TAYLOR' S
UNIVERSITY
Wisdom • Integrity • Excellence

Bachelor of Computer Science (Hons)
Bachelor of Software Engineering (Hons)
Bachelor of Information Technology (Hons)

Module Code : ITS66904 (January 2023)

Module Name : Big Data Technologies

Assignment No./Title	**Assignment - Group (30%)**
Course Tutor/Lecturer	Mr. Sham Shul Shukri Mat
Submission Date	26 February 2023 1159pm
Student Name, ID and Signature 1. 2. 3. 4. 5. 6.	

Declaration *(need to be signed by students. Otherwise, the assessment will not be evaluated)*

Certify that this assignment is entirely my own work, except where I have given fully documented references to the work of others, and that the material contained in this assignment has not previously been submitted for assessment in any other formal course of study.

Marks/Grade:	**Evaluated by:**
Evaluator's Comments:	

Table of Contents

Introduction

To complete this task, we will devise a prognostic model from a training corpus containing the pertinent parameters and records that are imperative for the model. We will harness diverse Python modules for analyzing, confirming, and sanitizing the corpus as well as for constructing the prognostic model with the sought-after result. The modules that will be employed include NumPy, Pandas, and KERAS. A neural system will be generated by utilizing the corpus as an input, where an output parameter will be utilized to generate predictions by comparing it to the initial input variables.

Prediction Model Design & Strategy

A predictive model has been developed to anticipate whether the real estate market in Kuala Lumpur is experiencing inflation in the future by utilizing several factors that exhibit strong correlations to the upward or downward trajectory of house prices over a given period. Specifically, the training dataset incorporates seven distinct attributes that enable the system to identify key aspects and characteristics of the housing property and factor in additional variables such as criminal activity in the region. Based on input variables, the model uses a total of 50 training examples that encompass the property attributes of each home to make predictions about the fluctuation of house prices over time.

Data mining execution

Brief Description of attributes

1. **id**: A unique identification number of the houses in the dataset.
2. **current_price**: The price of each of the houses.
3. **elapsed_years**: Indicates how long it has been since a house was built by calculating the elapsed years.
4. **square_feet**: A house is measured by the square feet that make up its total area to determine its size. As a rule of thumb, the larger the area for the construction of a house, the more expensive it will be to build.
5. **num_of_rooms**: In each house, this attribute shows how many bedrooms there are.
6. **crime_rates**: Indicates the occurrence rate of crime on the street around the houses.
7. **expensive**: Indicates whether the houses are expensive or not. It is indicated with Boolean which is 0 or 1 in our dataset.

Five (5) Items'/ Subjects' Details for Prediction Model Application

Using the prediction model, we will be assessing whether the house prices in Kuala Lumpur will be fluctuating or not based on the necessary attributes. The 5 subjects are described below:

Subject	id	Current_price	Elapsed_years	Square_feet	Num_of_rooms	Crime_rates	Expensive
1	13	466,300	9	1637		.34	0
2	14	320,800	10	1622	4	.37	1
3	30	500,000	8	960	3	.07	1
4	33	7,200,250	2	1400		.03	1
5	50	250,000	3	2500	3	.43	1

Subject 1: The current price of this house is RM 466,300 and 9 years have passed since it was built. The area of this house is 1637 square feet. but the number of rooms is not told, and it is vacant. The crime is also high at 34%. Even though this house is not expensive, it is less likely to fluctuate, and it is one of the dirty data of the dataset that needs to be cleaned.

Subject 2: The current price of this house is RM 320,000 and it has been 10 years since it was built. It is 1622 square feet, and the number of rooms is 4. The crime rate of the area of which this house is located is high at 37% and on top of that it is an expensive house. So, the prediction is that this house will not fluctuate.

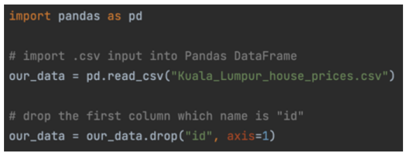
Subject 3: The current price of this house is 500,000 and it has been 8 years since it was built. It is 960 square feet and the number of bedrooms of this house is 3. The crime rate is very low in this location which on only 7%. Although this is an expensive house, the prediction is that it will inflate even more given the crime rate is very low.

Subject 4: This is a very expensive house priced at RM 7,200,250 and it is comparatively a new house as 2 years have elapsed since it was built. It is 1400 square feet. The number of rooms is not present. So, it is one of those dirty data of the datasets which needs to be cleaned.

Subject 5: This house is priced at RM 250,000. It is also comparatively new house. Its area is also large at 2500 square feet. Number of rooms is 3. The crime rate in its location is very high at 43% and on top of that it is an expensive house. So, the prediction is it is less likely to fluctuate.

Import the dirty datasets with the Pandas library

In the first step, we import the Pandas library and assigns it an alias 'pd' secondly, we loaded our dirty dataset csv file with Pandas DataFrame using the following code:

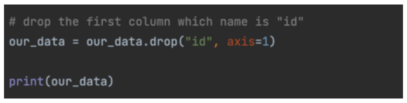


```python
import pandas as pd

# import .csv input into Pandas DataFrame
our_data = pd.read_csv("Kuala_Lumpur_house_prices.csv")

# drop the first column which name is "id"
our_data = our_data.drop("id", axis=1)
```

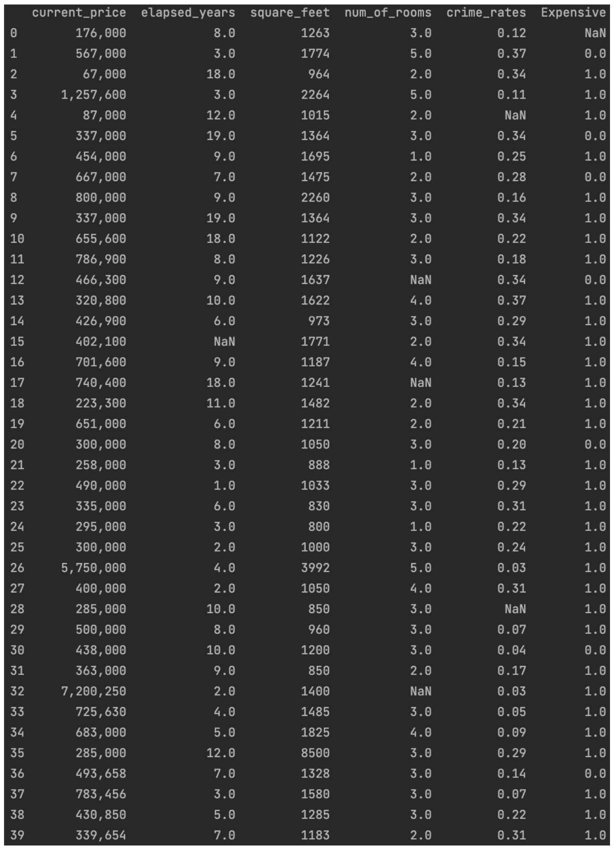
Our data set has an ID column which is not relevant, so we are going to drop as shown below which leaves us with the final Dirty Dataset that we are going to work with-



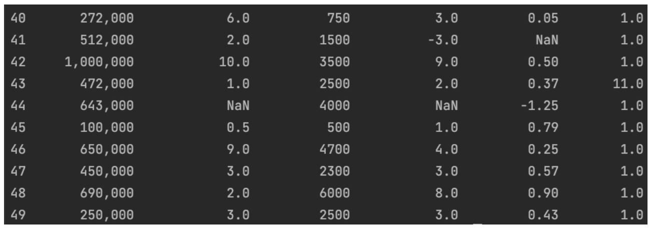
```python
# drop the first column which name is "id"
our_data = our_data.drop("id", axis=1)

print(our_data)
```

Then we got the output below:



```
   current_price  elapsed_years  square_feet  num_of_rooms  crime_rates  Expensive
0        176,000            8.0         1263           3.0         0.12        NaN
1        567,000            3.0         1774           5.0         0.37        0.0
2         67,000           18.0          964           2.0         0.34        1.0
3      1,257,600            3.0         2264           5.0         0.11        1.0
4         87,000           12.0         1015           2.0          NaN        1.0
5        337,000           19.0         1364           3.0         0.34        0.0
6        454,000            9.0         1695           1.0         0.25        1.0
7        667,000            7.0         1475           2.0         0.28        0.0
8        800,000            9.0         2260           3.0         0.16        1.0
9        337,000           19.0         1364           3.0         0.34        1.0
10       655,600           18.0         1122           2.0         0.22        1.0
11       786,900            8.0         1226           3.0         0.18        1.0
12       466,300            9.0         1637           NaN         0.34        0.0
13       320,800           10.0         1622           4.0         0.37        1.0
14       426,900            6.0          973           3.0         0.29        1.0
15       402,100            NaN         1771           2.0         0.34        1.0
16       701,600            9.0         1187           4.0         0.15        1.0
17       740,400           18.0         1241           NaN         0.13        1.0
18       223,300           11.0         1482           2.0         0.34        1.0
19       651,000            6.0         1211           2.0         0.21        1.0
20       300,000            8.0         1050           3.0         0.20        0.0
21       258,000            3.0          888           1.0         0.13        1.0
22       490,000            1.0         1033           3.0         0.29        1.0
23       335,000            6.0          830           3.0         0.31        1.0
24       295,000            3.0          800           1.0         0.22        1.0
25       300,000            2.0         1000           3.0         0.24        1.0
26     5,750,000            4.0         3992           5.0         0.03        1.0
27       400,000            2.0         1050           4.0         0.31        1.0
28       285,000           10.0          850           3.0          NaN        1.0
29       500,000            8.0          960           3.0         0.07        1.0
30       438,000           10.0         1200           3.0         0.04        0.0
31       363,000            9.0          850           2.0         0.17        1.0
32     7,200,250            2.0         1400           NaN         0.03        1.0
33       725,630            4.0         1485           3.0         0.05        1.0
34       683,000            5.0         1825           4.0         0.09        1.0
35       285,000           12.0         8500           3.0         0.29        1.0
36       493,658            7.0         1328           3.0         0.14        0.0
37       783,456            3.0         1580           3.0         0.07        1.0
38       430,850            5.0         1285           3.0         0.22        1.0
39       339,654            7.0         1183           2.0         0.31        1.0
```

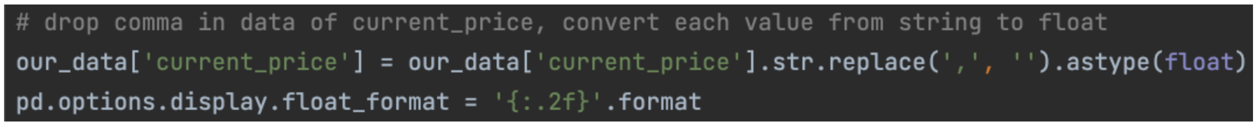


```
40      272,000          6.0       750        3.0      0.05      1.0
41      512,000          2.0      1500       -3.0       NaN      1.0
42    1,000,000         10.0      3500        9.0      0.50      1.0
43      472,000          1.0      2500        2.0      0.37     11.0
44      643,000          NaN      4000        NaN     -1.25      1.0
45      100,000          0.5       500        1.0      0.79      1.0
46      650,000          9.0      4700        4.0      0.25      1.0
47      450,000          3.0      2300        3.0      0.57      1.0
48      690,000          2.0      6000        8.0      0.90      1.0
49      250,000          3.0      2500        3.0      0.43      1.0
```

Eventually, the data has been successfully loaded into the Pandas DataFrame from the .csv file.

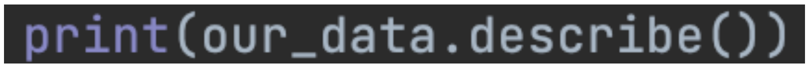
Convert String to Float and Validate the Data

Since the "current_price" column contains string values with commas, Pandas would recognize it as an object or string data type, rather than a numerical data type. Therefore, we need to convert the column to a float data type. We use the following code to convert the "current_price" column data from string to float:



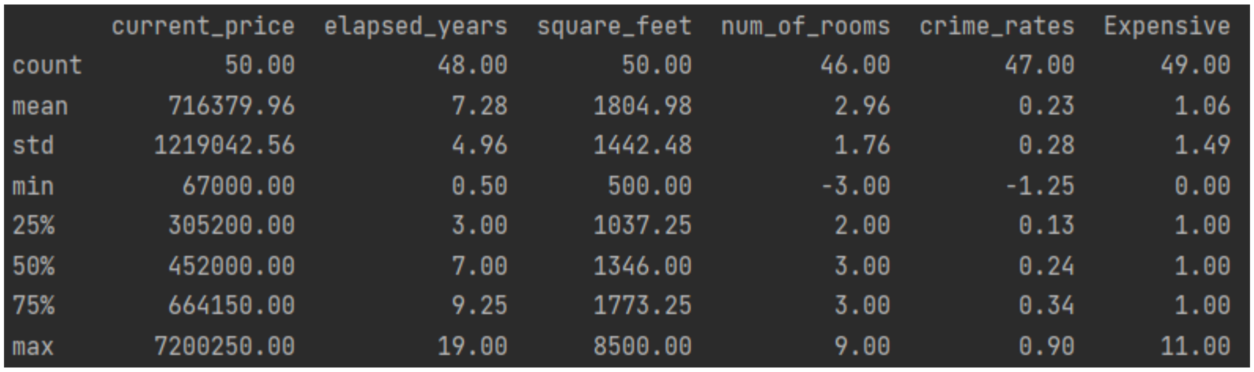
```
# drop comma in data of current_price, convert each value from string to float
our_data['current_price'] = our_data['current_price'].str.replace(',', '').astype(float)
pd.options.display.float_format = '{:.2f}'.format
```

Then we would use describe() to print a table with summary statistics for each numeric column in the DataFrame. It's a useful way to quickly understand the distribution and range of values in your data.
After that, we use the following code to check any invalid entries in our dataset:



```
print(our_data.describe())
```

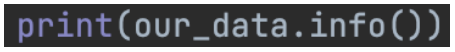
Then we got the output below:



```
      current_price  elapsed_years  square_feet  num_of_rooms  crime_rates  Expensive
count         50.00          48.00        50.00         46.00        47.00      49.00
mean      716379.96           7.28      1804.98          2.96         0.23       1.06
std      1219042.56           4.96      1442.48          1.76         0.28       1.49
min        67000.00           0.50       500.00         -3.00        -1.25       0.00
25%       305200.00           3.00      1037.25          2.00         0.13       1.00
50%       452000.00           7.00      1346.00          3.00         0.24       1.00
75%       664150.00           9.25      1773.25          3.00         0.34       1.00
max      7200250.00          19.00      8500.00          9.00         0.90      11.00
```

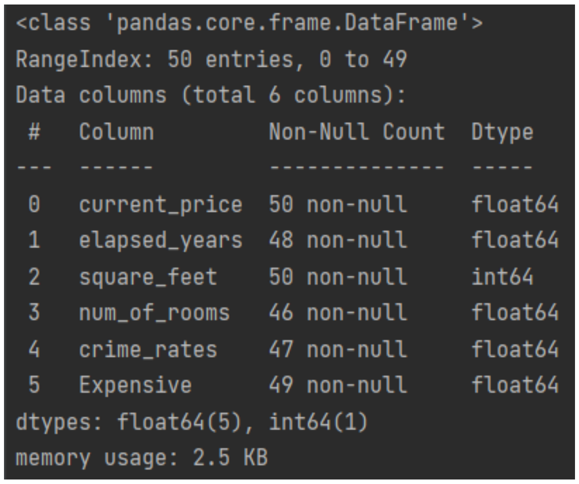
Find Dirty Data

We use the following code to determine the existing data types of our dataset:



```
print(our_data.info())
```

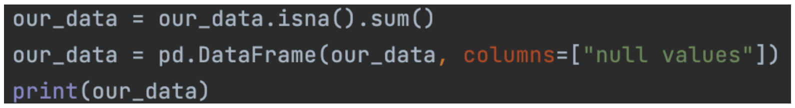
Then we got the output below:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   current_price  50 non-null     float64
 1   elapsed_years  48 non-null     float64
 2   square_feet    50 non-null     int64  
 3   num_of_rooms   46 non-null     float64
 4   crime_rates    47 non-null     float64
 5   Expensive      49 non-null     float64
dtypes: float64(5), int64(1)
memory usage: 2.5 KB
```

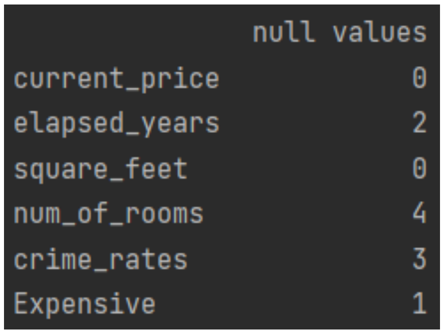
From the preceding output, we can perceive the various data formats employed in our dataset. Furthermore, we can notice that there are fewer non-empty values than anticipated, indicating that the dataset contains some unclean data, i.e., null values.

Then, we use the following code to determine the validate the existence of null values in our dataset:



```
our_data = our_data.isna().sum()
our_data = pd.DataFrame(our_data, columns=["null values"])
print(our_data)
```

Then we got the output below:



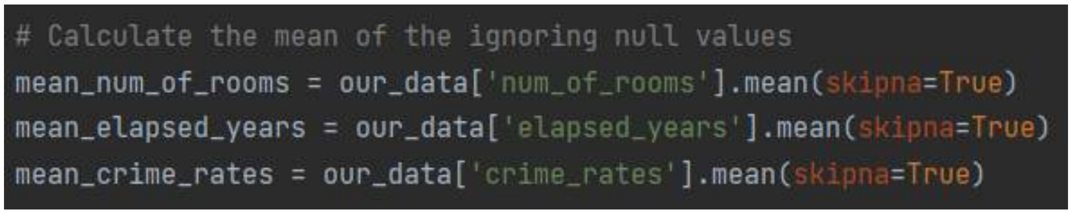
```
               null values
current_price            0
elapsed_years            2
square_feet              0
num_of_rooms             4
crime_rates              3
Expensive                1
```

From the output, we can observe the number of null values i.e., corrupted data in our training dataset. There are only 2 columns with complete data and the other 4 has some missing values. As a result, we can deduce that the dataset contains some soiled data. Consequently, we proceed to purify the dataset.

It’s also worth mentioning that ‘num_of_rooms’ and ‘crime_rates’ column has some outlier as shown on “our_data.discribe()” method output. They both have negative values which cannot be possible as the number of rooms in a house must be more than one and the crime rate can only go down to 0 indicating they are outliers in the dataset.

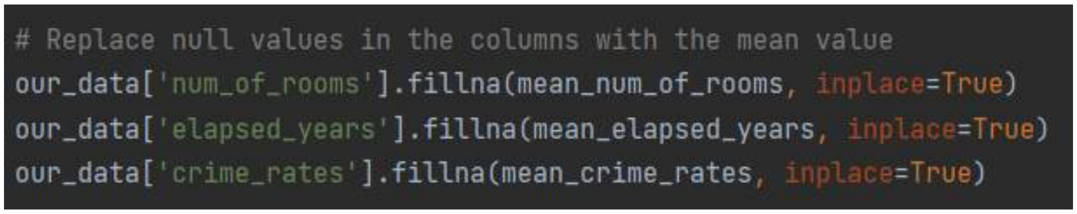
Clean Dirty Data

First, we focus on the null values in the dataset. To replace the null values of the three features ‘num_of_rooms’, ‘elapsed_years’, and ‘crime_rates’, we first calculate the mean values for these three features without considering the null values.



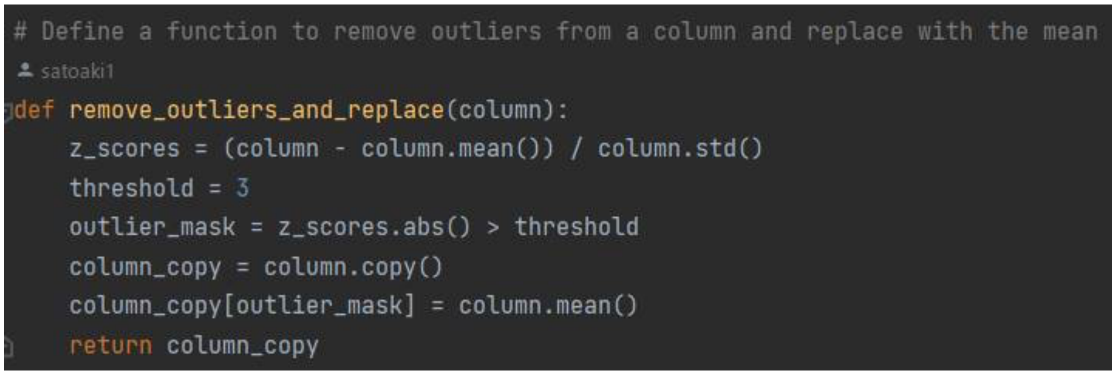
```python
# Calculate the mean of the ignoring null values
mean_num_of_rooms = our_data['num_of_rooms'].mean(skipna=True)
mean_elapsed_years = our_data['elapsed_years'].mean(skipna=True)
mean_crime_rates = our_data['crime_rates'].mean(skipna=True)
```

Then, we run the following code to replace the null values with the mean we received.



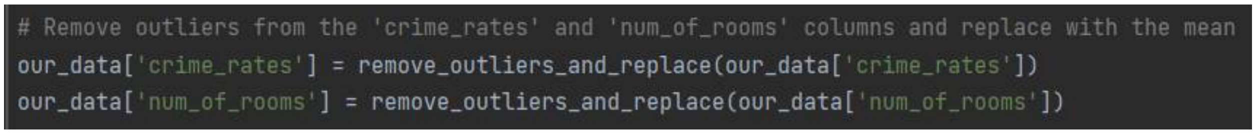
```python
# Replace null values in the columns with the mean value
our_data['num_of_rooms'].fillna(mean_num_of_rooms, inplace=True)
our_data['elapsed_years'].fillna(mean_elapsed_years, inplace=True)
our_data['crime_rates'].fillna(mean_crime_rates, inplace=True)
```

After replacing the null values, we still must deal with the outliers. As for our case, we created the function below to replace the outliers with the mean we received earlier.



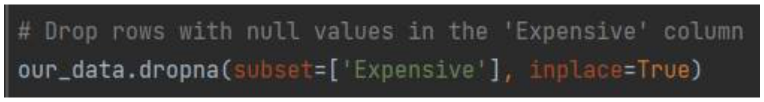
```python
# Define a function to remove outliers from a column and replace with the mean
def remove_outliers_and_replace(column):
    z_scores = (column - column.mean()) / column.std()
    threshold = 3
    outlier_mask = z_scores.abs() > threshold
    column_copy = column.copy()
    column_copy[outlier_mask] = column.mean()
    return column_copy
```

We replaced the outliers for both the features ‘crime_rates’ and ‘num_of_rooms’.



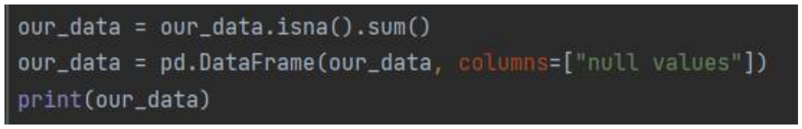
```python
# Remove outliers from the 'crime_rates' and 'num_of_rooms' columns and replace with the mean
our_data['crime_rates'] = remove_outliers_and_replace(our_data['crime_rates'])
our_data['num_of_rooms'] = remove_outliers_and_replace(our_data['num_of_rooms'])
```

As for the column ‘Expensive’, we decided to drop the null values instead of replacing them with mean as the ‘Expensive’ is the output we are trying to predict, so we did not consider replacing it with the mean value.



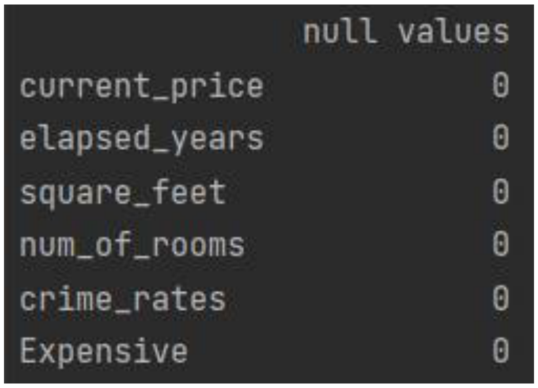
```
# Drop rows with null values in the 'Expensive' column
our_data.dropna(subset=['Expensive'], inplace=True)
```

After cleaning the null values and dirty data, we run the following codes to check for any null values left in the dataset.



```
our_data = our_data.isna().sum()
our_data = pd.DataFrame(our_data, columns=["null values"])
print(our_data)
```

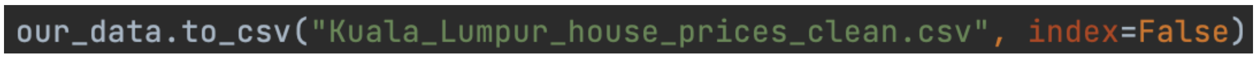
Then, we got the below output:



```
               null values
current_price            0
elapsed_years            0
square_feet              0
num_of_rooms             0
crime_rates              0
Expensive                0
```

Therefore, from the above output, we can determine the data has been cleaned.

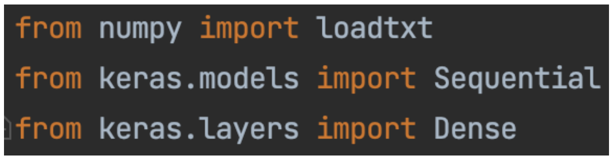
After we determine the data has indeed been cleaned, we save it into a new .csv file called “Kuala_Lumpur_house_prices_clean.csv”.



```
our_data.to_csv("Kuala_Lumpur_house_prices_clean.csv", index=False)
```

Developing the Prediction Model

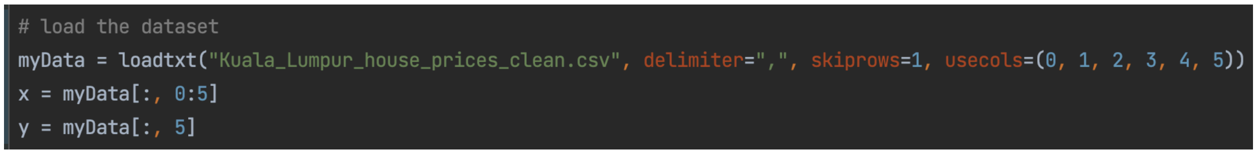
After cleaning our dataset, we have an appropriate dataset with proper entries that can now be used for making the prediction model. We import ‘loadtxt’ from NumPy package, ‘Sequential’ class, and ‘Dense’ class from Keras package.



```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
```

Firstly, we load the dataset from the clean .csv file using NumPy, and we drop the “id” column as it is irrelevant to the training dataset as it is a unique identifier for the records. We use the following to do the process

Then, the dataset is split into two sections i.e., x and y where x is the feature matrix and y are the output variable respectively.

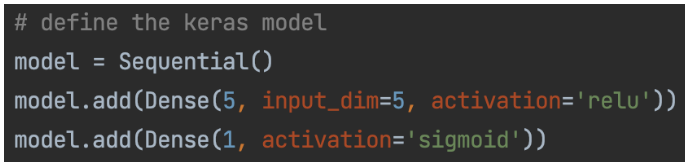


```
# load the dataset
myData = loadtxt("Kuala_Lumpur_house_prices_clean.csv", delimiter=",", skiprows=1, usecols=(0, 1, 2, 3, 4, 5))
x = myData[:, 0:5]
y = myData[:, 5]
```

Now, we build the prediction model using neural networks. We use the Sequential model for the neural networks and use three Dense layers having a certain number of links which are determined through testing. We must use an appropriate number of links and layers to ensure a balance between the prediction model accuracy and the time it takes to compute said model.

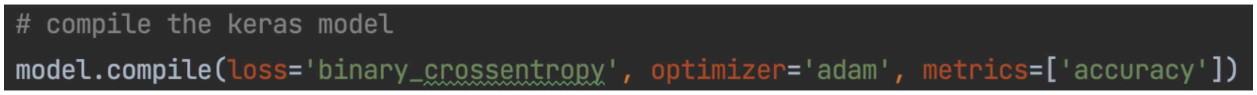
Therefore, we use 5 layers in the first Dense layer, and 1 link in the second Dense layer.

For the activation type, we use ‘relu’ for the first two layers and “sigmoid” for the third layer.



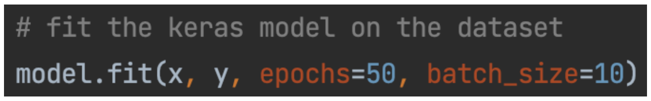
```
# define the keras model
model = Sequential()
model.add(Dense(5, input_dim=5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Our output parameters are to be defined with appropriate variables. For the "loss" parameter, we use "binary cross entropy" as the result can be either 0 or 1. Moving on, we use "Adam' mathematical model as our optimizer. And, lastly, to be able to see the accuracy, we use the "metrics" parameter.



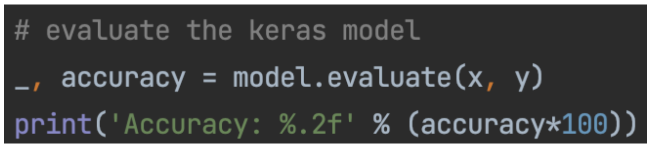
```
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

We fit the model to the dataset. The "epochs" parameter defines the number of runs the model will go through for training purposes. Now, a higher "epochs" number usually yields higher accuracy but also requires more computational power and time. Therefore, we need to have a balance between accuracy and the compute time. And the "batch_size" parameter describes the input size of the model where 10 is used. Therefore, it would take 10 inputs at a time for training purposes.



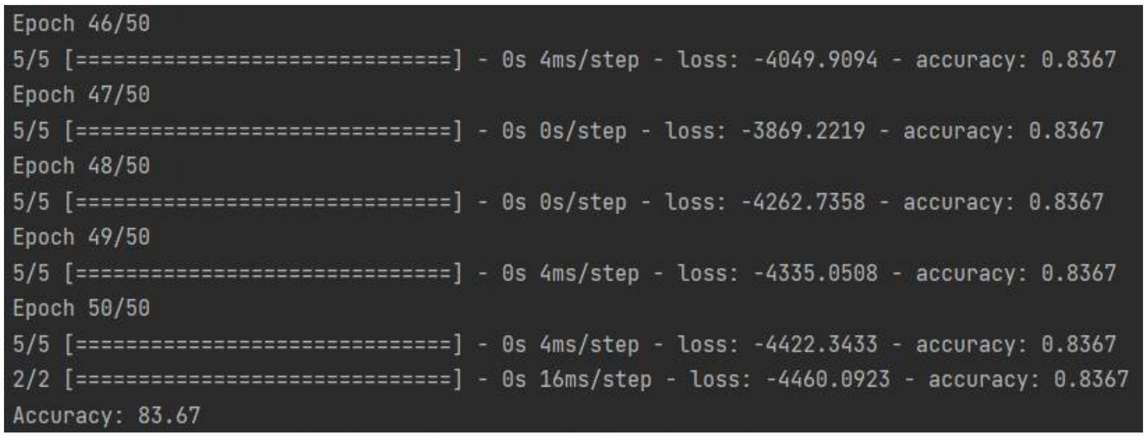
```
# fit the keras model on the dataset
model.fit(x, y, epochs=50, batch_size=10)
```

Now, we can move onto evaluating the prediction model and determining its accuracy.



```
# evaluate the keras model
_, accuracy = model.evaluate(x, y)
print('Accuracy: %.2f' % (accuracy*100))
```

Finally, we get the accuracy of the prediction model as our output.

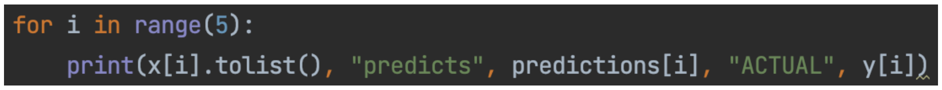


```
Epoch 46/50
5/5 [==============================] - 0s 4ms/step - loss: -4049.9094 - accuracy: 0.8367
Epoch 47/50
5/5 [==============================] - 0s 0s/step - loss: -3869.2219 - accuracy: 0.8367
Epoch 48/50
5/5 [==============================] - 0s 0s/step - loss: -4262.7358 - accuracy: 0.8367
Epoch 49/50
5/5 [==============================] - 0s 4ms/step - loss: -4335.0508 - accuracy: 0.8367
Epoch 50/50
5/5 [==============================] - 0s 4ms/step - loss: -4422.3433 - accuracy: 0.8367
2/2 [==============================] - 0s 16ms/step - loss: -4460.0923 - accuracy: 0.8367
Accuracy: 83.67
```

Therefore, the accuracy of our model is approximately 84% which concludes that our prediction model is precise and reliable.

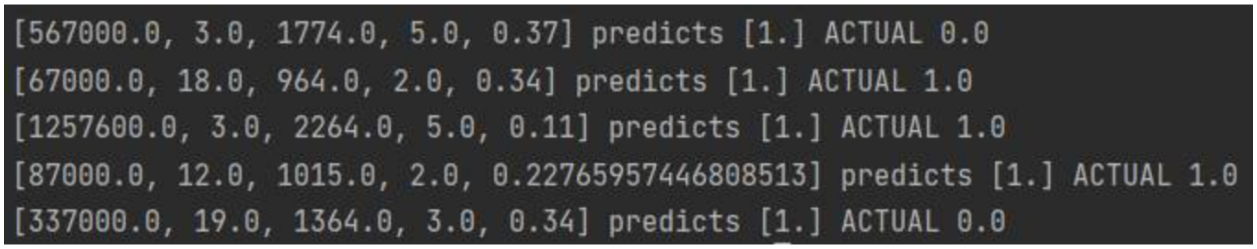
Prediction Model Application

We use our completed prediction model to predict values for our dataset. And we use the following code to do it:



```
for i in range(5):
    print(x[i].tolist(), "predicts", predictions[i], "ACTUAL", y[i])
```

And we get the following output:



```
[567000.0, 3.0, 1774.0, 5.0, 0.37] predicts [1.] ACTUAL 0.0
[67000.0, 18.0, 964.0, 2.0, 0.34] predicts [1.] ACTUAL 1.0
[1257600.0, 3.0, 2264.0, 5.0, 0.11] predicts [1.] ACTUAL 1.0
[87000.0, 12.0, 1015.0, 2.0, 0.22765957446808513] predicts [1.] ACTUAL 1.0
[337000.0, 19.0, 1364.0, 3.0, 0.34] predicts [1.] ACTUAL 0.0
```

From the output, we can see the prediction model is showing 5 records where it tries to predict the values using its trained model and shows a value that is closer to 1. Therefore, we can conclude that our prediction model is predicting the correct values for the dataset.

Python Source Codes

“clean_data.py”

```
import pandas as pd

# import .csv input into Pandas DataFrame
our_data = pd.read_csv("Kuala_Lumpur_house_prices.csv")

# drop the first column which name is "id"
our_data = our_data.drop("id", axis=1)

# drop comma in data of current_price, convert each value from string to float
our_data['current_price'] = our_data['current_price'].str.replace(',', '').astype(float)
pd.options.display.float_format = '{:.2f}'.format

print(our_data.info())

print(our_data.describe())

# Calculate the mean of the ignoring null values
mean_num_of_rooms = our_data['num_of_rooms'].mean(skipna=True)
mean_elapsed_years = our_data['elapsed_years'].mean(skipna=True)
mean_crime_rates = our_data['crime_rates'].mean(skipna=True)

# Replace null values in the columns with the mean value
our_data['num_of_rooms'].fillna(mean_num_of_rooms, inplace=True)
our_data['elapsed_years'].fillna(mean_elapsed_years, inplace=True)
our_data['crime_rates'].fillna(mean_crime_rates, inplace=True)


# Define a function to remove outliers from a column and replace with the mean
def remove_outliers_and_replace(column):
    z_scores = (column - column.mean()) / column.std()
    threshold = 3
    outlier_mask = z_scores.abs() > threshold
    column_copy = column.copy()
    column_copy[outlier_mask] = column.mean()
    return column_copy


# Remove outliers from the 'crime_rates' and 'num_of_rooms' columns and replace with the mean
our_data['crime_rates'] = remove_outliers_and_replace(our_data['crime_rates'])
our_data['num_of_rooms'] = remove_outliers_and_replace(our_data['num_of_rooms'])

# Write the updated DataFrame back to a new CSV file
our_data.to_csv('new_our_data.csv', index=False)

# Drop rows with null values in the 'Expensive' column
our_data.dropna(subset=['Expensive'], inplace=True)

# Write the updated DataFrame back to a new CSV file
our_data.to_csv('Kuala_Lumpur_house_prices_clean.csv', index=False)

print(our_data.info())
print(our_data.describe())

our_data = our_data.isna().sum()
our_data = pd.DataFrame(our_data, columns=["null values"])
print(our_data)
```

“predict.py”

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
myData = loadtxt("Kuala_Lumpur_house_prices_clean.csv",
                 delimiter=",",
                 skiprows=1,
                 usecols=(0, 1, 2, 3, 4, 5))
x = myData[:, 0:5]
y = myData[:, 5]

# define the keras model
model = Sequential()
model.add(Dense(5, input_dim=5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# fit the keras model on the dataset
model.fit(x, y, epochs=50, batch_size=10)

# evaluate the keras model
_, accuracy = model.evaluate(x, y)
print('Accuracy: %.2f' % (accuracy*100))

predictions = model.predict_on_batch(x)

for i in range(5):
    print(x[i].tolist(), "predicts", predictions[i], "ACTUAL", y[i])
```