Simultaneous Regression Run

Imports

In [ ]:
```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform as sp_randFloat
from scipy.stats import randint as sp_randInt
from sklearn.linear_model import SGDRegressor
from sklearn import neighbors
from sklearn.ensemble import RandomForestRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
```

Load and Describe Data

In [ ]:
```
def load_pp_data():
    csv_path = r"C:\Users\18123\OneDrive\Documents\IU Bloomington\Machine-Learning-Pro
    return pd.read_csv(csv_path)

pp = load_pp_data()
print(pp.describe())
```

```
                AT            V           AP           RH           PE
count  9568.000000  9568.000000  9568.000000  9568.000000  9568.000000
mean     19.651231    54.305804  1013.259078    73.308978   454.365009
std       7.452473    12.707893     5.938784    14.600269    17.066995
min       1.810000    25.360000   992.890000    25.560000   420.260000
25%      13.510000    41.740000  1009.100000    63.327500   439.750000
50%      20.345000    52.080000  1012.940000    74.975000   451.550000
75%      25.720000    66.540000  1017.260000    84.830000   468.430000
max      37.110000    81.560000  1033.300000   100.160000   495.760000
```

Split Train/Test Data

In [ ]:
```
pp["AT_cat"] = pd.cut(pp["AT"],bins=[0.,10.,20.,30.,np.inf],labels=[1,2,3,4])

split = StratifiedShuffleSplit(n_splits=1,test_size=0.2,random_state=42)
for train_index, test_index in split.split(pp,pp["AT_cat"]):
    train_set = pp.loc[train_index]
    test_set = pp.loc[test_index]

for set_ in(train_set,test_set):
    set_.drop("AT_cat",axis=1,inplace=True)
```

```
pptrain = train_set.copy()
pptest = test_set.copy()

pptrain_attrib = pptrain.drop("PE",axis=1)
pptrain_labels = pptrain["PE"].copy()
pptest_attrib = pptest.drop("PE",axis=1)
pptest_labels = pptest["PE"].copy()

scaler = StandardScaler()
scaler.fit_transform(pptrain_attrib)
```

Out[ ]:
```
array([[ 1.1978498 ,  0.96554795,  0.37377565, -2.67409022],
       [ 0.64009018, -1.03750958, -1.88469509, -2.35340963],
       [-1.82211612, -1.45609422, -0.36887464,  1.17611946],
       ...,
       [-1.07754063, -0.84989538,  0.57724148,  0.20454577],
       [-0.67971691, -0.96104497,  0.78748951,  0.87314098],
       [ 0.89545   ,  0.56351752, -0.13658448, -1.12107019]])
```

Dictionary of Regression Hyperparamter Ranges

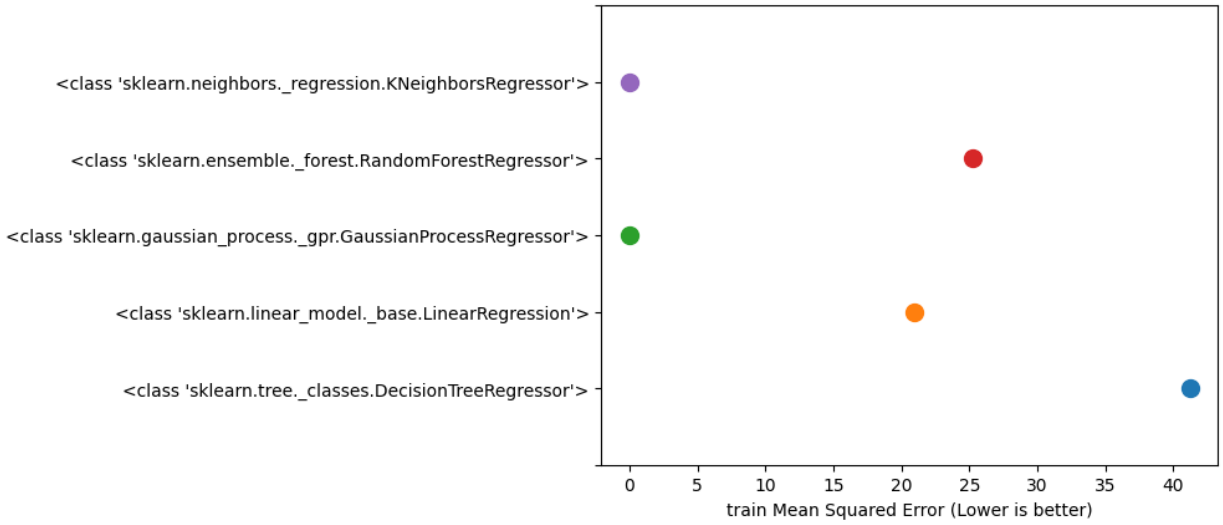
In [ ]:
```
regression_param_dict = {
DecisionTreeRegressor: [{'ccp_alpha': sp_randFloat(0,1), 'criterion': ['squared_error'
LinearRegression: [{'copy_X': [True], 'fit_intercept': [True, False], 'n_jobs': [None]
GaussianProcessRegressor: [{'alpha': sp_randFloat(1e-11,1e-9), 'copy_X_train': [True],
neighbors.KNeighborsRegressor: [{'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'
RandomForestRegressor: [{'bootstrap': [True,False], 'ccp_alpha': sp_randFloat(0,1), 'c
}
```

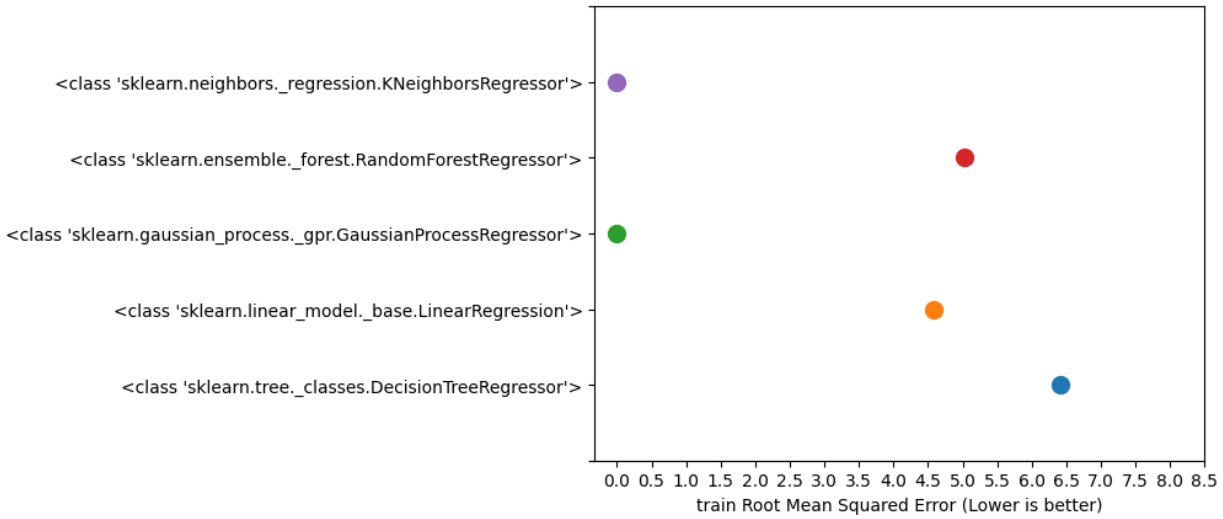
Function that Runs Regressions Simultaneously

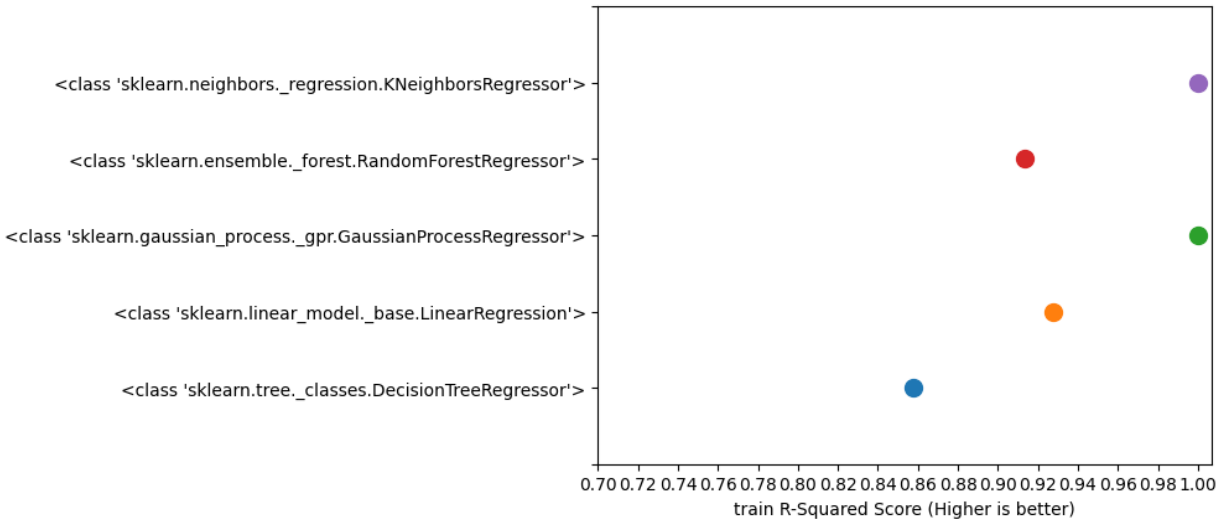
In [ ]:
```
def run(model):
    reg = model()
    param_grid = regression_param_dict[model]
    if model == LinearRegression:
        xiter = 4
    else:
        xiter = 10
    reg_hyper_search = RandomizedSearchCV(reg,param_grid,scoring="neg_mean_squared_err
    final_reg = reg_hyper_search.fit(pptrain_attrib,pptrain_labels)

    final_reg_train_predictions = final_reg.predict(pptrain_attrib)
    final_reg_train_mse = mean_squared_error(pptrain_labels,final_reg_train_prediction
    final_reg_train_rmse = np.sqrt(final_reg_train_mse)
    final_reg_train_r2 = r2_score(pptrain_labels,final_reg_train_predictions)
    final_reg_train_mae = mean_absolute_error(pptrain_labels,final_reg_train_predictic

    final_reg_test_predictions = final_reg.predict(pptest_attrib)

    final_reg_test_mse = mean_squared_error(pptest_labels,final_reg_test_predictions)
    final_reg_test_rmse = np.sqrt(final_reg_test_mse)
    final_reg_test_r2 = r2_score(pptest_labels,final_reg_test_predictions)
    final_reg_test_mae = mean_absolute_error(pptest_labels,final_reg_test_predictions)

    return [final_reg.best_params_, final_reg_train_mse, final_reg_train_rmse, final_r
```

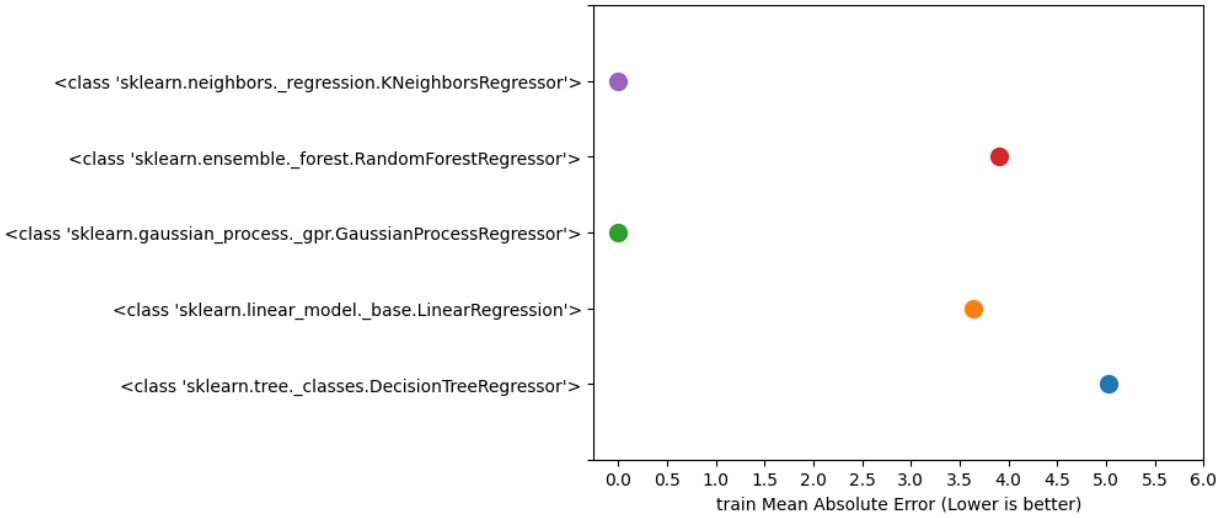
```python
def vizualizeMSE(regressions,modellst,data):
    MSEfig = plt.figure()
    for i in range(len(regressions)):
        if data == 'train':
            plt.scatter(regressions[i][1],i+1,s=100)
        if data == 'test':
            plt.scatter(regressions[i][5],i+1,s=100)
    plt.xticks([*range(0,45,5)])
    plt.yticks([*range(len(modellst)+2)],labels=['',*modellst,''])
    plt.xlabel(f'{data} Mean Squared Error (Lower is better)')
    return MSEfig

def vizualizeRMSE(regressions,modellst,data):
    RMSEfig = plt.figure()
    for i in range(len(regressions)):
        if data == 'train':
            plt.scatter(regressions[i][2],i+1,s=100)
        if data == 'test':
            plt.scatter(regressions[i][6],i+1,s=100)
    xticklst = []
    for j in range(0,90,5):
        xticklst += [j/10]
    plt.xticks(xticklst)
    plt.yticks([*range(len(modellst)+2)],labels=['',*modellst,''])
    plt.xlabel(f'{data} Root Mean Squared Error (Lower is better)')
    return RMSEfig

def vizualizeR2(regressions,modellst,data):
    R2fig = plt.figure()
    for i in range(len(regressions)):
        if data == 'train':
            plt.scatter(regressions[i][3],i+1,s=100)
        if data == 'test':
            plt.scatter(regressions[i][7],i+1,s=100)
    xticklst = []
    for j in range(70,102,2):
        xticklst += [j/100]
    plt.xticks(xticklst)
    plt.yticks([*range(len(modellst)+2)],labels=['',*modellst,''])
    plt.xlabel(f'{data} R-Squared Score (Higher is better)')
    return R2fig

def vizualizeMAE(regressions,modellst,data):
    MAEfig = plt.figure()
    for i in range(len(regressions)):
        if data == 'train':
            plt.scatter(regressions[i][4],i+1,s=100)
        if data == 'test':
            plt.scatter(regressions[i][8],i+1,s=100)
    xticklst = []
    for j in range(0,650,50):
        xticklst += [j/100]
    plt.xticks(xticklst)
    plt.yticks([*range(len(modellst)+2)],labels=['',*modellst,''])
    plt.xlabel(f'{data} Mean Absolute Error (Lower is better)')
```

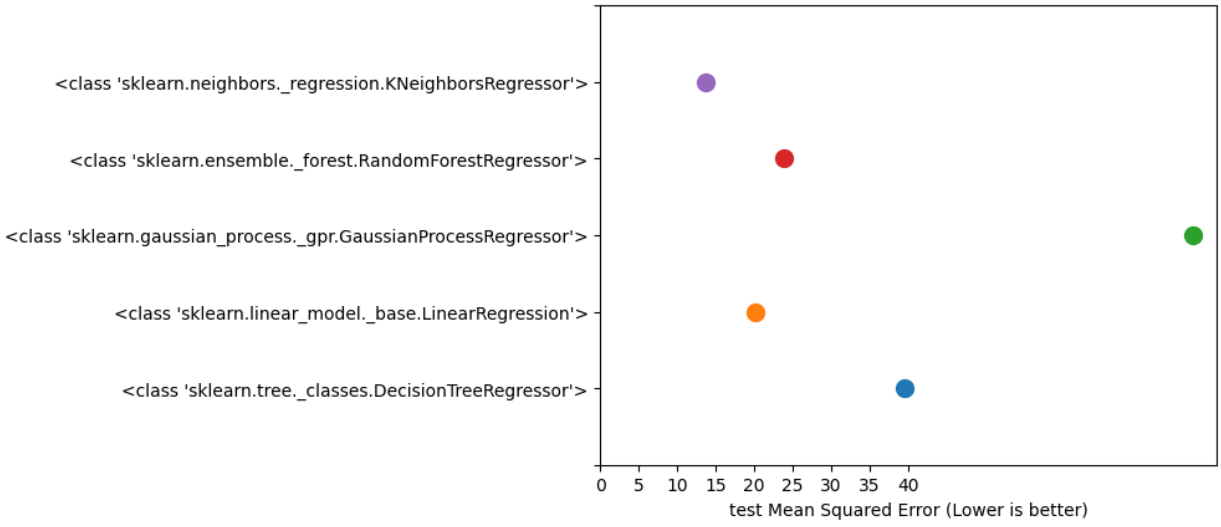
```
    return MAEfig

def comparison(modellst):
    regressions = []
    for i in modellst:
        regressions += [run(i)]
    trainmse = vizualizeMSE(regressions,modellst,'train')
    trainrmse = vizualizeRMSE(regressions,modellst,'train')
    trainr2 = vizualizeR2(regressions,modellst,'train')
    trainmae = vizualizeMAE(regressions,modellst,'train')
    testmse = vizualizeMSE(regressions,modellst,'test')
    testrmse = vizualizeRMSE(regressions,modellst,'test')
    testr2 = vizualizeR2(regressions,modellst,'test')
    testmae = vizualizeMAE(regressions,modellst,'test')
    return trainmse, trainrmse, trainr2, trainmae, testmse, testrmse, testr2, testmae

x = [DecisionTreeRegressor,LinearRegression,GaussianProcessRegressor,RandomForestRegre
comparison(x)
plt.show()
```

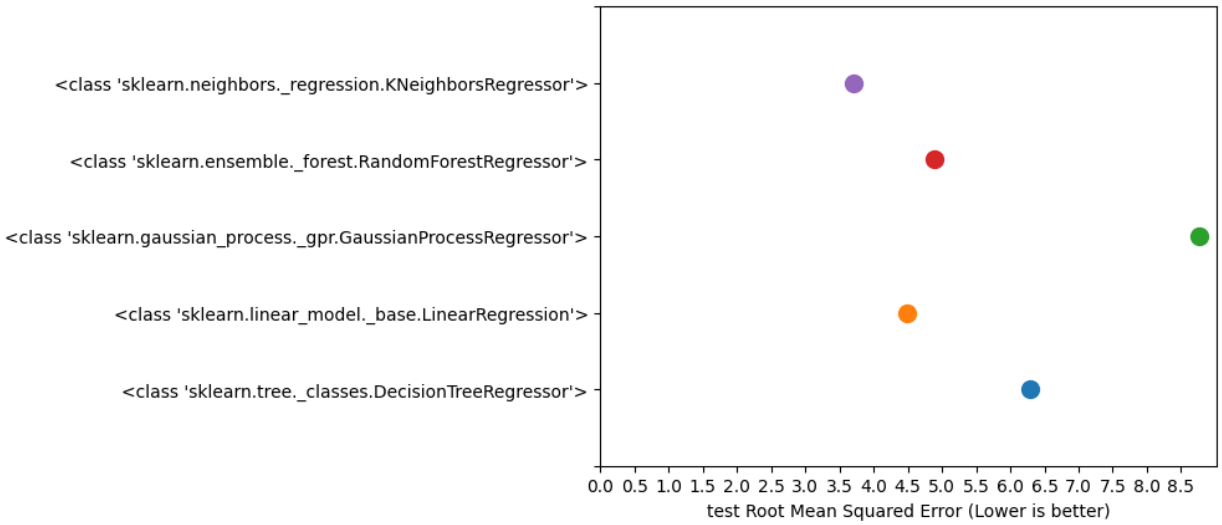


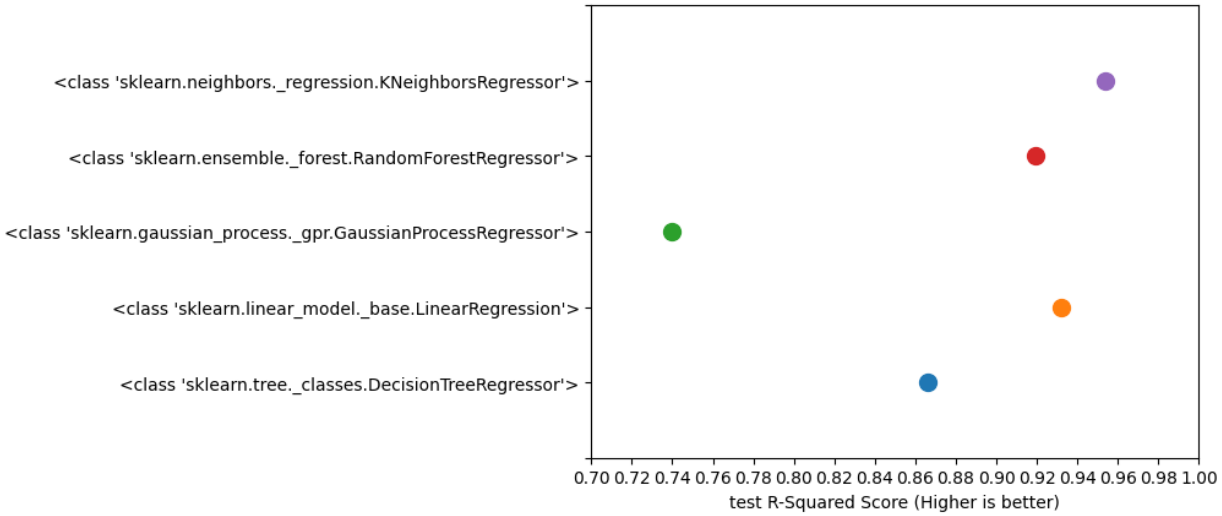


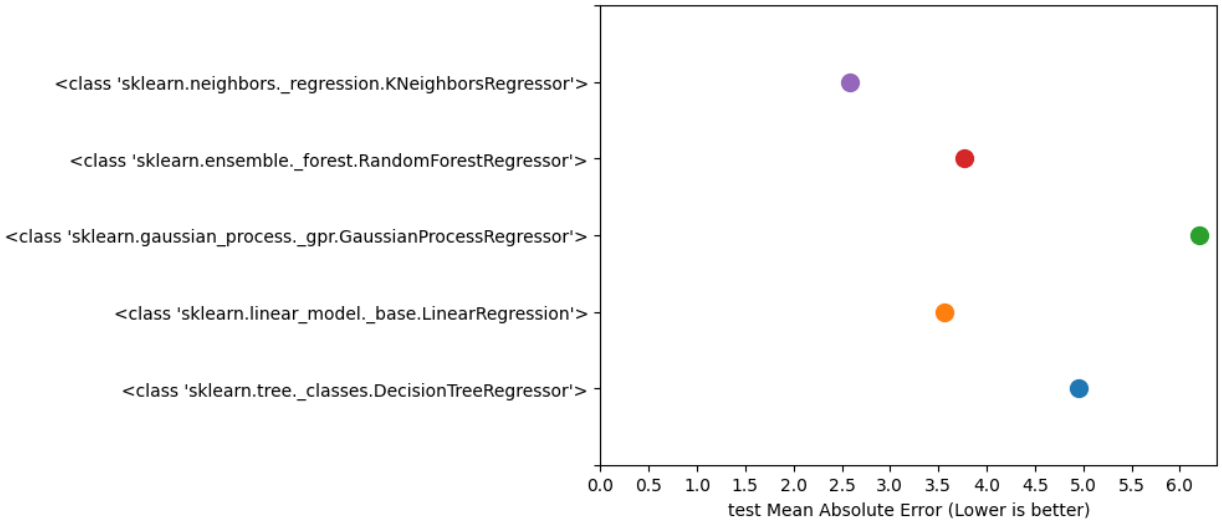

<class 'sklearn.neighbors._regression.KNeighborsRegressor'>
<class 'sklearn.ensemble._forest.RandomForestRegressor'>
<class 'sklearn.gaussian_process._gpr.GaussianProcessRegressor'>
<class 'sklearn.linear_model._base.LinearRegression'>
<class 'sklearn.tree._classes.DecisionTreeRegressor'>
0.70 0.72 0.74 0.76 0.78 0.80 0.82 0.84 0.86 0.88 0.90 0.92 0.94 0.96 0.98 1.00
train R-Squared Score (Higher is better)


<class 'sklearn.neighbors._regression.KNeighborsRegressor'>
<class 'sklearn.ensemble._forest.RandomForestRegressor'>
<class 'sklearn.gaussian_process._gpr.GaussianProcessRegressor'>
<class 'sklearn.linear_model._base.LinearRegression'>
<class 'sklearn.tree._classes.DecisionTreeRegressor'>
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
train Mean Absolute Error (Lower is better)


<class 'sklearn.neighbors._regression.KNeighborsRegressor'>
<class 'sklearn.ensemble._forest.RandomForestRegressor'>
<class 'sklearn.gaussian_process._gpr.GaussianProcessRegressor'>
<class 'sklearn.linear_model._base.LinearRegression'>
<class 'sklearn.tree._classes.DecisionTreeRegressor'>
0 5 10 15 20 25 30 35 40
test Mean Squared Error (Lower is better)


<class 'sklearn.neighbors._regression.KNeighborsRegressor'>
<class 'sklearn.ensemble._forest.RandomForestRegressor'>
<class 'sklearn.gaussian_process._gpr.GaussianProcessRegressor'>
<class 'sklearn.linear_model._base.LinearRegression'>
<class 'sklearn.tree._classes.DecisionTreeRegressor'>
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5
test Root Mean Squared Error (Lower is better)


<class 'sklearn.neighbors._regression.KNeighborsRegressor'>
<class 'sklearn.ensemble._forest.RandomForestRegressor'>
<class 'sklearn.gaussian_process._gpr.GaussianProcessRegressor'>
<class 'sklearn.linear_model._base.LinearRegression'>
<class 'sklearn.tree._classes.DecisionTreeRegressor'>
0.70 0.72 0.74 0.76 0.78 0.80 0.82 0.84 0.86 0.88 0.90 0.92 0.94 0.96 0.98 1.00
test R-Squared Score (Higher is better)


<class 'sklearn.neighbors._regression.KNeighborsRegressor'>
<class 'sklearn.ensemble._forest.RandomForestRegressor'>
<class 'sklearn.gaussian_process._gpr.GaussianProcessRegressor'>
<class 'sklearn.linear_model._base.LinearRegression'>
<class 'sklearn.tree._classes.DecisionTreeRegressor'>
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
test Mean Absolute Error (Lower is better)