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

from sklearn.model_selection import train_test_split, KFold, cross_val_sc
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import make_pipeline

from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import ElasticNet

from sklearn.metrics import mean_squared_error
from sklearn.metrics import make_scorer

import matplotlib.pyplot as plt
```

[2]
```
df = pd.read_csv("./clean_data.csv", index_col=0)
```

[3]
```
df.head()
```

	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Year
Id						
1	65.0	8450	7	5	2003	2003
2	80.0	9600	6	8	1976	1976
3	68.0	11250	7	5	2001	2002
4	60.0	9550	7	5	1915	1970
5	84.0	14260	8	5	2000	2000

5 rows × 546 columns

[4]
```
X = df.drop("SalePrice", axis=1)
y = np.log(df['SalePrice'])
```

Fit and Optimise Models

Linear Regression:

- sklearn.linear_model.Ridge
- sklearn.linear_model.Lasso
- sklearn.linear_model.ElasticNet

Support Vector Machines:

- sklearn.svm.LinearSVR
- sklearn.svm.SVR

Nearest Neighbours:

- sklearn.neighbors.KNearestNeighborsRegressor

Tree Based:

- sklearn.ensemble.RandomForestRegressor
- sklearn.ensemble.GradientBoostingRegressor
- xgboost.XGBRegressor

[5]
```
def rmse(y_true, y_pred):
    diff = y_pred - y_true
    sum_sq = sum(diff**2)
    n = len(y_pred)

    return np.sqrt(sum_sq/n)

rmse_scorer = make_scorer(rmse, greater_is_better=False)
```

[6]
```
def train_model(model, param_grid=[], X=[], y=[],
                splits=5, repeats=5):

    rkfold = RepeatedKFold(n_splits=splits, n_repeats=repeats)

    if len(param_grid)>0:
        gsearch = GridSearchCV(model, param_grid, cv=rkfold,
                               scoring=rmse_scorer, n_jobs = 8,
                               verbose=1, return_train_score=True)

        gsearch.fit(X,y)

        model = gsearch.best_estimator_
        best_idx = gsearch.best_index_

        grid_results = pd.DataFrame(gsearch.cv_results_)
        cv_mean = abs(grid_results.loc[best_idx,'mean_test_score'])
        cv_std = grid_results.loc[best_idx,'std_test_score']

    else:
        grid_results = []
        cv_results = cross_val_score(model, X, y, scoring=rmse_scorer, cv
        cv_mean = abs(np.mean(cv_results))
        cv_std = np.std(cv_results)

    cv_score = pd.Series({'mean':cv_mean,'std':cv_std})

    y_pred = model.predict(X)

    print('----------------------')
```

```
    print(model)
    print('----------------------')
    print('score=',model.score(X,y))
    print('rmse=',rmse(y, y_pred))
    print('cross_val: mean=',cv_mean,', std=',cv_std)

    y_pred = pd.Series(y_pred,index=y.index)
    resid = y - y_pred
    mean_resid = resid.mean()
    std_resid = resid.std()
    z = (resid - mean_resid)/std_resid
    n_outliers = sum(abs(z)>3)

    plt.figure(figsize=(15,5))
    ax_131 = plt.subplot(1,3,1)
    plt.plot(y,y_pred,'.')
    plt.xlabel('y')
    plt.ylabel('y_pred');
    plt.title('corr = {:.3f}'.format(np.corrcoef(y,y_pred)[0][1]))
    ax_132=plt.subplot(1,3,2)
    plt.plot(y,y-y_pred,'.')
    plt.xlabel('y')
    plt.ylabel('y - y_pred');
    plt.title('std resid = {:.3f}'.format(std_resid))

    ax_133=plt.subplot(1,3,3)
    z.plot.hist(bins=50,ax=ax_133)
    plt.xlabel('z')
    plt.title('{:.0f} samples with z>3'.format(n_outliers))

    return model, cv_score, grid_results
```

ElasticNet

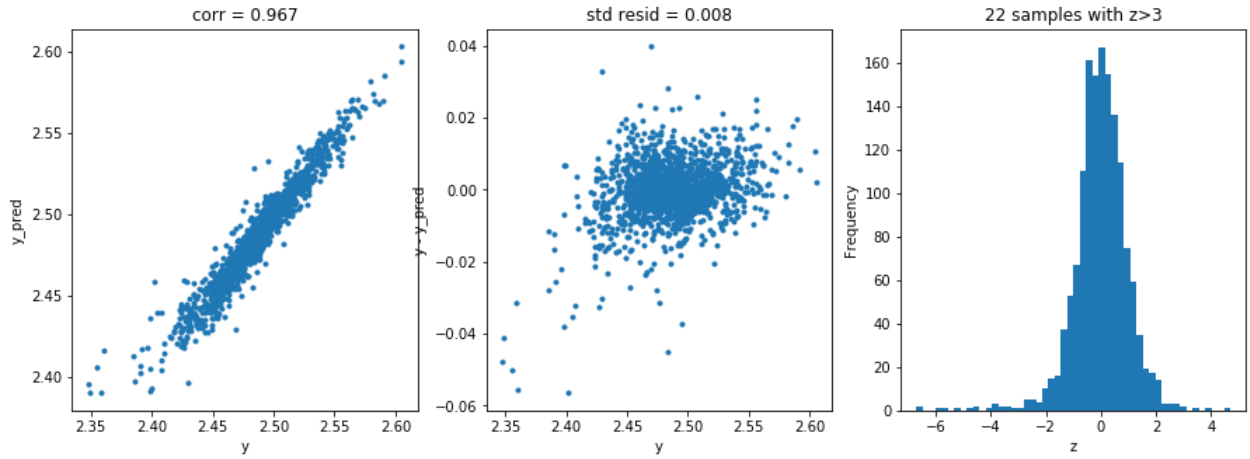
[7]
```
en_int = ElasticNet()

param_grid = {'alpha': np.arange(1e-4,1e-3,1e-4),
              'l1_ratio': np.arange(0.1,1.0,0.1),
              'max_iter':[100000]}
# param_grid = {'l1_ratio': np.arange(0.1,1.0,0.1),
#               'max_iter':[100000]}


en_int, cv_score, grid_results = train_model(en_int, X=X, y=y, param_grid
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
----------------------
ElasticNet(alpha=0.0001, copy_X=True, fit_intercept=True, l1_ratio=0.2,
      max_iter=100000, normalize=False, positive=False, precompute=False,
```

```
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
----------------------
score= 0.9353337276090429
rmse= 0.008443259909998552
cross_val: mean= 0.011679146088891477 , std= 0.0022280288068281012
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    3.7s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:    7.1s
[Parallel(n_jobs=8)]: Done 390 out of 405 | elapsed:   10.1s remaining:
0.4s
[Parallel(n_jobs=8)]: Done 405 out of 405 | elapsed:   10.2s finished
```



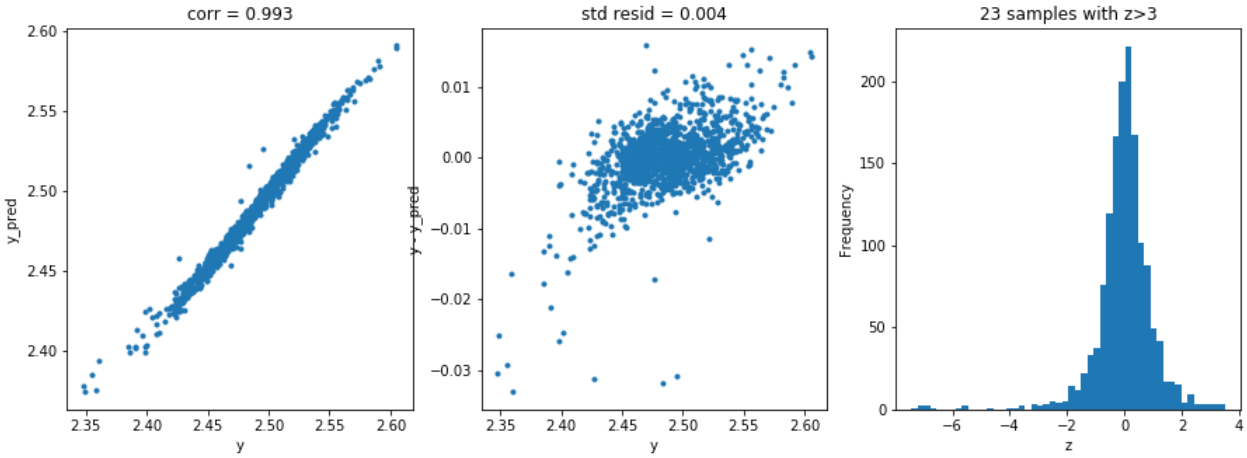
Random Forest Regression

[8]
```
rfr = RandomForestRegressor()

param_grid = {'n_estimators':[100,150,200],
              'max_features':[25,50,75],
              'min_samples_split':[2,4,6]}

rfr, cv_score, grid_results = train_model(rfr, X=X, y=y,param_grid=param_
                                          splits=5, repeats=1)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
----------------------
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=75, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=100, n_jobs=None, oob_score=False,
           random_state=None, verbose=0, warm_start=False)
----------------------
score= 0.981874803806623
rmse= 0.004470052284003907
cross_val: mean= 0.01170475735716495 , std= 0.0005079339374733876
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    3.9s
[Parallel(n_jobs=8)]: Done 135 out of 135 | elapsed:   19.5s finished
```


corr = 0.993
y_pred
y
std resid = 0.004
y - y_pred
y
23 samples with z>3
Frequency
z

[ ]