In [ ]:

```
# import pandas, numpy, matplotlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, BatchNormalization

# import scikit-learn
from sklearn.model_selection import train_test_split
```

In [ ]:

```
# load boston housing dataset
df_data = pd.read_csv('BostonHousing.csv')
print(df_data.shape)
```

In [ ]:

```
# display
df_data.head(10)
```

In [ ]:

```
# feature scaling
df_data2 = df_data.copy()
df_data2 = (df_data2 - df_data2.mean()) / df_data2.std()
df_data2 = 1.0 / (1 + np.exp(-df_data2)) # sigmoid function
df_data2.head(10)
```

In [ ]:

```
# calculate the correlation of MEDV with something
df_data.corrwith(df_data['MEDV'])
```

In [ ]:

```
# calculate the correlation of MEDV with something
df_data2.corrwith(df_data2['MEDV'])
```

In [ ]:

```
# convert pandas to numpy & split into input and label
bh_data = np.array(df_data.values, dtype=np.float32)
bh_data2 = np.array(df_data2.values, dtype=np.float32)

data_input = np.array(bh_data2[:,:13], dtype=np.float32).reshape(-1, 13)
data_label = np.array(bh_data[:,13], dtype=np.float32).reshape(-1, 1)

print(bh_data.shape, bh_data2.shape)
print(data_input.shape, data_label.shape)
```

In [ ]:

```
# split into train and test datasets
x_train, x_test, y_train, y_test = train_test_split(data_input, data_label, test_size=0.2)
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

In [ ]:

```
# neural network architecture
h_units = 32
activation_ = 'relu'

def dnn_model():
    model = Sequential()

    model.add(Dense(units=h_units, input_dim=13))
    model.add(BatchNormalization())
    model.add(Activation(activation_))

    model.add(Dense(units=h_units))
    model.add(BatchNormalization())
    model.add(Activation(activation_))

    model.add(Dense(units=h_units))
    model.add(BatchNormalization())
    model.add(Activation(activation_))

    model.add(Dense(units=h_units))
    model.add(BatchNormalization())
    model.add(Activation(activation_))

    model.add(Dense(units=1))

    return model
```

In [ ]:

```
# loss function & optimization method
model = dnn_model()
model.compile(loss='mean_squared_error', optimizer='adagrad')
```

In [ ]:

```
# training
hist = model.fit(x_train, y_train, epochs=200, batch_size=8, verbose=True, validation_data=(x_te
st, y_test), shuffle=True)
```

In [ ]:

```
# evaluation
y_pred = model.predict(x_test, batch_size=512)
print(y_pred.shape, y_test.shape)

model.evaluate(x=x_test, y=y_test, batch_size=128, verbose=1)
```

In [ ]:

```
# comparison of the true and estimated values & save the csv file
result = pd.DataFrame({
    "y_true" : y_test[:,0],
    "y_pred" : y_pred[:,0]
})
result.to_csv('result.csv', index=False)
result.head(10)
```

In [ ]:

```
# visualization for prediction result
# set figure size
plt.figure(figsize=(19,8))

# set font
font = {'family': 'Arial', 'weight': 'normal', 'size': 16}
plt.rc('font', **font)

# plot data
plt.plot(y_test, 'r', label='y_true')
plt.plot(y_pred, 'b', label='y_pred')

# set legend position
plt.legend(shadow=True, loc='upper right')

# set x-axis & y-axis titles
plt.title('prediction result')
plt.xlabel('House index')
plt.ylabel('MEDV')

# set axis-limits
plt.xlim(left = 0, right = len(y_test))
plt.ylim(bottom = 0)

# set grid
plt.grid(color='gray', linestyle='--', linewidth=1)

plt.xticks(np.arange(0,101,10))
plt.yticks(np.arange(0,51,10))

# save & display figure
plt.savefig('prediction_result.png')
plt.show()
```