic_net_coef)
```




```
AveBedrms     0.780693
MedInc        0.446304
HouseAge      0.009598
Population   -0.000001
AveOccup     -0.003434
AveRooms     -0.122219
Latitude     -0.417381
Longitude    -0.431973
dtype: float64
```

8 1.5 Inferences

The coefficients seem to agree on which features are important. All three models have high coefficients (in terms of magnitude) for the first 2 and the last 2 features while the middle ones are close to 0

In [ ]:

In [ ]:

In [ ]:

In [ ]:

9 2.1

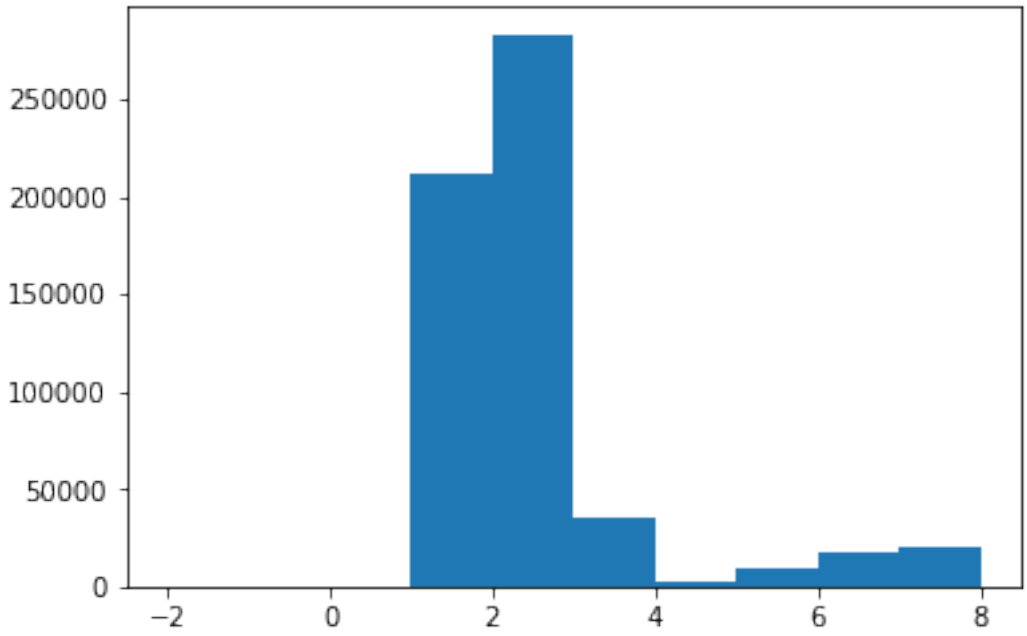
In [3]:
```
from sklearn.datasets import fetch_covtype
```

In [4]:
```
data = fetch_covtype()
```

In [5]:
```
print(data.DESCR)
```

```
Forest covertype dataset.

A classic dataset for classification benchmarks, featuring categorical and
real-valued features.

The dataset page is available from UCI Machine Learning Repository

    http://archive.ics.uci.edu/ml/datasets/Covertype

Courtesy of Jock A. Blackard and Colorado State University.
```

In [6]:
```
plt.hist(data.target, bins=range(-2,9))
plt.show()
#The target values are discrete integers representing the type of forest cover
#Thus, a majority of the values are either type 1 or 2 forest cover
```



```
In [202]: titles = ['Elevation','Aspect','Slope','Horizontal_Distance_To_Hydrology','Vertical_Di

In [7]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, random_state

In [8]: import pandas as pd

        cov_df = pd.DataFrame(data.data)

In [9]: target_df = pd.DataFrame(data.target)
        target_df.head
        data.data.shape
        #data.feature_names

Out[9]: (581012, 54)
```

- Here we observe that the data contains 2 features that are represented in 4 and 40 columns respectively in one hot encoded fashion. Thus in essence the data has 12 features, represented across 54 columns.
- Given the large dimensionality of the data, we will convert the one hot representation to a numerical category representation that occupies just one column per feature.
- This will reduce the dimensionality of the dataset and vastly improve computation time to fit our machine learning models

```
In [188]: wild_df = data_df[data_df.columns[10:14]]
          #wild_df.head
          #wild_df.shape

In [189]: soil_df = data_df[data_df.columns[14:54]]
          #soil_df.head
          #soil_df.shape

In [190]: print(wild_df.sum().sum())
          print(soil_df.sum().sum())

581012.0
581012.0
```

This means that the columns for wilderness area and soil type are one hot. We can convert them into a numerical categorical columns

```
In [191]: # Compressing all those columns into a single columns for the wilderness and soil type
          x_wild = wild_df.stack()
          wild_col = pd.Series(pd.Categorical(x_wild[x_wild!=0].index.get_level_values(1)))

          x_soil = soil_df.stack()
          soil_col = pd.Series(pd.Categorical(x_soil[x_soil!=0].index.get_level_values(1)))
```

```
In [192]: soil_col_i = soil_col.cat.codes
          wild_col_i = wild_col.cat.codes

In [196]: new_cov_data = cov_df[cov_df.columns[0:10]]
          new_cov_data.head
          #cov_df.head
          new_cov_data[10] = wild_col_i
          new_cov_data[11] = soil_col_i

/home/adi/.conda/envs/stuff/lib/python3.5/site-packages/ipykernel_launcher.py:4: SettingWithCopy


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#

/home/adi/.conda/envs/stuff/lib/python3.5/site-packages/ipykernel_launcher.py:5: SettingWithCopy


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#



In [197]: #This is the new dataset
          new_cov_data.head

Out[197]: <bound method NDFrame.head of             0      1     2      3      4       5      6
          0       2596.0   51.0   3.0  258.0    0.0   510.0  221.0  232.0  148.0
          1       2590.0   56.0   2.0  212.0   -6.0   390.0  220.0  235.0  151.0
          2       2804.0  139.0   9.0  268.0   65.0  3180.0  234.0  238.0  135.0
          3       2785.0  155.0  18.0  242.0  118.0  3090.0  238.0  238.0  122.0
          4       2595.0   45.0   2.0  153.0   -1.0   391.0  220.0  234.0  150.0
          5       2579.0  132.0   6.0  300.0  -15.0    67.0  230.0  237.0  140.0
          6       2606.0   45.0   7.0  270.0    5.0   633.0  222.0  225.0  138.0
          7       2605.0   49.0   4.0  234.0    7.0   573.0  222.0  230.0  144.0
          8       2617.0   45.0   9.0  240.0   56.0   666.0  223.0  221.0  133.0
          9       2612.0   59.0  10.0  247.0   11.0   636.0  228.0  219.0  124.0
          10      2612.0  201.0   4.0  180.0   51.0   735.0  218.0  243.0  161.0
          11      2886.0  151.0  11.0  371.0   26.0  5253.0  234.0  240.0  136.0
          12      2742.0  134.0  22.0  150.0   69.0  3215.0  248.0  224.0   92.0
          13      2609.0  214.0   7.0  150.0   46.0   771.0  213.0  247.0  170.0
          14      2503.0  157.0   4.0   67.0    4.0   674.0  224.0  240.0  151.0
          15      2495.0   51.0   7.0   42.0    2.0   752.0  224.0  225.0  137.0
          16      2610.0  259.0   1.0  120.0   -1.0   607.0  216.0  239.0  161.0
          17      2517.0   72.0   7.0   85.0    6.0   595.0  228.0  227.0  133.0
```

18	2504.0	0.0	4.0	95.0	5.0	691.0	214.0	232.0	156.0
19	2503.0	38.0	5.0	85.0	10.0	741.0	220.0	228.0	144.0
20	2501.0	71.0	9.0	60.0	8.0	767.0	230.0	223.0	126.0
21	2880.0	209.0	17.0	216.0	30.0	4986.0	206.0	253.0	179.0
22	2768.0	114.0	23.0	192.0	82.0	3339.0	252.0	209.0	71.0
23	2511.0	54.0	8.0	124.0	0.0	638.0	225.0	222.0	130.0
24	2507.0	22.0	9.0	120.0	14.0	732.0	215.0	221.0	143.0
25	2492.0	135.0	6.0	0.0	0.0	860.0	229.0	237.0	142.0
26	2489.0	163.0	10.0	30.0	-4.0	849.0	230.0	243.0	145.0
27	2962.0	148.0	16.0	323.0	23.0	5916.0	240.0	236.0	120.0
28	2811.0	135.0	1.0	212.0	30.0	3670.0	220.0	238.0	154.0
29	2739.0	117.0	24.0	127.0	53.0	3281.0	253.0	210.0	71.0
...	...	...	...	...	...	...	...	...	...
580982	2431.0	182.0	9.0	300.0	23.0	268.0	223.0	246.0	156.0
580983	2436.0	219.0	15.0	300.0	28.0	256.0	203.0	253.0	184.0
580984	2441.0	201.0	15.0	300.0	33.0	247.0	212.0	252.0	171.0
580985	2442.0	191.0	17.0	300.0	34.0	242.0	217.0	251.0	162.0
580986	2444.0	202.0	21.0	300.0	37.0	240.0	205.0	253.0	174.0
580987	2450.0	203.0	27.0	300.0	53.0	240.0	196.0	252.0	176.0
580988	2455.0	189.0	31.0	295.0	58.0	240.0	205.0	245.0	152.0
580989	2455.0	181.0	34.0	277.0	58.0	240.0	210.0	238.0	133.0
580990	2455.0	166.0	35.0	258.0	58.0	240.0	225.0	227.0	103.0
580991	2445.0	157.0	33.0	242.0	49.0	240.0	235.0	224.0	91.0
580992	2441.0	173.0	28.0	228.0	45.0	242.0	225.0	240.0	128.0
580993	2440.0	173.0	26.0	216.0	44.0	234.0	226.0	242.0	132.0
580994	2437.0	170.0	23.0	201.0	45.0	216.0	229.0	242.0	131.0
580995	2437.0	174.0	22.0	190.0	45.0	201.0	227.0	245.0	139.0
580996	2435.0	171.0	22.0	175.0	43.0	190.0	229.0	244.0	135.0
580997	2433.0	168.0	23.0	162.0	41.0	175.0	231.0	241.0	128.0
580998	2429.0	166.0	24.0	153.0	45.0	162.0	232.0	240.0	125.0
580999	2426.0	168.0	24.0	150.0	42.0	153.0	231.0	241.0	127.0
581000	2423.0	169.0	24.0	134.0	39.0	150.0	230.0	241.0	128.0
581001	2421.0	172.0	25.0	124.0	35.0	134.0	227.0	242.0	132.0
581002	2419.0	168.0	25.0	108.0	33.0	124.0	230.0	240.0	126.0
581003	2415.0	161.0	25.0	95.0	29.0	120.0	236.0	237.0	116.0
581004	2410.0	158.0	24.0	90.0	24.0	120.0	238.0	236.0	115.0
581005	2405.0	159.0	22.0	90.0	19.0	120.0	237.0	238.0	119.0
581006	2401.0	157.0	21.0	90.0	15.0	120.0	238.0	238.0	119.0
581007	2396.0	153.0	20.0	85.0	17.0	108.0	240.0	237.0	118.0
581008	2391.0	152.0	19.0	67.0	12.0	95.0	240.0	237.0	119.0
581009	2386.0	159.0	17.0	60.0	7.0	90.0	236.0	241.0	130.0
581010	2384.0	170.0	15.0	60.0	5.0	90.0	230.0	245.0	143.0
581011	2383.0	165.0	13.0	60.0	4.0	67.0	231.0	244.0	141.0

	9	10	11
0	6279.0	0	28
1	6225.0	0	28
2	6121.0	0	11

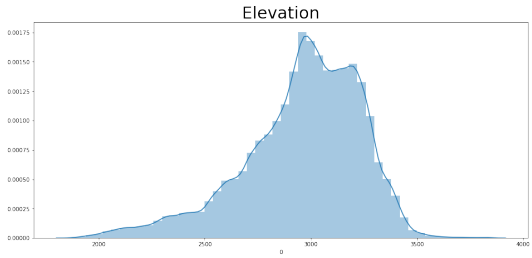
3	6211.0	0	29
4	6172.0	0	28
5	6031.0	0	28
6	6256.0	0	28
7	6228.0	0	28
8	6244.0	0	28
9	6230.0	0	28
10	6222.0	0	17
11	4051.0	0	29
12	6091.0	0	29
13	6211.0	0	17
14	5600.0	0	17
15	5576.0	0	15
16	6096.0	0	28
17	5607.0	0	17
18	5572.0	0	17
19	5555.0	0	17
20	5547.0	0	17
21	4323.0	0	29
22	5972.0	0	29
23	5569.0	0	17
24	5534.0	0	17
25	5494.0	0	17
26	5486.0	0	17
27	3395.0	0	28
28	5643.0	0	11
29	6033.0	0	29
...	...	..	..
580982	973.0	2	1
580983	957.0	2	1
580984	942.0	2	1
580985	927.0	2	1
580986	912.0	2	1
580987	899.0	2	1
580988	886.0	2	1
580989	875.0	2	1
580990	864.0	2	1
580991	854.0	2	1
580992	845.0	2	1
580993	837.0	2	1
580994	830.0	2	1
580995	824.0	2	1
580996	819.0	2	1
580997	815.0	2	1
580998	812.0	2	1
580999	811.0	2	1
581000	810.0	2	1
581001	811.0	2	1

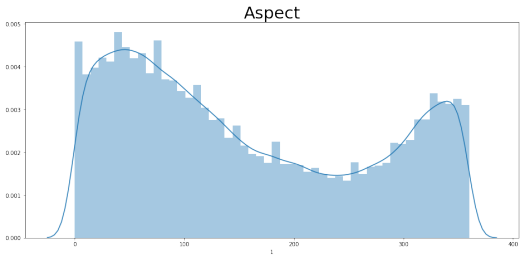
```
581002   812.0   2   1
581003   815.0   2   1
581004   819.0   2   1
581005   824.0   2   1
581006   830.0   2   1
581007   837.0   2   1
581008   845.0   2   1
581009   854.0   2   1
581010   864.0   2   1
581011   875.0   2   1

[581012 rows x 12 columns]>
```

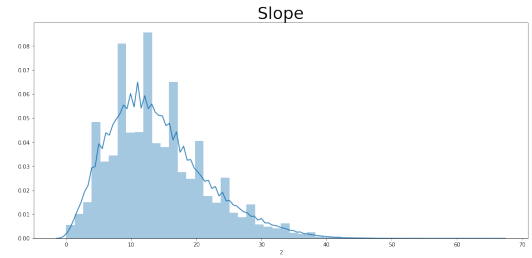
In [66]:
```
final_data = np.array(new_cov_data)
final_target = np.array(data.target)
```

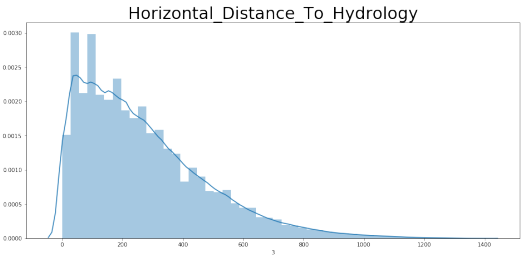
In [207]:
```
#Plotting univariate distribution of each feature:
fig,ax = plt.subplots(6,2, figsize=(35,50)) #initiating 3*3 grid of plots
k = 0
for i in range(6):
    for j in range(2):
        if(i+j<54):
            #print(i,j)
            sns.distplot(new_cov_data.iloc[:,k],ax = ax[i,j])
            ax[i,j].set_title(titles[k], fontsize = 30)
            k+=1

plt.show
```

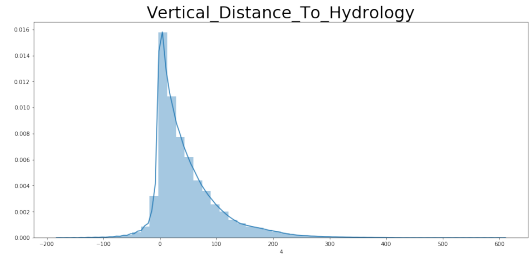
Out[207]:
```
<function matplotlib.pyplot.show>
```

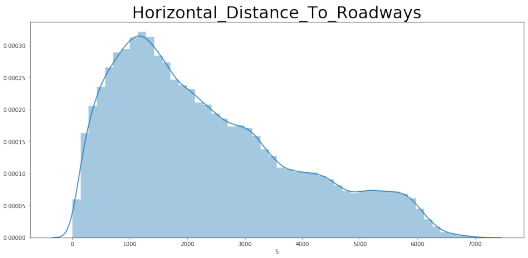

Elevation

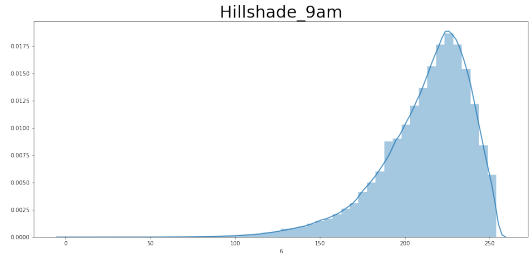

Aspect

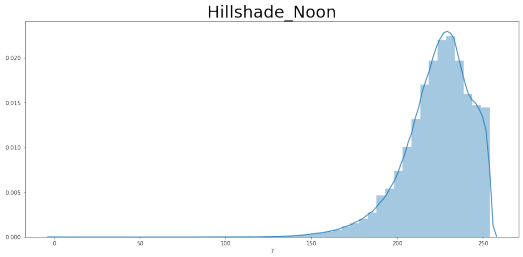

Slope

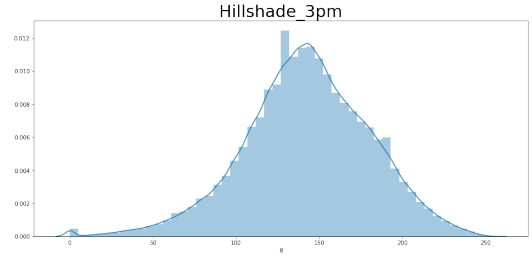

Horizontal_Distance_To_Hydrology

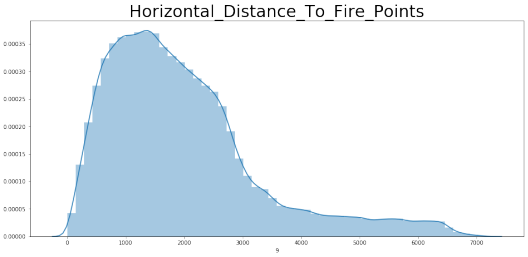

Vertical_Distance_To_Hydrology

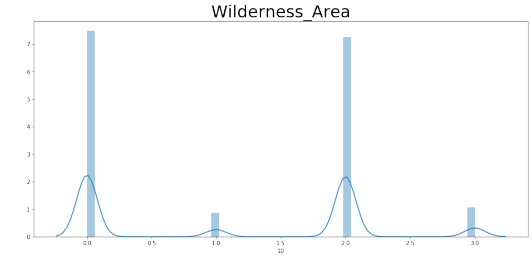

Horizontal_Distance_To_Roadways

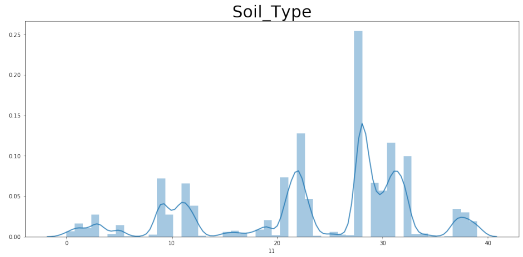

Hillshade_9am


Hillshade_Noon


Hillshade_3pm


Horizontal_Distance_To_Fire_Points


Wilderness_Area


Soil_Type

10 2.2

```
In [209]: X_train, X_test, y_train, y_test = train_test_split(new_cov_data, data.target, random_
```

```
In [35]: from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
         from sklearn.svm import LinearSVC
         from sklearn.neighbors import NearestCentroid
         from sklearn.model_selection import cross_val_score
         from sklearn.preprocessing import StandardScaler
         from sklearn.model_selection import GridSearchCV
```

```
In [ ]:
```

```
In [214]: lr = LogisticRegression()
          svc = LinearSVC()
          nc = NearestCentroid()
```

```
In [96]: lr.fit(X_train,y_train)
```

```
Out[96]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
In [99]: lr.score(X_test,y_test)
```

```
Out[99]: 0.6725988447742903
```

```
In [100]: print("Logistic Regression mean cross val score is:{:.3f}".format(np.mean(cross_val_sc
```

```
Logistic Regression mean cross val score is:0.674
```

11 Logistic Regression mean cross val score is: 0.674

```
In [22]: svc.fit(X_train,y_train)
```

```
Out[22]: LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
              intercept_scaling=1, loss='squared_hinge', max_iter=1000,
              multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
              verbose=0)
```

```
In [25]: print("Logistic Regression meanfrom sklearn.model_selection import cross_val_score cros
```

```
Logistic Regression meanfrom sklearn.model_selection import cross_val_score cross val score is:0
```

12 Linear SVC mean cross val score is 0.476

```
In [215]: nc.fit(X_train,y_train)
          nc.score(X_test,y_test)
          print("Nearest Centroid cross val score is:{:.3f}".format(np.mean(cross_val_score(nc,
```

```
Nearest Centroid cross val score is:0.193
```

13 Nearest Centroid cross val score is 0.193

```
In [216]: scaler = StandardScaler()
          scaler.fit(X_train)
          X_train_scaled = scaler.transform(X_train)
          X_test_scaled = scaler.transform(X_test)
```

```
In [217]: nc.fit(X_train_scaled,y_train)
          print(nc.score(X_test_scaled,y_test))
          print("Nearest Centroid cross val score is:{:.3f}".format(np.mean(cross_val_score(nc,
```

```
0.4219327655883183
Nearest Centroid cross val score is:0.424
```

14 Scaled nearest centroid cross val score is 0.424

```
In [218]: lr.fit(X_train_scaled,y_train)
          print(lr.score(X_test_scaled,y_test))
          print("Logistic Regression scaled cross val score is:{:.3f}".format(np.mean(cross_val_
```

```
0.6778586328681679
Logistic Regression scaled cross val score is:0.678
```

15 Scaled Logistic Regression cross val score is 0.678

```
In [213]: svc = LinearSVC(dual=False,tol=0.001)
          svc.fit(X_train_scaled,y_train)
          print(svc.score(X_test_scaled,y_test))
          print("Linear SVC scaled cross val score is:{:.3f}".format(np.mean(cross_val_score(svc
```

```
0.670258101381727
Linear SVC scaled cross val score is:0.670
```

16 Scaled Linear SVC has a cross val score of 0.670

Thus we see that scaling in this case greatly improves performance across all models

17 2.3

```
In [41]: lr = LogisticRegression(multi_class='multinomial', dual=False, solver='lbfgs', tol=0.00
         svc = LinearSVC(dual=False,tol=0.001)
         nc = NearestCentroid()
```

```
In [46]: #For Logistic
         param_grid_log = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
         print(param_grid_log)

         grid_log = GridSearchCV(lr, param_grid_log, cv=3,return_train_score=True)
         grid_log.fit(X_train_scaled, y_train)
         print(grid_log.best_params_)
         print(grid_log.best_score_)
         print("test-set score: {:.3f}".format(grid_log.score(X_test, y_test)))
         pd.DataFrame(grid_log.cv_results_)
```

```
{'C': [0.001, 0.01, 0.1, 1, 10, 100]}
{'C': 100}
0.7129927322212507
test-set score: 0.035
```

Out[46]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C
0	20.584035	0.022919	0.703719	0.703786	0.001
1	22.608849	0.022520	0.711260	0.711278	0.01
2	22.875478	0.022392	0.712552	0.712554	0.1
3	23.692736	0.023728	0.712850	0.712873	1
4	25.118319	0.026736	0.712885	0.712962	10
5	23.679013	0.022638	0.712993	0.712945	100

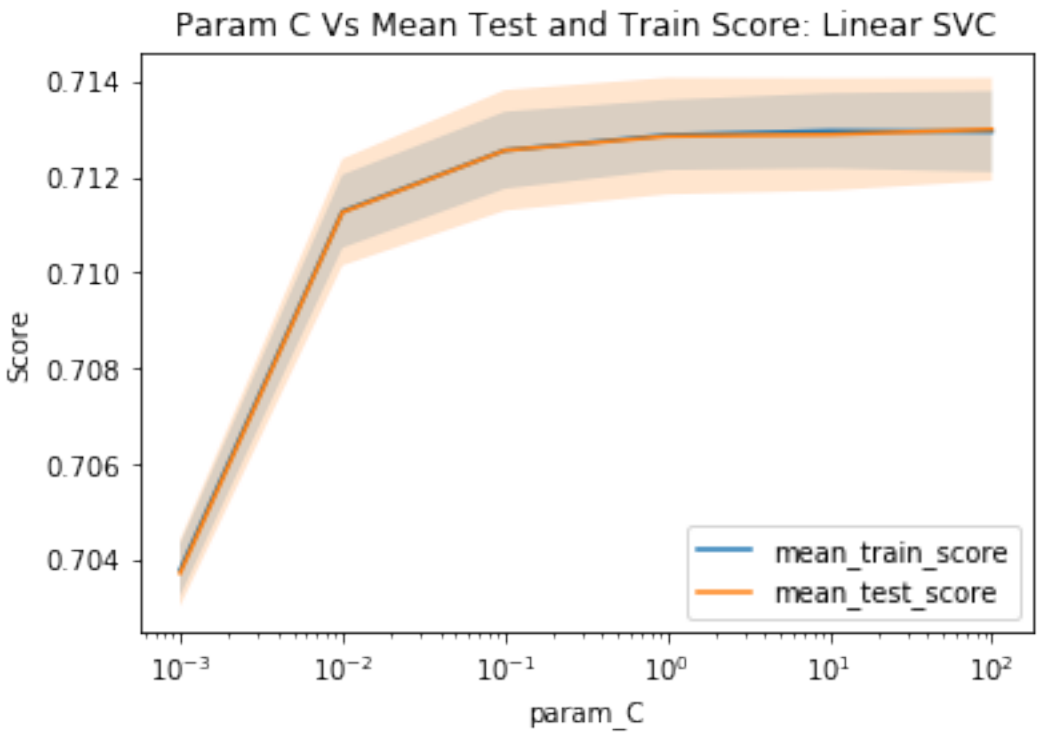
	params	rank_test_score	split0_test_score	split0_train_score
0	{'C': 0.001}	6	0.702814	0.704526
1	{'C': 0.01}	5	0.709699	0.712347
2	{'C': 0.1}	4	0.710828	0.713638
3	{'C': 1}	3	0.711165	0.713786
4	{'C': 10}	2	0.711296	0.714003
5	{'C': 100}	1	0.711599	0.714110

	split1_test_score	split1_train_score	split2_test_score
0	0.704488	0.703245	0.703855
1	0.712212	0.710622	0.711869
2	0.713803	0.711717	0.713026
3	0.713996	0.711996	0.713391
4	0.714092	0.712109	0.713267
5	0.714209	0.712137	0.713170

	split2_train_score	std_fit_time	std_score_time	std_test_score
0	0.703586	0.563584	0.000366	0.000690

```
1            0.710866      0.297692        0.000160        0.001113
2            0.712308      0.525774        0.000105        0.001260
3            0.712839      0.741672        0.001538        0.001217
4            0.712773      0.554335        0.003991        0.001173
5            0.712587      0.224756        0.000152        0.001073

   std_train_score
0         0.000542
1         0.000762
2         0.000803
3         0.000731
4         0.000785
5         0.000844
```

In [48]:
```
resultsSVC = pd.DataFrame(grid_log.cv_results_)

#Plotting Param_C vs Mean Test and Mean Train Score
resultsSVC.plot('param_C', 'mean_train_score')
resultsSVC.plot('param_C', 'mean_test_score', ax=plt.gca())
plt.fill_between(resultsSVC.param_C.astype(np.float),
                 resultsSVC['mean_train_score'] + resultsSVC['std_train_score'],
                 resultsSVC['mean_train_score'] - resultsSVC['std_train_score'], alpha=
plt.fill_between(resultsSVC.param_C.astype(np.float),
                 resultsSVC['mean_test_score'] + resultsSVC['std_test_score'],
                 resultsSVC['mean_test_score'] - resultsSVC['std_test_score'], alpha=0.
plt.legend()
plt.title("Param C Vs Mean Test and Train Score: Linear SVC")
plt.ylabel("Score")
plt.xscale("log")
plt.savefig("SVC_C_search.png")
```




```
In [43]: param_grid_svc = {'C': [0.01, 0.1, 1, 10, 100]}
         print(param_grid_svc)

         grid_svc = GridSearchCV(svc, param_grid_svc, cv=3,return_train_score=True)
         grid_svc.fit(X_train_scaled, y_train)
         print(grid_svc.best_params_)
         print(grid_svc.best_score_)
         print("test-set score: {:.3f}".format(grid_log.score(X_test, y_test)))
         pd.DataFrame(grid_svc.cv_results_)
```

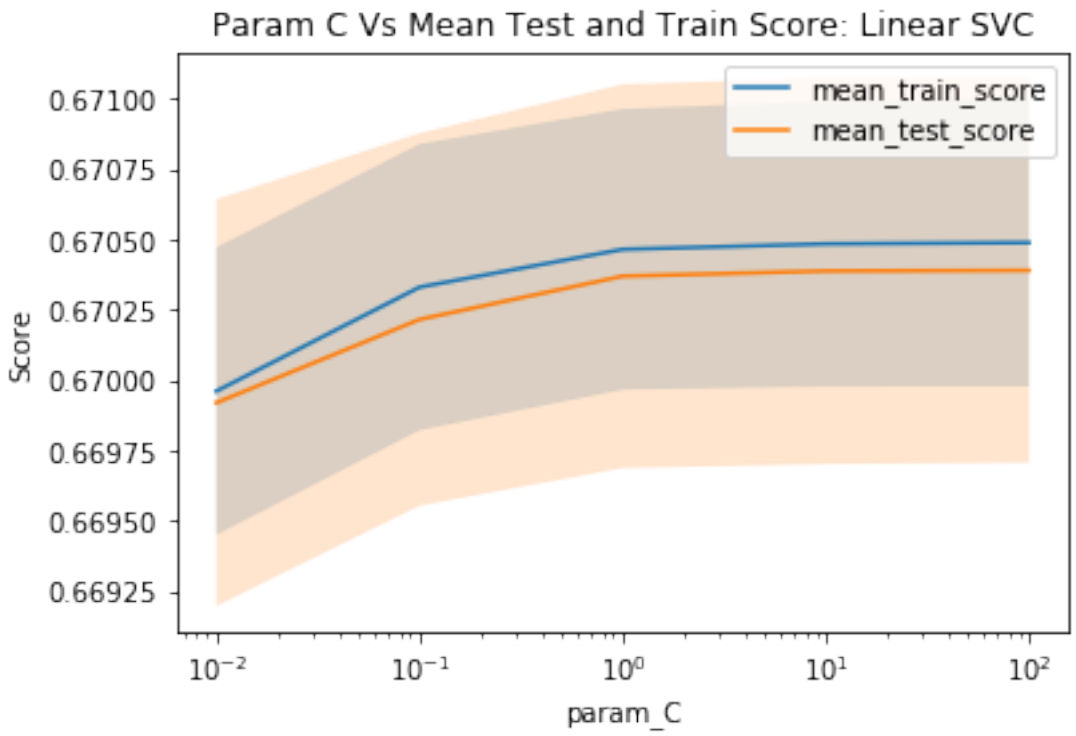
```
{'C': [0.01, 0.1, 1, 10, 100]}
{'C': 100}
0.6703912024765983
test-set score: 0.624
```

Out[43]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	\
0	6.876013	0.022590	0.669921	0.669962	0.01	
1	6.905701	0.023034	0.670217	0.670332	0.1	
2	7.768724	0.027496	0.670371	0.670466	1	
3	7.719738	0.024892	0.670389	0.670485	10	
4	6.850312	0.022570	0.670391	0.670490	100	

```
        params  rank_test_score  split0_test_score  split0_train_score  \
0  {'C': 0.01}                5           0.668971            0.670682
1   {'C': 0.1}                4           0.669308            0.671050
2     {'C': 1}                3           0.669432            0.671170
3    {'C': 10}                2           0.669446            0.671201
4   {'C': 100}                1           0.669453            0.671212

   split1_test_score  split1_train_score  split2_test_score  \
0           0.670072            0.669649           0.670719
1           0.670478            0.669983           0.670864
2           0.670651            0.670103           0.671029
3           0.670664            0.670110           0.671057
4           0.670664            0.670114           0.671057

   split2_train_score  std_fit_time  std_score_time  std_test_score  \
0            0.669556      0.253053        0.000290        0.000722
1            0.669962      0.041432        0.000965        0.000662
2            0.670124      0.586498        0.003033        0.000682
3            0.670144      0.538737        0.002450        0.000686
4            0.670144      0.184395        0.000120        0.000683

   std_train_score
0         0.000510
1         0.000508
2         0.000498
3         0.000507
4         0.000511
```

In [47]:
```
resultsSVC = pd.DataFrame(grid_svc.cv_results_)

#Plotting Param_C vs Mean Test and Mean Train Score
resultsSVC.plot('param_C', 'mean_train_score')
resultsSVC.plot('param_C', 'mean_test_score', ax=plt.gca())
plt.fill_between(resultsSVC.param_C.astype(np.float),
                 resultsSVC['mean_train_score'] + resultsSVC['std_train_score'],
                 resultsSVC['mean_train_score'] - resultsSVC['std_train_score'], alpha=
plt.fill_between(resultsSVC.param_C.astype(np.float),
                 resultsSVC['mean_test_score'] + resultsSVC['std_test_score'],
                 resultsSVC['mean_test_score'] - resultsSVC['std_test_score'], alpha=0.
plt.legend()
plt.title("Param C Vs Mean Test and Train Score: Linear SVC")
plt.ylabel("Score")
plt.xscale("log")
plt.savefig("SVC_C_search.png")
```




```
In [45]: param_grid_nc = {'shrink_threshold': [0.01, 0.1, 1, 10, 15, 20, 30]}
         print(param_grid_nc)

         grid_nc = GridSearchCV(nc, param_grid_nc, cv=3,return_train_score=True)
         grid_nc.fit(X_train_scaled, y_train)
         print(grid_nc.best_params_)
         print(grid_nc.best_score_)
         print("test-set score: {:.3f}".format(grid_nc.score(X_test, y_test)))
         pd.DataFrame(grid_nc.cv_results_)

{'shrink_threshold': [0.01, 0.1, 1, 10, 15, 20, 30]}
{'shrink_threshold': 0.01}
0.42425744505563856
test-set score: 0.035
```

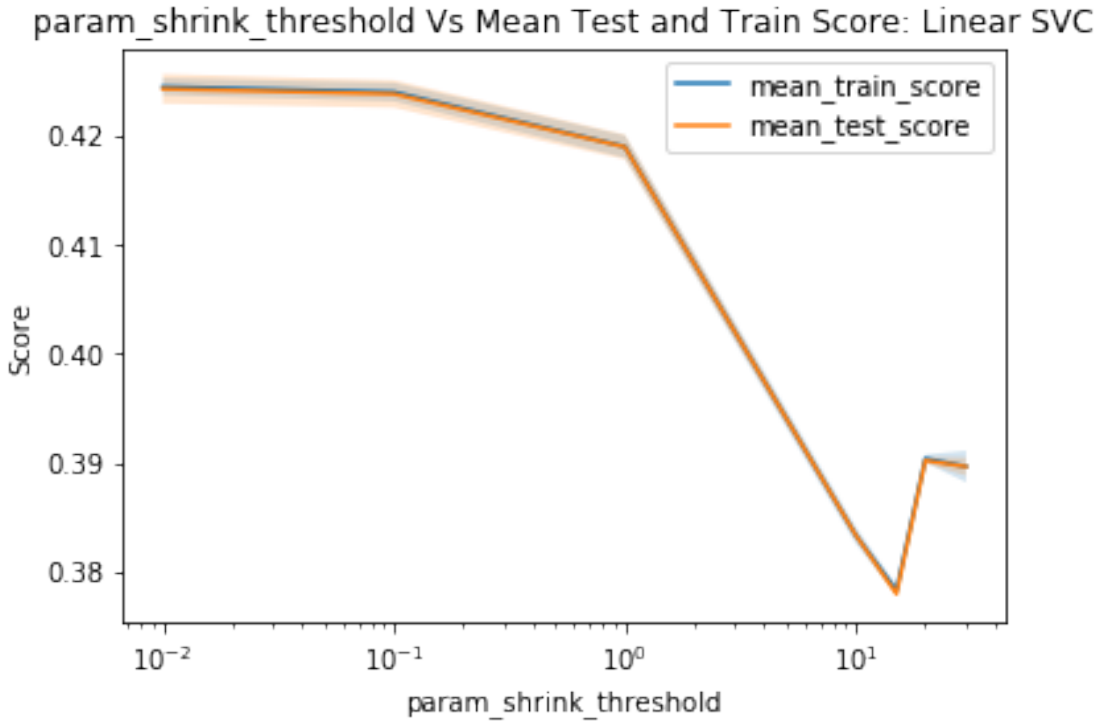
Out[45]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	\
0	0.129788	0.035022	0.424257	0.424447	
1	0.122453	0.035777	0.423787	0.423960	
2	0.122781	0.036052	0.418931	0.418964	
3	0.121249	0.035177	0.383347	0.383346	
4	0.122157	0.035026	0.377986	0.378287	
5	0.123654	0.035760	0.390200	0.390373	

```
6       0.120170         0.035071         0.389633          0.389644

  param_shrink_threshold                      params  rank_test_score  \
0                   0.01  {'shrink_threshold': 0.01}                1
1                    0.1   {'shrink_threshold': 0.1}                2
2                      1     {'shrink_threshold': 1}                3
3                     10    {'shrink_threshold': 10}                6
4                     15    {'shrink_threshold': 15}                7
5                     20    {'shrink_threshold': 20}                4
6                     30    {'shrink_threshold': 30}                5

   split0_test_score  split0_train_score  split1_test_score  \
0           0.425463            0.423837           0.424948
1           0.424926            0.423328           0.424404
2           0.420024            0.418354           0.419420
3           0.382945            0.382265           0.383620
4           0.378243            0.377869           0.378071
5           0.390600            0.389872           0.390085
6           0.388755            0.387717           0.390732

   split1_train_score  split2_test_score  split2_train_score  std_fit_time  \
0            0.425540           0.422361            0.423963      0.008907
1            0.425144           0.422031            0.423409      0.001867
2            0.420394           0.417349            0.418145      0.001678
3            0.383889           0.383477            0.383885      0.002542
4            0.378078           0.377645            0.378914      0.001311
5            0.390525           0.389914            0.390721      0.003044
6            0.390040           0.389411            0.391175      0.000741

   std_score_time  std_test_score  std_train_score
0        0.000052        0.001357         0.000775
1        0.000347        0.001260         0.000838
2        0.000571        0.001145         0.001015
3        0.000241        0.000291         0.000765
4        0.000313        0.000251         0.000452
5        0.000588        0.000292         0.000363
6        0.000154        0.000822         0.001439
```

In [49]:
```
resultsSVC = pd.DataFrame(grid_nc.cv_results_)

#Plotting Param_C vs Mean Test and Mean Train Score
resultsSVC.plot('param_shrink_threshold', 'mean_train_score')
resultsSVC.plot('param_shrink_threshold', 'mean_test_score', ax=plt.gca())
plt.fill_between(resultsSVC.param_shrink_threshold.astype(np.float),
                 resultsSVC['mean_train_score'] + resultsSVC['std_train_score'],
                 resultsSVC['mean_train_score'] - resultsSVC['std_train_score'], alpha=
plt.fill_between(resultsSVC.param_shrink_threshold.astype(np.float),
                 resultsSVC['mean_test_score'] + resultsSVC['std_test_score'],
```

```
                        resultsSVC['mean_test_score'] - resultsSVC['std_test_score'], alpha=0.
plt.legend()
plt.title("param_shrink_threshold Vs Mean Test and Train Score: Linear SVC")
plt.ylabel("Score")
plt.xscale("log")
plt.savefig("NC_Shrink_search.png")
```



18 2.3 Inference

The results improve marginally across for LinearSVC and Nearest Centroid. However, there is a sharp rise in performance for Logistic regression

19 2.4

```
In [50]: from sklearn.model_selection import KFold
```

```
In [52]: lr = LogisticRegression(multi_class='multinomial', dual=False, solver='lbfgs', tol=0.00
         svc = LinearSVC(dual=False,tol=0.001)
         nc = NearestCentroid()


         #For Logistic Reg kfold with suffling
```

```
kf = KFold(random_state=42, shuffle=True)
param_grid_log_k = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

#param_grid_log = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
print(param_grid_log)

grid_log_k = GridSearchCV(lr, param_grid_log_k, cv=kf,return_train_score=True)
grid_log_k.fit(X_train_scaled, y_train)
print(grid_log_k.best_params_)
print(grid_log_k.best_score_)
print("test-set score: {:.3f}".format(grid_log_k.score(X_test, y_test)))
pd.DataFrame(grid_log_k.cv_results_)
```

```
{'C': [0.001, 0.01, 0.1, 1, 10, 100]}
{'C': 100}
0.712848156894063
test-set score: 0.035
```

Out[52]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C
0	20.837193	0.022542	0.703607	0.703750	0.001
1	23.716816	0.023102	0.711189	0.711265	0.01
2	23.290579	0.023258	0.712410	0.712561	0.1
3	22.956206	0.022374	0.712761	0.712863	1
4	22.851773	0.022379	0.712731	0.712856	10
5	23.349914	0.022778	0.712848	0.712969	100

	params	rank_test_score	split0_test_score	split0_train_score
0	{'C': 0.001}	6	0.702189	0.704309
1	{'C': 0.01}	5	0.711228	0.711514
2	{'C': 0.1}	4	0.712591	0.712715
3	{'C': 1}	2	0.712784	0.712915
4	{'C': 10}	3	0.712688	0.712870
5	{'C': 100}	1	0.712825	0.712977

	split1_test_score	split1_train_score	split2_test_score
0	0.704137	0.703256	0.704495
1	0.711531	0.710932	0.710808
2	0.712701	0.712268	0.711937
3	0.713128	0.712770	0.712371
4	0.713225	0.712805	0.712281
5	0.713231	0.712805	0.712488

	split2_train_score	std_fit_time	std_score_time	std_test_score
0	0.703686	0.302540	0.000237	0.001013
1	0.711348	0.658001	0.000469	0.000296
2	0.712701	0.266904	0.001010	0.000337
3	0.712904	0.223298	0.000087	0.000310

```
4            0.712894      0.174427        0.000095        0.000386
5            0.713125      0.633673        0.000356        0.000304

   std_train_score
0         0.000432
1         0.000245
2         0.000208
3         0.000066
4         0.000038
5         0.000131
```

```
In [53]: #For Logistic Reg
         kf2 = KFold(random_state = 69, shuffle=True)
         param_grid_log_k = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

         #param_grid_log = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
         print(param_grid_log)

         grid_log_k2 = GridSearchCV(lr, param_grid_log_k, cv=kf2,return_train_score=True)
         grid_log_k2.fit(X_train_scaled, y_train)
         print(grid_log_k2.best_params_)
         print(grid_log_k2.best_score_)
         print("test-set score: {:.3f}".format(grid_log_k2.score(X_test, y_test)))
         pd.DataFrame(grid_log_k2.cv_results_)
```

```
{'C': [0.001, 0.01, 0.1, 1, 10, 100]}
{'C': 100}
0.7130363343040533
test-set score: 0.035
```

```
Out[53]:    mean_fit_time  mean_score_time  mean_test_score  mean_train_score param_C  \
         0      20.379207         0.023612         0.703634          0.703726   0.001
         1      23.311115         0.024384         0.711256          0.711237    0.01
         2      23.631054         0.022644         0.712513          0.712476     0.1
         3      23.632562         0.022595         0.712814          0.712696       1
         4      23.243504         0.022490         0.712857          0.712740      10
         5      24.192598         0.024090         0.713036          0.712850     100

                 params  rank_test_score  split0_test_score  split0_train_score  \
         0  {'C': 0.001}                6           0.704371            0.703263
         1   {'C': 0.01}                5           0.711173            0.710808
         2    {'C': 0.1}                4           0.712543            0.712020
         3      {'C': 1}                3           0.712949            0.712364
         4     {'C': 10}                2           0.712867            0.712171
         5    {'C': 100}                1           0.713417            0.712591

            split1_test_score  split1_train_score  split2_test_score  \
```

```
0           0.704722           0.703259           0.701810
1           0.712715           0.710901           0.709879
2           0.714264           0.712257           0.710732
3           0.714622           0.712371           0.710870
4           0.714712           0.712502           0.710994
5           0.714663           0.712447           0.711028

   split2_train_score  std_fit_time  std_score_time  std_test_score  \
0            0.704657      0.044640        0.000838        0.001298
1            0.712003      0.384218        0.001273        0.001159
2            0.713152      0.469125        0.000088        0.001442
3            0.713352      0.470034        0.000342        0.001535
4            0.713548      0.173501        0.000110        0.001518
5            0.713514      1.018058        0.001141        0.001508

   std_train_score
0         0.000658
1         0.000542
2         0.000488
3         0.000464
4         0.000587
5         0.000473
```

In [54]:
```
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(data.data, data.tar
scaler = StandardScaler()
scaler.fit(X_train_new)
X_train_scaled2 = scaler.transform(X_train_new)
X_test_scaled2 = scaler.transform(X_test_new)


#For Logistic Reg
kf2 = KFold(random_state = 69, shuffle=True)
param_grid_log_k = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

#param_grid_log = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
print(param_grid_log)

grid_log_k3 = GridSearchCV(lr, param_grid_log_k, cv=kf2,return_train_score=True)
grid_log_k3.fit(X_train_scaled2, y_train)
print(grid_log_k3.best_params_)
print(grid_log_k3.best_score_)
print("test-set score: {:.3f}".format(grid_log_k3.score(X_test_scaled2, y_test)))
pd.DataFrame(grid_log_k3.cv_results_)
```

```
{'C': [0.001, 0.01, 0.1, 1, 10, 100]}
{'C': 0.001}
0.48769618068703113
test-set score: 0.487
```

Out[54]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C
0	15.352440	0.037557	0.487696	0.487717	0.001
1	28.425483	0.041895	0.487694	0.487718	0.01
2	30.837087	0.037765	0.487689	0.487720	0.1
3	30.947627	0.040173	0.487692	0.487720	1
4	31.462450	0.039106	0.487692	0.487720	10
5	30.429259	0.039000	0.487692	0.487720	100

	params	rank_test_score	split0_test_score	split0_train_score
0	{'C': 0.001}	1	0.488582	0.487281
1	{'C': 0.01}	2	0.488575	0.487281
2	{'C': 0.1}	6	0.488561	0.487288
3	{'C': 1}	3	0.488568	0.487288
4	{'C': 10}	3	0.488568	0.487288
5	{'C': 100}	3	0.488568	0.487288

	split1_test_score	split1_train_score	split2_test_score
0	0.486654	0.488241	0.487852
1	0.486654	0.488241	0.487852
2	0.486654	0.488241	0.487852
3	0.486654	0.488241	0.487852
4	0.486654	0.488241	0.487852
5	0.486654	0.488241	0.487852

	split2_train_score	std_fit_time	std_score_time	std_test_score
0	0.487628	0.074040	0.000144	0.000795
1	0.487632	2.673127	0.002981	0.000792
2	0.487632	0.969374	0.000255	0.000787
3	0.487632	1.437076	0.002198	0.000790
4	0.487632	0.330687	0.001845	0.000790
5	0.487632	0.499907	0.001781	0.000790

	std_train_score
0	0.000397
1	0.000397
2	0.000394
3	0.000394
4	0.000394
5	0.000394

In [57]:
```
#For SVC
kf = KFold(random_state=42, shuffle=True)
param_grid_svc_k = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

#param_grid_log = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
print(param_grid_svc_k)

grid_svc_k = GridSearchCV(svc, param_grid_svc_k, cv=kf,return_train_score=True)
```

```
grid_svc_k.fit(X_train_scaled, y_train)
print(grid_svc_k.best_params_)
print(grid_svc_k.best_score_)
pd.DataFrame(grid_svc_k.cv_results_)
```

```
{'C': [0.001, 0.01, 0.1, 1, 10, 100]}
{'C': 100}
0.6703613694725754
```

Out[57]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C
0	6.444503	0.023908	0.665818	0.666040	0.001
1	6.920749	0.024916	0.669916	0.670025	0.01
2	6.730087	0.023700	0.670240	0.670335	0.1
3	6.804305	0.023620	0.670336	0.670506	1
4	6.862878	0.023600	0.670359	0.670531	10
5	6.845137	0.023622	0.670361	0.670532	100

	params	rank_test_score	split0_test_score	split0_train_score
0	{'C': 0.001}	6	0.666127	0.667198
1	{'C': 0.01}	5	0.670451	0.671132
2	{'C': 0.1}	4	0.670754	0.671404
3	{'C': 1}	3	0.670809	0.671583
4	{'C': 10}	2	0.670836	0.671604
5	{'C': 100}	1	0.670843	0.671604

	split1_test_score	split1_train_score	split2_test_score
0	0.665639	0.665291	0.665687
1	0.669935	0.669305	0.669363
2	0.670382	0.669628	0.669583
3	0.670540	0.669804	0.669659
4	0.670575	0.669835	0.669666
5	0.670575	0.669835	0.669666

	split2_train_score	std_fit_time	std_score_time	std_test_score
0	0.665632	0.231007	0.000237	0.000220
1	0.669638	0.133722	0.001850	0.000444
2	0.669972	0.200418	0.000011	0.000488
3	0.670131	0.171987	0.000138	0.000491
4	0.670155	0.238961	0.000030	0.000502
5	0.670158	0.194593	0.000135	0.000504

	std_train_score
0	0.000830
1	0.000795
2	0.000769
3	0.000773
4	0.000770
5	0.000769

In [58]:
```
#For SVC
kf2 = KFold(random_state=69, shuffle=True)
param_grid_svc_k = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

#param_grid_log = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
print(param_grid_svc_k)

grid_svc_k2 = GridSearchCV(svc, param_grid_svc_k, cv=kf,return_train_score=True)
grid_svc_k2.fit(X_train_scaled, y_train)
print(grid_svc_k2.best_params_)
print(grid_svc_k2.best_score_)
pd.DataFrame(grid_svc_k2.cv_results_)
```

```
{'C': [0.001, 0.01, 0.1, 1, 10, 100]}
{'C': 100}
0.6703613694725754
```

Out[58]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C
0	6.175060	0.022384	0.665818	0.666040	0.001
1	6.839324	0.022141	0.669916	0.670025	0.01
2	6.671474	0.021925	0.670240	0.670335	0.1
3	6.673582	0.021828	0.670336	0.670506	1
4	6.826746	0.023347	0.670359	0.670531	10
5	7.086383	0.026851	0.670361	0.670532	100

	params	rank_test_score	split0_test_score	split0_train_score
0	{'C': 0.001}	6	0.666127	0.667198
1	{'C': 0.01}	5	0.670451	0.671132
2	{'C': 0.1}	4	0.670754	0.671404
3	{'C': 1}	3	0.670809	0.671583
4	{'C': 10}	2	0.670836	0.671604
5	{'C': 100}	1	0.670843	0.671604

	split1_test_score	split1_train_score	split2_test_score
0	0.665639	0.665291	0.665687
1	0.669935	0.669305	0.669363
2	0.670382	0.669628	0.669583
3	0.670540	0.669804	0.669659
4	0.670575	0.669835	0.669666
5	0.670575	0.669835	0.669666

	split2_train_score	std_fit_time	std_score_time	std_test_score
0	0.665632	0.100034	0.000901	0.000220
1	0.669638	0.175879	0.000586	0.000444
2	0.669972	0.180436	0.000113	0.000488
3	0.670131	0.203472	0.000130	0.000491
4	0.670155	0.312098	0.001970	0.000502

```
5            0.670158      0.493710        0.004546        0.000504

   std_train_score
0         0.000830
1         0.000795
2         0.000769
3         0.000773
4         0.000770
5         0.000769
```

In [59]:
```
#For SVC
kf2 = KFold(random_state=69, shuffle=True)
param_grid_svc_k3 = {'C': [0.01, 0.1, 1, 10, 100, 1000]}

#param_grid_log = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
print(param_grid_svc_k3)

grid_svc_k3 = GridSearchCV(svc, param_grid_svc_k3, cv=kf,return_train_score=True)
grid_svc_k3.fit(X_train_scaled2, y_train)
pd.DataFrame(grid_svc_k3.cv_results_)
```

```
{'C': [0.01, 0.1, 1, 10, 100, 1000]}
```

Out[59]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	\
0	37.037791	0.038266	0.487698	0.487723	0.01	
1	28.529061	0.038738	0.487696	0.487721	0.1	
2	27.701664	0.038084	0.487696	0.487721	1	
3	26.535049	0.037932	0.487696	0.487721	10	
4	26.632158	0.038544	0.487696	0.487721	100	
5	31.361303	0.047957	0.487696	0.487721	1000	

	params	rank_test_score	split0_test_score	split0_train_score	\
0	{'C': 0.01}	1	0.488100	0.487529	
1	{'C': 0.1}	2	0.488093	0.487525	
2	{'C': 1}	2	0.488093	0.487525	
3	{'C': 10}	2	0.488093	0.487525	
4	{'C': 100}	2	0.488093	0.487525	
5	{'C': 1000}	2	0.488093	0.487525	

	split1_test_score	split1_train_score	split2_test_score	\
0	0.488589	0.487274	0.486406	
1	0.488589	0.487274	0.486406	
2	0.488589	0.487274	0.486406	
3	0.488589	0.487274	0.486406	
4	0.488589	0.487274	0.486406	
5	0.488589	0.487274	0.486406	

```
   split2_train_score  std_fit_time  std_score_time  std_test_score  \
0            0.488365      5.072516        0.000749        0.000935
1            0.488365      4.605452        0.001068        0.000934
2            0.488365      5.268161        0.000144        0.000934
3            0.488365      4.220784        0.000059        0.000934
4            0.488365      4.125851        0.000854        0.000934
5            0.488365      3.167772        0.006966        0.000934

   std_train_score
0         0.000466
1         0.000467
2         0.000467
3         0.000467
4         0.000467
5         0.000467
```

In [60]:
```
#For Nearest Controid

kf = KFold(random_state=42, shuffle=True)
param_grid_nc = {'shrink_threshold': [0.01, 0.1, 1, 10, 15, 20, 30, 100]}
print(param_grid_nc)

grid_nc_k = GridSearchCV(nc, param_grid_nc, cv=3,return_train_score=True)
grid_nc_k.fit(X_train_scaled, y_train)
pd.DataFrame(grid_nc_k.cv_results_)
```

```
{'shrink_threshold': [0.01, 0.1, 1, 10, 15, 20, 30, 100]}
```

Out[60]:
```
   mean_fit_time  mean_score_time  mean_test_score  mean_train_score  \
0       0.129763         0.036036         0.424257          0.424447
1       0.126100         0.035625         0.423787          0.423960
2       0.124257         0.035446         0.418931          0.418964
3       0.124510         0.035445         0.383347          0.383346
4       0.122507         0.034826         0.377986          0.378287
5       0.129629         0.037064         0.390200          0.390373
6       0.126681         0.037493         0.389633          0.389644
7       0.128099         0.035524         0.428675          0.428661

  param_shrink_threshold                    params  rank_test_score  \
0                   0.01  {'shrink_threshold': 0.01}                2
1                    0.1   {'shrink_threshold': 0.1}                3
2                      1     {'shrink_threshold': 1}                4
3                     10    {'shrink_threshold': 10}                7
4                     15    {'shrink_threshold': 15}                8
5                     20    {'shrink_threshold': 20}                5
6                     30    {'shrink_threshold': 30}                6
7                    100   {'shrink_threshold': 100}                1
```

```
   split0_test_score  split0_train_score  split1_test_score  \
0           0.425463            0.423837           0.424948
1           0.424926            0.423328           0.424404
2           0.420024            0.418354           0.419420
3           0.382945            0.382265           0.383620
4           0.378243            0.377869           0.378071
5           0.390600            0.389872           0.390085
6           0.388755            0.387717           0.390732
7           0.427432            0.428987           0.428686

   split1_train_score  split2_test_score  split2_train_score  std_fit_time  \
0            0.425540           0.422361            0.423963      0.006915
1            0.425144           0.422031            0.423409      0.003070
2            0.420394           0.417349            0.418145      0.001548
3            0.383889           0.383477            0.383885      0.000487
4            0.378078           0.377645            0.378914      0.000623
5            0.390525           0.389914            0.390721      0.004564
6            0.390040           0.389411            0.391175      0.002322
7            0.428910           0.429907            0.428087      0.002625

   std_score_time  std_test_score  std_train_score
0        0.001304        0.001357         0.000775
1        0.000521        0.001260         0.000838
2        0.000478        0.001145         0.001015
3        0.000713        0.000291         0.000765
4        0.000134        0.000251         0.000452
5        0.001508        0.000292         0.000363
6        0.001675        0.000822         0.001439
7        0.000883        0.001011         0.000408
```

In [62]:
```
#For Nearest Controid

kf2 = KFold(random_state=69, shuffle=True)
param_grid_nc = {'shrink_threshold': [0.01, 0.1, 1, 10, 15, 20, 30, 100]}
print(param_grid_nc)

grid_nc_k2 = GridSearchCV(nc, param_grid_nc, cv=3,return_train_score=True)
grid_nc_k2.fit(X_train_scaled, y_train)
pd.DataFrame(grid_nc_k2.cv_results_)
```

```
{'shrink_threshold': [0.01, 0.1, 1, 10, 15, 20, 30, 100]}
```

Out[62]:
```
   mean_fit_time  mean_score_time  mean_test_score  mean_train_score  \
0       0.126063         0.035162         0.424257          0.424447
1       0.124750         0.035919         0.423787          0.423960
2       0.141729         0.038850         0.418931          0.418964
```

```
3       0.132582         0.036957         0.383347          0.383346
4       0.130839         0.036198         0.377986          0.378287
5       0.123658         0.035235         0.390200          0.390373
6       0.123789         0.035200         0.389633          0.389644
7       0.126390         0.035432         0.428675          0.428661

  param_shrink_threshold                     params  rank_test_score  \
0                   0.01  {'shrink_threshold': 0.01}                2
1                    0.1   {'shrink_threshold': 0.1}                3
2                      1     {'shrink_threshold': 1}                4
3                     10    {'shrink_threshold': 10}                7
4                     15    {'shrink_threshold': 15}                8
5                     20    {'shrink_threshold': 20}                5
6                     30    {'shrink_threshold': 30}                6
7                    100   {'shrink_threshold': 100}                1

   split0_test_score  split0_train_score  split1_test_score  \
0           0.425463            0.423837           0.424948
1           0.424926            0.423328           0.424404
2           0.420024            0.418354           0.419420
3           0.382945            0.382265           0.383620
4           0.378243            0.377869           0.378071
5           0.390600            0.389872           0.390085
6           0.388755            0.387717           0.390732
7           0.427432            0.428987           0.428686

   split1_train_score  split2_test_score  split2_train_score  std_fit_time  \
0            0.425540           0.422361            0.423963      0.005148
1            0.425144           0.422031            0.423409      0.001594
2            0.420394           0.417349            0.418145      0.004519
3            0.383889           0.383477            0.383885      0.006924
4            0.378078           0.377645            0.378914      0.009987
5            0.390525           0.389914            0.390721      0.000733
6            0.390040           0.389411            0.391175      0.000284
7            0.428910           0.429907            0.428087      0.000224

   std_score_time  std_test_score  std_train_score
0        0.000415        0.001357         0.000775
1        0.000850        0.001260         0.000838
2        0.002859        0.001145         0.001015
3        0.002475        0.000291         0.000765
4        0.001278        0.000251         0.000452
5        0.000033        0.000292         0.000363
6        0.000206        0.000822         0.001439
7        0.000835        0.001011         0.000408
```

In [63]: *#For Nearest Controid*

```
kf3 = KFold(random_state=42, shuffle=True)
param_grid_nc = {'shrink_threshold': [0.01, 0.1, 1, 10, 15, 20, 30, 100]}
print(param_grid_nc)

grid_nc_k3 = GridSearchCV(nc, param_grid_nc, cv=3,return_train_score=True)
grid_nc_k3.fit(X_train_scaled2, y_train)
pd.DataFrame(grid_nc_k3.cv_results_)
```

```
{'shrink_threshold': [0.01, 0.1, 1, 10, 15, 20, 30, 100]}
```

Out[63]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	\
0	0.286566	0.061198	0.038354	0.040608	
1	0.278847	0.061549	0.038315	0.040537	
2	0.278288	0.058835	0.029592	0.030703	
3	0.282914	0.057302	0.364500	0.364500	
4	0.277939	0.057904	0.364500	0.364500	
5	0.280772	0.057152	0.364500	0.364500	
6	0.278669	0.057892	0.364500	0.364500	
7	0.278657	0.058116	0.364500	0.364500	

	param_shrink_threshold	params	rank_test_score	\
0	0.01	{'shrink_threshold': 0.01}	6	
1	0.1	{'shrink_threshold': 0.1}	7	
2	1	{'shrink_threshold': 1}	8	
3	10	{'shrink_threshold': 10}	1	
4	15	{'shrink_threshold': 15}	1	
5	20	{'shrink_threshold': 20}	1	
6	30	{'shrink_threshold': 30}	1	
7	100	{'shrink_threshold': 100}	1	

	split0_test_score	split0_train_score	split1_test_score	\
0	0.035207	0.036702	0.032991	
1	0.035964	0.037032	0.034354	
2	0.036474	0.037604	0.026712	
3	0.364494	0.364502	0.364502	
4	0.364494	0.364502	0.364502	
5	0.364494	0.364502	0.364502	
6	0.364494	0.364502	0.364502	
7	0.364494	0.364502	0.364502	

	split1_train_score	split2_test_score	split2_train_score	std_fit_time	\
0	0.035287	0.046864	0.049837	0.011733	
1	0.036949	0.044627	0.047630	0.002245	
2	0.028268	0.025590	0.026237	0.001642	
3	0.364498	0.364503	0.364498	0.000893	
4	0.364498	0.364503	0.364498	0.000381	
5	0.364498	0.364503	0.364498	0.003470	

```
6            0.364498           0.364503           0.364498      0.002700
7            0.364498           0.364503           0.364498      0.002167

   std_score_time  std_test_score  std_train_score
0        0.000637        0.006085         0.006551
1        0.000727        0.004511         0.005016
2        0.000497        0.004888         0.004950
3        0.000151        0.000004         0.000002
4        0.000308        0.000004         0.000002
5        0.000280        0.000004         0.000002
6        0.000653        0.000004         0.000002
7        0.000863        0.000004         0.000002
```

20 2.4 Inference

The parameters are found to sometimes change in the case of SVC they change only in the last case. For nearest centroid they vary from 1 to 100. And for Logistic Regression it is 100 in 2 cases and then drops to 0.01 for the last case. Thus, changing the random-state of the training-test split or the random seed of the shuffling has an unpredictable but marginal effect.

In [ ]:

In [ ]:

21 2.5

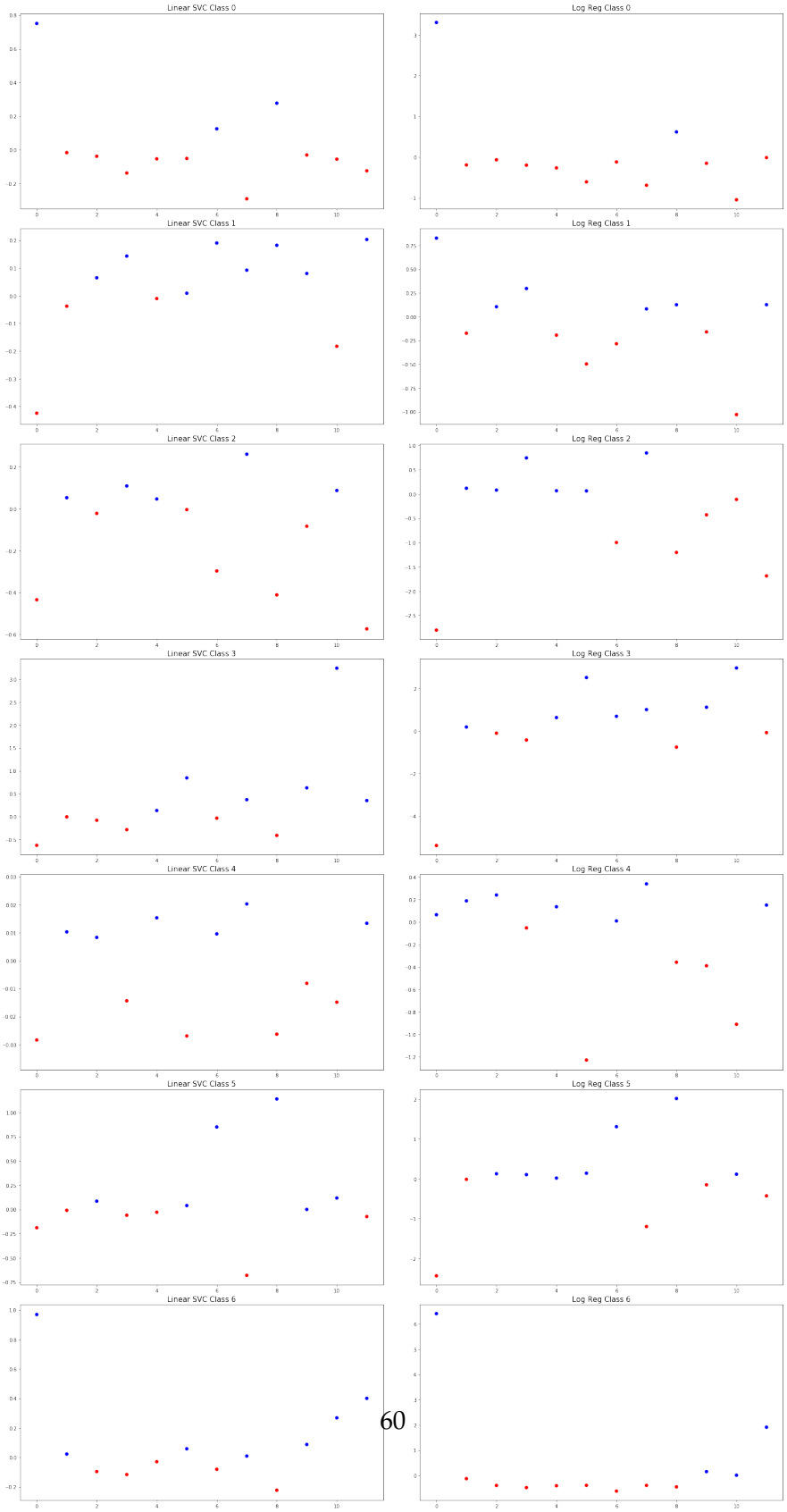
In [ ]:

In [76]:
```
grid_log_k.best_estimator_.coef_.shape
```

Out[76]: (7, 12)

In [77]:
```
#Visualizing coefficients for Support Vector Machines and Logistic Regression
fig25, axes25 = plt.subplots(7,2)
fig25.set_size_inches(28,56)
fig25.subplots_adjust(hspace=0.1, wspace=0.1)

for i in range(7):
    axes25[i,0].scatter(range(X_train_scaled.shape[1]),grid_svc_k2.best_estimator_.coef
    axes25[i,0].set_title("Linear SVC Class %s" %str(i),fontsize=15)
    axes25[i,1].scatter(range(X_train_scaled.shape[1]),grid_log_k.best_estimator_.coef_
    axes25[i,1].set_title("Log Reg Class %s" %str(i),fontsize=15)

fig25.suptitle('Coefficient Visualization for SVC and LR', fontsize=25)
fig25.savefig('task25.png')
```

Coefficient Visualization for SVC and LR



```
In [ ]:
```