In the notebook I will show how I iterate over multiple algorithms utilizing python's sklearn library. In the "Basic" section, I will iterate though each algorithm and nothing more. Afterwards I will scale the data and then apply principle component analysis.

In [1]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import SGDRegressor


from sklearn.model_selection import train_test_split

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
```

In [2]:
```
df1 = pd.read_csv('boston_housing.csv', usecols = ['CRIM', 'ZN', 'INDUS',
                                                   'RM', 'AGE', 'DIS', 'RA
                                                   'B', 'LSTAT', 'target']
```

In [3]:
```
df1.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.9
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.1
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.0
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.9
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.3

In [4]:
```
X = df1.drop(columns = ['target'])
y = df1['target']
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
```

Basic

In [6]:
```
models = {
          'RandomForestRegressor' : RandomForestRegressor(max_features='sq
          'GradientBoostingRegressor' : GradientBoostingRegressor(),
          'LinearRegression' : LinearRegression(),
          'DecisionTreeRegressor' : DecisionTreeRegressor(),
          'SGDRegressor' : SGDRegressor(max_iter = 2000)
         }

hyperparameters = {
                   'RandomForestRegressor' :
                                      {'max_depth': [6, 8, 10, 12],
                                       'n_estimators' : [165, 175, 17
                   'GradientBoostingRegressor' :
                                     {"max_depth": [2, 4, 6, 8], "mi
                                       "learning_rate": [0.001, 0.01,
                   'LinearRegression' : {'fit_intercept' : [True, False],

                   'DecisionTreeRegressor' :
                                      {'max_depth': [2, 4, 6, 8], 'm

                   'SGDRegressor' : {'alpha' : [0.15, 0.25, 0.30, 0.35],
                                     'learning_rate' :['constant', 'optima
                                      'penalty': ['l2', 'l1', 'elasticnet'

                  }
```

In [7]:
```
print('Algorithm, best hyperparameters, r-squared:')
i = 0
while i < len(list(models.values())):

    grid_rf_regression = GridSearchCV(
    estimator = list(models.values())[i],
    param_grid = list(hyperparameters.values())[i],
    #scoring='neg_mean_squared_error',
    scoring='r2',
    n_jobs=4,
    cv = 3,
    refit=True,
    return_train_score=True)

    grid_rf_regression.fit(X_train, y_train)
    #display(grid_rf_regression.get_params())

    print(list(models.keys())[i].upper(),':',grid_rf_regression.best_param

    i += 1
```

```
Algorithm, best hyperparameters, r-squared:
RANDOMFORESTREGRESSOR : {'max_depth': 12, 'min_samples_leaf': 1, 'n_estimator
s': 180} 0.8247038461819008
GRADIENTBOOSTINGREGRESSOR : {'learning_rate': 0.1, 'max_depth': 4, 'min_sampl
es_split': 3} 0.8431710173025984
LINEARREGRESSION : {'fit_intercept': True, 'positive': False} 0.6864013112944
148
DECISIONTREEREGRESSOR : {'max_depth': 8, 'min_samples_leaf': 2} 0.70146898777
66709
SGDREGRESSOR : {'alpha': 0.3, 'learning_rate': 'adaptive', 'penalty': 'l2'} -
4.5017376544305714e+21
```

In [8]:
```
#display(DecisionTreeRegressor().get_params().keys())
#display(DecisionTreeRegressor().get_params())
#display(grid_rf_regression.get_params())
```

Scaling

In [9]:

```
scaler = StandardScaler()
models = {
          'RandomForestRegressor' : Pipeline([('scaler', scaler),
                                              ('RandomForestRegressor', Ra
          'GradientBoostingRegressor' : Pipeline([('scaler', scaler),
                                                  ('GradientBoostingRegres
          'LinearRegression' : Pipeline([('scaler', scaler),
                                         ('LinearRegression', LinearRegres
          'DecisionTreeRegressor' : Pipeline([('scaler', scaler),
                                              ('DecisionTreeRegressor', De
          'SGDRegressor' :  Pipeline([('scaler', scaler),
                                      ('SGDRegressor', SGDRegressor())])
          }

hyperparameters = {
                   'RandomForestRegressor' :
                                        {'RandomForestRegressor__max_d
                                         'RandomForestRegressor__min_s
                                         'RandomForestRegressor__n_esti
                   'GradientBoostingRegressor' :
                                       {"GradientBoostingRegressor__ma
                                        "GradientBoostingRegressor__mi
                                        "GradientBoostingRegressor__le
                   'LinearRegression' :
                                     {'LinearRegression__fit_intercept'
                                       'LinearRegression__positive' : [
                   'DecisionTreeRegressor' :
                                        {'DecisionTreeRegressor__max_d
                                         'DecisionTreeRegressor__min_s
                   'SGDRegressor' :
                                    {'SGDRegressor__alpha' : [0.15, 0.25,
                                     'SGDRegressor__learning_rate' :['cons
                                      'SGDRegressor__penalty': ['l2', 'l1'
                    }
```

In [10]:
```
print('Algorithm, best hyperparameters, r-squared:')
i = 0
while i < len(list(models.values())):

    grid_rf_regression = GridSearchCV(
    estimator = list(models.values())[i],
    param_grid = list(hyperparameters.values())[i],
    #scoring='neg_mean_squared_error',
    scoring='r2',
    n_jobs=4,
    cv = 3,
    refit=True,
    return_train_score=True)

    grid_rf_regression.fit(X_train, y_train)
    #display(grid_rf_regression.get_params())

    print(list(models.keys())[i].upper(),':',grid_rf_regression.best_param
    print('  ')
    i += 1
```

```
Algorithm, best hyperparameters, r-squared:
RANDOMFORESTREGRESSOR : {'RandomForestRegressor__max_depth': 8, 'RandomForest
Regressor__min_samples_leaf': 1, 'RandomForestRegressor__n_estimators': 175}
0.826227448699833

GRADIENTBOOSTINGREGRESSOR : {'GradientBoostingRegressor__learning_rate': 0.1,
'GradientBoostingRegressor__max_depth': 4, 'GradientBoostingRegressor__min_sa
mples_split': 8} 0.843835347505636

LINEARREGRESSION : {'LinearRegression__fit_intercept': True, 'LinearRegressio
n__positive': False} 0.6864013112944152

DECISIONTREEREGRESSOR : {'DecisionTreeRegressor__max_depth': 8, 'DecisionTree
Regressor__min_samples_leaf': 1} 0.7417983886289493

SGDREGRESSOR : {'SGDRegressor__alpha': 0.35, 'SGDRegressor__learning_rate':
'adaptive', 'SGDRegressor__penalty': None} 0.6877920559477668
```

Scaler and Principle Component Analysis (PCA)

In [11]:

```
scaler = StandardScaler()
pca = PCA()

models = {
          'RandomForestRegressor' : Pipeline([('scaler', scaler),
                                              ('pca', pca),
                                              ('RandomForestRegressor', Ra
          'GradientBoostingRegressor' : Pipeline([('scaler', scaler),
                                                  ('pca', pca),
                                                  ('GradientBoostingRegres
          'LinearRegression' : Pipeline([('scaler', scaler),
                                         ('pca', pca),
                                         ('LinearRegression', LinearRegres
          'DecisionTreeRegressor' : Pipeline([('scaler', scaler),
                                              ('pca', pca),
                                              ('DecisionTreeRegressor', De
          'SGDRegressor' :  Pipeline([('scaler', scaler),
                                      ('pca', pca),
                                      ('SGDRegressor', SGDRegressor())]),
          }

hyperparameters = {
                   'RandomForestRegressor' :
                                             {
                                              'pca__n_components' : np.aran
                                              'RandomForestRegressor__max_d
                                              'RandomForestRegressor__min_s
                                              'RandomForestRegressor__n_est
                                             },
                   'GradientBoostingRegressor' :
                                            {'pca__n_components' : np.arang
                                             'GradientBoostingRegressor__ma
                                             'GradientBoostingRegressor__mi
                                             'GradientBoostingRegressor__le
                                            },
                   'LinearRegression' :
                                        {
                                          'pca__n_components' : np.arange(
                                          'LinearRegression__fit_intercept
                                          'LinearRegression__positive' : [
                                        },
                   'DecisionTreeRegressor' :
                                             {
                                              'pca__n_components' : np.aran
                                              'DecisionTreeRegressor__max_d
                                              'DecisionTreeRegressor__min_s
                                             },
                   'SGDRegressor' :
                                    {
                                     'pca__n_components' : np.arange(1, le
                                     'SGDRegressor__alpha' : [0.15, 0.25,
                                     'SGDRegressor__learning_rate' :['cons
                                     'SGDRegressor__penalty': ['l2', 'l1',
                                    }
```

```
55                          }
```

In [12]:

```
print('Algorithm, best hyperparameters, r-squared:')
i = 0
while i < len(list(models.values())):

    grid_rf_regression = GridSearchCV(
    estimator = list(models.values())[i],
    param_grid = list(hyperparameters.values())[i],
    #scoring='neg_mean_squared_error',
    scoring='r2',
    n_jobs=4,
    cv = 3,
    refit=True,
    return_train_score=True)

    grid_rf_regression.fit(X_train, y_train)
    #display(grid_rf_regression.get_params())

    print(list(models.keys())[i].upper(),':',grid_rf_regression.best_param
    print('  ')
    i += 1
```

```
Algorithm, best hyperparameters, r-squared:
RANDOMFORESTREGRESSOR : {'RandomForestRegressor__max_depth': 12, 'RandomFores
tRegressor__min_samples_leaf': 1, 'RandomForestRegressor__n_estimators': 165,
'pca__n_components': 12} 0.7396813003224555

GRADIENTBOOSTINGREGRESSOR : {'GradientBoostingRegressor__learning_rate': 0.1,
'GradientBoostingRegressor__max_depth': 4, 'GradientBoostingRegressor__min_sa
mples_split': 10, 'pca__n_components': 12} 0.7943943739600443

LINEARREGRESSION : {'LinearRegression__fit_intercept': True, 'LinearRegressio
n__positive': False, 'pca__n_components': 13} 0.6864013112944151

DECISIONTREEREGRESSOR : {'DecisionTreeRegressor__max_depth': 8, 'DecisionTree
Regressor__min_samples_leaf': 2, 'pca__n_components': 7} 0.5889123570638041

SGDREGRESSOR : {'SGDRegressor__alpha': 0.3, 'SGDRegressor__learning_rate': 'a
daptive', 'SGDRegressor__penalty': None, 'pca__n_components': 13} 0.687486882
3240415
```

The tree based estimators - xgbooster, (not in this notebook) decision tree, and random forest - often do not benefit from scaling and pca, which I why I prefer them. Nonetheless, if you wish you use an algorithm that requires scaling or pca, you can use the code in this notebook as a guide.

Below is an example of how I might instigate a

In [13]:
```
pipe = Pipeline([('scaler', StandardScaler()),
                 ('reducer', PCA(n_components=3)),
                 ('regressor', RandomForestRegressor())])
```

In [14]:
```
pipe.fit(X_train, y_train)
```

Out[14]:
```
Pipeline(steps=[('scaler', StandardScaler()), ('reducer', PCA(n_components=
3)),
                ('regressor', RandomForestRegressor())])
```

In [15]:
```
print(pipe.score(X_test, y_test))
```

```
0.7161943144071752
```

In [ ]:
```

```