Classification

Abstract: The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y)

Source Link: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing (https://archive.ics.uci.edu/ml/datasets/Bank+Marketing)

File name used: bank.csv

Evaluation strategy: Accuracy is used as an evaluation strategy because accuracy is easy to understand and easily suited for binary as well as a multiclass classification problem.

Regression

Abstratct: The data is related with various models of Toyota cars. The goal is to predict the prices(y) of the cars.

Source Link: https://www.kaggle.com/klkwak/toyotacorollacsv (https://www.kaggle.com/klkwak/toyotacorollacsv)

MinMaxScaler is used for both the tasks because StandardScaler cannot guarantee balanced feature scales in the presence of outliers.

There are no missing values in the original datasets, values are manually removed.

Classification

Importing libraries and reading data

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import graphviz
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
bank = pd.read_csv('bank_new.csv')
bank.head()
```

Out[2]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	car
0	30.0	unemployed	married	primary	no	1787.0	no	no	cellular	19	oct	79.0	
1	33.0	services	married	secondary	no	4789.0	yes	yes	cellular	11	may	220.0	
2	35.0	management	single	tertiary	no	1350.0	yes	no	cellular	16	apr	185.0	
3	30.0	management	married	tertiary	no	1476.0	yes	yes	unknown	3	jun	NaN	
4	59.0	blue-collar	married	secondary	no	0.0	yes	no	unknown	5	may	226.0	

In [3]:
```
bank.shape
```
Out[3]:
```
(4521, 17)
```

In [4]:
```
bank.describe()
```
Out[4]:

	age	balance	day	duration	campaign	pdays	previous
count	4261.000000	4437.000000	4521.000000	4427.000000	4521.000000	4521.000000	4521.000000
mean	41.179535	1420.179851	15.915284	264.670431	2.793630	39.766645	0.542579
std	10.543468	3013.249912	8.247667	260.517101	3.109807	100.121124	1.693562
min	19.000000	-3313.000000	1.000000	4.000000	1.000000	-1.000000	0.000000
25%	33.000000	68.000000	9.000000	104.000000	1.000000	-1.000000	0.000000
50%	39.000000	443.000000	16.000000	185.000000	2.000000	-1.000000	0.000000
75%	49.000000	1469.000000	21.000000	330.000000	3.000000	-1.000000	0.000000
max	87.000000	71188.000000	31.000000	3025.000000	50.000000	871.000000	25.000000

In [5]:
```
bank.isna().sum()
```
Out[5]:
```
age          260
job            0
marital        0
education      0
default        0
balance       84
housing        0
loan           0
contact        0
day            0
month          0
duration      94
campaign       0
pdays          0
previous       0
poutcome       0
y              0
dtype: int64
```

Dropping time series variables and job description

In [6]:
```
bank = bank.drop(['job','day','month'], axis=1)
```

Creating dummy variables for categorical variables

In [7]:
```
pd.set_option('display.max_columns', None)
bank = pd.get_dummies(bank, columns = ['marital', 'education', 'default', 'housing
', 'loan', 'contact', 'poutcome'], drop_first = True)
```

Mapping target valriable to 1 and 0

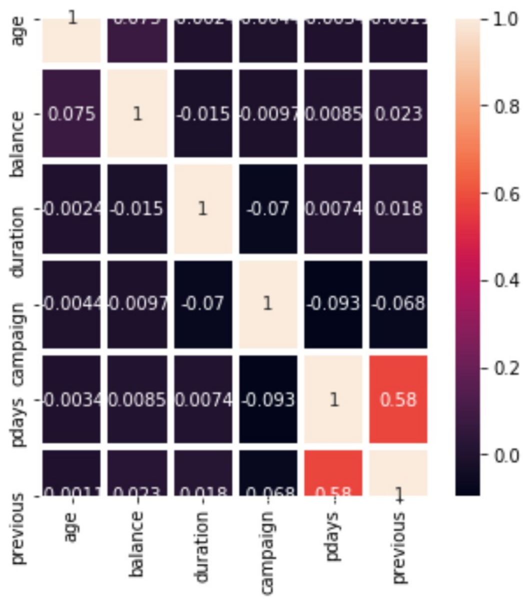
In [8]:
```
bank['y'] = bank.y.map({'yes':1, 'no':0})
```

Creating a list of numerical variables to find the correlation

In [9]:
```
bank_numerical = bank[['age', 'balance', 'duration', 'campaign', 'pdays', 'previous
']]
```

In [10]:
```
a = bank_numerical.corr()
fig=plt.figure(figsize=(5,5))
sns.heatmap(a,annot= True,linewidths=3)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x28758f122c8>



There is no correlation between the variables, 0.58 is not being considered as high correlation in this case.

Splitting the data into Train and Test

In [11]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X = bank.drop('y',axis=1)
y = bank['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_
state = 5)
```

Data Imputation

In [12]:
```
#Replacing blanks in age column with its median in Train and Test data
X_train['age'].fillna(X_train['age'].median(), inplace = True)
```

In [13]:
```
X_test['age'].fillna(X_test['age'].median(), inplace = True)
```

In [14]:
```
#Replacing blanks in balance column with its median in Train and Test data
X_train['balance'].fillna(X_train['balance'].median(), inplace = True)
```

In [15]:
```
X_test['balance'].fillna(X_test['balance'].median(), inplace = True)
```

In [16]:
```
#Replacing blanks in duration column with its median in Train and Test data
X_train['duration'].fillna(X_train['duration'].median(), inplace = True)
```

In [17]:
```
X_test['duration'].fillna(X_test['duration'].median(), inplace = True)
```

Feature scaling using MinMaxScaler

In [18]:
```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
X_train[['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']] = sc.fit_t
ransform(X_train[['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']])
X_test[['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']] = sc.transf
orm(X_test[['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']])
```

1. SVC with kernels = linear, rbf, poly

In [19]:
```
# Fitting Kernel SVM to the Training set
from sklearn.svm import SVC
classifier = SVC()
classifier.fit(X_train, y_train)
print('Train score: {:.4f}'.format(classifier.score(X_train, y_train)))
print('Test score: {:.4f}'.format(classifier.score(X_test, y_test)))
```

```
Train score: 0.8927
Test score: 0.8939
```

This is an underfitting model as the train score is less than the test score

SVC Confusion matrix

In [20]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
accuracy=accuracy_score(y_test,y_pred)
print('accuracy: {:.4f}'.format(accuracy))
```

```
              precision    recall  f1-score   support

           0       0.90      0.99      0.94       803
           1       0.62      0.15      0.24       102

    accuracy                           0.89       905
   macro avg       0.76      0.57      0.59       905
weighted avg       0.87      0.89      0.86       905

accuracy: 0.8939
```

SVC Gridsearch

In [21]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'C': [1, 10, 100], 'kernel': ['linear']},
              {'C': [1, 10, 100], 'kernel': ['rbf'], 'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]},
            {'C': [1, 10, 100], 'kernel': ['poly'], 'degree': [0, 1, 2, 3, 4, 5, 6]}]
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy')
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Out[21]:
```
{'C': 10, 'gamma': 0.2, 'kernel': 'rbf'}
```

In [22]:
```
accuracy = grid_search.best_score_
accuracy
```

Out[22]:
```
0.8987831858407079
```

Fitting SVC with the best parameters from Gridsearch

In [23]:
```
# Fitting Kernel SVM to the Training set with the grid search parameters
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', gamma = 0.2, C = 10)
classifier.fit(X_train, y_train)
print('Train score: {:.4f}'.format(classifier.score(X_train, y_train)))
print('Test score: {:.4f}'.format(classifier.score(X_test, y_test)))
```

```
Train score: 0.9043
Test score: 0.8972
```

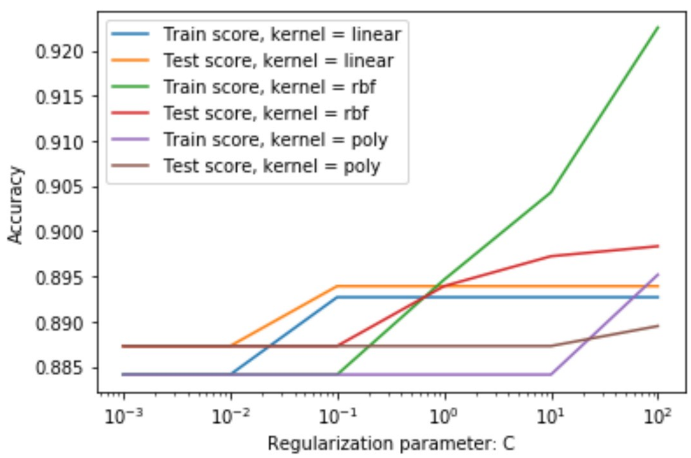
SVC Cross-Validation (kernel = 'rbf', gamma = 0.2, C = 10)

In [24]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(classifier,X_train,y_train,cv=5)
test_score_list = cross_val_score(classifier,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8927
Avg Test Score:0.8928
```

In [25]:
```
from sklearn.svm import SVC

c_range = [0.001, 0.01, 0.1, 1, 10, 100]
train_score_linear = []
train_score_rbf = []
train_score_poly = []
test_score_linear = []
test_score_rbf = []
test_score_poly = []

for c in c_range:
    linear = SVC(kernel = 'linear', C = c)
    rbf = SVC(kernel = 'rbf', C = c, gamma = 0.2)
    poly = SVC(kernel = 'poly', C = c)
    linear.fit(X_train, y_train)
    rbf.fit(X_train, y_train)
    poly.fit(X_train, y_train)
    train_score_linear.append(linear.score(X_train, y_train))
    train_score_rbf.append(rbf.score(X_train, y_train))
    train_score_poly.append(poly.score(X_train, y_train))
    test_score_linear.append(linear.score(X_test, y_test))
    test_score_rbf.append(rbf.score(X_test, y_test))
    test_score_poly.append(poly.score(X_test, y_test))
```

In [26]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(c_range, train_score_linear, label = 'Train score, kernel = linear')
plt.plot(c_range, test_score_linear, label = 'Test score, kernel = linear')
plt.plot(c_range, train_score_rbf, label = 'Train score, kernel = rbf')
plt.plot(c_range, test_score_rbf, label = 'Test score, kernel = rbf')
plt.plot(c_range, train_score_poly, label = 'Train score, kernel = poly')
plt.plot(c_range, test_score_poly, label = 'Test score, kernel = poly')
plt.legend()
plt.xlabel('Regularization parameter: C')
plt.ylabel('Accuracy')
plt.xscale('log')
```



- Graph shows that C=100 gives best test accuracy but this should be verified with cross-validation

In [27]:
```
# Fitting Kernel SVM to the Training set with the grid search parameters
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', gamma = 0.2, C = 100)
classifier.fit(X_train, y_train)
print('Train score: {:.4f}'.format(classifier.score(X_train, y_train)))
print('Test score: {:.4f}'.format(classifier.score(X_test, y_test)))
```

```
Train score: 0.9226
Test score: 0.8983
```

SVC Cross-Validation (kernel = 'rbf', gamma = 0.2, C = 100)

In [28]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(classifier,X_train,y_train,cv=5)
test_score_list = cross_val_score(classifier,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8957
Avg Test Score:0.8829
```

SVM-Kernal Summary

- From the gridsearch, the best parameters are: {'C': 10, 'gamma': 0.2, 'kernel': 'rbf'}.
- A visual representation of various kernels with parameters shows that C:100, gamma:0.2, kernal:rbf improves the test accuracy but with these parameters, the average test accuracy decreases.
- The best train and test accuracies of SVM-kernal are Train score: 0.9043, Test score: 0.8972.
- Average Train and test scrores are Avg Train Score:0.8927, Avg Test Score:0.8928

2. Logistic Regression

In [29]:
```
# Fitting logistic regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier_Log = LogisticRegression()
classifier_Log.fit(X_train, y_train)
print('Train score: {:.4f}'.format(classifier_Log.score(X_train, y_train)))
print('Test score: {:.4f}'.format(classifier_Log.score(X_test, y_test)))
```

```
Train score: 0.8999
Test score: 0.8972
```

Logistic regression confusion Matrix

In [30]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
y_pred = classifier_Log.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
accuracy=accuracy_score(y_test,y_pred)
print('accuracy: {:.4f}'.format(accuracy))
```

```
              precision    recall  f1-score   support

           0       0.91      0.99      0.94       803
           1       0.64      0.21      0.31       102

    accuracy                           0.90       905
   macro avg       0.77      0.60      0.63       905
weighted avg       0.88      0.90      0.87       905

accuracy: 0.8972
```

Logistic regression Gridsearch

In [31]:
```
# Applying Grid Search to find the best model and the best parameters
from sklearn.model_selection import GridSearchCV
parameters = [{'C': [0.001, 0.01, 0.1, 1, 10, 100,1000], 'penalty': ['l2','l1']}]
grid_search = GridSearchCV(estimator = classifier_Log,
                           param_grid = parameters,
                           scoring = 'accuracy', cv=5, n_jobs = -1)
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Out[31]:
```
{'C': 1, 'penalty': 'l1'}
```

In [32]:
```
accuracy = grid_search.best_score_
accuracy
```

Out[32]:
```
0.9026548672566371
```

Fitting Logistic regression with the best parameters from Gridsearch

In [33]:
```
from sklearn.linear_model import LogisticRegression
classifier_Log = LogisticRegression(penalty = 'l1', C = 1)
classifier_Log.fit(X_train, y_train)
print('Train score: {:.4f}'.format(classifier_Log.score(X_train, y_train)))
print('Test score: {:.4f}'.format(classifier_Log.score(X_test, y_test)))
```

```
Train score: 0.9035
Test score: 0.8983
```

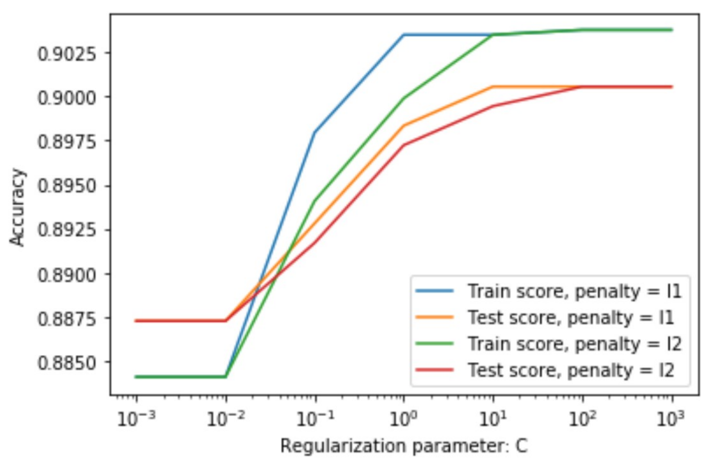
Logistic Cross validation (penality:l1, C:1)

In [34]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(classifier_Log,X_train,y_train,cv=5)
test_score_list = cross_val_score(classifier_Log,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.9027
Avg Test Score:0.8972
```

In [35]:
```
from sklearn.linear_model import LogisticRegression

c_range = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
train_score_l1 = []
train_score_l2 = []
test_score_l1 = []
test_score_l2 = []

for c in c_range:
    log_l1 = LogisticRegression(penalty = 'l1', C = c)
    log_l2 = LogisticRegression(penalty = 'l2', C = c)
    log_l1.fit(X_train, y_train)
    log_l2.fit(X_train, y_train)
    train_score_l1.append(log_l1.score(X_train, y_train))
    train_score_l2.append(log_l2.score(X_train, y_train))
    test_score_l1.append(log_l1.score(X_test, y_test))
    test_score_l2.append(log_l2.score(X_test, y_test))
```

In [36]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(c_range, train_score_l1, label = 'Train score, penalty = l1')
plt.plot(c_range, test_score_l1, label = 'Test score, penalty = l1')
plt.plot(c_range, train_score_l2, label = 'Train score, penalty = l2')
plt.plot(c_range, test_score_l2, label = 'Test score, penalty = l2')
plt.legend()
plt.xlabel('Regularization parameter: C')
plt.ylabel('Accuracy')
plt.xscale('log')
```



- This graph shows that the test accuracy peaks at C=10 and penalty= l1,this should be verified with cross-validation.

In [37]:
```
# Fitting the parameters from the graph
from sklearn.linear_model import LogisticRegression
classifier_Log = LogisticRegression(penalty = 'l1', C = 10)
classifier_Log.fit(X_train, y_train)
print('Train score: {:.4f}'.format(classifier_Log.score(X_train, y_train)))
print('Test score: {:.4f}'.format(classifier_Log.score(X_test, y_test)))
```

```
Train score: 0.9035
Test score: 0.9006
```

Logistic regression Cross-Validation (penalty:l1, C:10)

In [38]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(classifier_Log,X_train,y_train,cv=5)
test_score_list = cross_val_score(classifier_Log,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.9007
Avg Test Score:0.8950
```

Logistic Regression Summary

- Though the Logistic regression yields high test accuracy at penalty=l1, C = 10 as shown, the average test accuracy is less than penalty =l1, C = 1 parameters' average test accuracy.
- The best parameters for Logistic Regression are {'C': 1, 'penalty': 'l1'}
- Before parameter tuning: Train score: 0.8999, Test score: 0.8972
- After parameter tuning: Train score: 0.9035, Test score: 0.8983
- Average scores: Avg Train Score:0.9027, Avg Test Score:0.8972

3. LinearSVC

In [39]:
```
from sklearn.svm import LinearSVC

clf = LinearSVC(dual = False)
clf.fit(X_train,y_train)
print('Train score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test score: {:.4f}'.format(clf.score(X_test, y_test)))
```

```
Train score: 0.9002
Test score: 0.8994
```

LinearSVC Confusion matrix

In [40]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
y_pred = clf.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
accuracy=accuracy_score(y_test,y_pred)
print('accuracy: {:.4f}'.format(accuracy))
```

```
              precision    recall  f1-score   support

           0       0.91      0.98      0.95       803
           1       0.65      0.24      0.35       102

    accuracy                           0.90       905
   macro avg       0.78      0.61      0.65       905
weighted avg       0.88      0.90      0.88       905

accuracy: 0.8994
```

LinearSVC Gridsearch

In [41]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'C': [0.001, 0.01, 0.1, 1, 10, 100], 'penalty': ['l1','l2']}]
grid_search = GridSearchCV(estimator = clf,
                           param_grid = parameters,
                           scoring = 'accuracy')
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Out[41]:
```
{'C': 1, 'penalty': 'l1'}
```

In [42]:
```
accuracy = grid_search.best_score_
accuracy
```

Out[42]:
```
0.9007190265486725
```

Fitting LinearSVC with best parameters from gridsearch

In [43]:
```
from sklearn.svm import LinearSVC

clf = LinearSVC(penalty = 'l1', C= 1, dual=False)
clf.fit(X_train,y_train)
print('Train score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test score: {:.4f}'.format(clf.score(X_test, y_test)))
```

```
Train score: 0.9004
Test score: 0.8994
```

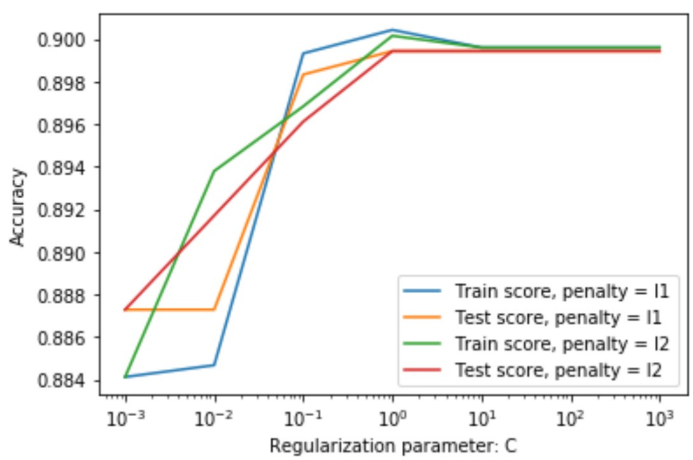
LinearSVC Cross-validation

In [44]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(clf,X_train,y_train,cv=5)
test_score_list = cross_val_score(clf,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.9004
Avg Test Score:0.8939
```

In [45]:
```
from sklearn.svm import LinearSVC

c_range = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
train_score_l1 = []
train_score_l2 = []
test_score_l1 = []
test_score_l2 = []

for c in c_range:
    lin_l1 = LinearSVC(penalty = 'l1', C = c, dual = False)
    lin_l2 = LinearSVC(penalty = 'l2', C = c, dual = False)
    lin_l1.fit(X_train, y_train)
    lin_l2.fit(X_train, y_train)
    train_score_l1.append(lin_l1.score(X_train, y_train))
    train_score_l2.append(lin_l2.score(X_train, y_train))
    test_score_l1.append(lin_l1.score(X_test, y_test))
    test_score_l2.append(lin_l2.score(X_test, y_test))
```

In [46]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(c_range, train_score_l1, label = 'Train score, penalty = l1')
plt.plot(c_range, test_score_l1, label = 'Test score, penalty = l1')
plt.plot(c_range, train_score_l2, label = 'Train score, penalty = l2')
plt.plot(c_range, test_score_l2, label = 'Test score, penalty = l2')
plt.legend()
plt.xlabel('Regularization parameter: C')
plt.ylabel('Accuracy')
plt.xscale('log')
```



Fitting LinearSVC with l2 penalty to check the average scores

In [47]:
```
from sklearn.svm import LinearSVC

clf = LinearSVC(penalty = 'l2', C= 1, dual=False)
clf.fit(X_train,y_train)
print('Train score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test score: {:.4f}'.format(clf.score(X_test, y_test)))
```

```
Train score: 0.9002
Test score: 0.8994
```

LinearSVC Cross-Validation with l2 penalty

In [48]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(clf,X_train,y_train,cv=5)
test_score_list = cross_val_score(clf,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8996
Avg Test Score:0.8950
```

Linear SVC Summary

- At C=1, both l1 and l2 yield same test accuracies, but l2 yields high average test accuracy.
- Best parameters for Linear SVC are {C=1, penalty='l2'}.
- Best accuracies are: Train score: 0.9002, Test score: 0.8994
- Best average scores are: Avg Train Score:0.8996 Avg Test Score:0.8950

4. KNN

In [49]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
print('Train score: {:.4f}'.format(knn.score(X_train, y_train)))
print('Test score: {:.4f}'.format(knn.score(X_test, y_test)))
```

```
Train score: 0.9112
Test score: 0.8873
```

KNN Confusion Matrix

In [50]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
accuracy=accuracy_score(y_test,y_pred)
print('accuracy: {:.4f}'.format(accuracy))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       803
           1       0.50      0.16      0.24       102

    accuracy                           0.89       905
   macro avg       0.70      0.57      0.59       905
weighted avg       0.86      0.89      0.86       905

accuracy: 0.8873
```

KNN Gridsearch

In [51]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'n_neighbors': list(range(3,10)), 'leaf_size': list(range(1,5)), 'we
ights':['uniform', 'distance'], 'p': [1,2]}]
grid_search = GridSearchCV(estimator = knn,
                           param_grid = parameters,
                           scoring = 'accuracy')
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Out[51]:
```
{'leaf_size': 1, 'n_neighbors': 9, 'p': 1, 'weights': 'uniform'}
```

In [52]:
```
accuracy = grid_search.best_score_
accuracy
```

Out[52]:
```
0.8896570796460177
```

Fitting KNN with the best parameters from Gridsearch

In [53]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(leaf_size= 1, n_neighbors = 9, p = 1, weights = 'uniform
')
knn.fit(X_train, y_train)
print('Train score: {:.4f}'.format(knn.score(X_train, y_train)))
print('Test score: {:.4f}'.format(knn.score(X_test, y_test)))
```

```
Train score: 0.9007
Test score: 0.8895
```

KNN Cross-validation with leaf_size= 1, n_neighbors = 9, p = 1, weights = 'uniform'

In [54]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(knn,X_train,y_train,cv=5)
test_score_list = cross_val_score(knn,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

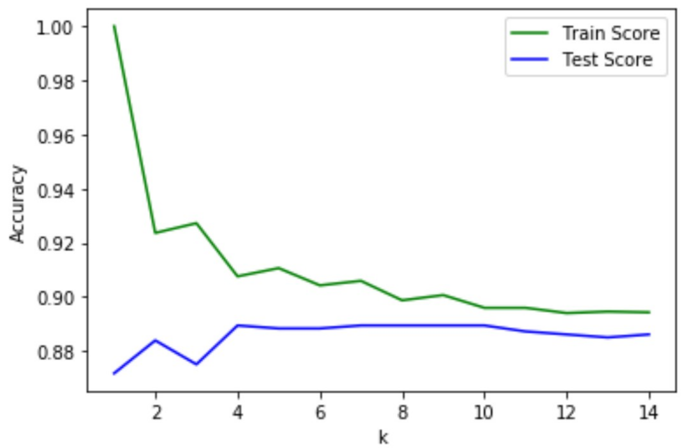
```
Avg Train Score:0.8886
Avg Test Score:0.8829
```

Finding best K using visualization

In [55]:
```
train_score_array = []
test_score_array = []

for k in range(1,15):
    knn = KNeighborsClassifier(leaf_size= 1, n_neighbors = k, p = 1, weights = 'uni
form')
    knn.fit(X_train, y_train)
    train_score_array.append(knn.score(X_train, y_train))
    test_score_array.append(knn.score(X_test, y_test))
```

In [56]:
```
x_axis = range(1,15)
%matplotlib inline
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend()
```

Out[56]: `<matplotlib.legend.Legend at 0x2875c2d87c8>`



Test score peaks at k=4

In [57]:
```
knn = KNeighborsClassifier(leaf_size= 1, n_neighbors = 4, p = 1, weights = 'uniform
')
knn.fit(X_train, y_train)
print('Train score: {:.4f}'.format(knn.score(X_train, y_train)))
print('Test score: {:.4f}'.format(knn.score(X_test, y_test)))
```

```
Train score: 0.9076
Test score: 0.8895
```

KNN Cross-Validation with leaf_size= 1, n_neighbors = 4, p = 1, weights = 'uniform'

In [58]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(knn,X_train,y_train,cv=5)
test_score_list = cross_val_score(knn,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8872
Avg Test Score:0.8840
```

KNN Summary

- Though the test accuracies are same at k = 9 and 4, the average test accuracy when k=4 is higher.
- The best parameters are leaf_size= 1, n_neighbors = 4, p = 1, weights = 'uniform'
- The best accuracies are: Train score: 0.9076, Test score: 0.8895
- The best average scores are: Avg Train Score:0.8872, Avg Test Score:0.8840

5. Decision Tree

In [59]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
print('Train score: {:.4f}'.format(dtree.score(X_train, y_train)))
print('Test score: {:.4f}'.format(dtree.score(X_test, y_test)))
```

```
Train score: 1.0000
Test score: 0.8718
```

Decision Tree Confusion matrix

In [60]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
accuracy=accuracy_score(y_test,y_pred)
print('accuracy: {:.4f}'.format(accuracy))
```

```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       803
           1       0.42      0.37      0.40       102

    accuracy                           0.87       905
   macro avg       0.67      0.65      0.66       905
weighted avg       0.87      0.87      0.87       905

accuracy: 0.8718
```

Decision Tree Gridsearch

In [61]:
```
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint
parameters = [{"max_depth": [1,2,3,4,5,6,7], "criterion": ["gini", "entropy"]}]
grid_search = GridSearchCV(estimator = dtree,
                           param_grid = parameters,
                           scoring = 'accuracy')
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Out[61]:
```
{'criterion': 'entropy', 'max_depth': 3}
```

In [62]:
```
accuracy = grid_search.best_score_
accuracy
```

Out[62]:
```
0.9001659292035398
```

Fitting Decision Tree with the best parameters from Gridsearch

In [63]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 3)
dtree.fit(X_train, y_train)
print('Train score: {:.4f}'.format(dtree.score(X_train, y_train)))
print('Test score: {:.4f}'.format(dtree.score(X_test, y_test)))
```

```
Train score: 0.9038
Test score: 0.8972
```

Decision Tree Cross-Validation

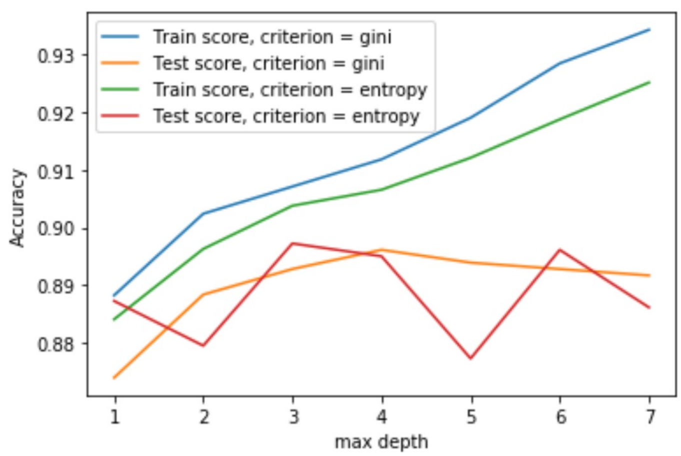
In [64]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(dtree,X_train,y_train,cv=5)
test_score_list = cross_val_score(dtree,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8977
Avg Test Score:0.8928
```

In [65]:
```
from sklearn.tree import DecisionTreeClassifier

depth = [1,2,3,4,5,6,7]
train_score_g = []
train_score_e = []
test_score_g = []
test_score_e = []

for d in depth:
    dec_g = DecisionTreeClassifier(criterion = 'gini', max_depth = d)
    dec_e = DecisionTreeClassifier(criterion = 'entropy', max_depth = d)
    dec_g.fit(X_train, y_train)
    dec_e.fit(X_train, y_train)
    train_score_g.append(dec_g.score(X_train, y_train))
    train_score_e.append(dec_e.score(X_train, y_train))
    test_score_g.append(dec_g.score(X_test, y_test))
    test_score_e.append(dec_e.score(X_test, y_test))
```

In [66]:
```
x_axis = range(1,8)
%matplotlib inline
plt.plot(x_axis, train_score_g, label = 'Train score, criterion = gini')
plt.plot(x_axis, test_score_g, label = 'Test score, criterion = gini')
plt.plot(x_axis, train_score_e, label = 'Train score, criterion = entropy')
plt.plot(x_axis, test_score_e, label = 'Test score, criterion = entropy')
plt.legend()
plt.xlabel('max depth')
plt.ylabel('Accuracy')
```

Out[66]:
```
Text(0, 0.5, 'Accuracy')
```



Decision Tree summary

- Both the graph and gridsearch show that the best parameters are criterion = 'entropy', max_depth = 3
- The best accuracy scores are: Train score: 0.9038, Test score: 0.8972
- The best average scores are: Avg Train Score:0.8977, Avg Test Score:0.8928

Models Summary

In [67]:

```
Models= {'Models':['SVC Kernels','Logistic Regrerssion','Linear SVC', 'KNN','Decisi
on Tree'], 'Without_Hyperparameter_Tuning':[0.8939,0.8972,0.8994,0.8873,0.8796],
          'With_Hyperparameter_Tuning': [0.8972,0.8983,0.8994,0.8895,0.8972], 'Aver
age Test score': [0.8928,0.8972,0.8950,0.8840,0.8929]}
Models_scores = pd.DataFrame(Models)
Models_scores
```

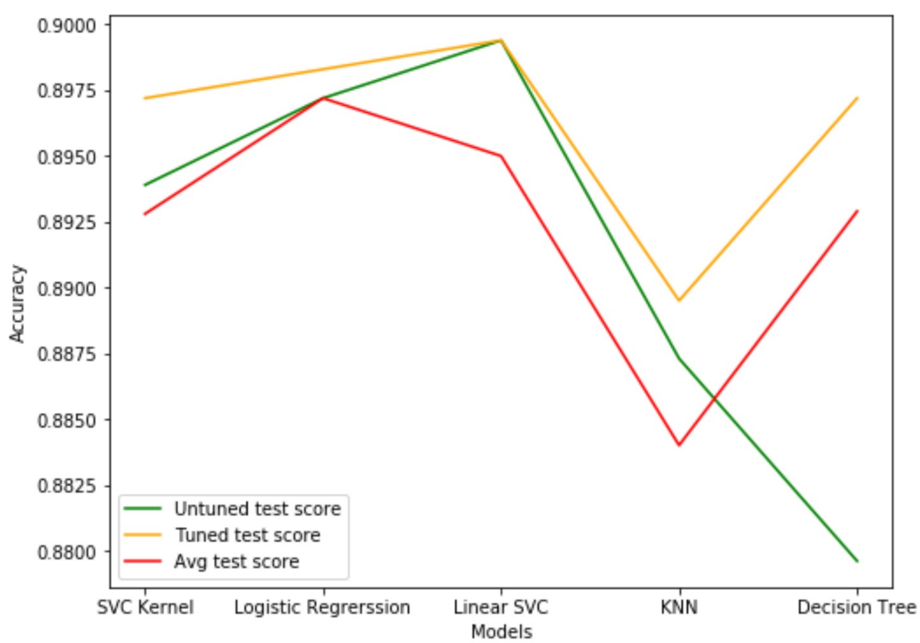
Out[67]:

	Models	Without_Hyperparameter_Tuning	With_Hyperparameter_Tuning	Average Test score
0	SVC Kernels	0.8939	0.8972	0.8928
1	Logistic Regrerssion	0.8972	0.8983	0.8972
2	Linear SVC	0.8994	0.8994	0.8950
3	KNN	0.8873	0.8895	0.8840
4	Decision Tree	0.8796	0.8972	0.8929

In [68]:

```
import matplotlib.pyplot as plt
Models = ['SVC Kernel','Logistic Regrerssion','Linear SVC', 'KNN','Decision Tree']
Without_Hyperparameter = [0.8939,0.8972,0.8994,0.8873,0.8796]
With_Hyperparameter = [0.8972,0.8983,0.8994,0.8895,0.8972]
Average_Test_score = [0.8928,0.8972,0.8950,0.8840,0.8929]

fig=plt.figure(figsize=(8,6))
plt.plot(Models,Without_Hyperparameter, label = 'Untuned test score', color='g')
plt.plot(Models, With_Hyperparameter, label = 'Tuned test score', color='orange')
plt.plot(Models, Average_Test_score, label = 'Avg test score', color='red')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.legend()
```

Out[68]: <matplotlib.legend.Legend at 0x2875c29d208>



- All the models have almost similar accuracies, LinearSVC has the highest accuracy with Logistic next to it.
- But, the average test accuracy of Logistic is better than LinearSVC, so, Logistic regression is the best model.

Regression

Importing Libraries and Reading Data

In [69]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [70]:
```
toyota = pd.read_csv('ToyotaCorolla1.csv')
```

In [71]:
```
toyota.shape
```

Out[71]: (1436, 37)

In [72]:
```
toyota.describe()
```

Out[72]:

	Id	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	HP	Met_C
count	1436.000000	1436.000000	1408.000000	1436.000000	1436.000000	1419.000000	1402.000000	1436.00
mean	721.555014	10730.824513	56.337358	5.548747	1999.625348	68820.710359	101.551355	0.67
std	416.476890	3626.964585	18.496331	3.354085	1.540722	37489.323349	14.788318	0.46
min	1.000000	4350.000000	1.000000	1.000000	1998.000000	1.000000	69.000000	0.00
25%	361.750000	8450.000000	44.000000	3.000000	1998.000000	43060.000000	90.000000	0.00
50%	721.500000	9900.000000	61.000000	5.000000	1999.000000	63792.000000	110.000000	1.00
75%	1081.250000	11950.000000	70.000000	8.000000	2001.000000	87316.000000	110.000000	1.00
max	1442.000000	32500.000000	80.000000	12.000000	2004.000000	243000.000000	192.000000	1.00

In [73]:
```
toyota.isna().sum()
```

Out[73]:
```
Id                    0
Model                 0
Price                 0
Age_08_04            28
Mfg_Month             0
Mfg_Year              0
KM                   17
Fuel_Type             0
HP                   34
Met_Color             0
Automatic             0
cc                   15
Doors                 0
Cylinders             0
Gears                 0
Quarterly_Tax        11
Weight                8
Mfr_Guarantee         0
BOVAG_Guarantee       0
Guarantee_Period     14
ABS                   0
Airbag_1              0
Airbag_2              0
Airco                 0
Automatic_airco       0
Boardcomputer         0
CD_Player             0
Central_Lock          0
Powered_Windows       0
Power_Steering        0
Radio                 0
Mistlamps             0
Sport_Model           0
Backseat_Divider      0
Metallic_Rim          0
Radio_cassette        0
Tow_Bar               0
dtype: int64
```

Dropping Time Series Variables and ID

In [74]:
```
toyota = toyota.drop(['Id','Model','Mfg_Month','Mfg_Year'], axis=1)
```

In [75]:
```
toyota['Cylinders'].value_counts()
```

Out[75]:
```
4    1436
Name: Cylinders, dtype: int64
```

In [76]:
```
#since columns have same values there is no effect of this column
toyota = toyota.drop(['Cylinders'], axis = 1)
```

Creating a dummy variables for categorical variables

In [77]:
```
pd.set_option('display.max_columns', None)
toyota = pd.get_dummies(toyota, columns = ['Fuel_Type', 'Doors','Gears'], drop_firs
t = True)
```

In [78]:
```
toyota.head()
```

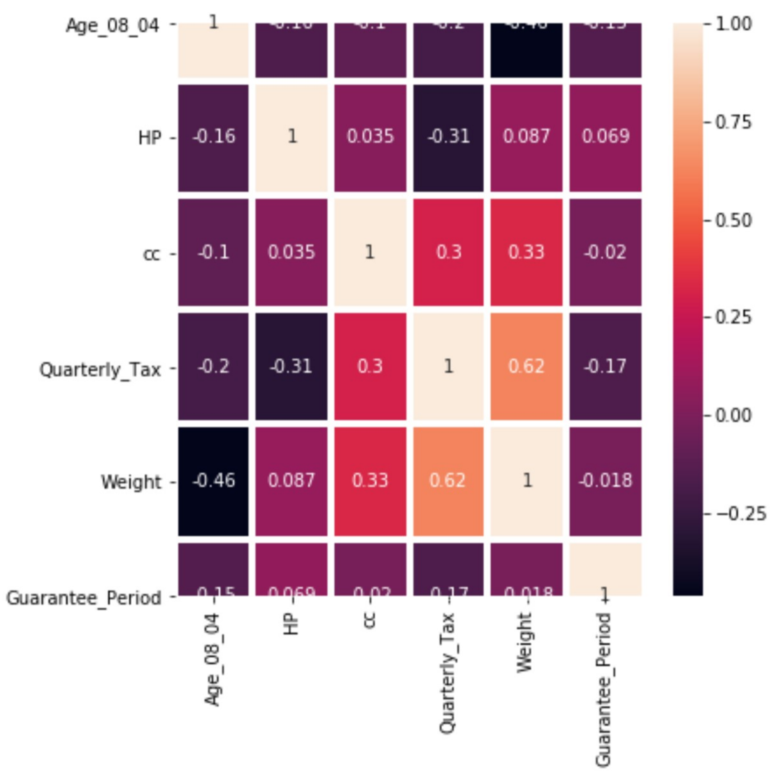
Out[78]:

	Price	Age_08_04	KM	HP	Met_Color	Automatic	cc	Quarterly_Tax	Weight	Mfr_Guarantee	BO'
0	13500	23.0	46986.0	90.0	1	0	2000.0	210.0	1165.0	0	
1	13750	23.0	72937.0	90.0	1	0	2000.0	210.0	1165.0	0	
2	13950	24.0	41711.0	NaN	1	0	2000.0	210.0	1165.0	1	
3	14950	26.0	48000.0	90.0	0	0	2000.0	210.0	1165.0	1	
4	13750	30.0	38500.0	90.0	0	0	2000.0	210.0	1170.0	1	

Checking all the numerical variables for correlation

In [79]:
```
toyota_numerical = toyota[['Age_08_04','HP','cc','Quarterly_Tax','Weight','Guarante
e_Period']]
b= toyota_numerical.corr()
fig=plt.figure(figsize=(6,6))
sns.heatmap(b,annot= True,linewidths=3)
```

Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x2875c765788>



In [80]:
```
b
```

Out[80]:

	Age_08_04	HP	cc	Quarterly_Tax	Weight	Guarantee_Period
Age_08_04	1.000000	-0.162115	-0.100913	-0.196853	-0.463758	-0.148153
HP	-0.162115	1.000000	0.035102	-0.306595	0.087366	0.068516
cc	-0.100913	0.035102	1.000000	0.303811	0.333492	-0.019978
Quarterly_Tax	-0.196853	-0.306595	0.303811	1.000000	0.622841	-0.165954
Weight	-0.463758	0.087366	0.333492	0.622841	1.000000	-0.017601
Guarantee_Period	-0.148153	0.068516	-0.019978	-0.165954	-0.017601	1.000000

Splitting the dataset into the Training set and Test set

In [81]:
```
from sklearn.model_selection import train_test_split
X = toyota.drop('Price',axis=1)
y = toyota['Price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_
state = 5)
```

Imputing the missing data in train and test

In [82]:
```
import warnings
warnings.filterwarnings('ignore')
X_train['Age_08_04'].fillna(X_train['Age_08_04'].median(), inplace = True)
X_test['Age_08_04'].fillna(X_test['Age_08_04'].median(), inplace = True)
```

In [83]:
```
X_train['KM'].fillna(X_train['KM'].median(), inplace = True)
X_test['KM'].fillna(X_test['KM'].median(), inplace = True)
```

In [84]:
```
X_train['HP'].fillna(X_train['HP'].median(), inplace = True)
X_test['HP'].fillna(X_test['HP'].median(), inplace = True)
```

In [85]:
```
X_train['cc'].fillna(X_train['cc'].median(), inplace = True)
X_test['cc'].fillna(X_test['cc'].median(), inplace = True)
```

In [86]:
```
X_train['Weight'].fillna(X_train['Weight'].median(), inplace = True)
X_test['Weight'].fillna(X_test['Weight'].median(), inplace = True)
```

In [87]:
```
X_train['Quarterly_Tax'].fillna(X_train['Quarterly_Tax'].median(), inplace = True)
X_test['Quarterly_Tax'].fillna(X_test['Quarterly_Tax'].median(), inplace = True)
```

In [88]:
```
X_train['Guarantee_Period'].fillna(X_train['Guarantee_Period'].median(), inplace =
True)
X_test['Guarantee_Period'].fillna(X_test['Guarantee_Period'].median(), inplace = Tr
ue)
```

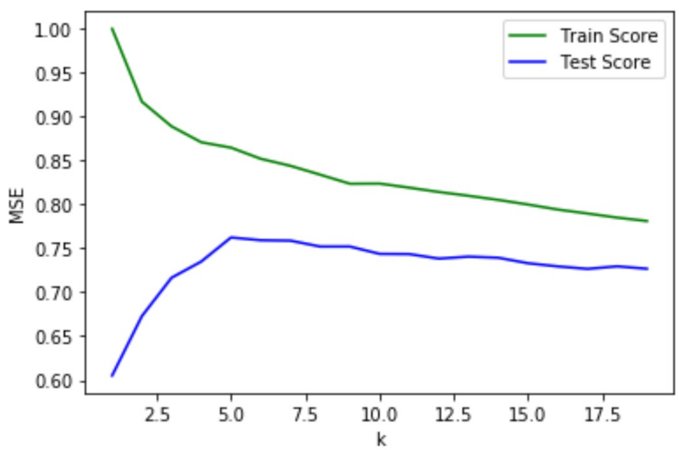
Fetaure Scaling Using MinMaxScaler

In [89]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

KNN Regressor

In [90]:
```
from sklearn.neighbors import KNeighborsRegressor
%matplotlib inline
train_score_array = []
test_score_array = []

for k in range(1,20):
    knn = KNeighborsRegressor(k)
    knn.fit(X_train, y_train)
    train_score_array.append(knn.score(X_train, y_train))
    test_score_array.append(knn.score(X_test, y_test))

x_axis = range(1,20)
plt.plot(x_axis, train_score_array, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_array, c = 'b', label = 'Test Score')
plt.legend()
plt.xlabel('k')
plt.ylabel('MSE')
```

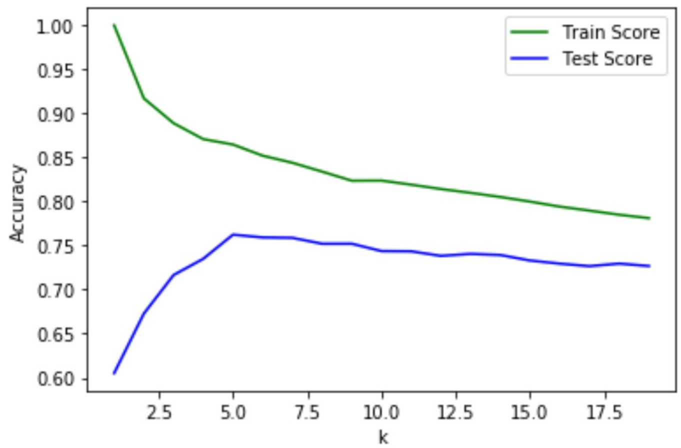
Out[90]: Text(0, 0.5, 'MSE')



As seen in graph the MSE value of k equals 5 is less than other values of k

In [91]:
```
x_axis = range(1,20)
%matplotlib inline
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend()
```

Out[91]: <matplotlib.legend.Legend at 0x2875bfaa608>



Test Scores peaks at k=5

In [92]:
```
knn = KNeighborsRegressor(5)
knn.fit(X_train, y_train)
print('Train score: {:.4f}'.format(knn.score(X_train, y_train)))
print('Test score: {:.4f}'.format(knn.score(X_test, y_test)))
```

```
Train score: 0.8647
Test score: 0.7624
```

Finding best k value using Grid Search

In [93]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import neighbors
params = {'n_neighbors':[2,3,4,5,6,7,8,9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
0]}

knn = neighbors.KNeighborsRegressor()

model = GridSearchCV(knn, params, cv=5)
model.fit(X_train,y_train)
model.best_params_
```

Out[93]: {'n_neighbors': 5}

In [94]:
```
accuracy = model.best_score_
accuracy
```

Out[94]: 0.776380238353679

For k value of 5, cross validation

In [95]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(model,X_train,y_train,cv=5)
test_score_list = cross_val_score(model,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.7721
Avg Test Score:0.6367
```

KNN SUMMARY

1) The average test accuracy when k=5 is higher.(Using Visualization).

2) The best parameter is n_neighbors = 5 using GridSearch.

3) The best accuracies are: Train score: 0.8647, Test score: 0.7624.

4) The best average scores are: Avg Train Score:0.7721, Avg Test Score:0.6367

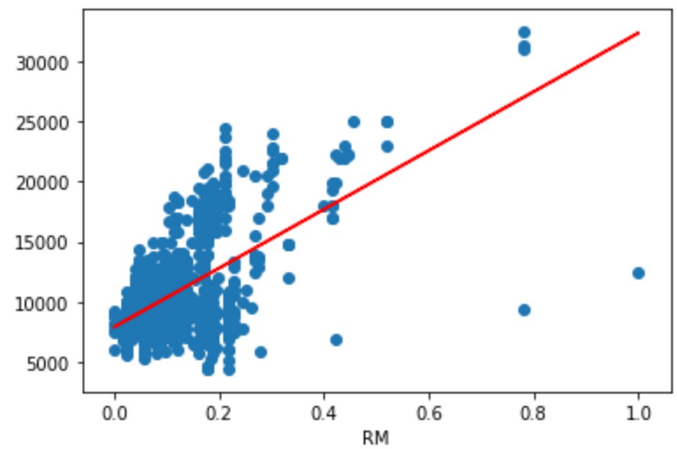
Linear Regressor

In [96]:
```
from sklearn.linear_model import LinearRegression

lreg = LinearRegression()
lreg.fit(X_train, y_train)
print('Train score: {:.4f}'.format(lreg.score(X_train, y_train)))
print('Test score: {:.4f}'.format(lreg.score(X_test, y_test)))
```

```
Train score: 0.8928
Test score: 0.8645
```

In [97]:
```
%matplotlib inline
import matplotlib.pyplot as plt

X_train_rm = X_train[:,7].reshape(-1,1)
lreg.fit(X_train_rm, y_train)
y_predict = lreg.predict(X_train_rm)

plt.plot(X_train_rm, y_predict, c = 'r')
plt.scatter(X_train_rm,y_train)
plt.xlabel('RM')
```

Out[97]:
```
Text(0.5, 0, 'RM')
```



Linear Regression using Grid Search

In [98]:
```
model = LinearRegression()
parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[Tru
e, False]}
grid = GridSearchCV(model,parameters, cv= 5)
grid.fit(X_train, y_train)
print (grid.best_score_)
print (grid.best_params_)
```

```
0.6954581049627842
{'copy_X': True, 'fit_intercept': True, 'normalize': True}
```

In [99]:
```
from sklearn import metrics
from sklearn.model_selection import cross_val_score, cross_val_predict
scores = cross_val_score(lreg, X_train, y_train, cv = 5)     #cv is the number of fo
lds, scores will give an array of scores

print (scores)
```

```
[-0.04920183  0.89178726  0.84693389  0.88983513  0.89966907]
```

In [100]:
```
predictions = cross_val_predict(lreg, X_test, y_test, cv =5)
```

In [101]:
```
accuracy = metrics.r2_score(y_test, predictions)
accuracy
```

Out[101]:
```
0.8477142535903984
```

Fitting model with best parameter in Linear Regression

In [102]:
```
lreg = LinearRegression(copy_X= True, fit_intercept= True, normalize= True)
lreg.fit(X_train, y_train)
print('Train score: {:.4f}'.format(lreg.score(X_train, y_train)))
print('Test score: {:.4f}'.format(lreg.score(X_test, y_test)))
```

```
Train score: 0.8928
Test score: 0.8645
```

Cross Validation for Linear Regression

In [103]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(lreg,X_train,y_train,cv= 5)
test_score_list = cross_val_score(lreg,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.6958
Avg Test Score:0.8365
```

Linear Regression Summary

1) The Linear Regression: Train score: 0.8928 and Test score: 0.8645

2) Using Grid Search, the best parameter we got is copy_X=True, fit_intercept=True, n_jobs=None, normalize=True

3) Fitting the model with best parameters we got, Train score: 0.8928 and Test score: 0.8645

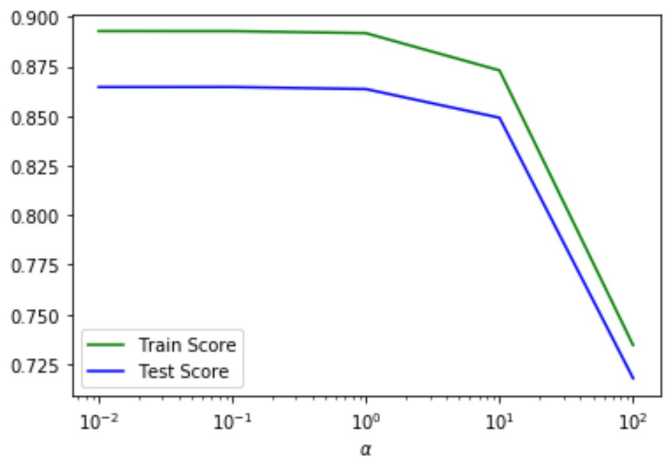
4) The best average scores are: Avg Train Score:0.6958 and Avg Test Score:0.8365

Ridge

In [104]:
```
from  sklearn.linear_model import Ridge

x_range = [0.01, 0.1, 1, 10, 100]
train_score_list = []
test_score_list = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train,y_train)
    train_score_list.append(ridge.score(X_train,y_train))
    test_score_list.append(ridge.score(X_test, y_test))
```

In [105]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

Out[105]:
```
Text(0.5, 0, '$\\alpha$')
```



In [106]:
```
print(train_score_list)
print(test_score_list)
```

```
[0.8927691459326847, 0.8927473125454589, 0.8917299979903542, 0.8729593522349502,
0.7344840326245498]
[0.8645524712373145, 0.8645933472820702, 0.8635536666419096, 0.8491107355888738,
0.7177511832009609]
```

In [107]:
```
ridge = Ridge()
ridge.fit(X_train,y_train)
print('Train score: {:.4f}'.format(ridge.score(X_train,y_train)))
print('Test score: {:.4f}'.format(ridge.score(X_test, y_test)))
```

```
Train score: 0.8917
Test score: 0.8636
```

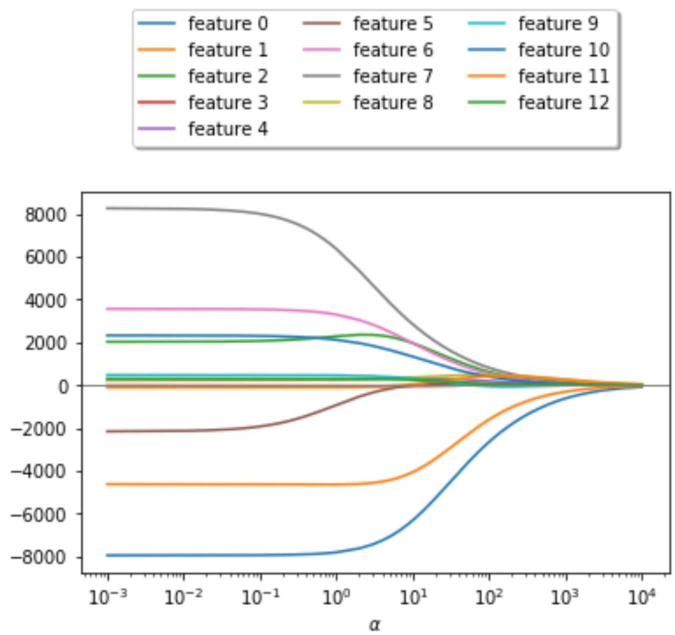
In [108]:
```
ridge.coef_
```

Out[108]:
```
array([-7.81186956e+03, -4.64195077e+03,  2.28805244e+03, -2.75296659e+01,
        2.65590801e+02, -9.22945412e+02,  3.30142672e+03,  6.37045708e+03,
        2.75877067e+02,  4.65458550e+02,  2.12232340e+03, -1.02189102e+02,
        3.01787224e+02, -4.78788952e+01,  2.60668406e+02,  2.87776466e+03,
        1.41411525e+02,  5.26835365e+02, -7.78303007e+00,  4.88618125e+02,
       -1.26368115e+02, -3.08207871e+02, -1.89170569e+02,  3.82688453e+02,
       -3.40088575e+02,  1.91720049e+02,  2.73772152e+02, -2.86363943e+02,
        8.94794108e+02,  1.70861463e+03, -3.69865726e+02, -2.69868609e+02,
       -1.60490711e+02,  1.31033402e+02,  4.74725192e+02,  6.90094514e+02])
```

In [109]:
```
ridge.intercept_
```

Out[109]:
```
12421.415647168014
```

In [110]:
```
%matplotlib inline
import numpy as np

x_range1 = np.linspace(0.001, 1, 100).reshape(-1,1)
x_range2 = np.linspace(1, 10000, 10000).reshape(-1,1)

x_range = np.append(x_range1, x_range2)
coeff = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train,y_train)
    coeff.append(ridge.coef_ )

coeff = np.array(coeff)

for i in range(0,13):
    plt.plot(x_range, coeff[:,i], label = 'feature {:d}'.format(i))

plt.axhline(y=0, xmin=0.001, xmax=9999, linewidth=1, c ='gray')
plt.xlabel(r'$\alpha$')
plt.xscale('log')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.5),
          ncol=3, fancybox=True, shadow=True)
plt.show()
```



Grid Search for Ridge

In [112]:
```
from sklearn.model_selection import GridSearchCV
params_Ridge = {'alpha': [1,0.1,0.01,0.001,0.0001,0] , "fit_intercept": [True, Fal
se], "solver": ['svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']}
Ridge_GS = GridSearchCV(ridge, param_grid=params_Ridge, n_jobs=-1)
Ridge_GS.fit(X_train,y_train)
Ridge_GS.best_params_
```

Out[112]:
```
{'alpha': 1, 'fit_intercept': True, 'solver': 'svd'}
```

In [113]:
```
Ridge_GS.best_score_
```

Out[113]:
```
0.8758306176829231
```

Fitting model with best parameter

In [114]:
```
ridge = Ridge(alpha = 1,fit_intercept= True, solver ='svd')
ridge.fit(X_train,y_train)
print('Train score: {:.4f}'.format(ridge.score(X_train,y_train)))
print('Test score: {:.4f}'.format(ridge.score(X_test, y_test)))
```

```
Train score: 0.8917
Test score: 0.8636
```

In [115]:
```
Ridgeregression = Ridge(random_state=3, **Ridge_GS.best_params_)
from sklearn.model_selection import cross_val_score
all_accuracies = cross_val_score(estimator=Ridgeregression, X=X_train, y=y_train,
cv=5)
all_accuracies
```

Out[115]:
```
array([0.84377403, 0.88864518, 0.86697679, 0.89098162, 0.89754775])
```

In [116]:
```
print(all_accuracies.mean())
```

```
0.8775850742098846
```

Ridge Cross validation

In [117]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(Ridgeregression,X_train,y_train,cv= 5)
test_score_list = cross_val_score(Ridgeregression,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8776
Avg Test Score:0.8259
```

Ridge Summary

1) For Ridge model : Train score: 0.8917 and Test score: 0.8636

2) The best parameters using grid search were:alpha = 1, fit_intercept = True, solver = svd

3) The average train and test score using cross validation was: Avg Train Score:0.8776 and Avg Test Score:0.8259

Lasso

In [118]:

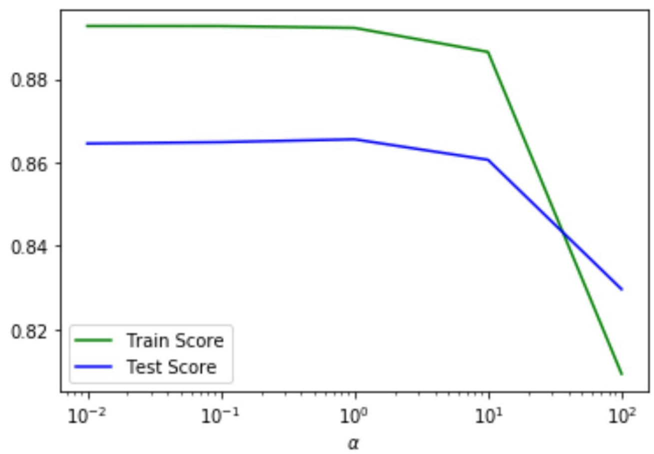
```
from sklearn.linear_model import Lasso
x_range = [0.01, 0.1, 1, 10, 100]
train_score_list = []
test_score_list = []

for alpha in x_range:
    lasso = Lasso(alpha)
    lasso.fit(X_train,y_train)
    train_score_list.append(lasso.score(X_train,y_train))
    test_score_list.append(lasso.score(X_test, y_test))
```

In [119]:

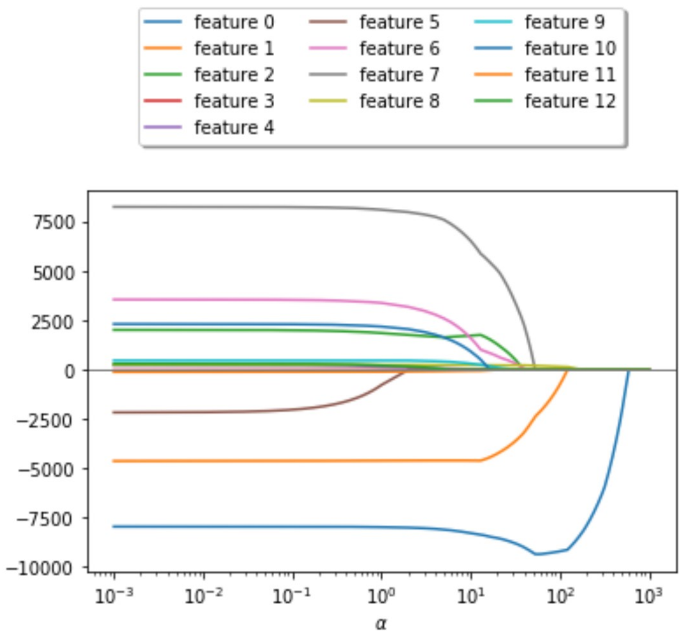
```
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

Out[119]:

```
Text(0.5, 0, '$\\alpha$')
```



In [120]:
```
%matplotlib inline

x_range1 = np.linspace(0.001, 1, 1000).reshape(-1,1)
x_range2 = np.linspace(1, 1000, 1000).reshape(-1,1)

x_range = np.append(x_range1, x_range2)
coeff = []

for alpha in x_range:
    lasso = Lasso(alpha)
    lasso.fit(X_train,y_train)
    coeff.append(lasso.coef_ )

coeff = np.array(coeff)

for i in range(0,13):
    plt.plot(x_range, coeff[:,i], label = 'feature {:d}'.format(i))

plt.axhline(y=0, xmin=0.001, xmax=9999, linewidth=1, c ='gray')
plt.xlabel(r'$\alpha$')
plt.xscale('log')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.5),
          ncol=3, fancybox=True, shadow=True)
plt.show()
```



Lasso Gridsearch

In [121]:
```
from sklearn.linear_model import Lasso
lasso = Lasso()
from sklearn.model_selection import GridSearchCV
params_Lasso = {'alpha': [1,0.1,0.01,0.001,0.0001,0] , "fit_intercept": [True, Fal
se]}
Lasso_GS = GridSearchCV(lasso, param_grid=params_Lasso, n_jobs=-1)
Lasso_GS.fit(X_train,y_train)
Lasso_GS.best_params_
```

Out[121]:
```
{'alpha': 1, 'fit_intercept': True}
```

In [122]:
```
Lasso_GS.best_score_
```
Out[122]:
```
0.8750388756806711
```

In [123]:
```
lasso =Lasso(alpha = 1, fit_intercept =True)
lasso.fit(X_train, y_train)
print('Train score: {:.4f}'.format(lasso.score(X_train,y_train)))
print('Test score: {:.4f}'.format(lasso.score(X_test, y_test)))
```
```
Train score: 0.8923
Test score: 0.8656
```

Lasso Cross Validation

In [124]:
```
Lassoregression = Lasso(random_state=3, **Lasso_GS.best_params_)
from sklearn.model_selection import cross_val_score
all_accuracies = cross_val_score(estimator=Lassoregression, X=X_train, y=y_train,
cv=5)
all_accuracies
```
Out[124]:
```
array([0.84630185, 0.89175263, 0.85085261, 0.89040334, 0.89912033])
```

In [125]:
```
print(all_accuracies.mean())
```
```
0.8756861491728142
```

In [126]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(Lassoregression,X_train,y_train,cv= 5)
test_score_list = cross_val_score(Lassoregression,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```
```
Avg Train Score:0.8757
Avg Test Score:0.8407
```

Lasso Summary

1) The train and test score of the model was: Train score: 0.8923 and Test score: 0.8656

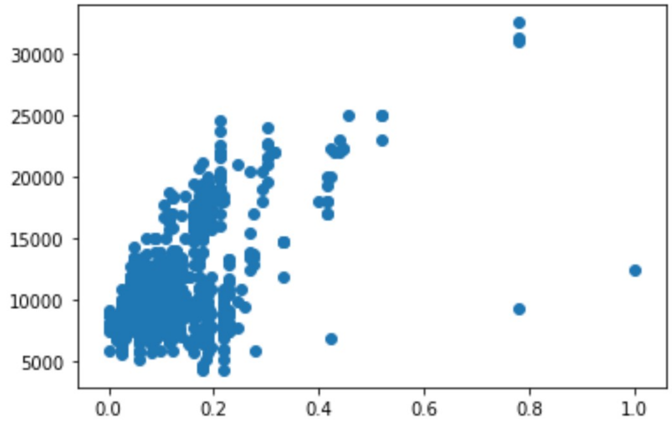
2) The best parameters using grid search was: alpha = 1, fit_intercept = True

3) The average train and test scores using cross validation was:Avg Train Score:0.8757 and Avg Test Score:0.8407

Polynomial Regressor

In [127]:
```
from sklearn.preprocessing import PolynomialFeatures

X_train_1 = X_train[:,7].reshape(-1,1)   #here is was[:,5] idk why
plt.scatter(X_train_1,y_train)
```

Out[127]: <matplotlib.collections.PathCollection at 0x2875e392908>



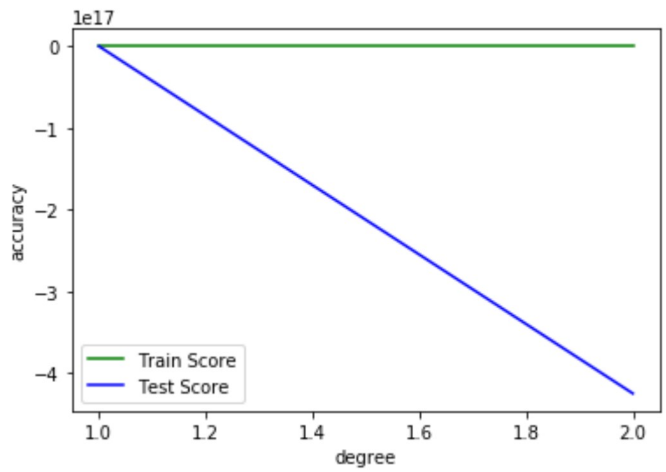
In [128]:
```
train_score_list = []
test_score_list = []

for n in range(1,3):
    poly = PolynomialFeatures(n)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)
    lreg.fit(X_train_poly, y_train)
    train_score_list.append(lreg.score(X_train_poly, y_train))
    test_score_list.append(lreg.score(X_test_poly, y_test))
```

In [129]:
```
print(train_score_list)
print(test_score_list)
```

```
[0.892769400855392, 0.9693234249393261]
[0.8645412404946087, -4.248193992374181e+17]
```

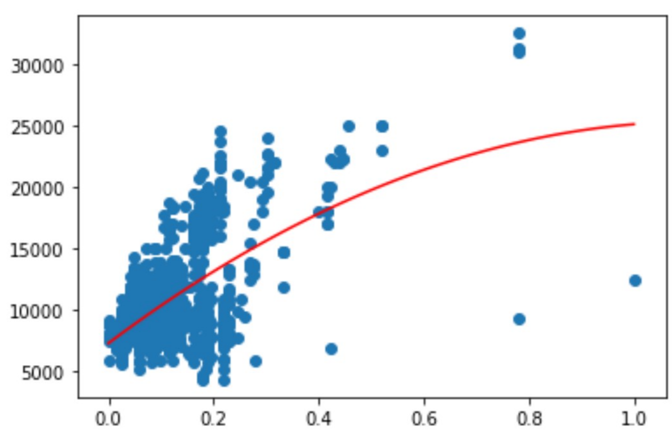
In [130]:
```
%matplotlib inline

x_axis = range(1,3)
plt.plot(x_axis, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel('degree')
plt.ylabel('accuracy')
plt.legend()
```

Out[130]: <matplotlib.legend.Legend at 0x2875c69b288>



In [131]:
```
poly = PolynomialFeatures(n)
X_train_poly = poly.fit_transform(X_train_1)
lreg.fit(X_train_poly, y_train)

x_axis = np.linspace(0,1,100).reshape(-1,1)
x_poly = poly.transform(x_axis)
y_predict = lreg.predict(x_poly)

X_train_1 = X_train[:,7].reshape(-1,1)
plt.scatter(X_train_1,y_train)
plt.plot(x_axis, y_predict, c = 'r')
```

Out[131]: [<matplotlib.lines.Line2D at 0x2875e44b588>]



In [132]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline

def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree), LinearRegression(**kwargs))

param_grid = {'polynomialfeatures__degree': np.arange(5), 'linearregression__fit_intercept': [True, False], 'linearregression__normalize': [True, False]}

poly_grid = GridSearchCV(PolynomialRegression(), param_grid, cv=10, scoring='neg_mean_squared_error')
```

In [133]:
```
# poly_grid.fit(X_train, y_train)
# Stopping the model here as the kernel is breaking after executing for 3+ hours
```

Polynomial Summary

- Did not work

SVR Regressor

In [134]:
```
from sklearn.svm import SVR
regressor = SVR(kernel ='linear')
regressor.fit(X_train,y_train)
print('Train score: {:.4f}'.format(regressor.score(X_train, y_train)))
print('Test score: {:.4f}'.format(regressor.score(X_test, y_test)))
```

```
Train score: 0.0951
Test score: 0.1214
```

SVR Gridsearch

In [135]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'C': 10. ** np.arange(-2, 5), 'epsilon': [0.1,0.2,0.3,0.4,0.5]}]
grid_search = GridSearchCV(estimator = regressor,param_grid = parameters)
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Out[135]:
```
{'C': 10000.0, 'epsilon': 0.1}
```

Fitting SVR with gridsearch parameters

In [136]:
```
from sklearn.svm import SVR
regressor = SVR(kernel ='linear', C=10000, epsilon=0.1)
regressor.fit(X_train,y_train)
print('Train score: {:.4f}'.format(regressor.score(X_train, y_train)))
print('Test score: {:.4f}'.format(regressor.score(X_test, y_test)))
```

```
Train score: 0.8873
Test score: 0.8722
```

Cross validation

In [137]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(regressor,X_train,y_train,cv=5)
test_score_list = cross_val_score(regressor,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8757
Avg Test Score:0.8494
```

SVR Kernel =linear Summary

1) The train and test score was:Train score: 0.0951 and Test score: 0.1214 which is an underfitting model

2) By Grid search, the best parameter is C =10000.0 and epsilon = 0.1

3) Then the train and test score was: Train score: 0.8873 and Test score: 0.8722

4) Using Cross validation, the average train and test score is Avg Train Score:0.8757 and Avg Test Score:0.8494

SVR rbf

In [138]:
```
from sklearn.svm import SVR
regressor = SVR(kernel ='rbf')
regressor.fit(X_train,y_train)
print('Train score: {:.4f}'.format(regressor.score(X_train, y_train)))
print('Test score: {:.4f}'.format(regressor.score(X_test, y_test)))
```

```
Train score: -0.0485
Test score: -0.0212
```

SVR Gridsearch

In [139]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'C': 10. ** np.arange(-2, 5), 'gamma': 10. ** np.arange(-5, 4)}]
grid_search = GridSearchCV(estimator = regressor,param_grid = parameters)
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Out[139]:
```
{'C': 10000.0, 'gamma': 0.1}
```

In [140]:
```
from sklearn.svm import SVR
regressor = SVR(kernel ='rbf', C=10000, gamma=0.1)
regressor.fit(X_train,y_train)
print('Train score: {:.4f}'.format(regressor.score(X_train, y_train)))
print('Test score: {:.4f}'.format(regressor.score(X_test, y_test)))
```

```
Train score: 0.9397
Test score: 0.8557
```

SVR Cross validation

In [141]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(regressor,X_train,y_train,cv=5)
test_score_list = cross_val_score(regressor,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8802
Avg Test Score:0.7535
```

SVR Kernel=Rbf Summary

1) Using the grid Search the best parameters for Rbf kernel is:C= 10000.0 and gamma= 0.1

2) The train and test score for the model is Train score: 0.9397 and Test score: 0.8557

3) Using Cross Validation, the average train and test score is Avg Train Score:0.8802 and Avg Test Score:0.7535

SVR Poly

In [142]:
```
from sklearn.svm import SVR
regressor = SVR(kernel ='poly')
regressor.fit(X_train,y_train)
print('Train score: {:.4f}'.format(regressor.score(X_train, y_train)))
print('Test score: {:.4f}'.format(regressor.score(X_test, y_test)))
```

```
Train score: -0.0560
Test score: -0.0288
```

Gridsearch

In [143]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'C': 10. ** np.arange(-3, 6), 'degree':[0,1,2,3,4,5,6]}]
grid_search = GridSearchCV(estimator = regressor,param_grid = parameters)
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Out[143]:
```
{'C': 100000.0, 'degree': 2}
```

Fitting with best parameters

In [144]:
```
from sklearn.svm import SVR
regressor = SVR(kernel ='poly', C=100000,degree=2)
regressor.fit(X_train,y_train)
print('Train score: {:.4f}'.format(regressor.score(X_train, y_train)))
print('Test score: {:.4f}'.format(regressor.score(X_test, y_test)))
```

```
Train score: 0.9240
Test score: 0.8490
```

Cross Validation

In [145]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(regressor,X_train,y_train,cv=5)
test_score_list = cross_val_score(regressor,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8847
Avg Test Score:0.7635
```

SVR Kernel=Poly Summary

1) Using the grid Search the best parameters for Poly kernel is: C= 100000.0 and degree= 2

2) The train and test score for the model is Train score: Train score: 0.9240 and Test score: 0.8490

3) Using Cross Validation, the average train and test score is Avg Train Score:0.8847 and Avg Test Score:0.7635

SVR Gridsearch

In [146]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'kernel': ['linear'], 'C': 10. ** np.arange(-2, 5), 'epsilon': [0.
1,0.2,0.3,0.4,0.5]},
              {'kernel': ['rbf'], 'C': 10. ** np.arange(-2, 5), 'gamma': 10. ** n
p.arange(-5, 4)},
             {'kernel': ['poly'], 'C': 10. ** np.arange(-2, 5),'degree': [0, 1, 2,
3, 4, 5, 6]}]
grid_search = GridSearchCV(estimator = regressor,param_grid = parameters)
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Out[146]:
```
{'C': 10000.0, 'epsilon': 0.1, 'kernel': 'linear'}
```

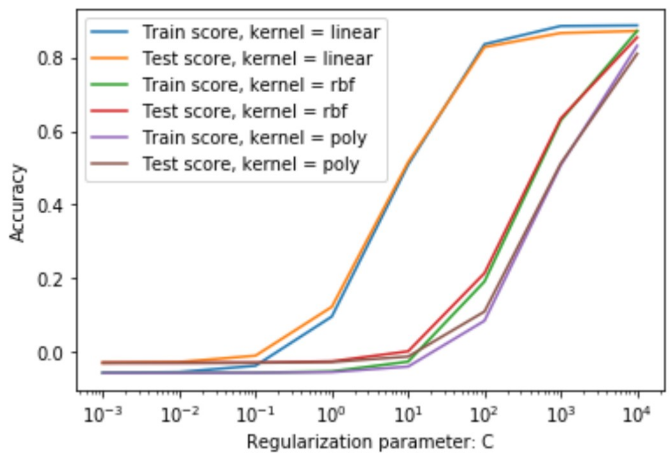
In [147]:
```
accuracy = grid_search.best_score_
accuracy
```

Out[147]:
```
0.8747485540122213
```

SVR Visualization

In [148]:
```
from sklearn.svm import SVR

c_range = [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
train_score_linear = []
train_score_rbf = []
train_score_poly = []
test_score_linear = []
test_score_rbf = []
test_score_poly = []

for c in c_range:
    linear = SVR(kernel = 'linear', C = c)
    rbf = SVR(kernel = 'rbf', C = c, gamma = 0.01)
    poly = SVR(kernel = 'poly', C = c)
    linear.fit(X_train, y_train)
    rbf.fit(X_train, y_train)
    poly.fit(X_train, y_train)
    train_score_linear.append(linear.score(X_train, y_train))
    train_score_rbf.append(rbf.score(X_train, y_train))
    train_score_poly.append(poly.score(X_train, y_train))
    test_score_linear.append(linear.score(X_test, y_test))
    test_score_rbf.append(rbf.score(X_test, y_test))
    test_score_poly.append(poly.score(X_test, y_test))
```

In [149]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(c_range, train_score_linear, label = 'Train score, kernel = linear')
plt.plot(c_range, test_score_linear, label = 'Test score, kernel = linear')
plt.plot(c_range, train_score_rbf, label = 'Train score, kernel = rbf')
plt.plot(c_range, test_score_rbf, label = 'Test score, kernel = rbf')
plt.plot(c_range, train_score_poly, label = 'Train score, kernel = poly')
plt.plot(c_range, test_score_poly, label = 'Test score, kernel = poly')
plt.legend()
plt.xlabel('Regularization parameter: C')
plt.ylabel('Accuracy')
plt.xscale('log')
```



In [150]:
```
from sklearn.svm import SVR
regressor = SVR(kernel = 'linear', C = 10000)
regressor.fit(X_train, y_train)
print('Train score: {:.4f}'.format(regressor.score(X_train, y_train)))
print('Test score: {:.4f}'.format(regressor.score(X_test, y_test)))
```

```
Train score: 0.8873
Test score: 0.8722
```

SVR Summary

1) Out of linear, rbf, poly svr regressor, linear regressor works best.

2) The best parameters were C = 10000.0 and epsilon= 0.1

3) Accuracy found is Train score: 0.8873 and Test score: 0.8722

SVR-Simple

In [151]:
```
from sklearn.svm import LinearSVR
linsvr = LinearSVR()
linsvr.fit(X_train, y_train)
```

Out[151]:
```
LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True,
          intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000,
          random_state=None, tol=0.0001, verbose=0)
```

In [152]:
```
print('Train score: {:.4f}'.format(linsvr.score(X_train, y_train)))
print('Test score: {:.4f}'.format
      (linsvr.score(X_test, y_test)))
```

```
Train score: -0.3492
Test score: -0.4876
```

Grid Search

In [153]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'C': 10. ** np.arange(-3, 6), 'epsilon':[0.1,0.2,0.3,0.4,0.5,0.6],
'fit_intercept':[True, False]}]
grid_search = GridSearchCV(estimator = linsvr,param_grid = parameters)
grid_search = grid_search.fit(X_train, y_train)
grid_search.best_params_
```

Out[153]:
```
{'C': 10000.0, 'epsilon': 0.1, 'fit_intercept': True}
```

Fitting with best parameters

In [154]:
```
linsvr = LinearSVR(C = 10000.0, epsilon = 0.1, fit_intercept = True)
linsvr.fit(X_train, y_train)
print('Train score: {:.4f}'.format(linsvr.score(X_train, y_train)))
print('Test score: {:.4f}'.format(linsvr.score(X_test, y_test)))
```

```
Train score: 0.8840
Test score: 0.8711
```

Cross Validation

In [155]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(linsvr,X_train,y_train,cv=5)
test_score_list = cross_val_score(linsvr,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8745
Avg Test Score:0.8521
```

Linear SVR Summary

1) The train and test score of the model is Train score: -0.3486 and Test score: -0.4868

2) Using Grid search, the best parameters are C = 10000.0, epsilon= 0.1 and fit_intercept = True

3) Then the train and test score improved to: Train score: 0.8863 and Test score: 0.8714

4) Using Cross Validation, the average train and test score is Avg Train Score:0.8745 and Avg Test Score:0.8521

Decision Tree Regressor

In [156]:
```
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(X_train, y_train)
print('Train score: {:.4f}'.format(regressor.score(X_train, y_train)))
print('Test score: {:.4f}'.format(regressor.score(X_test, y_test)))
```

```
Train score: 1.0000
Test score: 0.6518
```

It is an overfitting model

Gridsearch

In [157]:
```
from sklearn.model_selection import GridSearchCV, cross_val_score, cross_val_predi
ct
param_grid = {"criterion": ["mse", "mae"],
              "min_samples_split": [10, 20, 40],
              "max_depth": [2, 6, 8],
              "min_samples_leaf": [20, 40, 100],
              "max_leaf_nodes": [5, 20, 100],
              }

grid_cv_dtm = GridSearchCV(regressor, param_grid, cv=5)

grid_cv_dtm.fit(X_train,y_train)
print("Best Hyperparameters::\n{}".format(grid_cv_dtm.best_params_))
```

```
Best Hyperparameters::
{'criterion': 'mse', 'max_depth': 8, 'max_leaf_nodes': 100, 'min_samples_leaf':
20, 'min_samples_split': 10}
```

In [158]:
```
accuracy = grid_cv_dtm.best_score_
accuracy
```

Out[158]: 0.8357723570722227

Fitting with best parameters

In [159]:
```
regressor = DecisionTreeRegressor(criterion = 'mse', max_depth = 8, max_leaf_nodes
= 100, min_samples_leaf = 20, min_samples_split = 10)

# fit the regressor with X and Y data
regressor.fit(X_train, y_train)
print('Train score: {:.4f}'.format(regressor.score(X_train, y_train)))
print('Test score: {:.4f}'.format(regressor.score(X_test, y_test)))
```

```
Train score: 0.8807
Test score: 0.8487
```

Cross validation

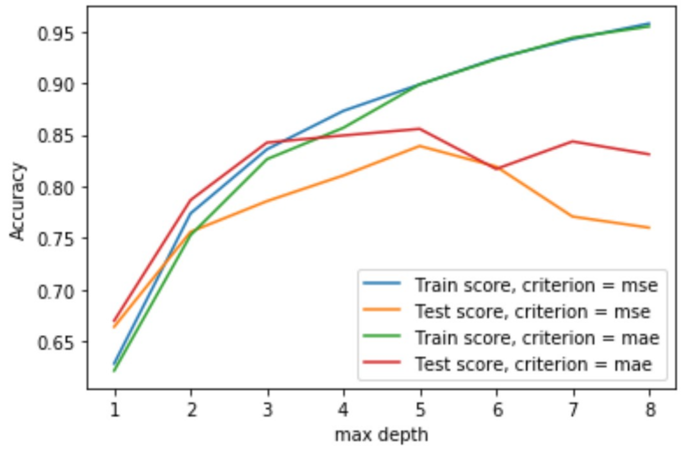
In [160]:
```
from sklearn.model_selection import cross_val_score
train_score_list = cross_val_score(regressor,X_train,y_train,cv=5)
test_score_list = cross_val_score(regressor,X_test,y_test,cv=5)
print("Avg Train Score:%.4f"%train_score_list.mean())
print("Avg Test Score:%.4f"%test_score_list.mean())
```

```
Avg Train Score:0.8358
Avg Test Score:0.7838
```

In [161]:
```
from sklearn.tree import DecisionTreeRegressor

depth = [1,2,3,4,5,6,7,8]
train_score_g = []
train_score_e = []
test_score_g = []
test_score_e = []

for d in depth:
    dec_g = DecisionTreeRegressor(criterion = 'mse', max_depth = d)
    dec_e = DecisionTreeRegressor(criterion = 'mae', max_depth = d)
    dec_g.fit(X_train, y_train)
    dec_e.fit(X_train, y_train)
    train_score_g.append(dec_g.score(X_train, y_train))
    train_score_e.append(dec_e.score(X_train, y_train))
    test_score_g.append(dec_g.score(X_test, y_test))
    test_score_e.append(dec_e.score(X_test, y_test))
```

In [162]:
```
x_axis = range(1,9)
%matplotlib inline
plt.plot(x_axis, train_score_g, label = 'Train score, criterion = mse')
plt.plot(x_axis, test_score_g, label = 'Test score, criterion = mse')
plt.plot(x_axis, train_score_e, label = 'Train score, criterion = mae')
plt.plot(x_axis, test_score_e, label = 'Test score, criterion = mae')
plt.legend()
plt.xlabel('max depth')
plt.ylabel('Accuracy')
```

Out[162]:
```
Text(0, 0.5, 'Accuracy')
```



Decision Tree Summary

1) The train and test score for the Decison tree was Train score: 1.0000 and Test score: 0.6518 which is overfitting.

2) After applying grid search, we found out the best parameters: criterion = 'mse', max_depth = 8, max_leaf_nodes = 100, min_samples_leaf = 20, min_samples_split = 10

3) The train and test score for this hyperparameter is Train score: 0.8807 and Test score: 0.8487.

4) Using Cross Validation, the average train and test score is Avg Train Score:0.8356 and Avg Test Score:0.7838

Predicting test dataset prices from the best model

In [163]:
```
model = LinearSVR(C = 10000.0, epsilon = 0.1, fit_intercept = True)
model.fit(X_train, y_train)
pred_test_SVR= model.predict(X_test)
```

In [164]:
```
pred_test_SVR
```

```
Out[164]: array([ 8672.90099104, 10003.06356697, 15375.94468695,  8519.82419381,
                  9062.88275255,  8827.63426372,  8657.43673579,  8297.79106632,
                 10746.91918491,  9044.86407593,  9921.54696596,  6839.43383946,
                  9243.64232952,  9493.66363785,  7977.04177001,  7126.1010759 ,
                 10160.82994359,  9549.99225781,  7722.14794521, 13180.58754038,
                  7800.88695023, 12001.44333266,  9332.43990355,  8528.06750345,
                  9827.72525329, 16800.85563512, 14983.19989784,  7596.44608694,
                  8562.82436393,  6980.22398361, 10305.37656809,  8169.72292226,
                  9048.2599994 , 10742.97377666, 10685.21046345,  8705.79240432,
                 15326.13385719,  8654.14982479, 10456.49497389,  8952.22741716,
                 10706.39635161,  8608.14410151,  8000.5683134 ,  8339.34749436,
                 10391.45326179, 12847.88112585, 10101.65493317, 11130.71728722,
                  7827.40060399,  9977.91823687,  7263.43015429, 10829.63450726,
                  7478.47860552,  9467.02532025, 11129.30282859, 11699.80831269,
                  7878.47906813,  8252.81810846, 17470.35021814,  8840.92228258,
                  8739.63041165,  9984.8372587 ,  8661.47768686,  7255.1810711 ,
                  9934.34591167, 10409.62177955,  8516.3054226 , 10053.13887058,
                 10259.9734165 ,  8291.53182935, 20211.7558513 ,  7969.00375067,
                  6899.94653701,  7144.04239382, 10200.31468255,  6739.4650028 ,
                  6287.52925238,  8314.89228153,  9744.74918325,  8079.46137693,
                  9577.09714898, 10712.60989432, 12202.75011345, 11290.76794072,
                 10204.04426749,  5949.21306159,  9164.4803324 , 10008.45732686,
                  9581.91826513,  9828.3687554 ,  6015.53562728,  7074.83984687,
                  9696.81749422,  7193.53393603,  9137.89976714, 10108.09805468,
                  9944.96444414, 15630.70649506,  9836.95796055,  6373.8158604 ,
                  8721.46835563,  6798.95035517,  6872.36744476,  9732.85131375,
                  9159.66565976,  8620.39414648,  9200.46680757,  7967.79953862,
                 12499.53676565, 10986.73168242,  8979.94856449, 11027.8833848 ,
                 11993.28092529, 11178.75817581,  8696.22022016,  9686.17806327,
                  7314.5047692 ,  9974.27647789,  9658.39333434, 19193.3913887 ,
                  8377.5705622 , 13300.54114153,  9213.78913403,  7773.15852168,
                  9698.30947171, 15009.64956905,  9337.04587539,  7190.09762615,
                  8443.5421227 ,  8507.84245429, 11334.24299019, 10675.28824686,
                 10868.67678848, 11522.05968178,  9650.72379477, 10716.9803321 ,
                 12201.64726281, 12071.28143956, 11319.95694253, 11311.62558976,
                  7973.92262825,  9975.34852637, 10993.61409345, 10146.35914493,
                 18981.05403943,  8474.21599922, 17835.980098  ,  8719.78891737,
                  9887.62530886, 10320.28052911, 19281.61891888,  7527.82779775,
                  9734.22606688,  8572.74186133,  8645.76289501, 10071.81667788,
                 12527.00793778,  6378.24306924, 11265.6332674 , 11618.0337953 ,
                 11823.1998558 , 11076.69697005,  6018.79178955,  8220.06760267,
                  9118.04290496, 18455.36135975, 16914.4657637 ,  5915.22960379,
                 10349.38030059,  7854.75779505,  8542.05786757,  9880.65344787,
                  9935.40598999,  8219.53169591, 10882.54468864,  9474.3170416 ,
                  8208.8343527 , 11822.8985683 ,  7198.47186148,  7756.75309818,
                  8512.75690891,  7347.55983732, 17278.36628   , 19745.93583241,
                  7909.56680535, 21867.88915236, 12808.95764123,  8653.26074404,
                  8110.93983254,  7654.72286102, 15352.98523673,  8840.21043829,
                 15048.76279281, 13919.50570396,  9547.513902  , 11262.62627829,
                  7690.28050224,  7768.46650258,  8909.28489042, 12199.63873789,
                 12230.04127277,  8121.43673442, 10918.4923334 ,  9339.994926  ,
                  6211.99448232, 11715.40345212, 10507.75440377, 11009.35068412,
                 10802.27788308,  9378.85992801, 10348.8701091 ,  8170.94936889,
                 10441.16678069, 12382.67153532,  7106.73327873, 11435.64715689,
                 10453.89465355, 10415.84237173,  7934.20321997, 12068.70212851,
                 14059.66896506, 12770.93623   , 12225.61626377,  9602.0925173 ,
                  8697.24236587, 20526.58569521, 12341.26352902,  8405.68083192,
                 15500.72865272, 10483.24194925, 10916.54408094, 10720.09091975,
                 19898.91382552,  9999.68919917, 11129.92653371, 18399.47421845,
                 13596.60401992, 11698.23280886, 18920.22329007,  8870.74098028,
                 10580.50369335,  8103.94194021, 20154.29679641, 10590.56050676,
                 16169.20444028,  7183.85574431, 12731.05978769, 11166.43297692,
                 21110.21416312,  8828.81557403,  7571.79609845,  8759.19855221,
                  8348.06201177, 10950.59865528,  9346.84203217,  8543.33631811,
```

Models Summary

In [165]:

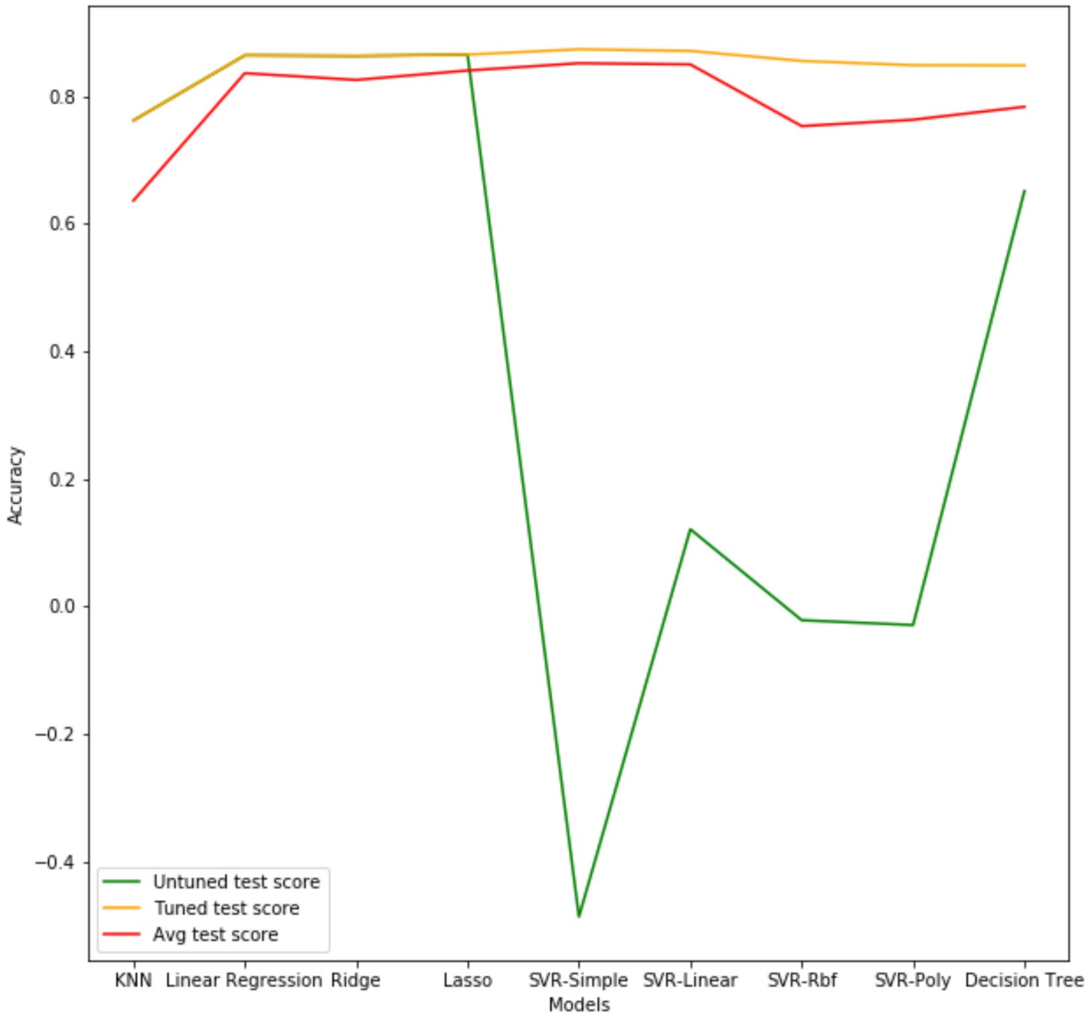
```
Models= {'Models':['KNN','Linear Regression','Ridge', 'Lasso','SVR-Simple','SVR-Linear','SVR-Rbf','SVR-Poly','Decision Tree'], 'Without_Hyperparameter_Tuning':[0.7624,0.8645,0.8636,0.8656,-0.4860,0.1214,-0.0212,-0.0288,0.6518],
         'With_Hyperparameter_Tuning': [0.7624,0.8645,0.8636,0.8656,0.8740,0.8714,0.8557,0.8490,0.8487], 'Average Test score': [0.6367,0.8365,0.8259,0.8407,0.8521,0.8503,0.7535,0.7635,0.7838]}
Models_scores = pd.DataFrame(Models)
Models_scores
```

Out[165]:

	Models	Without_Hyperparameter_Tuning	With_Hyperparameter_Tuning	Average Test score
0	KNN	0.7624	0.7624	0.6367
1	Linear Regression	0.8645	0.8645	0.8365
2	Ridge	0.8636	0.8636	0.8259
3	Lasso	0.8656	0.8656	0.8407
4	SVR-Simple	-0.4860	0.8740	0.8521
5	SVR-Linear	0.1214	0.8714	0.8503
6	SVR-Rbf	-0.0212	0.8557	0.7535
7	SVR-Poly	-0.0288	0.8490	0.7635
8	Decision Tree	0.6518	0.8487	0.7838

In [166]:
```
import matplotlib.pyplot as plt
Models = ['KNN','Linear Regression','Ridge', 'Lasso','SVR-Simple','SVR-Linear','SV
R-Rbf','SVR-Poly','Decision Tree']
Without_Hyperparameter = [0.7624,0.8645,0.8636,0.8656,-0.4860,0.1214,-0.0212,-0.02
88,0.6518]
With_Hyperparameter = [0.7624,0.8645,0.8636,0.8656,0.8740,0.8714,0.8557,0.8490,0.8
487]
Average_Test_score = [0.6367,0.8365,0.8259,0.8407,0.8521,0.8503,0.7535,0.7635,0.78
38]

fig=plt.figure(figsize=(10,10))
plt.plot(Models,Without_Hyperparameter, label = 'Untuned test score', color='g')
plt.plot(Models, With_Hyperparameter, label = 'Tuned test score', color='orange')
plt.plot(Models, Average_Test_score, label = 'Avg test score', color='red')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.legend()
```

Out[166]: <matplotlib.legend.Legend at 0x2875c51bb08>



1) Out of all the models, the SVR-Simple has the highest accuracy with SVR-linear kernel being next.

2) The average test accuracy of SVR-Simple is also better than SVR-linear kernel, so, SVR-Simple regression is the best model

In [ ]: