First 20 Models Simultaneous Run

General Imports

In [ ]:
```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.utils import all_estimators
import warnings
warnings.filterwarnings('ignore')
```

Automatically Importing All Regressions

In [ ]:
```
from sklearn.linear_model import QuantileRegressor


estimators = all_estimators(type_filter='regressor')

all_regs = []
for name, RegressorClass in estimators:
    try:
        if name != 'DummyRegressor' and name != 'GaussianProcessRegressor' and name !=
            print('Appending', name)
            reg = RegressorClass()
            all_regs.append(reg)
    except Exception as e:
        print(e)

print(all_regs)
```

```
Appending ARDRegression
Appending AdaBoostRegressor
Appending BaggingRegressor
Appending BayesianRidge
Appending CCA
Appending DecisionTreeRegressor
Appending ElasticNet
Appending ElasticNetCV
Appending ExtraTreeRegressor
Appending ExtraTreesRegressor
Appending GammaRegressor
Appending GradientBoostingRegressor
Appending HistGradientBoostingRegressor
Appending HuberRegressor
Appending IsotonicRegression
Appending KNeighborsRegressor
Appending KernelRidge
Appending Lars
Appending LarsCV
Appending Lasso
Appending LassoCV
Appending LassoLars
Appending LassoLarsCV
Appending LassoLarsIC
Appending LinearRegression
Appending LinearSVR
Appending MLPRegressor
Appending MultiOutputRegressor
__init__() missing 1 required positional argument: 'estimator'
Appending MultiTaskElasticNet
Appending MultiTaskElasticNetCV
Appending MultiTaskLasso
Appending MultiTaskLassoCV
Appending NuSVR
Appending OrthogonalMatchingPursuit
Appending OrthogonalMatchingPursuitCV
Appending PLSCanonical
Appending PLSRegression
Appending PassiveAggressiveRegressor
Appending PoissonRegressor
Appending RANSACRegressor
Appending RadiusNeighborsRegressor
Appending RandomForestRegressor
Appending RegressorChain
__init__() missing 1 required positional argument: 'base_estimator'
Appending Ridge
Appending RidgeCV
Appending SVR
Appending StackingRegressor
__init__() missing 1 required positional argument: 'estimators'
Appending TheilSenRegressor
Appending TransformedTargetRegressor
Appending TweedieRegressor
Appending VotingRegressor
__init__() missing 1 required positional argument: 'estimators'
[ARDRegression(), AdaBoostRegressor(), BaggingRegressor(), BayesianRidge(), CCA(), De
cisionTreeRegressor(), ElasticNet(), ElasticNetCV(), ExtraTreeRegressor(), ExtraTrees
```

```
Regressor(), GammaRegressor(), GradientBoostingRegressor(), HistGradientBoostingRegre
ssor(), HuberRegressor(), IsotonicRegression(), KNeighborsRegressor(), KernelRidge(),
Lars(), LarsCV(), Lasso(), LassoCV(), LassoLars(), LassoLarsCV(), LassoLarsIC(), Line
arRegression(), LinearSVR(), MLPRegressor(), MultiTaskElasticNet(), MultiTaskElasticN
etCV(), MultiTaskLasso(), MultiTaskLassoCV(), NuSVR(), OrthogonalMatchingPursuit(), O
rthogonalMatchingPursuitCV(), PLSCanonical(), PLSRegression(), PassiveAggressiveRegre
ssor(), PoissonRegressor(), RANSACRegressor(), RadiusNeighborsRegressor(), RandomFore
stRegressor(), Ridge(), RidgeCV(), SVR(), TheilSenRegressor(), TransformedTargetRegre
ssor(), TweedieRegressor()]
```

Load and Describe Data

In [ ]:
```
def load_pp_data():
    csv_path = r"C:\Users\18123\OneDrive\Documents\IUBloomington\Machine-Learning-Proj
    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
```

Train/Test Split and Preprocess 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]])
```

Simultaneous Run

In [ ]:
```
def comparison(modellst):
    cv_data = []
    errors = []
    passed_models = []
    for i in range(len(modellst)):
        x = run(modellst[i])
        if type(x) == dict:
            cv_data += [x]
        else:
            errors += [i]
    for j in range(len(modellst)):
        if j not in errors:
            passed_models += [modellst[j]]
    display(test_best(cv_data, passed_models))
    return box_rmse(cv_data, passed_models), box_r2(cv_data, passed_models), box_mae(c


def run(model):
    print(f"checking {model}")
    try:
        cv_outer = KFold(n_splits=10, shuffle=True, random_state=2)
        cv_output_dict = cross_validate(model, pptrain_attrib, pptrain_labels, scoring
        return cv_output_dict
    except:
        pass


def runtime(cv_data, passed_models):
    timefig = plt.figure(constrained_layout=True)
    df = pd.DataFrame()
    for i,j in zip(cv_data,passed_models):
        df[j] = list(i[('fit_time')])
    sorted_index = df.median().sort_values().index
    df_sorted=df[sorted_index]
    top20 = df_sorted.drop(columns=df_sorted.columns[20:])
    top20_sorted_index = top20.median().sort_values(ascending=False).index
    top20_sorted=top20[top20_sorted_index]
    top20_sorted.boxplot(vert=False,grid=False)
    plt.xlabel('Run Time')
    plt.ylabel('Models')
    return timefig


def box_rmse(cv_data, passed_models):
    RMSEfig = plt.figure(constrained_layout=True)
    df = pd.DataFrame()
    for i,j in zip(cv_data,passed_models):
```

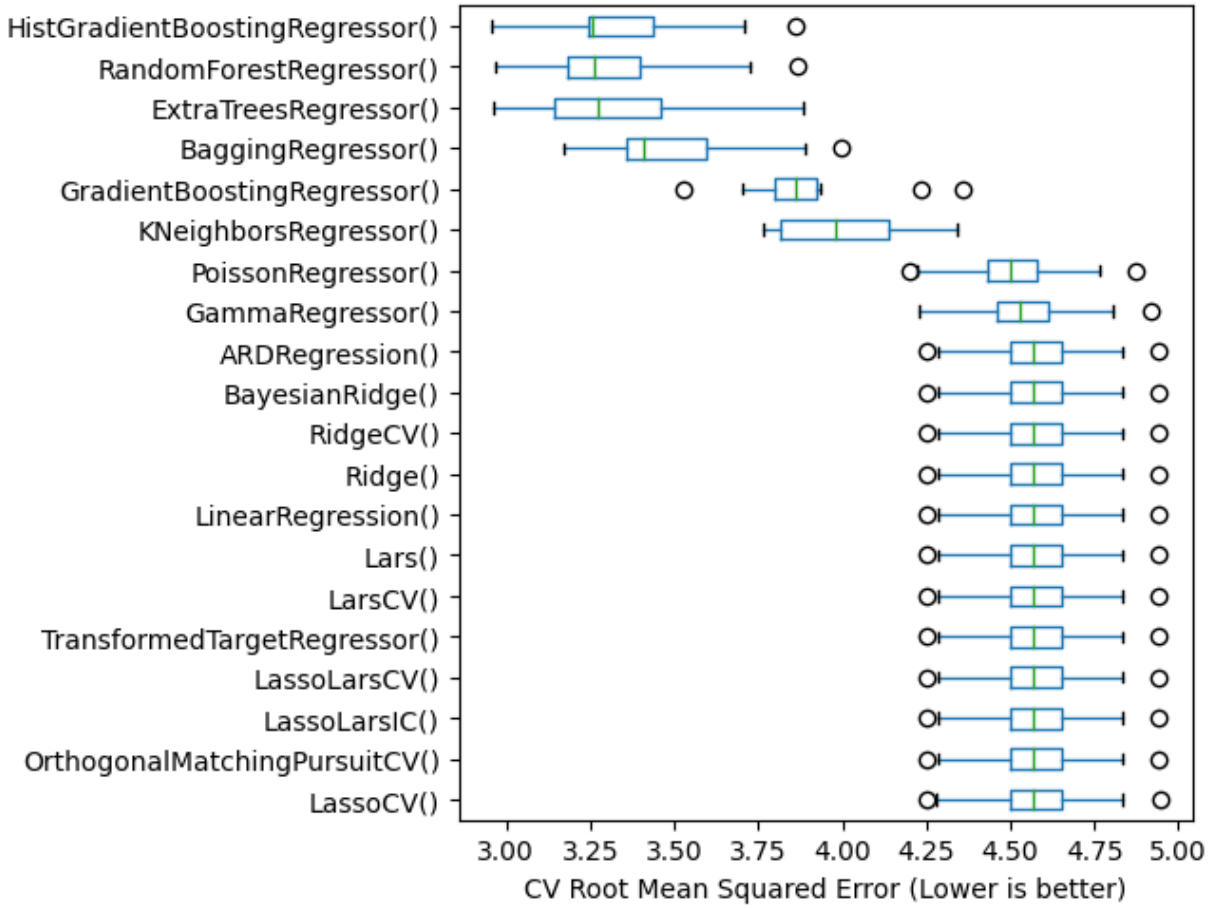
```
        df[j] = list(np.sqrt(i['test_neg_mean_squared_error']*-1))
    sorted_index = df.median().sort_values().index
    df_sorted=df[sorted_index]
    top20 = df_sorted.drop(columns=df_sorted.columns[20:])
    top20_sorted_index = top20.median().sort_values(ascending=False).index
    top20_sorted=top20[top20_sorted_index]
    top20_sorted.boxplot(vert=False,grid=False)
    plt.xlabel(f'CV Root Mean Squared Error (Lower is better)')
    return RMSEfig


def box_r2(cv_data, passed_models):
    R2fig = plt.figure(constrained_layout=True)
    df = pd.DataFrame()
    for i,j in zip(cv_data,passed_models):
        df[j] = list(i['test_r2'])
    sorted_index = df.median().sort_values(ascending=False).index
    df_sorted=df[sorted_index]
    top20 = df_sorted.drop(columns=df_sorted.columns[20:])
    top20_sorted_index = top20.median().sort_values().index
    top20_sorted=top20[top20_sorted_index]
    top20_sorted.boxplot(vert=False,grid=False)
    plt.xlabel(f'CV R-Squared Score (Higher is better)')
    return R2fig


def box_mae(cv_data, passed_models):
    MAEfig = plt.figure(constrained_layout=True)
    df = pd.DataFrame()
    for i,j in zip(cv_data,passed_models):
        df[j] = list(i['test_neg_mean_absolute_error']*-1)
    sorted_index = df.median().sort_values().index
    df_sorted=df[sorted_index]
    top20 = df_sorted.drop(columns=df_sorted.columns[20:])
    top20_sorted_index = top20.median().sort_values(ascending=False).index
    top20_sorted=top20[top20_sorted_index]
    top20_sorted.boxplot(vert=False,grid=False)
    plt.xlabel(f'CV Mean Absolute Error (Lower is better)')
    return MAEfig


def test_best(cv_data, passed_models):
    rmse = []
    r2 = []
    mae = []
    for i in cv_data:
        x = list((np.sqrt(i['test_neg_mean_squared_error']*-1)))
        y = list(i['estimator'])
        for j in range(len(x)):
            if x[j] == min(x):
                best = y[j]
        predictions = best.predict(pptest_attrib)
        rmse += [np.sqrt(mean_squared_error(pptest_labels,predictions))]
        r2 += [r2_score(pptest_labels,predictions)]
        mae += [mean_absolute_error(pptest_labels,predictions)]
    columnnames = ['rmse','r2','mae']
    df = pd.DataFrame(np.array([rmse,r2,mae]).T,index=passed_models,columns=columnname
```

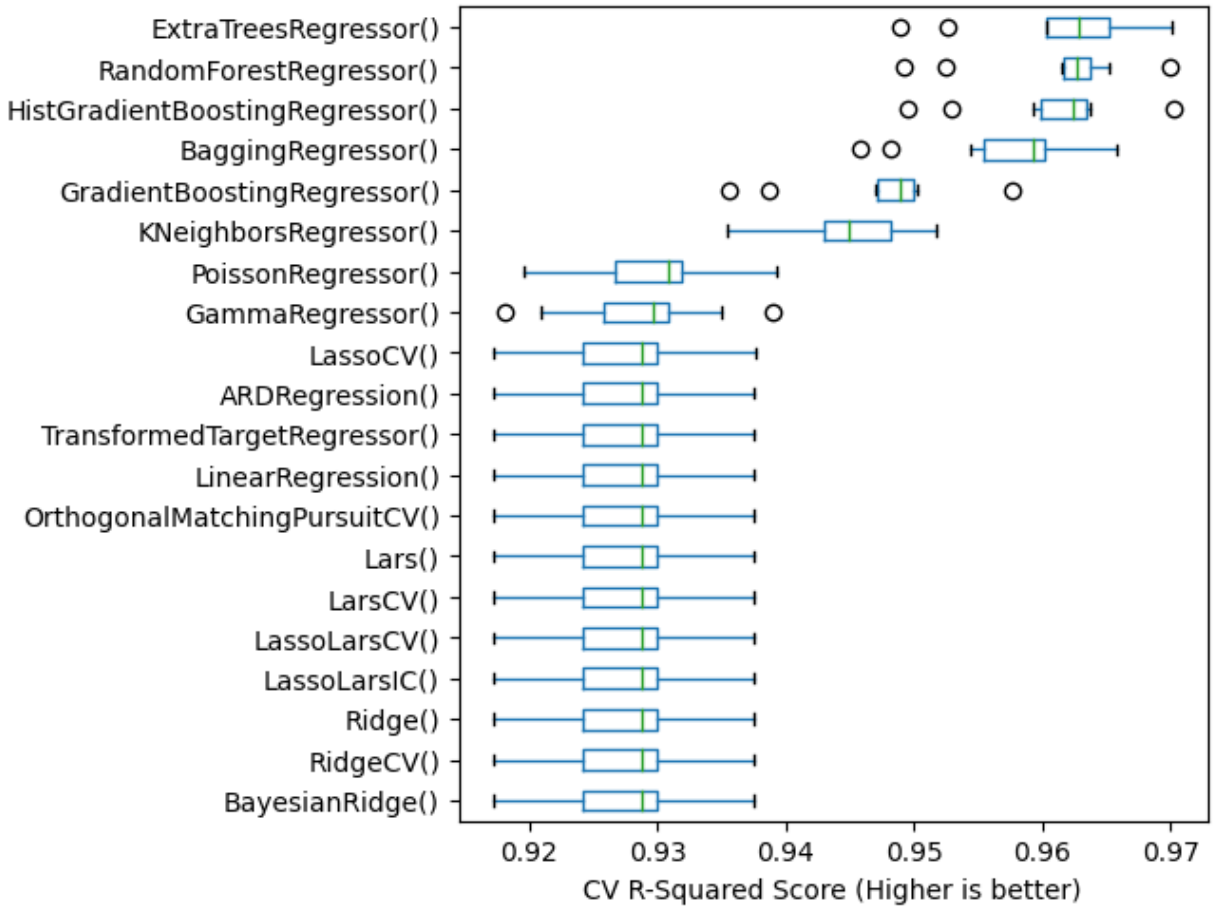
```
    sorted_df = df.sort_values(by="rmse",ascending=True)
    final_df = sorted_df.style.format("{:.4f}")
    return final_df


y = all_regs
x = all_regs[0:8]
comparison(y)
plt.show()
```

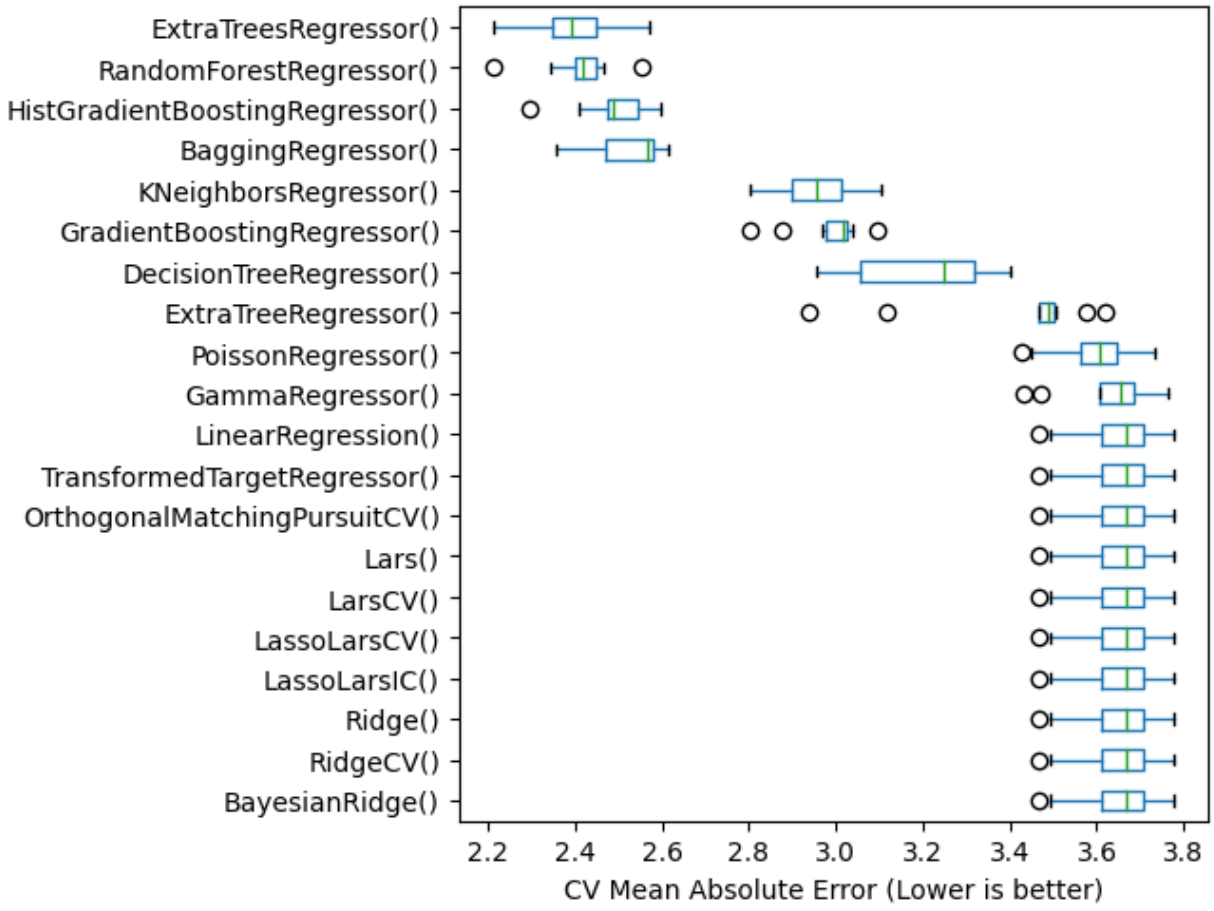
```
checking ARDRegression()
checking AdaBoostRegressor()
checking BaggingRegressor()
checking BayesianRidge()
checking CCA()
checking DecisionTreeRegressor()
checking ElasticNet()
checking ElasticNetCV()
checking ExtraTreeRegressor()
checking ExtraTreesRegressor()
checking GammaRegressor()
checking GradientBoostingRegressor()
checking HistGradientBoostingRegressor()
checking HuberRegressor()
checking IsotonicRegression()
checking KNeighborsRegressor()
checking KernelRidge()
checking Lars()
checking LarsCV()
checking Lasso()
checking LassoCV()
checking LassoLars()
checking LassoLarsCV()
checking LassoLarsIC()
checking LinearRegression()
checking LinearSVR()
checking MLPRegressor()
checking MultiTaskElasticNet()
checking MultiTaskElasticNetCV()
checking MultiTaskLasso()
checking MultiTaskLassoCV()
checking NuSVR()
checking OrthogonalMatchingPursuit()
checking OrthogonalMatchingPursuitCV()
checking PLSCanonical()
checking PLSRegression()
checking PassiveAggressiveRegressor()
checking PoissonRegressor()
checking RANSACRegressor()
checking RadiusNeighborsRegressor()
checking RandomForestRegressor()
checking Ridge()
checking RidgeCV()
checking SVR()
checking TheilSenRegressor()
checking TransformedTargetRegressor()
checking TweedieRegressor()
```

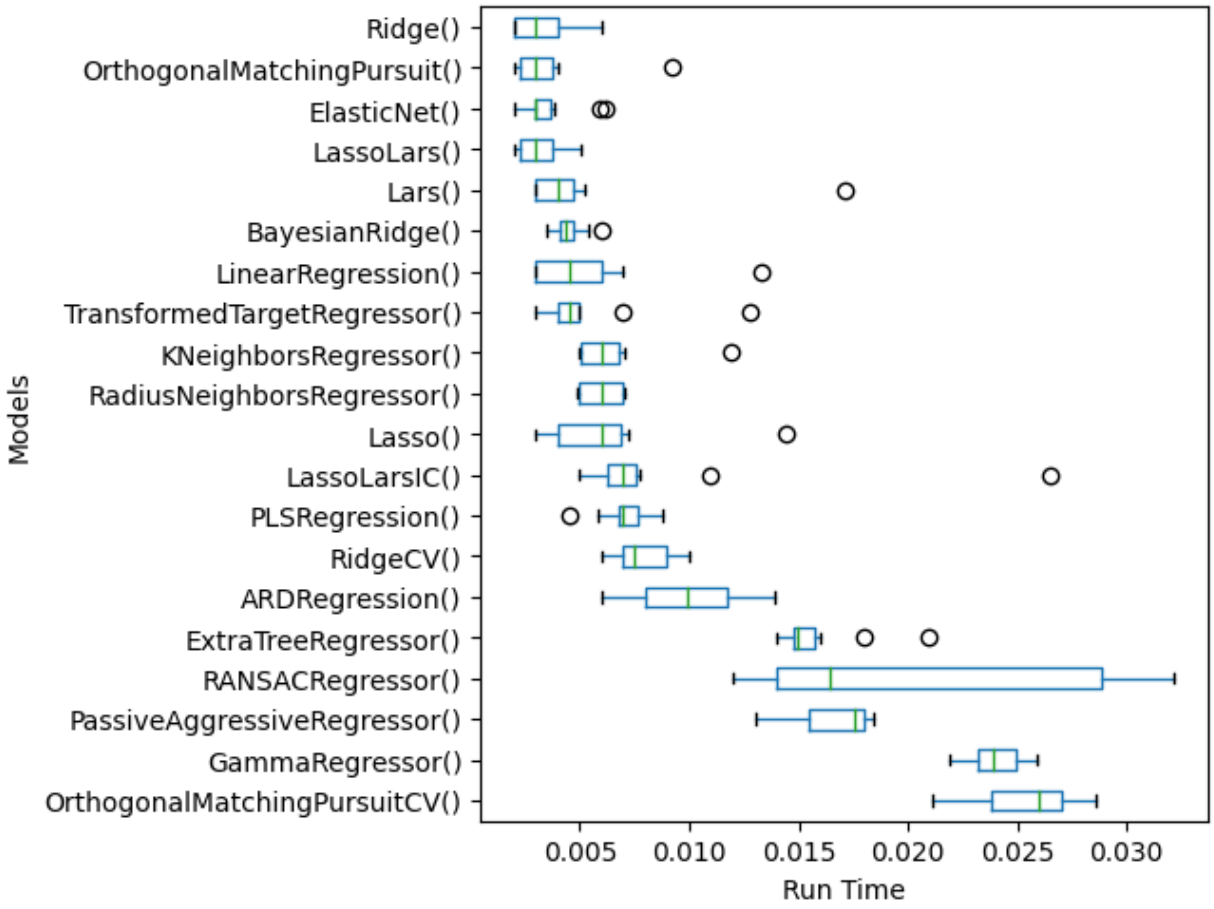
	rmse	r2	mae
HistGradientBoostingRegressor()	3.3676	0.9616	2.3837
RandomForestRegressor()	3.4110	0.9606	2.3139
ExtraTreesRegressor()	3.4211	0.9604	2.3178
BaggingRegressor()	3.5922	0.9563	2.4876
GradientBoostingRegressor()	3.8396	0.9501	2.8735
KNeighborsRegressor()	4.0942	0.9433	2.9300
PoissonRegressor()	4.4227	0.9338	3.5091
GammaRegressor()	4.4433	0.9332	3.5354
ElasticNetCV()	4.4850	0.9319	3.5634
LassoCV()	4.4869	0.9319	3.5609
BayesianRidge()	4.4870	0.9319	3.5608
RidgeCV()	4.4870	0.9319	3.5608
Ridge()	4.4870	0.9319	3.5608
LassoLarsCV()	4.4870	0.9319	3.5608
LarsCV()	4.4870	0.9319	3.5608
OrthogonalMatchingPursuitCV()	4.4870	0.9319	3.5608
LassoLarsIC()	4.4870	0.9319	3.5608
LinearRegression()	4.4870	0.9319	3.5608
Lars()	4.4870	0.9319	3.5608
TransformedTargetRegressor()	4.4870	0.9319	3.5608
ARDRegression()	4.4875	0.9319	3.5606
RadiusNeighborsRegressor()	4.4884	0.9318	3.5626
RANSACRegressor()	4.4884	0.9318	3.5626
ElasticNet()	4.4917	0.9317	3.5731
Lasso()	4.4918	0.9317	3.5654
HuberRegressor()	4.4945	0.9316	3.5519
TweedieRegressor()	4.5003	0.9315	3.5881
TheilSenRegressor()	4.5129	0.9311	3.5715
MLPRegressor()	4.7019	0.9252	3.6967
DecisionTreeRegressor()	4.7308	0.9243	3.0925
KernelRidge()	4.9334	0.9176	3.9177
LinearSVR()	4.9452	0.9172	3.9419
AdaBoostRegressor()	5.0095	0.9151	3.9485

	rmse	r2	mae
ExtraTreeRegressor()	5.0504	0.9137	3.3712
PLSRegression()	5.0933	0.9122	4.0585
OrthogonalMatchingPursuit()	5.3416	0.9034	4.1911
PassiveAggressiveRegressor()	5.3476	0.9032	4.1984
SVR()	13.8241	0.3533	11.6661
NuSVR()	14.6625	0.2725	12.7126
LassoLars()	17.1909	-0.0000	15.0920




ExtraTreesRegressor()
RandomForestRegressor()
HistGradientBoostingRegressor()
BaggingRegressor()
GradientBoostingRegressor()
KNeighborsRegressor()
PoissonRegressor()
GammaRegressor()
LassoCV()
ARDRegression()
TransformedTargetRegressor()
LinearRegression()
OrthogonalMatchingPursuitCV()
Lars()
LarsCV()
LassoLarsCV()
LassoLarsIC()
Ridge()
RidgeCV()
BayesianRidge()
0.92
0.93
0.94
0.95
0.96
0.97
CV R-Squared Score (Higher is better)


ExtraTreesRegressor()
RandomForestRegressor()
HistGradientBoostingRegressor()
BaggingRegressor()
KNeighborsRegressor()
GradientBoostingRegressor()
DecisionTreeRegressor()
ExtraTreeRegressor()
PoissonRegressor()
GammaRegressor()
LinearRegression()
TransformedTargetRegressor()
OrthogonalMatchingPursuitCV()
Lars()
LarsCV()
LassoLarsCV()
LassoLarsIC()
Ridge()
RidgeCV()
BayesianRidge()
2.2
2.4
2.6
2.8
3.0
3.2
3.4
3.6
3.8
CV Mean Absolute Error (Lower is better)


Ridge()
OrthogonalMatchingPursuit()
ElasticNet()
LassoLars()
Lars()
BayesianRidge()
LinearRegression()
TransformedTargetRegressor()
KNeighborsRegressor()
RadiusNeighborsRegressor()
Lasso()
LassoLarsIC()
PLSRegression()
RidgeCV()
ARDRegression()
ExtraTreeRegressor()
RANSACRegressor()
PassiveAggressiveRegressor()
GammaRegressor()
OrthogonalMatchingPursuitCV()
Models
0.005
0.010
0.015
0.020
0.025
0.030
Run Time