1.Preprocessing the Dataset

```
#1.Importing required libraries
import pandas as pd
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt


#2.Loading the dataset
data = pd.read_csv('/content/House Price India.csv')
print(data)
```

```
                 id   Date  number of bedrooms  number of bathrooms  \
0      6762810145  42491                   5                 2.50
1      6762810635  42491                   4                 2.50
2      6762810998  42491                   5                 2.75
3      6762812605  42491                   4                 2.50
4      6762812919  42491                   3                 2.00
...           ...    ...                 ...                  ...
14615  6762830250  42734                   2                 1.50
14616  6762830339  42734                   3                 2.00
14617  6762830618  42734                   2                 1.00
14618  6762830709  42734                   4                 1.00
14619  6762831463  42734                   3                 1.00

       living area  lot area  number of floors  waterfront present  \
0             3650      9050               2.0                   0
1             2920      4000               1.5                   0
2             2910      9480               1.5                   0
3             3310     42998               2.0                   0
4             2710      4500               1.5                   0
...            ...       ...               ...                 ...
14615         1556     20000               1.0                   0
14616         1680      7000               1.5                   0
14617         1070      6120               1.0                   0
14618         1030      6621               1.0                   0
14619          900      4770               1.0                   0

       number of views  condition of the house  ...  Built Year  \
0                    4                       5  ...        1921
1                    0                       5  ...        1909
2                    0                       3  ...        1939
3                    0                       3  ...        2001
4                    0                       4  ...        1929
...                ...                     ...  ...         ...
14615                0                       4  ...        1957
14616                0                       4  ...        1968
14617                0                       3  ...        1962
14618                0                       4  ...        1955
14619                0                       3  ...        1969

       Renovation Year  Postal Code  Lattitude  Longitude  living_area_renov  \
0                    0       122003    52.8645   -114.557               2880
1                    0       122004    52.8878   -114.470               2470
2                    0       122004    52.8852   -114.468               2940
3                    0       122005    52.9532   -114.321               3350
4                    0       122006    52.9047   -114.485               2060
...                ...          ...        ...        ...                ...
14615                0       122066    52.6191   -114.472               2250
14616                0       122072    52.5075   -114.393               1540
14617                0       122056    52.7289   -114.507               1130
14618                0       122042    52.7157   -114.411               1420
14619             2009       122018    52.5338   -114.552                900

       lot_area_renov  Number of schools nearby  Distance from the airport  \
0                5400                         2                         58
1                4000                         2                         51
2                6600                         1                         53
3               42847                         3                         76
4                4500                         1                         51
```

```
#3.task assigned in dataset
data['number of bathrooms'] = data['number of bathrooms'].astype(int)
data['number of bedrooms'] = data['number of bedrooms'].astype(int)
data['waterfront present'] = data['waterfront present'].astype(int)
data = data.drop("Date" , axis=1)
data.head()
```

	id	number of bedrooms	number of bathrooms	living area	lot area	number of floors	waterfront present	number of views	con
0	6762810145	5	2	3650	9050	2.0	0	4	
1	6762810635	4	2	2920	4000	1.5	0	0	
2	6762810998	5	2	2910	9480	1.5	0	0	
3	6762812605	4	2	3310	42998	2.0	0	0	
4	6762812919	3	2	2710	4500	1.5	0	0	

5 rows × 22 columns

```
#4.Checking missing values
print(data.isnull().any())#there is no missing  values
```

```
id                                    False
number of bedrooms                    False
number of bathrooms                   False
living area                           False
lot area                              False
number of floors                      False
waterfront present                    False
number of views                       False
condition of the house                False
grade of the house                    False
Area of the house(excluding basement)    False
Area of the basement                  False
Built Year                            False
Renovation Year                       False
Postal Code                           False
Lattitude                             False
Longitude                             False
living_area_renov                     False
lot_area_renov                        False
Number of schools nearby              False
Distance from the airport             False
Price                                 False
dtype: bool
```

```
#5.Encoding the categorical variables using one-hot encoding
categorical_cols = data.select_dtypes(include=['object']).columns
print(categorical_cols)#there is no categorical variables
```

```
Index([], dtype='object')
```

```
#6.Spliting the dataset into training and testing sets
X = data.iloc[:, :-1].values # Features
y = data.iloc[:, -1].values # Target Variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

2.Building the ANN Model

```
model = Sequential()
#a.Input Layer
model.add(Dense(128, input_dim=X.shape[1], activation='relu'))
#b.Min of 2 hidden layers
model.add(Dense(64, activation='relu')) # Hidden Layer 1
model.add(Dense(32, activation='relu')) # Hidden Layer 2
#c.Output Layer
model.add(Dense(1))
#Compiling the model
model.compile(loss='mean_squared_error', optimizer='adam')
#Training the model
model.fit(X_train, y_train, epochs=100)
```

```
Epoch 1/100
366/366 [==============================] - 2s 2ms/step - loss: 260774861733888.0000
Epoch 2/100
366/366 [==============================] - 1s 2ms/step - loss: 200320696320.0000
Epoch 3/100
366/366 [==============================] - 1s 2ms/step - loss: 284910583808.0000
Epoch 4/100
366/366 [==============================] - 1s 2ms/step - loss: 556119425024.0000
Epoch 5/100
366/366 [==============================] - 1s 2ms/step - loss: 18878560206848.0000
```

```
Epoch 6/100
366/366 [==============================] - 1s 2ms/step - loss: 9042706563072.0000
Epoch 7/100
366/366 [==============================] - 1s 2ms/step - loss: 216702423924736.0000
Epoch 8/100
366/366 [==============================] - 1s 3ms/step - loss: 195402973184.0000
Epoch 9/100
366/366 [==============================] - 1s 3ms/step - loss: 205263044608.0000
Epoch 10/100
366/366 [==============================] - 1s 3ms/step - loss: 218544439296.0000
Epoch 11/100
366/366 [==============================] - 1s 2ms/step - loss: 242096783360.0000
Epoch 12/100
366/366 [==============================] - 1s 2ms/step - loss: 272602202112.0000
Epoch 13/100
366/366 [==============================] - 1s 2ms/step - loss: 532396048384.0000
Epoch 14/100
366/366 [==============================] - 1s 2ms/step - loss: 3070672764928.0000
Epoch 15/100
366/366 [==============================] - 1s 2ms/step - loss: 5831012122624.0000
Epoch 16/100
366/366 [==============================] - 1s 2ms/step - loss: 5214868865024.0000
Epoch 17/100
366/366 [==============================] - 1s 2ms/step - loss: 4505096683520.0000
Epoch 18/100
366/366 [==============================] - 1s 2ms/step - loss: 3972480106496.0000
Epoch 19/100
366/366 [==============================] - 1s 2ms/step - loss: 3088315318272.0000
Epoch 20/100
366/366 [==============================] - 1s 2ms/step - loss: 109353541566464.0000
Epoch 21/100
366/366 [==============================] - 1s 2ms/step - loss: 197107302400.0000
Epoch 22/100
366/366 [==============================] - 1s 2ms/step - loss: 229114281984.0000
Epoch 23/100
366/366 [==============================] - 1s 2ms/step - loss: 322975432704.0000
Epoch 24/100
366/366 [==============================] - 1s 2ms/step - loss: 439197663232.0000
Epoch 25/100
366/366 [==============================] - 1s 3ms/step - loss: 944752099328.0000
Epoch 26/100
366/366 [==============================] - 1s 3ms/step - loss: 2177487077376.0000
Epoch 27/100
366/366 [==============================] - 1s 2ms/step - loss: 2834713018368.0000
Epoch 28/100
366/366 [==============================] - 1s 2ms/step - loss: 2813497966592.0000
Epoch 29/100
366/366 [==============================] - 1s 2ms/step - loss: 2636359139328.0000
```

3.Testing the Model

```
#Using MSE
y_pred = model.predict(X_test)
mse = np.mean((y_pred-y_test)**2)
print("Mean Squared Error:", mse)
```

```
92/92 [==============================] - 0s 1ms/step
Mean Squared Error: 125172501348.40625
```

```
# Making predictions on test data
test_data = X[:10]
predictions = model.predict(test_data)
```

```
1/1 [==============================] - 0s 22ms/step
```

```
# Visualizing predicted values against actual values
sns.scatterplot(x=y[:10], y=predictions.flatten(),color='cyan')
plt.xlabel('Actual prices')
plt.ylabel('Predicted prices')
plt.show()
```