```
 1 import numpy
 2 import pandas
 3 import pop_coding
 4 from keras.models import Sequential
 5 from keras.layers import Dense
 6 from keras.wrappers.scikit_learn import KerasRegressor
 7 from sklearn import preprocessing
 8 from sklearn.model_selection import cross_val_score
 9 from sklearn.model_selection import KFold
10 from sklearn.preprocessing import StandardScaler
11 from sklearn.pipeline import Pipeline
12 from sklearn.metrics import accuracy_score
13 from sklearn.metrics import mean_squared_error
14
15 # load dataset
16 dataframe = pandas.read_csv("housing.csv", delim_whitespace
   =True, header=None)
17 dataset = dataframe.values
18 data_scaler = pop_coding.code_dataframe(dataframe)
19 coded_dataframe = data_scaler[0]
20 min_max_scaler = data_scaler[1]
21 coded_dataset = coded_dataframe.values
22
23 # split into input (X) and output (Y)
24 X = coded_dataset[:, 0:130]
25 Y = coded_dataset[:, 130:140]
26
27
28 # define base model
29 def baseline_model():
30     # create model
31     model = Sequential()
32     model.add(Dense(130, input_dim=130, kernel_initializer=
   'normal', activation='relu'))
33     model.add(Dense(10, kernel_initializer='normal'))
34     # compile model
35     model.compile(loss='mean_squared_error', optimizer='
   adam')
36     return model
37
38 # fix random seed for reproducibility
39 seed = 7
40 numpy.random.seed(seed)
41 # evaluate model with standardized dataset
42 # estimator = KerasRegressor(build_fn=baseline_model,
   epochs=100, batch_size=5, verbose=0)
43
44 estimator = KerasRegressor(build_fn=baseline_model, epochs=
   1000, batch_size=5, verbose=1)
45 # estimators.append(('standardize', StandardScaler()))
```

```
# estimators.append(('mlp', KerasRegressor(build_fn=
  baseline_model, epochs=10, batch_size=5, verbose=1)))
# pipeline = Pipeline(estimators)

# kfold = KFold(n_splits=2, random_state=seed)
# results = cross_val_score(estimator, X, Y, cv=kfold)
# print("Results: %.2f (%.2f) MSE" % (results.mean(),
  results.std()))

estimator.fit(X, Y)
prediction = estimator.predict(X)

decoded_prediction = numpy.array(pop_coding.
  decode_prediction(prediction))


a = numpy.zeros([506, 14])
a[:,13] = decoded_prediction
rescaled_decoded_prediction = min_max_scaler.
  inverse_transform(a)[:,13]

numpy.savetxt("housing_prediction_coded.csv", prediction,
  delimiter=",")
numpy.savetxt("housing_prediction_decoded.csv",
  decoded_prediction, delimiter=",")
numpy.savetxt("housing_prediction_decoded_rescaled.csv",
  rescaled_decoded_prediction, delimiter=",")

# Error between rescaled prediction and real values

print(mean_squared_error(dataset[:, 13],
  rescaled_decoded_prediction))

```