BANGALORE HOUSE PRICE PREDICTION

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Buying a home, especially in a city like Bengaluru, is a tricky choice. While the major factors are usually the same for all metros, there are others to be considered for the Silicon Valley of India. With its help millennial crowd, vibrant culture, great climate, and a slew of job opportunities, it is difficult to ascertain the price of a house in Bengaluru.

By cleaning and analyzing the Bangalore home dataset, we will be able to understand how house prices are affected by various factors, and then apply machine learning regression to establish an approximate price for the properties. In this work, I presented a study approach that utilizes a variety of methods (Logistic Regression, K-Folds, and DNN) to predict the houses price, and the deep learning model agreed on the others with an accuracy of 91.2 %.

1. PROPLEM STATEMENT

What are the things that a potential home buyer considers before purchasing a house? The location, the size of the property, vicinity to offices, schools, parks, restaurants, hospitals or the stereotypical white picket fence? What about the most important factor — the price?

Now with the lingering impact of demonetization, the enforcement of the Real Estate (Regulation and Development) Act (RERA), and the lack of trust in property developers in the city, housing units sold across India in 2017 dropped by 7 percent. In fact, the property prices in Bengaluru fell by almost 5 percent in the second half of 2017, said a study published by property consultancy Knight Frank.

For example, for a potential homeowner, over 9,000 apartment projects and flats for sale are available in the range of ₹42-52 lakh, followed by over 7,100 apartments that are in the ₹52-62 lakh budget segment, says a report by property website Makana. According to the study, there are over 5,000 projects in the ₹15-25 lakh budget segment followed by those in the ₹34-43 lakh budget category.

2. PROPOSED SOLUTION

Cleaning and analyzing Bangalore house dataset we will be able to understand how house prices depend on other factors, then determine the approximate price for the houses using machine learning regression. In this work, I provided a study methodology that employs a range of algorithms (Logistic Regression, K-Folds, and DNN) to assess the dataset's most vital parts or attributes and compare their accuracy.

3. TOOLS

In this work, a several APIs and libraries used listed as follows:

- Pandas Library
- NumPy Library
- Matplotlib Library
- Time Library
- Sklearn Library
- Keras APIs (model_selection, models, layers, callbacks)

4. METHODOLOGY

Data Science is an interdisciplinary field that focuses on extracting knowledge from data sets that are typically huge in amount. The field encompasses analysis, preparing data for analysis, and presenting findings to inform high-level decisions in an organization.

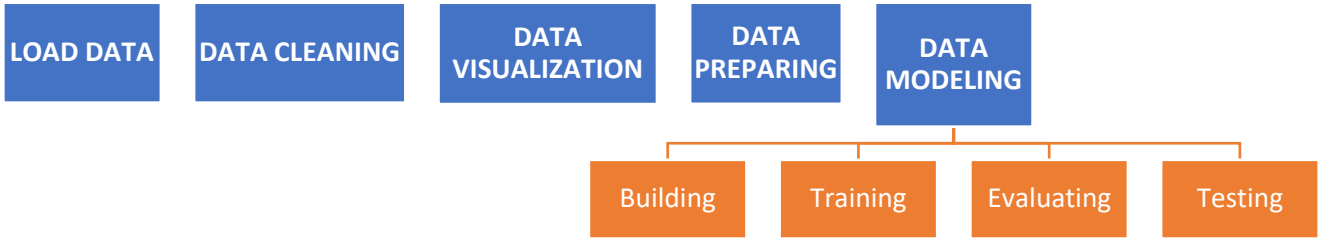

Figure 1: Project Pipeline.

4.1 Load Dataset

In this work, the dataset consists of houses features and price from Bangalore, which includes area type, location, size, and etc.

Dataset columns

- area_type
- availability
- location
- size
- society
- total_sqft
- bath
- balcony
- price

Load banglore home prices dataset

```
In [2]:
1 df = pd.read_csv('Bengaluru_House_Data.csv')
2 df.head()
```

Out[2]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00

Figure 2: Load dataset

4.2 Data Cleaning

1. Handle null values

2. Feature Engineering
    - Add new feature (bedrooms)
    - Fix (total_sqft) feature
    - Add new feature called (price_per_sqft)
3. Dimensionality reduction for categorical feature (location)
4. Outlier removal
    - Outlier removal from (price_per_sqft) feature according to business logic
    - Outlier removal from (price_per_sqft) feature according to std and mean
    - Outlier removal from (bedroom) feature according to business logic
    - Outlier removal from (bath) feature

4.3 Data Visualization

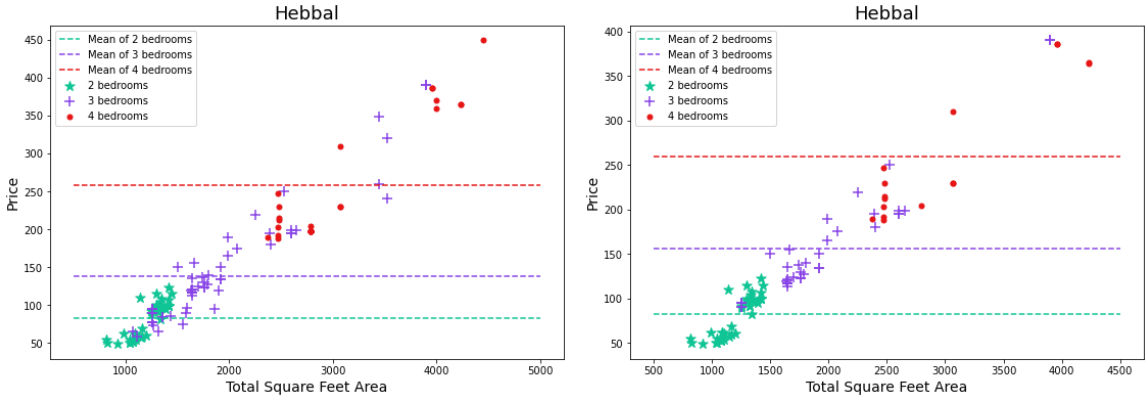

Figure 3: Scatter chart for (bedroom) feature before and after outliers' removal.

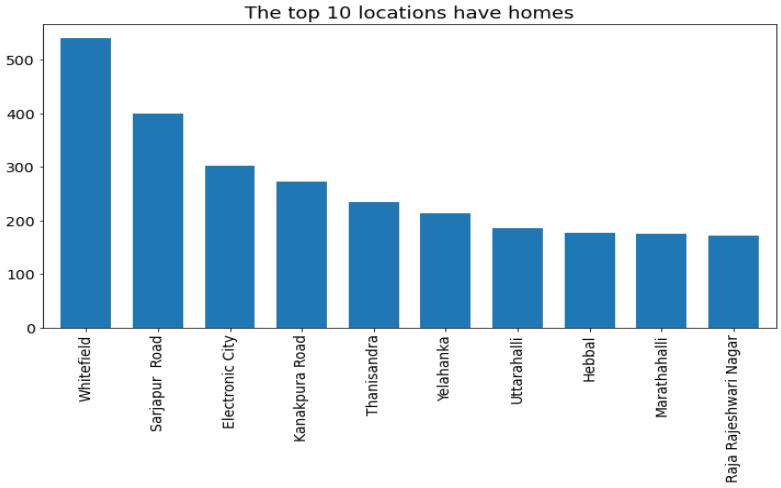

Figure 4: Bar Chart for the top 5 values of (location) feature.

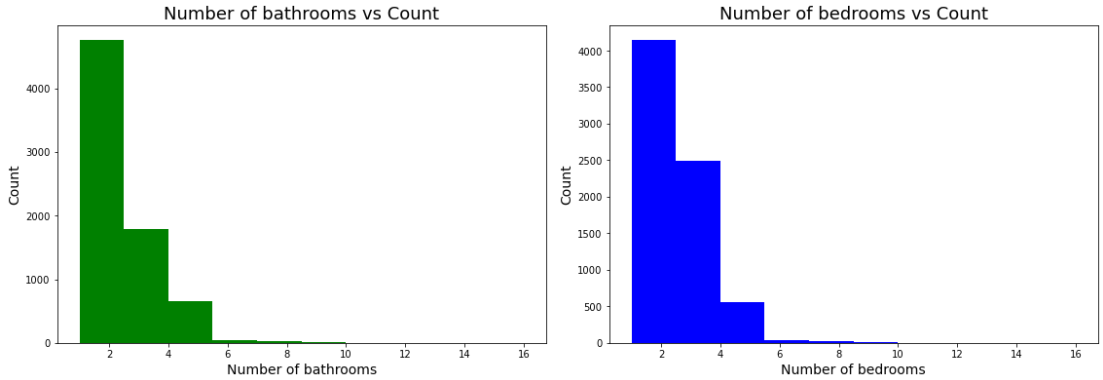

Figure 5: Histogram chart for Number of bedrooms and bathrooms vs Count.

Figure 6: Line chart for the relation between Total price vs Number of bedrooms and bathrooms.

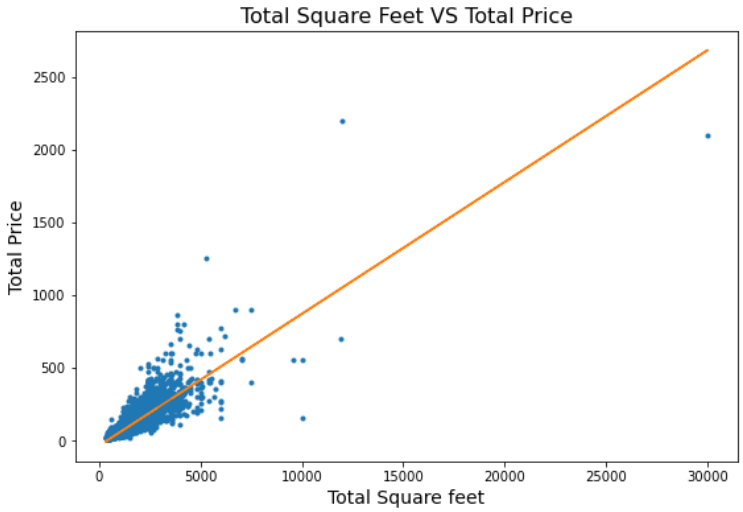

Figure 7: Trend chart for Total square feet vs Total price.

4.4 Data Preparing

Use One Hot Encoding For (location) feature

In [56]:
```
len(df9.location.unique())
```

Out[56]: 241

In [57]:
```
dummies = pd.get_dummies(df9.location)
dummies = dummies.drop('other', axis = 1)
dummies.head()
```

Out[57]:

	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar	9th Phase JP Nagar	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 240 columns

Figure 8: One Hot Encoding implementation for (location) feature

4.5 Data Modeling

4.5.1 Multivariate Linear Regression model

Multivariate linear regression is an approach for predicting a quantitative response using multiple feature (or "predictor" or "input variable"). It takes the following form:

$$Y_i = \alpha + \beta_1 x_i^1 + \beta_2 x_i^2 + \cdots . + \beta_n x_i^n$$

Where:

- Y_i is the response
- x_i is the feature
- α is the intercept
- β is the coefficient for x
- n number of features

Build LinearRegression model

In [63]:
```
# building the regression model
LR_model = LinearRegression()
```

Train LinearRegression model

In [64]:
```
LR_model.fit(X_train,y_train)
```

Out[64]:
```
LinearRegression()
```

Evaluate LinearRegression model

In [65]:
```
# calculate testing accuacy
sc =LR_model.score(X_test,y_test)
print('The regression model has validation score = {}'.format(sc))
```

```
The regression model has validation score = 0.7717304950660657
```

> The accuracy of the model is low a little bit

Figure 9: Multivariate Linear Regression model

This model give accuracy = 77.2% which is pretty low, so I will go and try the regression model using K-Folds.

4.5.2 Best Regression Model Using K-Fold Cross Validation.

Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods. The general procedure is as follows:

1. Shuffle the dataset randomly with $test\ size\ =\ [0.3,0.2,0.15]$.
2. Split the dataset into k-folds groups $folds\ =\ [2,3,4,5,6,7,8,9,10]$.
3. For each unique group:
    a. Take the group as a hold out or test data set.
    b. Take the remaining groups as a training data set
    c. Fit a model on the training set and evaluate it on the test set
    d. Retain the evaluation score and discard the model
    e. Plot evaluation chart for all the groups

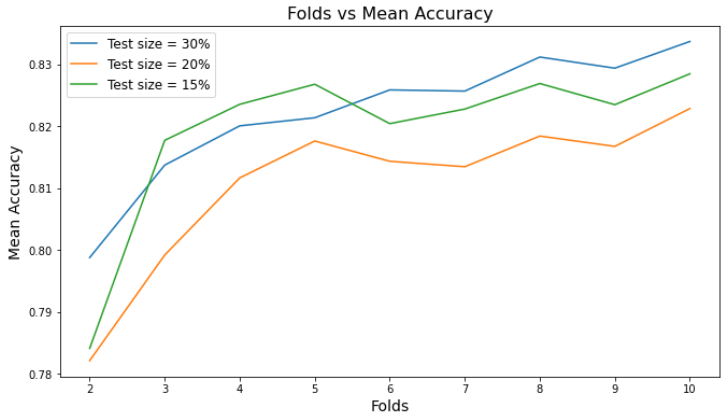

Figure 10: Line chart for k-Fold models' accuracies.

4. Applying the best k-folds and test size for the regression model.

Apply the best LinearRegression model

In [71]:
```
cv = ShuffleSplit(n_splits = folds, test_size = test_size, random_state=2021)
final_RL_model = LinearRegression()
```

Testing the best LinearRegression mode

In [72]:
```
y_pred = cross_val_predict(final_RL_model,X_test, y_test, cv=folds)
print('Frist 5 prediction values is :\n{}'.format(y_pred[:5]))
print('\nFrist 5 actual values is :\n{}'.format(y_test[:5]))
print('\nFinal accuracy: ',round(100*max_acc))
```

```
Frist 5 prediction values is :
[ 61.49133566 149.20143774 113.23506048  73.154372    43.45685079]

Frist 5 actual values is :
[ 74.  115.  100.   73.   54.5]

Final accuracy:  83.0
```

Figure 11: Applying the best k-fold regression model.

The accuracy of the model is pretty good, but I will improve it using deep learning technique.

4.5.3 Deep Learning Model

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks. Since it is a regression problem with Rectified linear unit (ReLU) activation function.

Building Architecture

Using Keras Sequential model I could stack of layers where each layer has exactly one input tensor and one output tensor.

Table 1: Deep leaning model Hyperparameters.

Hyperparameter	**values**
Test size	20%
Number of hidden layers	10
Hidden layers size	[100,90,80,70,60,50,40,30,20,10]
Activation function	ReLU
Optimizer	Adam
learning rate	0.0001
Number of epochs	500
Callbacks	Model Check Point
Metric	Mean Square Error

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 110)               26840
_________________________________________________________________
dense_1 (Dense)              (None, 100)               11100
_________________________________________________________________
dense_2 (Dense)              (None, 90)                9090
_________________________________________________________________
dense_3 (Dense)              (None, 80)                7280
_________________________________________________________________
dense_4 (Dense)              (None, 70)                5670
_________________________________________________________________
dense_5 (Dense)              (None, 60)                4260
_________________________________________________________________
dense_6 (Dense)              (None, 50)                3050
_________________________________________________________________
dense_7 (Dense)              (None, 40)                2040
_________________________________________________________________
dense_8 (Dense)              (None, 30)                1230
_________________________________________________________________
dense_9 (Dense)              (None, 20)                620
_________________________________________________________________
dense_10 (Dense)             (None, 10)                210
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 11
=================================================================
Total params: 71,401
Trainable params: 71,401
Non-trainable params: 0
_________________________________________________________________
```

Figure 12: Deep Learning Model Architecture.

Training

During the training process, known data is fed to the DNN, and the DNN makes a prediction about what the data represents. Any error in the prediction is used to update the strength of the connections between the artificial neurons. As the training process continues, the connections are further adjusted until the DNN is making predictions with sufficient accuracy.

Optimization algorithms used for training of deep models differ from traditional optimization algorithms in several ways. Machine learning usually acts indirectly. In most machine learning scenarios, we care about some performance measure P, that is defined with respect to the test set and may also be intractable. We therefore optimize P only indirectly.

Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters. Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network.

In [77]:

```
n_epochs = 500

history = DL_model.fit(
    X_train, y_train,
    validation_data = (X_test, y_test),
    verbose = 1,
    epochs = n_epochs,
   callbacks=[mc],
)
saved_model = load_model('best_model.h5')
```

```
Epoch 1/500
194/194 [==============================] - 0s 2ms/step - loss: 1.0155 - mse: 1.0155 - val_loss: 0.7235 - val_mse: 0.7235
Epoch 2/500
194/194 [==============================] - 0s 2ms/step - loss: 0.9250 - mse: 0.9250 - val_loss: 0.6476 - val_mse: 0.6476
Epoch 3/500
194/194 [==============================] - 0s 2ms/step - loss: 0.8725 - mse: 0.8725 - val_loss: 0.6063 - val_mse: 0.6063
Epoch 4/500
194/194 [==============================] - 0s 2ms/step - loss: 0.7035 - mse: 0.7035 - val_loss: 0.2498 - val_mse: 0.2498
Epoch 5/500
194/194 [==============================] - 0s 1ms/step - loss: 0.3011 - mse: 0.3011 - val_loss: 0.1316 - val_mse: 0.1316
Epoch 6/500
194/194 [==============================] - 0s 1ms/step - loss: 0.2079 - mse: 0.2079 - val_loss: 0.1432 - val_mse: 0.1432
Epoch 7/500
194/194 [==============================] - 0s 1ms/step - loss: 0.1563 - mse: 0.1563 - val_loss: 0.1427 - val_mse: 0.1427
Epoch 8/500
194/194 [==============================] - 0s 1ms/step - loss: 0.1319 - mse: 0.1319 - val_loss: 0.1323 - val_mse: 0.1323
Epoch 9/500
194/194 [==============================] - 0s 2ms/step - loss: 0.1195 - mse: 0.1195 - val_loss: 0.1277 - val_mse: 0.1277
Epoch 10/500
194/194 [==============================] - 0s 1ms/step - loss: 0.1084 - mse: 0.1084 - val_loss: 0.1567 - val_mse: 0.1567
Epoch 11/500
194/194 [==============================] - 0s 1ms/step - loss: 0.1082 - mse: 0.1082 - val_loss: 0.1372 - val_mse: 0.1372
Epoch 12/500
194/194 [==============================] - 0s 1ms/step - loss: 0.0997 - mse: 0.0997 - val loss: 0.1297 - val mse: 0.1297
```

Figure 13: Training phase implementation and results.

Plotting history

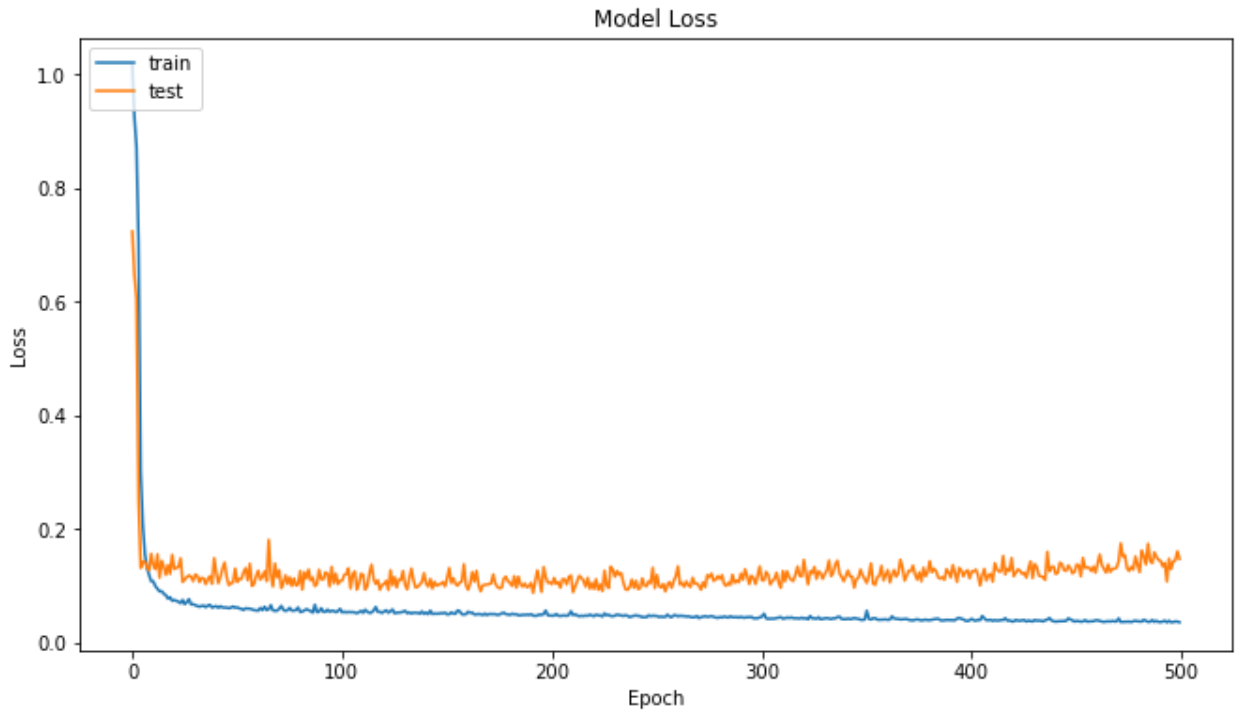

Figure 14: Line Chart for deep learning history according to loss function.

Testing

So, because best model was preserved during training as a consequence of the use of the model check point API, I utilized it for testing and documenting the final findings.

```
In [81]:
1 _, train_err = saved_model.evaluate(X_train, y_train, verbose=0)
2 _, test_err = saved_model.evaluate(X_test, y_test, verbose=0)
3 print('Train ACCuracy: %0.2f%%, Test Accuracy: %.2f%%' % (100-100*train_err, 100-100*test_err))

Train ACCuracy: 95.02%, Test Accuracy: 91.20%
```

Figure 15: Testing and evaluating implementation and results.

5. CONCLUSION

After studying, analysis, cleansing the data, as well as understanding the correlation between the features and the impact of each on the pricing value of each house, I created a three prediction models, the results of each were as follows in Table 2:

Table 2: The Final Accuracy of the three models.

Model	**Accuracy**

Multivariate Regression	77.17%
K-Fold Regression	83.0%
Deep Learning	91.20%

Therefore, it is clear that the third model that was built using deep learning is the most efficient among the three, with an accuracy of 91.20%.