Michael Ferko

Dr. Jacek Zurada

ECE 614 Deep Learning

23 February 2020

Lab 1: Assignment

Real Estate Evaluation of housing prices in Taipei Taiwan

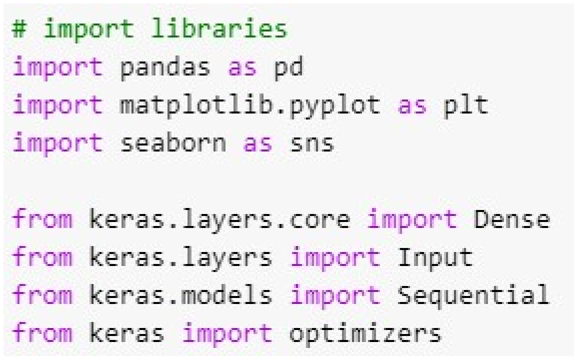
Objective:

Use the Mean Square Error Loss Function for training on 80% of data pairs. Set aside 20% of data for testing. Compare the actual 20% of house prices with the predicted 20% of house prices. The predicted 20% is computed from modeling the Neural network based on the initial 80% of normalized random data.

Tasks:

1. Try several architectures for this 5-inputs 1-output model. Select the most accurate model and for the best one implement tasks 2 and 3. Be flexible with the number of training epochs to lower the error - if needed.

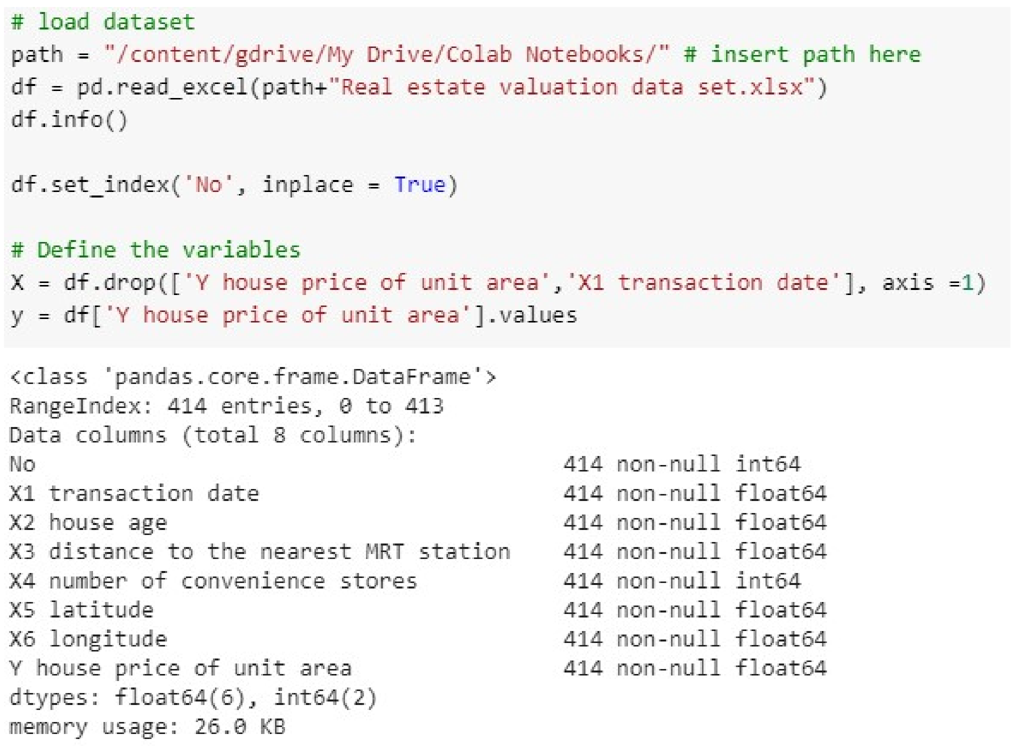
These are the imports used:



```
# import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from keras.layers.core import Dense
from keras.layers import Input
from keras.models import Sequential
from keras import optimizers
```

Figure 1: Imports Used

Data was loaded from the google drive environment and the index "No" was defined to be taken note of by the system but ignored as a data column. Used data format panda read excel to get data. Used data format drop to get rid of house prices and transaction date from the input X. Therefore, X is a 5 input vector column, 414 row matrix. As shown in Figure 2:

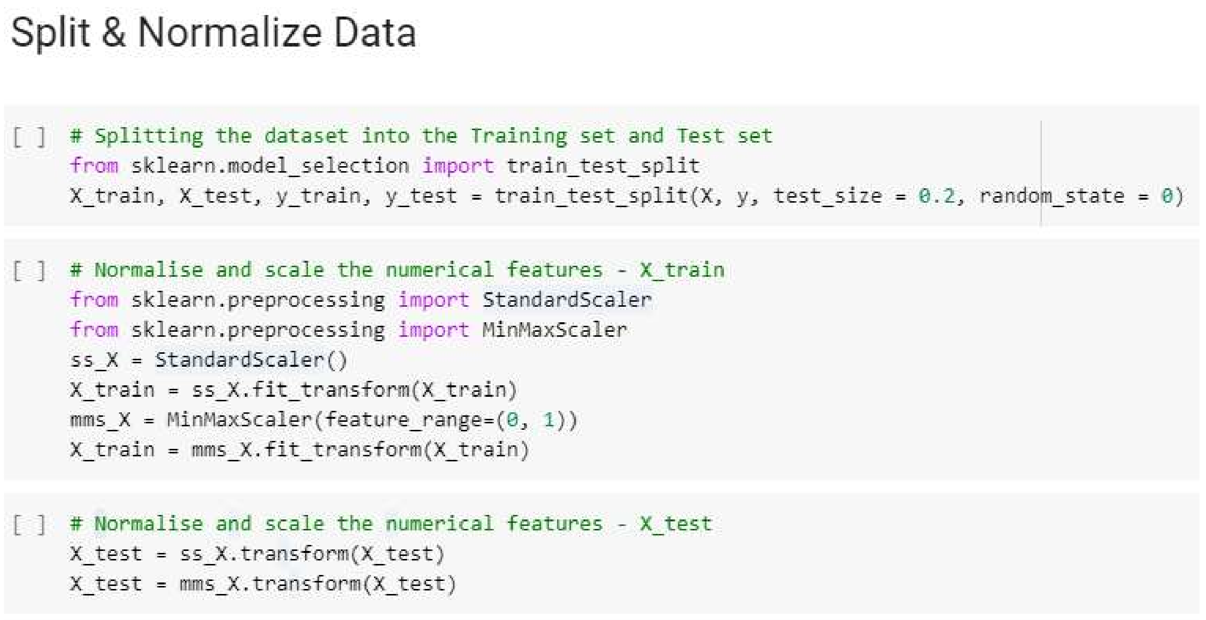


```
# load dataset
path = "/content/gdrive/My Drive/Colab Notebooks/" # insert path here
df = pd.read_excel(path+"Real estate valuation data set.xlsx")
df.info()

df.set_index('No', inplace = True)

# Define the variables
X = df.drop(['Y house price of unit area','X1 transaction date'], axis =1)
y = df['Y house price of unit area'].values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 414 entries, 0 to 413
Data columns (total 8 columns):
No                                        414 non-null int64
X1 transaction date                       414 non-null float64
X2 house age                              414 non-null float64
X3 distance to the nearest MRT station    414 non-null float64
X4 number of convenience stores           414 non-null int64
X5 latitude                               414 non-null float64
X6 longitude                              414 non-null float64
Y house price of unit area                414 non-null float64
dtypes: float64(6), int64(2)
memory usage: 26.0 KB
```

Figure 2: Load Data formatting

Data needed to be split into 80% training data and 20% testing data for the housing prices y and the 5-input column matrix X. Then the data needed to be normalized for training and testing data. As shown in Figure 3:



Split & Normalize Data

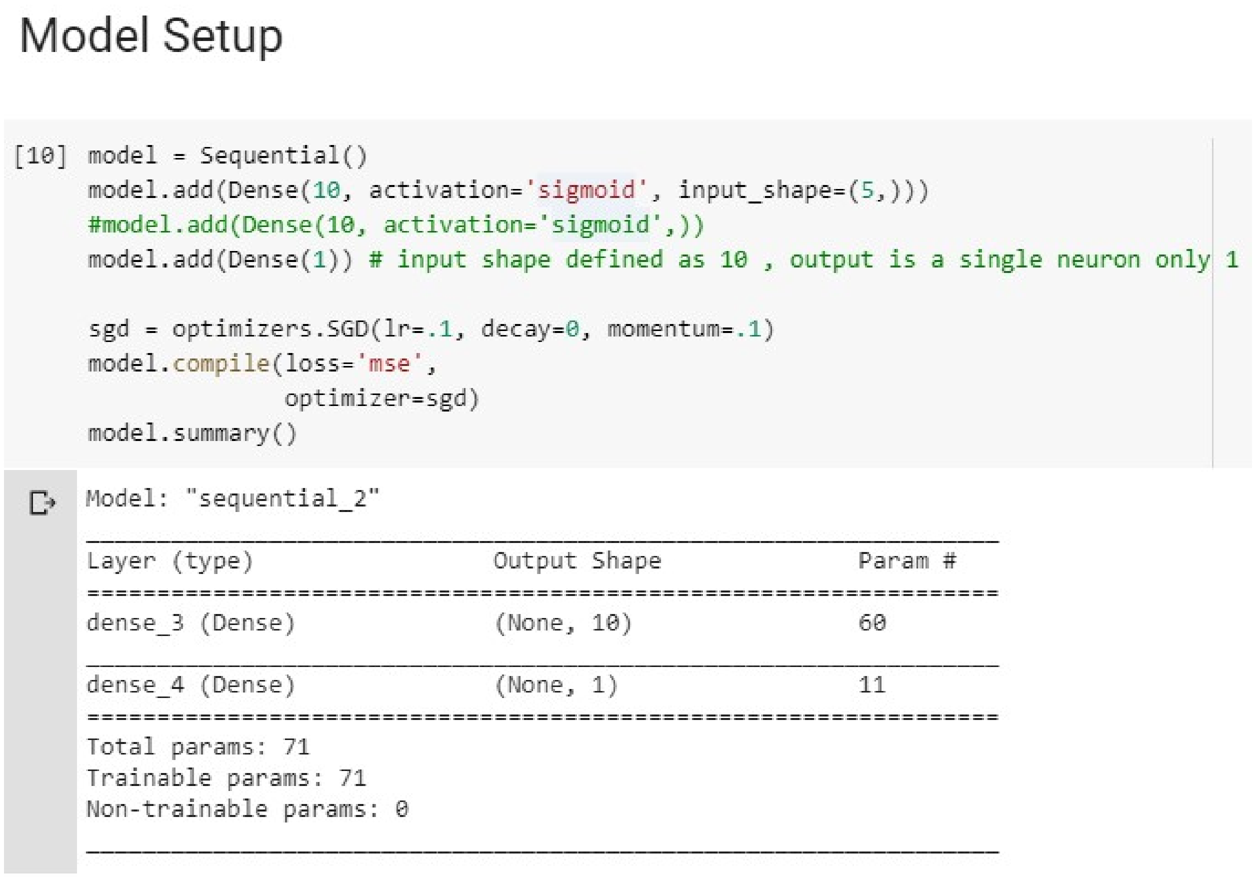
```
[ ] # Splitting the dataset into the Training set and Test set
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
[ ] # Normalise and scale the numerical features - X_train
    from sklearn.preprocessing import StandardScaler
    from sklearn.preprocessing import MinMaxScaler
    ss_X = StandardScaler()
    X_train = ss_X.fit_transform(X_train)
    mms_X = MinMaxScaler(feature_range=(0, 1))
    X_train = mms_X.fit_transform(X_train)
```

```
[ ] # Normalise and scale the numerical features - X_test
    X_test = ss_X.transform(X_test)
    X_test = mms_X.transform(X_test)
```

Figure 3: Splitting and normalization of Data

The only changes necessary for Model setup is to change the input shape from 1 column vector to 5 column vectors:



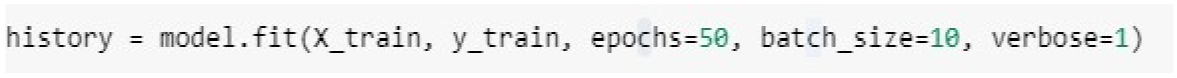
Model Setup

```
[10] model = Sequential()
     model.add(Dense(10, activation='sigmoid', input_shape=(5,)))
     #model.add(Dense(10, activation='sigmoid',))
     model.add(Dense(1)) # input shape defined as 10 , output is a single neuron only 1

     sgd = optimizers.SGD(lr=.1, decay=0, momentum=.1)
     model.compile(loss='mse',
                   optimizer=sgd)
     model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 10)                60
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 71
Trainable params: 71
Non-trainable params: 0
_________________________________________________________________
```

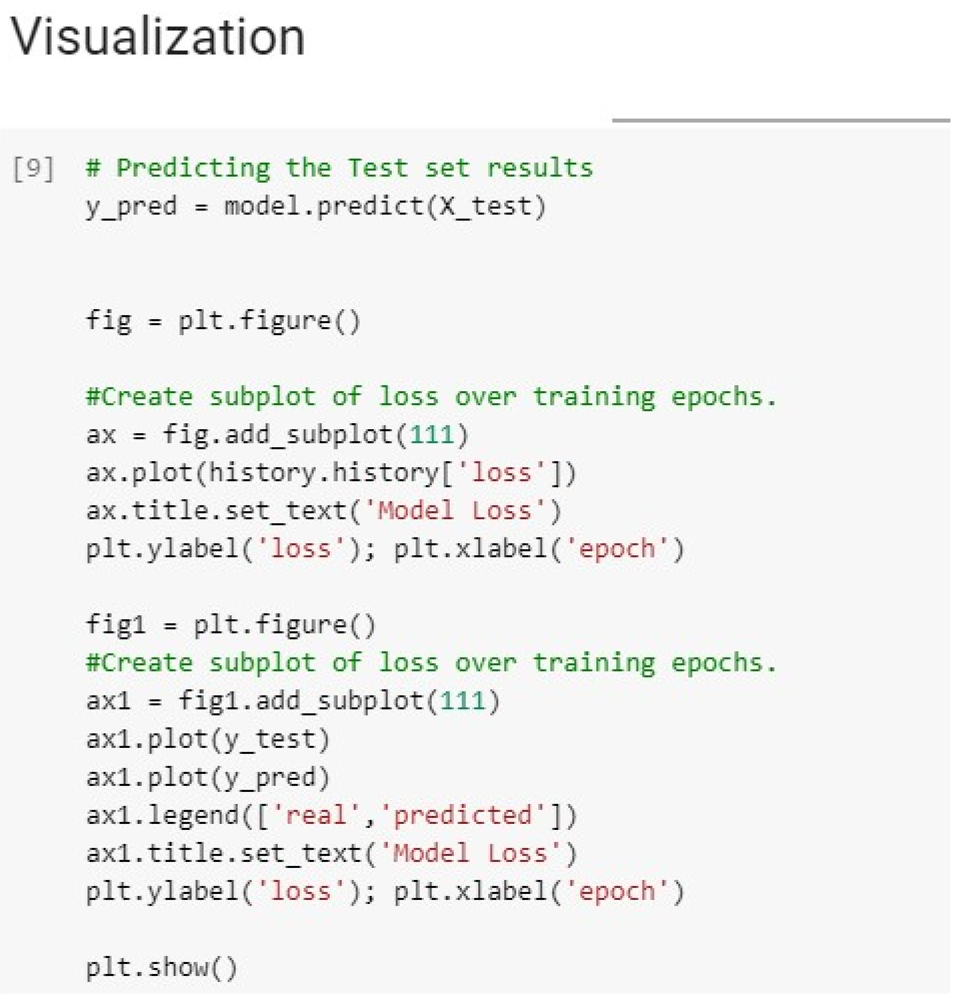
Figure 4: Model Setup Changes

This Model will be suing 50 epochs. 10 is too small to observe the changes and 100 is too many for unnecessary observation:



```
history = model.fit(X_train, y_train, epochs=50, batch_size=10, verbose=1)
```

Figure 5: Model Training code changes



Visualization

```
[9] # Predicting the Test set results
    y_pred = model.predict(X_test)


    fig = plt.figure()

    #Create subplot of loss over training epochs.
    ax = fig.add_subplot(111)
    ax.plot(history.history['loss'])
    ax.title.set_text('Model Loss')
    plt.ylabel('loss'); plt.xlabel('epoch')

    fig1 = plt.figure()
    #Create subplot of loss over training epochs.
    ax1 = fig1.add_subplot(111)
    ax1.plot(y_test)
    ax1.plot(y_pred)
    ax1.legend(['real','predicted'])
    ax1.title.set_text('Model Loss')
    plt.ylabel('loss'); plt.xlabel('epoch')

    plt.show()
```

Figure 6: Changes to Visualization code

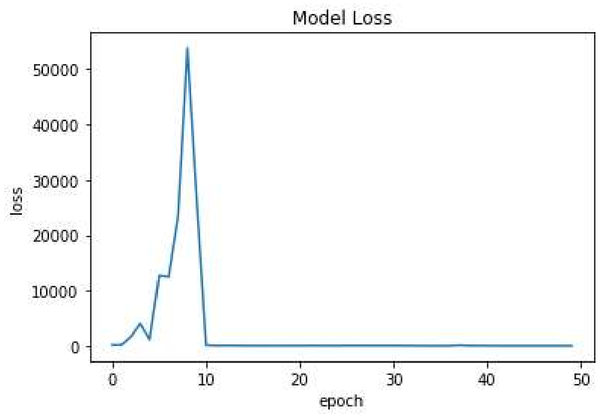


Figure 7(a): Model Loss

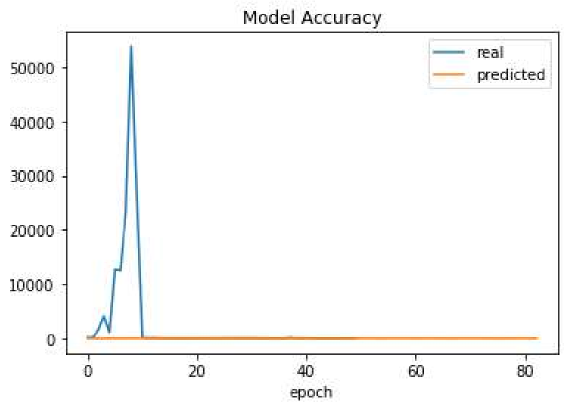


Figure 7(b): Model Accuracy

Task 2: With ReLU:

In the Model Setup code block, change the activation functions of the hidden layer to 'relu' and train both models for the two z functions. How accurate is the regression for each z compared with Task 1 (example of the relu function is below in Appendix A).

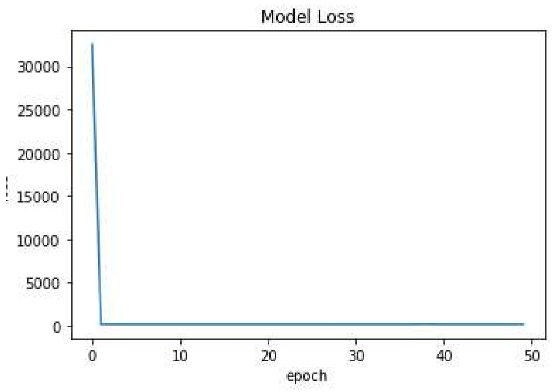


Figure 8(a): Model Loss

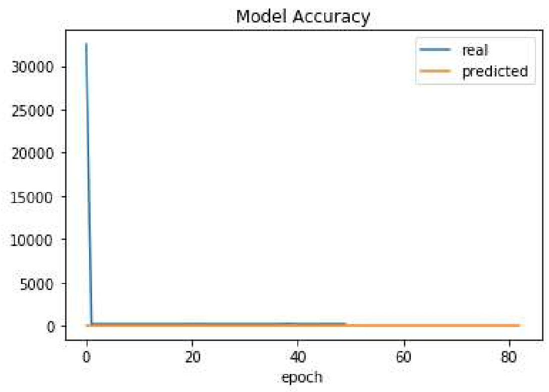


Figure 8(b): Model Accuracy

Task 3: 100 Neurons:

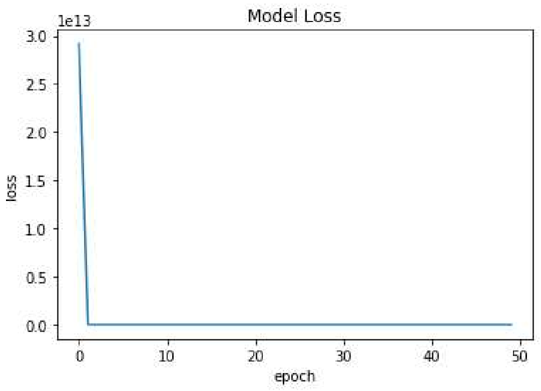


Figure 8(a): Model Loss

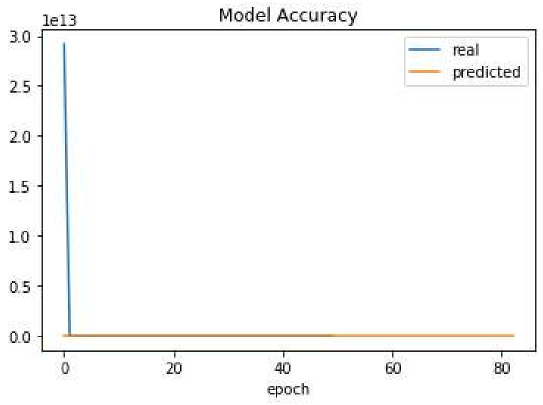


Figure 8(b): Model Accuracy

2. Produce a cross-scatter plot *price*$_{pred}$ *vs price*$_{actual}$. Mark the training points blue and the test point red. Make sure to denormalize the outputs to get actual process on both axes.

   To denormalize the data, just reverse the normalization actions from earlier. Then scatter plot the data where training points are blue and the test points are red:

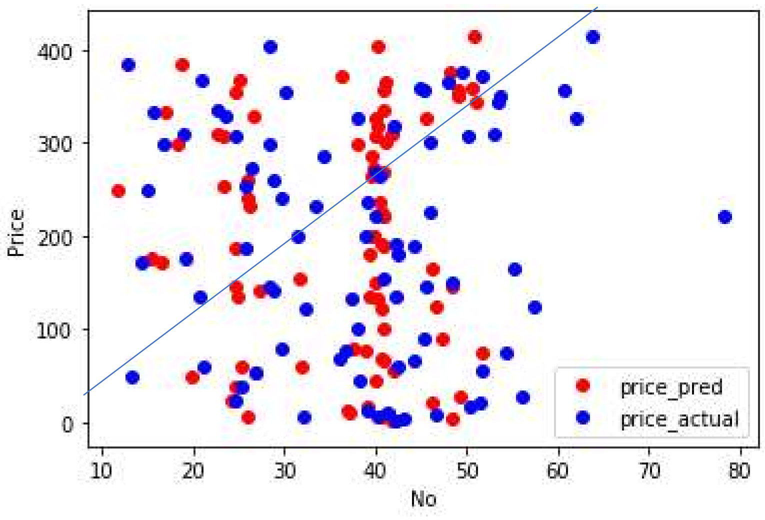
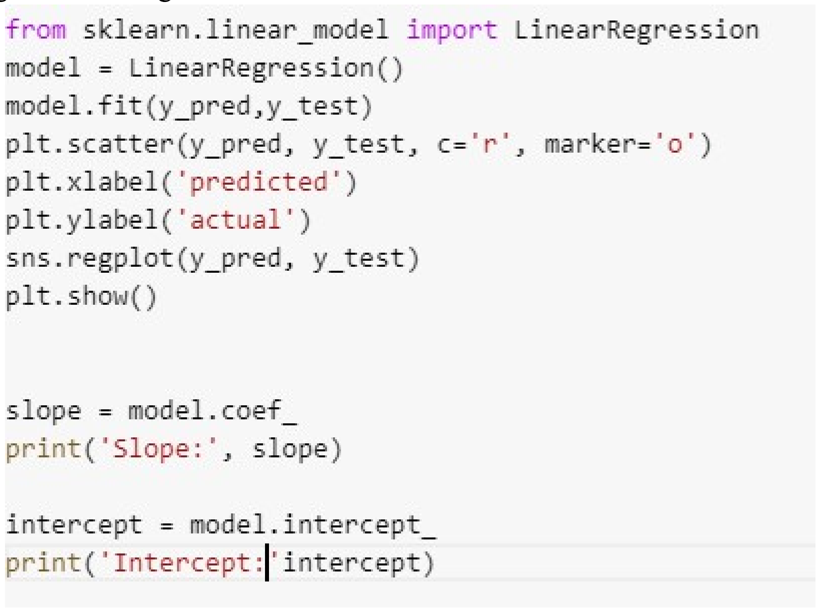


Figure 9: cross-scatter plot *$price_{pred}$ vs $price_{actual}$*

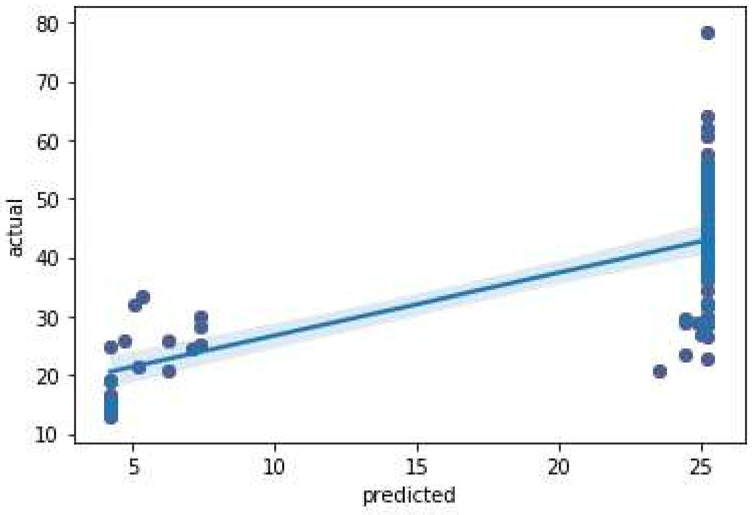
After denormalize, a linear regression modeling of data was added and extracted the slope and intercept using the following code:



```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(y_pred,y_test)
plt.scatter(y_pred, y_test, c='r', marker='o')
plt.xlabel('predicted')
plt.ylabel('actual')
sns.regplot(y_pred, y_test)
plt.show()


slope = model.coef_
print('Slope:', slope)

intercept = model.intercept_
print('Intercept:'intercept)
```

Figure 10: Code used for linear regression of denormalizaed data

3. Find the slope and the intercept of the linear regression line $price_{pred} = m\ price_{actual} + b$ (and draw the line) for the model-produced cross-scatter cloud



```
Slope: [1.0647504]
Intercept: 16.041925
```

Figure 11: Linear Regression, slope and intercept