In [46]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
```

In [47]:
```
# Loading the California Housing Dataset
housing = datasets.fetch_california_housing()
```

In [48]:
```
cali_housing = housing.data
```

In [49]:
```
features = housing.feature_names
```

In [50]:
```
features
```

Out[50]:
```
['MedInc',
 'HouseAge',
 'AveRooms',
 'AveBedrms',
 'Population',
 'AveOccup',
 'Latitude',
 'Longitude']
```

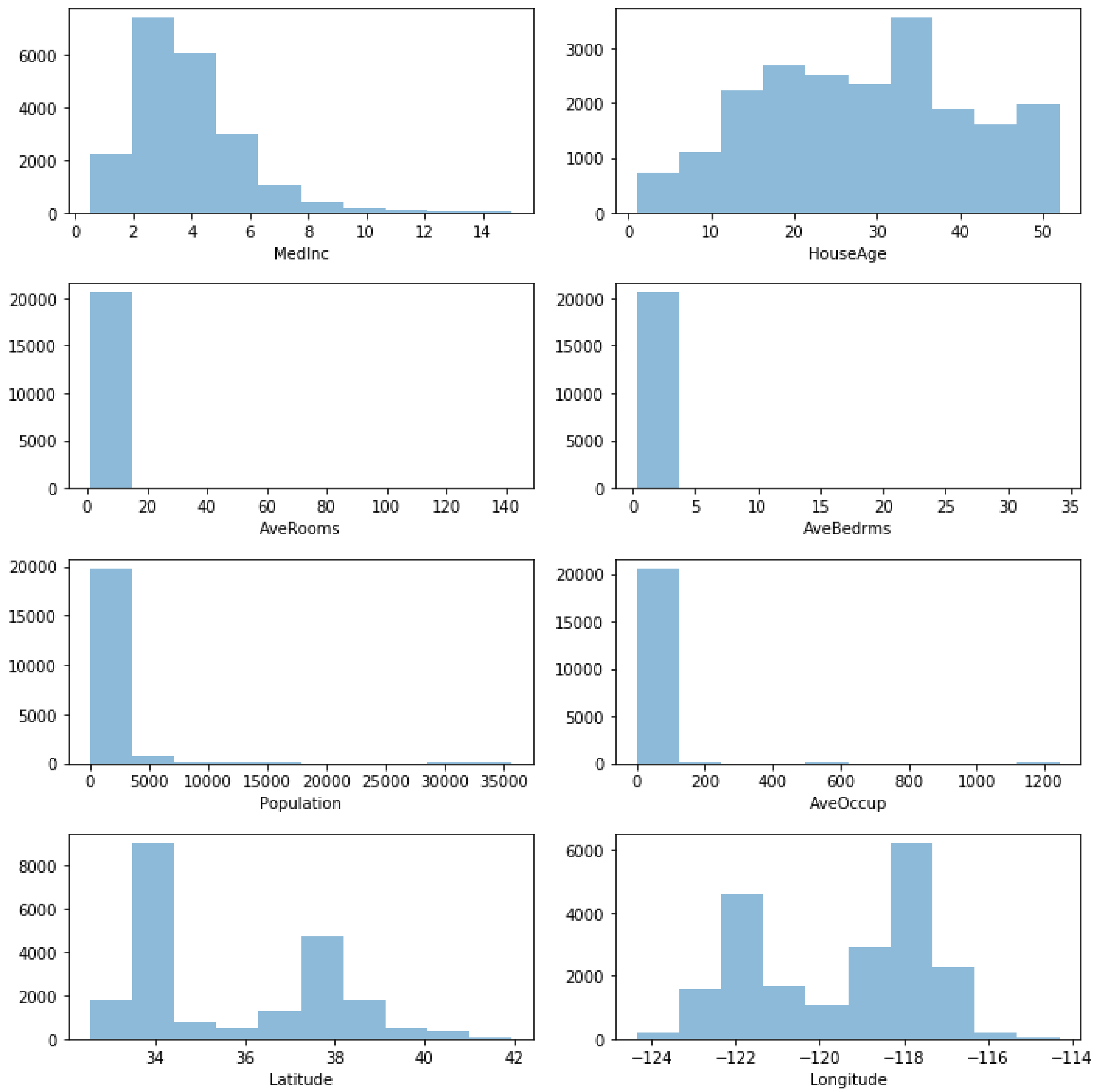
In [51]:
```
cali_housing.shape
```

Out[51]:
```
(20640, 8)
```

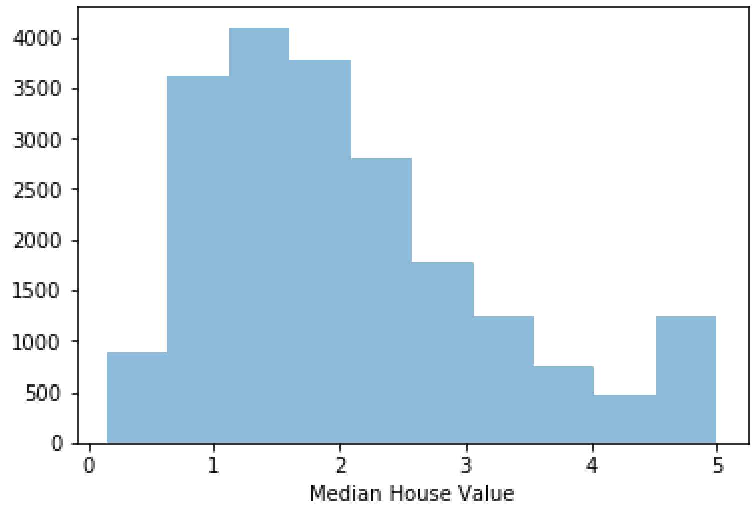
In [52]:
```
median_house_value = housing.target
```

## 1.1

In [53]:
```
# Visualizing univariate distribution of each feature

count = 0
fig = plt.figure(figsize=(10,10))
for i in range(1,9):
    fig.add_subplot(4,2,i)
    plt.hist(cali_housing[:,count], alpha=0.5)
    plt.xlabel(features[count])
    #plt.ylabel("Median House Value")
    plt.tight_layout()
    count += 1
plt.show()
```



In [56]:
```
# Visualizing univariate distribution of target

plt.hist(median_house_value, alpha=0.5)
plt.xlabel("Median House Value")
plt.show()
```




**Observations:**

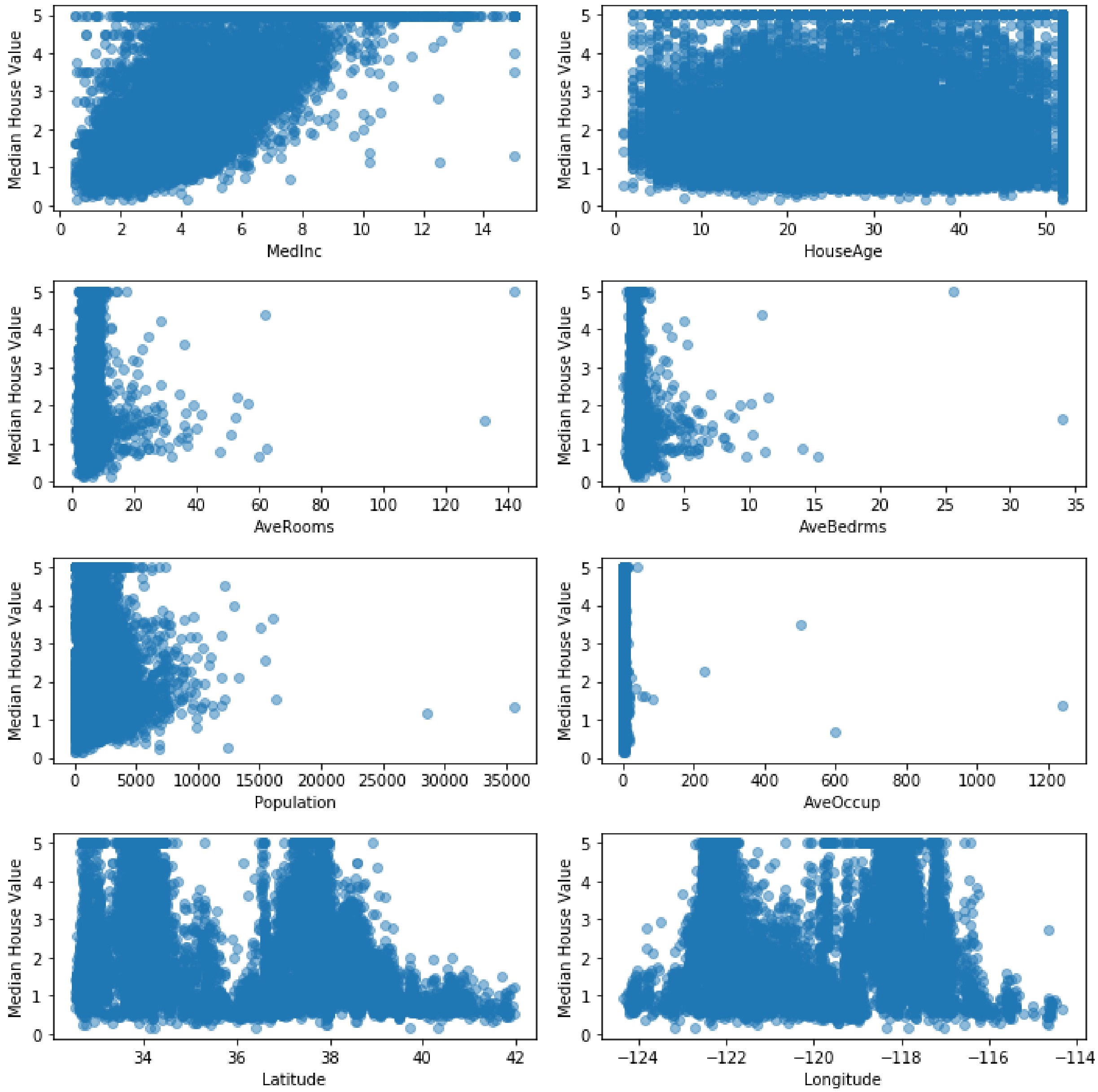
1) There are outliers in Average Occupancy of houses

2) Latitude and Longitude are bimodal which suggests that there are a couple of densely populated areas in California

## 1.2

In [57]:
```
# Visualizing the dependency of target on each feature

count = 0
fig = plt.figure(figsize=(10,10))
for i in range(1,9):
    fig.add_subplot(4,2,i)
    plt.scatter(cali_housing[:,count], median_house_value, alpha=0.5)
    plt.xlabel(features[count])
    plt.ylabel("Median House Value")
    plt.tight_layout()
    count += 1
```

In [58]:
```
plt.show()
```




## 1.3

In [59]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
```

In [60]:
```
X_train, X_test, y_train, y_test = train_test_split(
    cali_housing, median_house_value, random_state=0)
```

In [61]:
```
models = [LinearRegression(),Ridge(),Lasso(),ElasticNet()]
```

In [62]:
```
models_names = ['LinearRegression','Ridge','Lasso','ElasticNet']
```

In [64]:
```
for i in range(len(models)):
    score_train = np.mean(cross_val_score(models[i], X_train, y_train, cv=1(
    print("Mean training score of " + models_names[i] + " : " + str(score_tr
```

```
Mean training score of LinearRegression : 0.606136752898
Mean training score of Ridge : 0.606151468466
Mean training score of Lasso : 0.291549120926
Mean training score of ElasticNet : 0.429496808434
```

In [65]:
```
# Scaling the data with Standard Scaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [66]:
```
for i in range(len(models)):
    score_train_scaled = np.mean(cross_val_score(models[i], X_train_scaled,
    print("Mean training score of " + models_names[i] + " : " + str(score_tr
```

```
Mean training score of LinearRegression : 0.606136752898
Mean training score of Ridge : 0.606143117729
Mean training score of Lasso : -0.000601150113709
Mean training score of ElasticNet : 0.209910128135
```

Observation: scaling the data with StandardScaler doesn't help

# 1.4

In [67]:
```
param = [{'alpha': np.logspace(-3, 3, 13)},{'alpha': np.logspace(-3, 0, 13)]
              'l1_ratio': [0.01, .1, .5, .9, .98, 1]}]
print(param)
```

```
[{'alpha': array([  1.00000000e-03,   3.16227766e-03,   1.00000000e-02,
         3.16227766e-02,   1.00000000e-01,   3.16227766e-01,
         1.00000000e+00,   3.16227766e+00,   1.00000000e+01,
         3.16227766e+01,   1.00000000e+02,   3.16227766e+02,
         1.00000000e+03])}, {'alpha': array([ 0.001     ,  0.00177828,
0.00316228,  0.00562341,  0.01      ,
        0.01778279,  0.03162278,  0.05623413,  0.1       ,  0.17782794,
        0.31622777,  0.56234133,  1.        ])}, {'l1_ratio': [0.01, 0.1,
0.5, 0.9, 0.98, 1], 'alpha': array([ 0.0001    ,  0.00021544,  0.0004641
6,  0.001     ,  0.00215443,
        0.00464159,  0.01      ,  0.02154435,  0.04641589,  0.1
])}]
```

In [113]:
```
# Tuning the parameters of the models using GridSearchCV
list_of_result_dataframes = list()
for i in range(len(models)-1):
    grid = GridSearchCV(models[i+1], param[i], cv=10)
    grid.fit(X_train, y_train)
    list_of_result_dataframes.append(pd.DataFrame(grid.cv_results_))
```

In [114]:
```
# Results for Grid Search on Ridge
list_of_result_dataframes[0]
```

Out[114]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_alpha | pa |
|---|---|---|---|---|---|---|
| **0** | 0.001910 | 0.000502 | 0.606137 | 0.611152 | 0.001 | {'alpha': ( |
| **1** | 0.001699 | 0.000280 | 0.606137 | 0.611152 | 0.00316228 | {'a 0.0031622776 |
| **2** | 0.002480 | 0.000433 | 0.606137 | 0.611152 | 0.01 | {'alpha': |
| **3** | 0.002957 | 0.000491 | 0.606137 | 0.611152 | 0.0316228 | {'a 0.031622776 |
| **4** | 0.001602 | 0.000277 | 0.606138 | 0.611152 | 0.1 | {'alpha |
| **5** | 0.001261 | 0.000210 | 0.606141 | 0.611152 | 0.316228 | {'a 0.31622776 |
| **6** | 0.001223 | 0.000207 | 0.606151 | 0.611151 | 1 | {'alpha |
| **7** | 0.001208 | 0.000204 | 0.606183 | 0.611151 | 3.16228 | {'a 3.1622776 |
| **8** | 0.001204 | 0.000203 | 0.606274 | 0.611146 | 10 | {'alpha': |
| **9** | 0.001598 | 0.000283 | 0.606504 | 0.611104 | 31.6228 | {'a 31.622776 |
| **10** | 0.001210 | 0.000217 | 0.606807 | 0.610777 | 100 | {'alpha': |
| **11** | 0.001296 | 0.000227 | 0.605999 | 0.609052 | 316.228 | {'a 316.22776 |
| **12** | 0.001229 | 0.000222 | 0.600725 | 0.603141 | 1000 | {'alpha': 1( |

13 rows × 31 columns

In [115]:
```
# Results for Grid Search on Lasso
list_of_result_dataframes[1]
```

Out[115]:

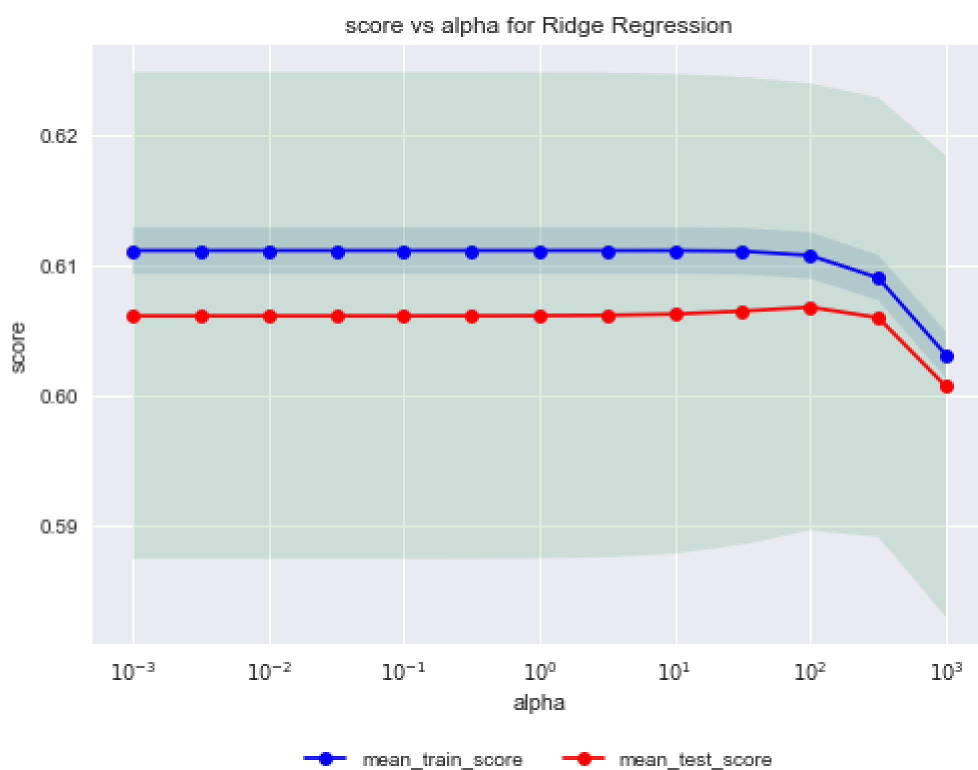
| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_alpha | pa |
|---|---|---|---|---|---|---|
| **0** | 0.009053 | 0.000232 | 0.606374 | 0.611114 | 0.001 | {'alpha': ( |
| **1** | 0.008635 | 0.000231 | 0.606498 | 0.611033 | 0.00177828 | {'a 0.0017782794 |
| **2** | 0.009546 | 0.000295 | 0.606589 | 0.610777 | 0.00316228 | {'a 0.0031622776 |
| **3** | 0.009629 | 0.000356 | 0.606337 | 0.609967 | 0.00562341 | {'a 0.005623413 |
| **4** | 0.008558 | 0.000255 | 0.604583 | 0.607406 | 0.01 | {'alpha': |
| **5** | 0.006588 | 0.000794 | 0.600121 | 0.602024 | 0.0177828 | {'a 0.017782794 |
| **6** | 0.005416 | 0.000228 | 0.594291 | 0.595973 | 0.0316228 | {'a 0.031622776 |
| **7** | 0.003864 | 0.000238 | 0.583396 | 0.584626 | 0.0562341 | {'a 0.05623413 |
| **8** | 0.003424 | 0.000226 | 0.549299 | 0.550440 | 0.1 | {'alpha |
| **9** | 0.001455 | 0.000220 | 0.513368 | 0.514470 | 0.177828 | {'a 0.17782794 |
| **10** | 0.001450 | 0.000262 | 0.494713 | 0.495351 | 0.316228 | {'a 0.31622776 |
| **11** | 0.001385 | 0.000223 | 0.445874 | 0.446528 | 0.562341 | {'a 0.5623413 |
| **12** | 0.001390 | 0.000239 | 0.291549 | 0.292297 | 1 | {'alpha |

13 rows × 31 columns

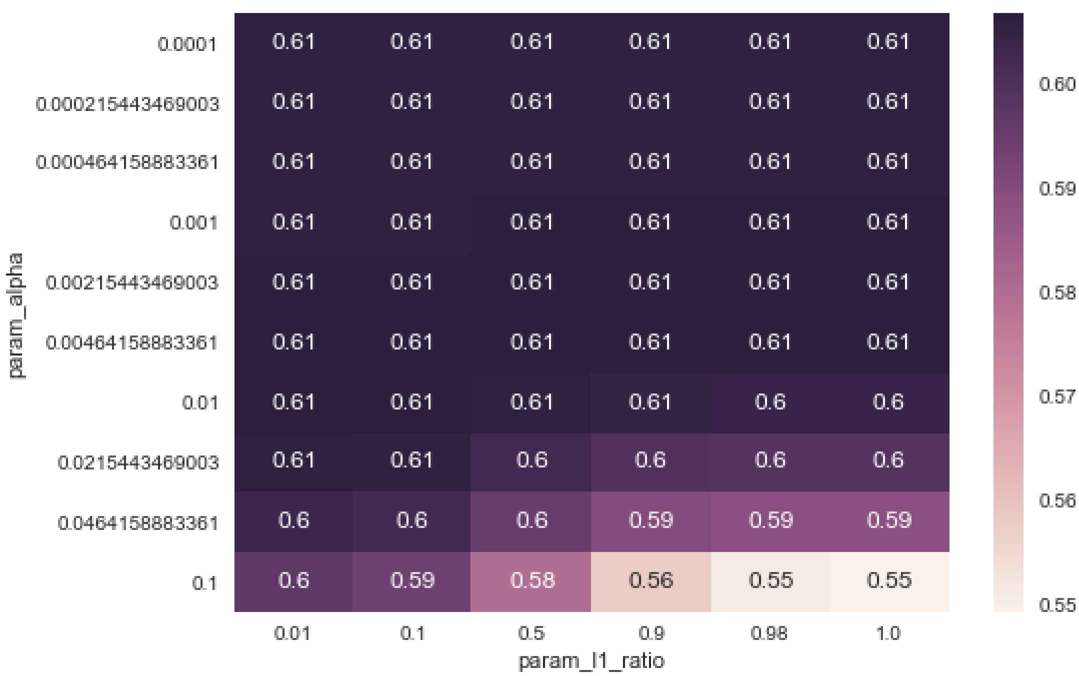
In [116]:
```
# Results for Grid Search on Elastic Net
list_of_result_dataframes[2]
```

| | | | | | | |
|---|---|---|---|---|---|---|
| **41** | 0.007717 | 0.000231 | 0.604583 | 0.607406 | 0.01 | 1 |
| **42** | 0.008841 | 0.000252 | 0.606061 | 0.609142 | 0.0215443 | 0.01 |
| **43** | 0.007478 | 0.000231 | 0.605683 | 0.608647 | 0.0215443 | 0.1 |
| **44** | 0.006884 | 0.000235 | 0.602653 | 0.605055 | 0.0215443 | 0.5 |
| **45** | 0.005949 | 0.000234 | 0.599351 | 0.601212 | 0.0215443 | 0.9 |
| **46** | 0.005848 | 0.000234 | 0.598845 | 0.600683 | 0.0215443 | 0.98 |

In [134]:
```
save_images = ['Ridge.png','Lasso.png']
for i in range(0,2):
    plt.figure()
    ax1 = plt.gca()
    line1, = ax1.plot(list_of_result_dataframes[i]['param_alpha'], list_of_r
    line2, = ax1.plot(list_of_result_dataframes[i]['param_alpha'], list_of_r

    plt.fill_between(list_of_result_dataframes[i].param_alpha.astype(np.floa
                     list_of_result_dataframes[i]['mean_train_score'] + list_of_
                     list_of_result_dataframes[i]['mean_train_score'] - list_of_
    plt.fill_between(list_of_result_dataframes[i].param_alpha.astype(np.floa
                     list_of_result_dataframes[i]['mean_test_score'] + list_of_r
                     list_of_result_dataframes[i]['mean_test_score'] - list_of_r


    plt.legend([line1, line2], ["mean_train_score", "mean_test_score"], loc=
    ax1.set_ylabel("score")
    ax1.set_xlabel("alpha")
    ax1.set_xscale("log")
    if i==0:
        plt.title("score vs alpha for Ridge Regression")
    else:
        plt.title("score vs alpha for Lasso Regression")

    plt.savefig(save_images[i], bbox_inches = 'tight')
    plt.show()
```




In [135]:
```
res = pd.pivot_table(list_of_result_dataframes[2],
    values='mean_test_score', index='param_alpha', columns='param_l1_ratio')

import seaborn as sns; sns.set()
sns.heatmap(res, annot = True)
plt.show()
```



In [119]:
```
res
```

Out[119]:

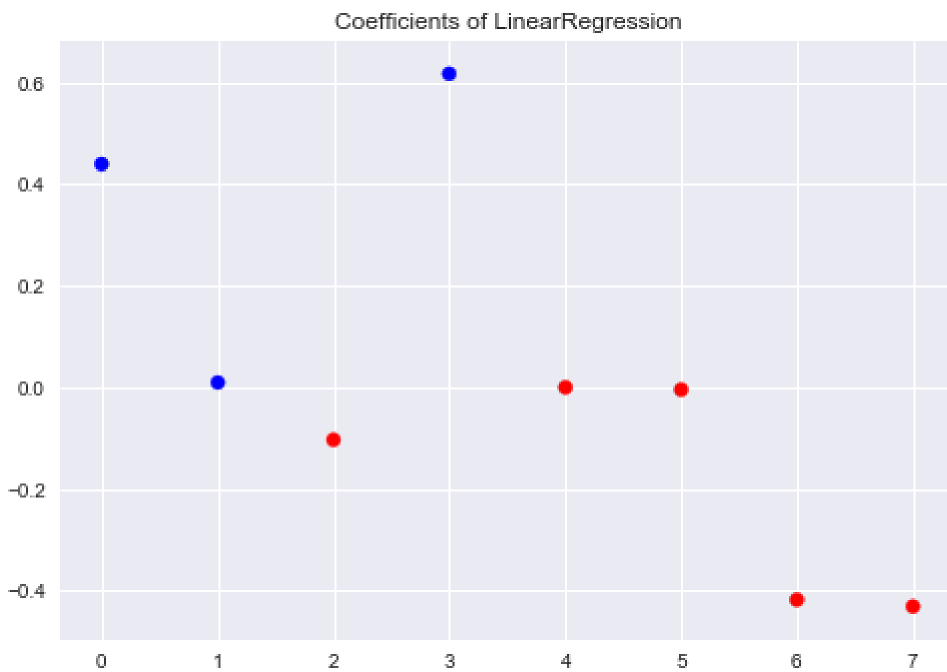
| param_l1_ratio | 0.01 | 0.1 | 0.5 | 0.9 | 0.98 | 1.0 |
|---|---|---|---|---|---|---|
| **param_alpha** | | | | | | |
| **0.000100** | 0.606157 | 0.606158 | 0.606161 | 0.606164 | 0.606164 | 0.606164 |
| **0.000215** | 0.606180 | 0.606182 | 0.606188 | 0.606194 | 0.606195 | 0.606195 |
| **0.000464** | 0.606228 | 0.606231 | 0.606243 | 0.606255 | 0.606257 | 0.606258 |
| **0.001000** | 0.606323 | 0.606328 | 0.606350 | 0.606369 | 0.606373 | 0.606374 |
| **0.002154** | 0.606490 | 0.606498 | 0.606524 | 0.606538 | 0.606539 | 0.606539 |
| **0.004642** | 0.606716 | 0.606715 | 0.606671 | 0.606547 | 0.606510 | 0.606501 |
| **0.010000** | 0.606787 | 0.606716 | 0.606155 | 0.605007 | 0.604673 | 0.604583 |
| **0.021544** | 0.606061 | 0.605683 | 0.602653 | 0.599351 | 0.598845 | 0.598710 |
| **0.046416** | 0.603436 | 0.602144 | 0.595231 | 0.589820 | 0.588683 | 0.588383 |
| **0.100000** | 0.597221 | 0.593782 | 0.580172 | 0.557644 | 0.551079 | 0.549299 |

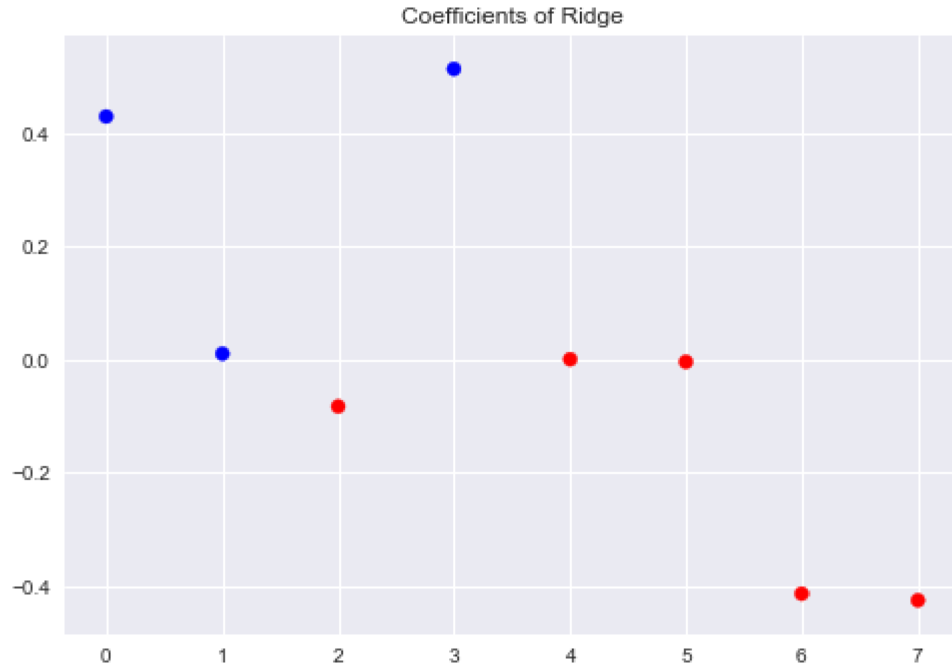
Observations:

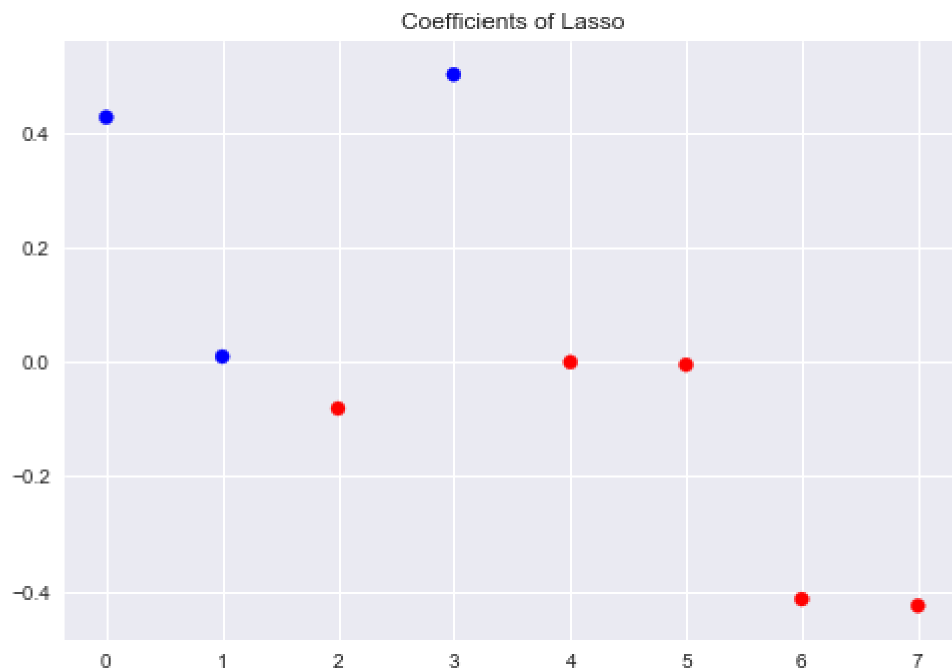
There is significant improvement in results for Lasso and Elastic Net whereas for Ridge regression the results are almost the same.

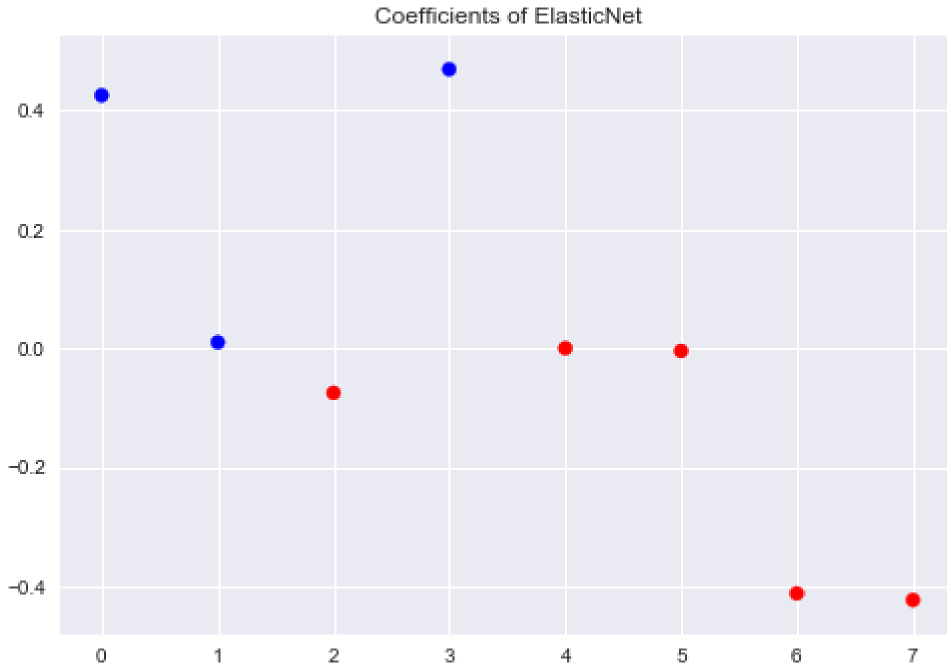
# 1.5

In [133]:
```
save_images_names = ['LinearRegression_Scatter','Ridge_Scatter','Lasso_Scatt
for i in range(len(models)):
    if i==0:
        grid = LinearRegression().fit(X_train, y_train, sample_weight=None)
        plt.scatter(range(8), grid.coef_,
                c=np.sign(grid.coef_), cmap="bwr_r")
        plt.title("Coefficients of " + str(models_names[i]))
        plt.savefig(save_images_names[i], bbox_inches = 'tight')
        plt.show()
    else:
        grid = GridSearchCV(models[i], param[i-1], cv=10)
        grid.fit(X_train, y_train)

        plt.scatter(range(8), grid.best_estimator_.coef_,
                c=np.sign(grid.best_estimator_.coef_), cmap="bwr_r")
        plt.title("Coefficients of " + str(models_names[i]))
        plt.savefig(save_images_names[i], bbox_inches = 'tight')
        plt.show()
```




Coefficients of Ridge
0.4
0.2
0.0
−0.2
−0.4
0
1
2
3
4
5
6
7


Coefficients of Lasso
0.4
0.2
0.0
−0.2
−0.4
0
1
2
3
4
5
6
7




Observation: All the models agree on which features are important

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
```

In [2]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
```

## 2.1

In [3]:
```
forest = datasets.fetch_covtype()
```

In [4]:
```
forest_data = forest.data
#forest_features = forest.feature_names
cover_type = forest.target
```

In [5]:
```
forest_data.shape
```

Out[5]:
```
(581012, 54)
```

In [6]:
```
forest_features = ['elevation','aspect','slope','hdist_to_hydrology','vdist_
              'hill_shade_9','hill_shade_12','hill_shade_3','hdist_to_fire']
```

In [7]:
```
for i in range(1,5):
    forest_features.append('wilderness' + str(i))
```

In [8]:
```
forest_features
```

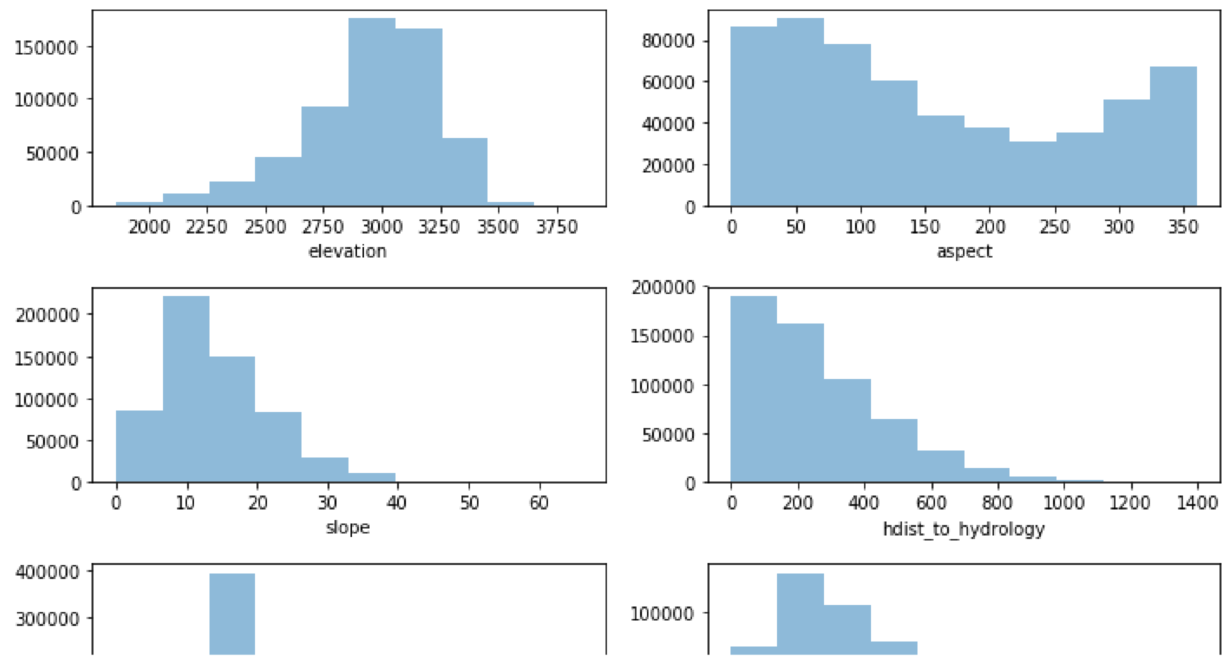
Out[8]:
```
['elevation',
 'aspect',
 'slope',
 'hdist_to_hydrology',
 'vdist_to_hydrology',
 'hdist_to_road',
 'hill_shade_9',
 'hill_shade_12',
 'hill_shade_3',
 'hdist_to_fire',
 'wilderness1',
 'wilderness2',
 'wilderness3',
 'wilderness4']
```

In [9]:
```
for i in range(1,41):
    forest_features.append('soil_type' + str(i))
```

In [10]:
```
len(forest_features)
```

Out[10]: 54

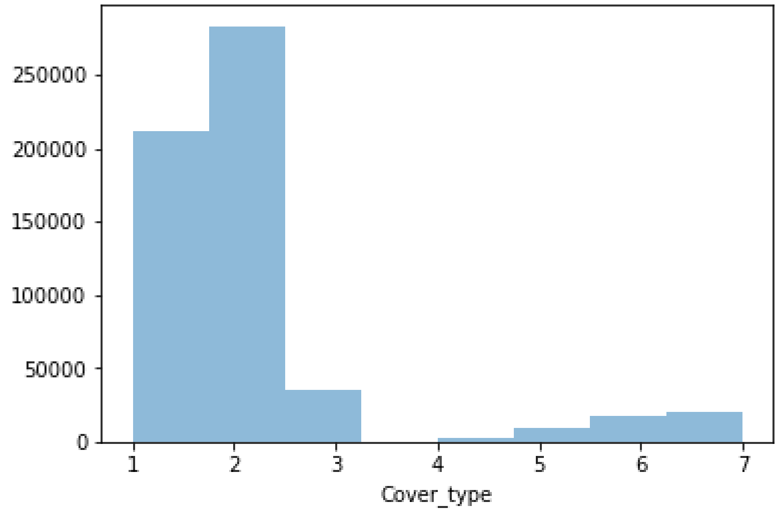
In [11]:
```
# Visualizing univariate distribution of each feature
count = 0
fig = plt.figure(figsize=(10,60))
for i in range(1,55):
    fig.add_subplot(27,2,i)
    plt.hist(forest_data[:,count], alpha=0.5)
    plt.xlabel(forest_features[count])
    #plt.ylabel("Median House Value")
    plt.tight_layout()
    count += 1
plt.show()
```




In [12]:
```
count
```

Out[12]: 54

In [13]:
```
# Visualizing univariate distribution of target
plt.hist(cover_type, alpha=0.5, bins=8)
plt.xlabel('Cover_type')
plt.show()
```




## 2.2

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(
    forest_data, cover_type, random_state=0)
```

In [15]:
```
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.svm import LinearSVC
from sklearn.neighbors.nearest_centroid import NearestCentroid
```

In [44]:
```
#logreg = LogisticRegressionCV(multi_class='multinomial', dual=False).fit(X_
```

In [45]:
```
# Accuracy over train data
#print(logreg.score(X_train,y_train))
```

In [16]:
```
cross_validation_scores_logreg = cross_val_score(LogisticRegressionCV(multi_
```

In [22]:
```
mean_cross_validation_score_logreg = np.mean(cross_validation_scores_logreg)
print("mean cross validation score of logreg is " + str(mean_cross_validatic
```

```
mean cross validation score of logreg is 0.672098597965
```

In [23]:
```
cross_validation_scores_linearsvm = cross_val_score(LinearSVC(dual=False,tol
```

In [25]:
```
mean_cross_validation_score_linearsvm = np.mean(cross_validation_scores_line
print("mean cross validation score of linearsvm is " + str(mean_cross_valida
```

```
mean cross validation score of linearsvm is 0.680220061782
```

In [27]:
```
cross_validation_scores_nearest_centroid = cross_val_score(NearestCentroid()
```

In [28]:
```
mean_cross_validation_score_nearest_centroid = np.mean(cross_validation_scor
print("mean cross validation score of nearest centroid is " + str(mean_cross
```

```
mean cross validation score of nearest centroid is 0.193586356619
```

In [29]:
```
# Scaling the data with StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [30]:
```
cross_validation_scores_logreg = cross_val_score(LogisticRegressionCV(multi_
mean_cross_validation_score_logreg = np.mean(cross_validation_scores_logreg)
print("mean cross validation score of logreg is " + str(mean_cross_validatic
```

```
mean cross validation score of logreg is 0.72434305351
```

In [31]:
```
cross_validation_scores_linearsvm = cross_val_score(LinearSVC(dual=False,tol
mean_cross_validation_score_linearsvm = np.mean(cross_validation_scores_line
print("mean cross validation score of linearsvm is " + str(mean_cross_valida
```

```
mean cross validation score of linearsvm is 0.712719657497
```

In [32]:
```
cross_validation_scores_nearest_centroid = cross_val_score(NearestCentroid()
mean_cross_validation_score_nearest_centroid = np.mean(cross_validation_scor
print("mean cross validation score of nearest centroid is " + str(mean_cross
```

```
mean cross validation score of nearest centroid is 0.549861256828
```

Observations:

1) Logistic Regression and Linear Support Vector Machines are giving significantly higher cross validation scores than Nearest Centroid. (Best Score: linear support vector machine: 0.68)

2) Scaling the data using StandardScaler gives better cross validation score than without scaling the data in case of all three models and even in this case, logistic regression and linear support vector machines are performing better than nearest centroid. (Best Score: logistic regression: 0.724)

As the models are performing better when X_train is scaled, we use the standard scaled X_data from now.

# 2.3

In [34]:
```
params = [{'C': [0.01, 1, 10, 100, 1000]}]
grid_logreg = GridSearchCV(LogisticRegression(multi_class='multinomial', dua
grid_logreg.fit(X_train_scaled, y_train)
pd.DataFrame(grid_logreg.cv_results_)
```

Out[34]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_C | params | rank_tes |
|---|---|---|---|---|---|---|---|
| **0** | 14.930232 | 0.029203 | 0.722741 | 0.722941 | 0.01 | {'C': 0.01} | |
| **1** | 14.429111 | 0.027336 | 0.723873 | 0.724003 | 1 | {'C': 1} | |
| **2** | 14.432691 | 0.029546 | 0.724065 | 0.724046 | 10 | {'C': 10} | |
| **3** | 14.886415 | 0.027300 | 0.724081 | 0.723992 | 100 | {'C': 100} | |
| **4** | 15.148282 | 0.027407 | 0.723958 | 0.723974 | 1000 | {'C': 1000} | |

In [35]:
```
params = [{'C': [0.01, 1, 10, 100, 1000]}]
grid_linearsvc = GridSearchCV(LinearSVC(dual=False, tol=0.001), params, cv=3
grid_linearsvc.fit(X_train_scaled, y_train)
pd.DataFrame(grid_linearsvc.cv_results_)
```

Out[35]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_C | params | rank_tes |
|---|---|---|---|---|---|---|---|
| **0** | 37.866422 | 0.027945 | 0.712424 | 0.712354 | 0.01 | {'C': 0.01} | |
| **1** | 40.058371 | 0.028552 | 0.712720 | 0.712685 | 1 | {'C': 1} | |
| **2** | 39.194059 | 0.028445 | 0.712738 | 0.712701 | 10 | {'C': 10} | |
| **3** | 39.344713 | 0.028619 | 0.712736 | 0.712707 | 100 | {'C': 100} | |
| **4** | 38.208435 | 0.027860 | 0.712736 | 0.712699 | 1000 | {'C': 1000} | |

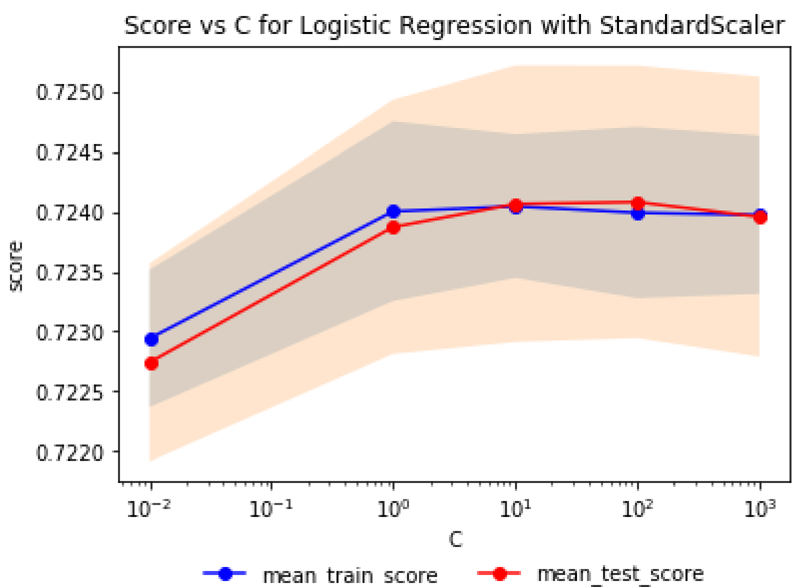
In [50]:
```
params = [{'shrink_threshold': [0,0.5,1,10, 15, 20, 50, 100]}]
grid_nearest_centroid = GridSearchCV(NearestCentroid(), params, cv=3)
grid_nearest_centroid.fit(X_train_scaled,y_train)
pd.DataFrame(grid_nearest_centroid.cv_results_)
```

Out[50]:

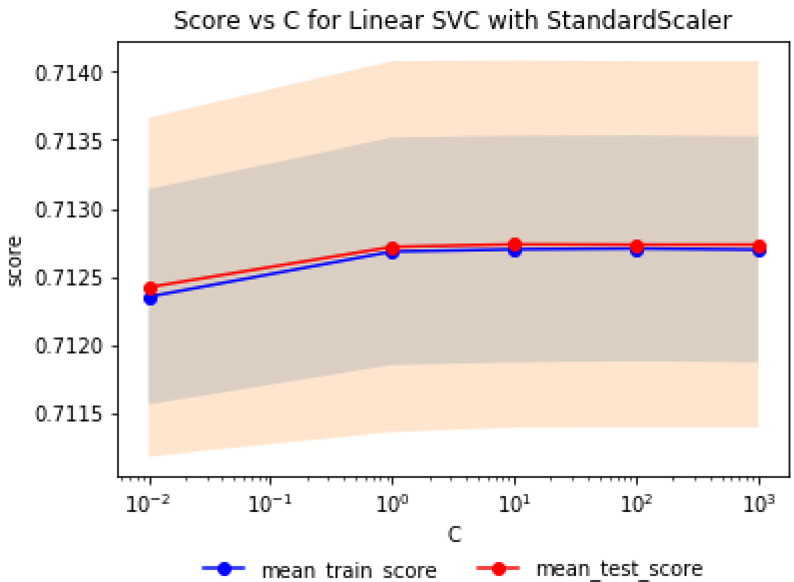
| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_shrink_threshold | |
|---|---|---|---|---|---|---|
| **0** | 0.187555 | 0.036439 | 0.549861 | 0.549946 | 0 | {'sh |
| **1** | 0.344647 | 0.036744 | 0.548528 | 0.548660 | 0.5 | {'sh |
| **2** | 0.341735 | 0.035756 | 0.547204 | 0.547211 | 1 | {'sh |
| **3** | 0.342701 | 0.035268 | 0.542938 | 0.543083 | 10 | {'sh |
| **4** | 0.339407 | 0.034746 | 0.547222 | 0.547348 | 15 | {'sh |
| **5** | 0.338873 | 0.034924 | 0.552881 | 0.552903 | 20 | {'sh |
| **6** | 0.336124 | 0.034707 | 0.633974 | 0.633935 | 50 | {'sh |
| **7** | 0.335678 | 0.035027 | 0.514564 | 0.514576 | 100 | {'sh |

Observation: There is a significant improvement in case of Nearest Centroid whereas the others are pretty much the same.

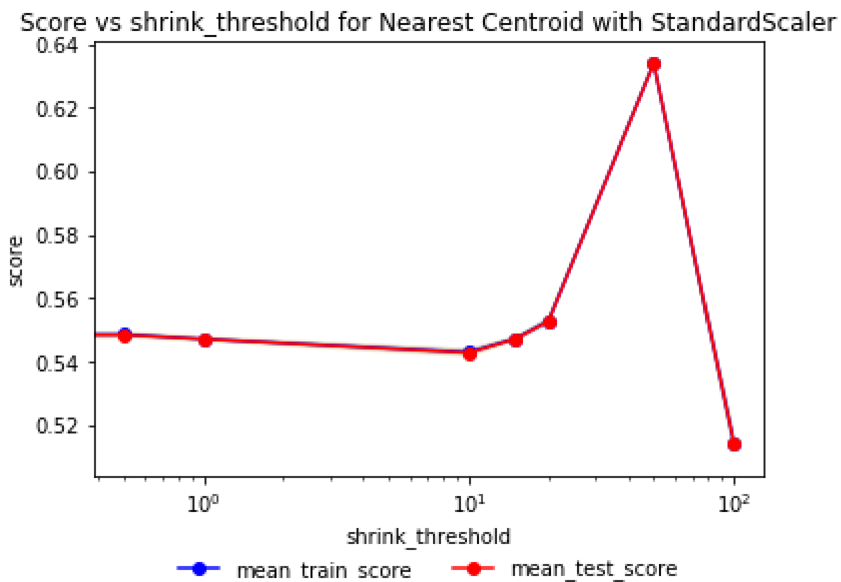
In [42]:
```
plt.figure()
ax1 = plt.gca()
logreg_dataframe = pd.DataFrame(grid_logreg.cv_results_)
line1, = ax1.plot(logreg_dataframe['param_C'], logreg_dataframe['mean_train_
line2, = ax1.plot(logreg_dataframe['param_C'], logreg_dataframe['mean_test_s

plt.fill_between(logreg_dataframe.param_C.astype(np.float),
                 logreg_dataframe['mean_train_score'] + logreg_dataframe['st
                 logreg_dataframe['mean_train_score'] - logreg_dataframe['st
plt.fill_between(logreg_dataframe.param_C.astype(np.float),
                 logreg_dataframe['mean_test_score'] + logreg_dataframe['std
                 logreg_dataframe['mean_test_score'] - logreg_dataframe['std

plt.legend([line1, line2], ["mean_train_score", "mean_test_score"], loc="upp
ax1.set_ylabel("score")
ax1.set_xlabel("C")
ax1.set_xscale("log")
plt.title("Score vs C for Logistic Regression with StandardScaler")
plt.savefig("Logistic Regression score vs C", bbox_inches = 'tight')
plt.show()
```



In [46]:
```
plt.figure()
ax1 = plt.gca()
linearsvc_dataframe = pd.DataFrame(grid_linearsvc.cv_results_)
line1, = ax1.plot(linearsvc_dataframe['param_C'], linearsvc_dataframe['mean_
line2, = ax1.plot(linearsvc_dataframe['param_C'], linearsvc_dataframe['mean_

plt.fill_between(linearsvc_dataframe.param_C.astype(np.float),
                 linearsvc_dataframe['mean_train_score'] + linearsvc_datafra
                 linearsvc_dataframe['mean_train_score'] - linearsvc_datafra
plt.fill_between(linearsvc_dataframe.param_C.astype(np.float),
                 linearsvc_dataframe['mean_test_score'] + linearsvc_datafran
                 linearsvc_dataframe['mean_test_score'] - linearsvc_datafran

plt.legend([line1, line2], ["mean_train_score", "mean_test_score"], loc="upp
ax1.set_ylabel("score")
ax1.set_xlabel("C")
ax1.set_xscale("log")
plt.title("Score vs C for Linear SVC with StandardScaler")
plt.savefig("Linear SVC score vs C", bbox_inches = 'tight')
plt.show()
```



In [52]:
```
plt.figure()
ax1 = plt.gca()
nearest_centroid_dataframe = pd.DataFrame(grid_nearest_centroid.cv_results_)
line1, = ax1.plot(nearest_centroid_dataframe['param_shrink_threshold'], near
line2, = ax1.plot(nearest_centroid_dataframe['param_shrink_threshold'], near

plt.fill_between(nearest_centroid_dataframe.param_shrink_threshold.astype(np
                 nearest_centroid_dataframe['mean_train_score'] + nearest_ce
                 nearest_centroid_dataframe['mean_train_score'] - nearest_ce
plt.fill_between(nearest_centroid_dataframe.param_shrink_threshold.astype(np
                 nearest_centroid_dataframe['mean_test_score'] + nearest_cen
                 nearest_centroid_dataframe['mean_test_score'] - nearest_cen

plt.legend([line1, line2], ["mean_train_score", "mean_test_score"], loc="upp
ax1.set_ylabel("score")
ax1.set_xlabel("shrink_threshold")
ax1.set_xscale("log")
plt.title("Score vs shrink_threshold for Nearest Centroid with StandardScale
plt.savefig("Nearest Centroid score vs C", bbox_inches = 'tight')
plt.show()
```




## 2.4

In [55]:

```
# Kfold for Logistic Regression
from sklearn.model_selection import KFold
params = [{'C': [0.01, 1, 10, 100, 1000]}]
kf = KFold(shuffle=True, random_state = 0)
grid_logreg_kfold = GridSearchCV(LogisticRegression(multi_class='multinomial
grid_logreg_kfold.fit(X_train_scaled,y_train)
pd.DataFrame(grid_logreg_kfold.cv_results_)
```

Out[55]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_C | params | rank_tes |
|---|---|---|---|---|---|---|---|
| **0** | 13.819211 | 0.026531 | 0.722734 | 0.722888 | 0.01 | {'C': 0.01} | |
| **1** | 13.279655 | 0.025883 | 0.723951 | 0.724030 | 1 | {'C': 1} | |
| **2** | 13.156065 | 0.026231 | 0.723930 | 0.724046 | 10 | {'C': 10} | |
| **3** | 13.596971 | 0.026013 | 0.723680 | 0.723847 | 100 | {'C': 100} | |
| **4** | 13.549845 | 0.026424 | 0.723705 | 0.723838 | 1000 | {'C': 1000} | |

In [56]:

```
from sklearn.model_selection import KFold
params = [{'C': [0.01, 1, 10, 100, 1000]}]
kf = KFold(shuffle=True, random_state = 23)
grid_logreg_kfold = GridSearchCV(LogisticRegression(multi_class='multinomial
grid_logreg_kfold.fit(X_train_scaled,y_train)
pd.DataFrame(grid_logreg_kfold.cv_results_)
```

Out[56]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_C | params | rank_tes |
|---|---|---|---|---|---|---|---|
| **0** | 14.881437 | 0.029927 | 0.722764 | 0.723051 | 0.01 | {'C': 0.01} | |
| **1** | 13.973566 | 0.029743 | 0.723671 | 0.724040 | 1 | {'C': 1} | |
| **2** | 13.885630 | 0.028339 | 0.723762 | 0.724038 | 10 | {'C': 10} | |
| **3** | 15.164577 | 0.028938 | 0.723845 | 0.724104 | 100 | {'C': 100} | |
| **4** | 14.234138 | 0.025845 | 0.723836 | 0.724089 | 1000 | {'C': 1000} | |

In [57]:

```
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(
    forest_data, cover_type, random_state=24)
#scaler = StandardScaler()
scaler.fit(X_train_new)
X_train_new_scaled = scaler.transform(X_train_new)
X_test_new_scaled = scaler.transform(X_test_new)
```

In [58]:
```
params = [{'C': [0.01, 1, 10, 100, 1000]}]
kf = KFold(shuffle=True, random_state = 23)
grid_logreg_kfold_new_scaled = GridSearchCV(LogisticRegression(multi_class='
grid_logreg_kfold_new_scaled.fit(X_train_new_scaled,y_train)
pd.DataFrame(grid_logreg_kfold_new_scaled.cv_results_)
```

Out[58]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_C | params | rank_tes |
|---|---|---|---|---|---|---|---|
| **0** | 12.749484 | 0.024265 | 0.487389 | 0.487438 | 0.01 | {'C': 0.01} | |
| **1** | 12.939968 | 0.024612 | 0.487386 | 0.487437 | 1 | {'C': 1} | |
| **2** | 13.010075 | 0.024633 | 0.487386 | 0.487436 | 10 | {'C': 10} | |
| **3** | 13.607497 | 0.024616 | 0.487386 | 0.487437 | 100 | {'C': 100} | |
| **4** | 13.669615 | 0.027509 | 0.487386 | 0.487437 | 1000 | {'C': 1000} | |

In [ ]:

In [60]:
```
# Kfold for Linear SVC
from sklearn.model_selection import KFold
params = [{'C': [0.01, 1, 10, 100, 1000]}]
kf = KFold(shuffle=True, random_state = 0)
grid_linearsvc_kfold = GridSearchCV(LinearSVC(dual=False, tol=0.001), params
grid_linearsvc_kfold.fit(X_train_scaled,y_train)
pd.DataFrame(grid_linearsvc_kfold.cv_results_)
```

Out[60]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_C | params | rank_tes |
|---|---|---|---|---|---|---|---|
| **0** | 38.502326 | 0.036306 | 0.712297 | 0.712397 | 0.01 | {'C': 0.01} | |
| **1** | 43.159197 | 0.028188 | 0.712607 | 0.712786 | 1 | {'C': 1} | |
| **2** | 44.027135 | 0.033794 | 0.712605 | 0.712793 | 10 | {'C': 10} | |
| **3** | 45.255886 | 0.036994 | 0.712600 | 0.712787 | 100 | {'C': 100} | |
| **4** | 42.431367 | 0.029711 | 0.712607 | 0.712793 | 1000 | {'C': 1000} | |

In [61]:

```
params = [{'C': [0.01, 1, 10, 100, 1000]}]
kf = KFold(shuffle=True, random_state = 23)
grid_linearsvc_kfold = GridSearchCV(LinearSVC(dual=False, tol=0.001), params
grid_linearsvc_kfold.fit(X_train_scaled,y_train)
pd.DataFrame(grid_linearsvc_kfold.cv_results_)
```

Out[61]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_C | params | rank_tes |
|---|---|---|---|---|---|---|---|
| **0** | 38.096436 | 0.029992 | 0.712316 | 0.712480 | 0.01 | {'C': 0.01} | |
| **1** | 41.421902 | 0.029080 | 0.712605 | 0.712834 | 1 | {'C': 1} | |
| **2** | 41.952436 | 0.037596 | 0.712616 | 0.712849 | 10 | {'C': 10} | |
| **3** | 40.118394 | 0.031115 | 0.712614 | 0.712850 | 100 | {'C': 100} | |
| **4** | 42.095113 | 0.027708 | 0.712612 | 0.712849 | 1000 | {'C': 1000} | |

In [62]:

```
params = [{'C': [0.01, 1, 10, 100, 1000]}]
kf = KFold(shuffle=True, random_state = 23)
grid_linearsvc_kfold_new_scaled = GridSearchCV(LinearSVC(dual=False, tol=0.(
grid_linearsvc_kfold_new_scaled.fit(X_train_new_scaled,y_train)
pd.DataFrame(grid_linearsvc_kfold_new_scaled.cv_results_)
```

Out[62]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_C | params | rank_tes |
|---|---|---|---|---|---|---|---|
| **0** | 30.087380 | 0.026045 | 0.487386 | 0.487439 | 0.01 | {'C': 0.01} | |
| **1** | 23.956915 | 0.026351 | 0.487386 | 0.487437 | 1 | {'C': 1} | |
| **2** | 25.441062 | 0.032273 | 0.487386 | 0.487437 | 10 | {'C': 10} | |
| **3** | 25.502872 | 0.033514 | 0.487386 | 0.487437 | 100 | {'C': 100} | |
| **4** | 24.968369 | 0.028285 | 0.487386 | 0.487437 | 1000 | {'C': 1000} | |

In [65]:
```
# Kfold for Nearest Centroid
params = [{'shrink_threshold': [0,0.5,1,10, 15, 20, 50, 100]}]
kf = KFold(shuffle=True, random_state = 0)
grid_nearest_centroid_kfold = GridSearchCV(NearestCentroid(), params, cv=kf)
grid_nearest_centroid_kfold.fit(X_train_scaled,y_train)
pd.DataFrame(grid_nearest_centroid_kfold.cv_results_)
```

Out[65]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_shrink_threshold | |
|---|---|---|---|---|---|---|
| **0** | 0.195823 | 0.038003 | 0.549905 | 0.550000 | 0 | {'sh |
| **1** | 0.395064 | 0.046341 | 0.548634 | 0.548724 | 0.5 | {'sh |
| **2** | 0.398716 | 0.035100 | 0.547112 | 0.547431 | 1 | {'sh |
| **3** | 0.377510 | 0.035181 | 0.542933 | 0.543051 | 10 | {'sh |
| **4** | 0.369522 | 0.038149 | 0.547571 | 0.547768 | 15 | {'sh |
| **5** | 0.360834 | 0.041314 | 0.553226 | 0.553234 | 20 | {'sh |
| **6** | 0.354040 | 0.038796 | 0.634119 | 0.634307 | 50 | {'sh |
| **7** | 0.358631 | 0.034428 | 0.515140 | 0.515244 | 100 | {'sh |

In [66]:
```
params = [{'shrink_threshold': [0,0.5,1,10, 15, 20, 50, 100]}]
kf = KFold(shuffle=True, random_state = 23)
grid_nearest_centroid_kfold = GridSearchCV(NearestCentroid(), params, cv=kf)
grid_nearest_centroid_kfold.fit(X_train_scaled,y_train)
pd.DataFrame(grid_nearest_centroid_kfold.cv_results_)
```

Out[66]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_shrink_threshold | |
|---|---|---|---|---|---|---|
| **0** | 0.181291 | 0.036835 | 0.549948 | 0.549852 | 0 | {'sh |
| **1** | 0.426806 | 0.039336 | 0.548735 | 0.548708 | 0.5 | {'sh |
| **2** | 0.350931 | 0.034518 | 0.547264 | 0.547296 | 1 | {'sh |
| **3** | 0.349073 | 0.033866 | 0.542974 | 0.542902 | 10 | {'sh |
| **4** | 0.338475 | 0.035174 | 0.547314 | 0.547286 | 15 | {'sh |
| **5** | 0.346626 | 0.033742 | 0.552748 | 0.552847 | 20 | {'sh |
| **6** | 0.343126 | 0.033329 | 0.633919 | 0.633970 | 50 | {'sh |
| **7** | 0.346839 | 0.033320 | 0.514897 | 0.515001 | 100 | {'sh |

In [67]:
```
params = [{'shrink_threshold': [0,0.5,1,10, 15, 20, 50, 100]}]
kf = KFold(shuffle=True, random_state = 23)
grid_nearest_centroid_kfold_new_scaled = GridSearchCV(NearestCentroid(), par
grid_nearest_centroid_kfold_new_scaled.fit(X_train_new_scaled,y_train)
pd.DataFrame(grid_nearest_centroid_kfold_new_scaled.cv_results_)
```

Out[67]:

| | mean_fit_time | mean_score_time | mean_test_score | mean_train_score | param_shrink_threshold | |
|---|---|---|---|---|---|---|
| **0** | 0.202094 | 0.044167 | 0.035937 | 0.037689 | 0 | {'sh |
| **1** | 0.408201 | 0.038316 | 0.032440 | 0.034003 | 0.5 | {'sh |
| **2** | 0.422890 | 0.044169 | 0.040481 | 0.040756 | 1 | {'sh |
| **3** | 0.423583 | 0.045564 | 0.364612 | 0.364613 | 10 | {'sh |
| **4** | 0.383388 | 0.037520 | 0.364612 | 0.364613 | 15 | {'sh |
| **5** | 0.396229 | 0.034246 | 0.364612 | 0.364613 | 20 | {'sh |
| **6** | 0.362487 | 0.033999 | 0.364612 | 0.364613 | 50 | {'sh |
| **7** | 0.369381 | 0.034582 | 0.364612 | 0.364613 | 100 | {'sh |

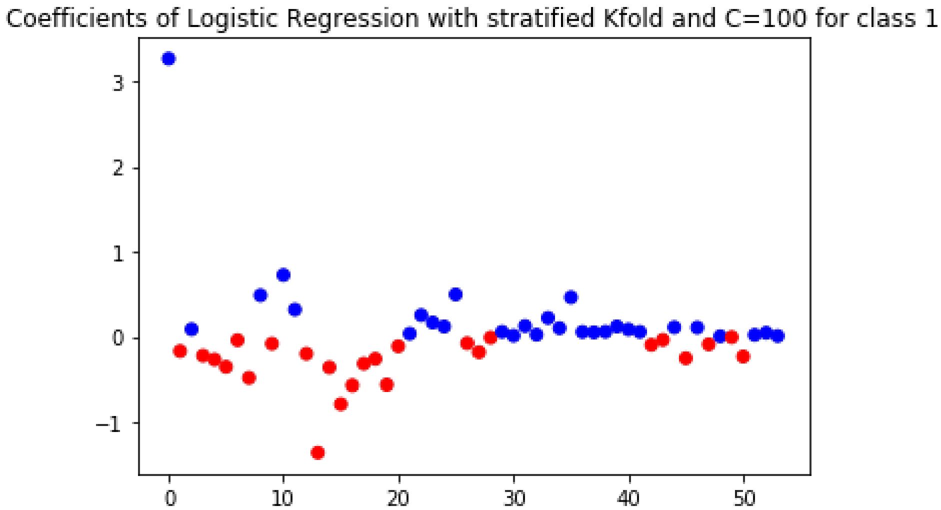
# 2.5

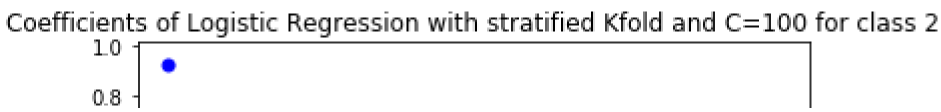
In [82]:
```
shape = grid_logreg.best_estimator_.coef_.shape
len(grid_logreg.best_estimator_.coef_[0])
```

Out[82]: 54

For Logistic Regression, stratified Kfold with C=100 and standard scaling gave the best score and hence visualizing the coefficients for that model

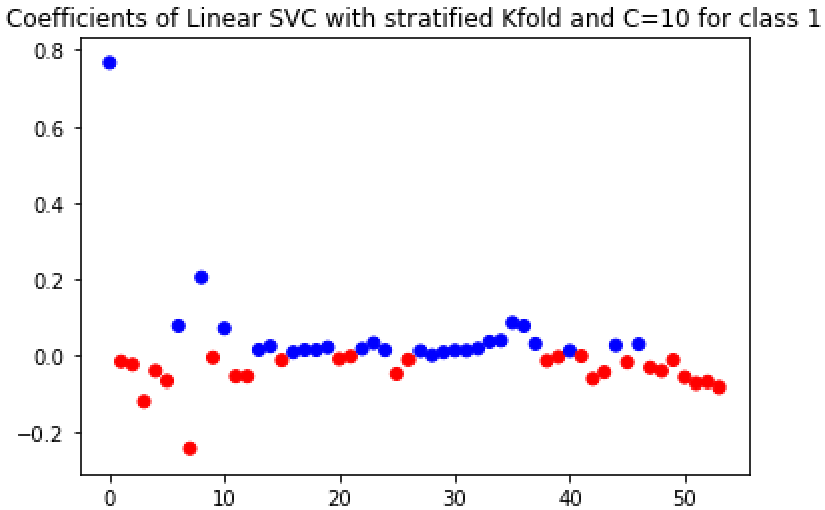
In [85]:
```
for i in range(0,7):
    plt.scatter(range(54), grid_logreg.best_estimator_.coef_[i],
                c=np.sign(grid_logreg.best_estimator_.coef_[i]), cmap="k
    plt.title("Coefficients of Logistic Regression with stratified Kfold and
    plt.savefig("Coefficients of Logistic Regression with stratified Kfold a
    plt.show()
```




For Linear SVC, stratified Kfold with C=10 and standard scaling gave the best score and hence visualizing the coefficients for that model

In [86]:
```
for i in range(0,7):
    plt.scatter(range(54), grid_linearsvc.best_estimator_.coef_[i],
                    c=np.sign(grid_linearsvc.best_estimator_.coef_[i]), cmap
    plt.title("Coefficients of Linear SVC with stratified Kfold and C=10 for
    plt.savefig("Coefficients of Linear SVC with stratified Kfold and C=10 f
    plt.show()
```




In [103]:
```
count = 0
j=0
k=0
fig = plt.figure(figsize=(10,50))
for i in range(0,14):
    fig.add_subplot(14,1,i+1)
    if i%2 == 0:
        plt.scatter(range(54), grid_logreg.best_estimator_.coef_[j],
                    c=np.sign(grid_logreg.best_estimator_.coef_[j]), cmap="b
        plt.title("Coefficients of Logistic Regression with stratified Kfold
        j+=1
    else:
        plt.scatter(range(54), grid_linearsvc.best_estimator_.coef_[k],
                    c=np.sign(grid_linearsvc.best_estimator_.coef_[k]), cmap
        plt.title("Coefficients of Linear SVC with stratified Kfold and C=1(
        k+=1

    plt.tight_layout()
    count += 1
plt.show()
```

```
<matplotlib.figure.Figure at 0x12db2ca90>
```

Logistic Regression and Linear SVM are agreeing on which features are important

In [ ]: