

# Housing Price Prediction: Recurrent Neural Networks vs. Structured Neural Network

Mohammed Yaseen 1PE15CS086

Shubharthi Dey 1PE15CS157

Saleha Afsari 1PE16CS421

Ms. Lakshmi Nagaprasanna R.

Batch No. - 26

May 21, 2019

## 1 First Section

- This project directs to the best application of recurrent neural networks models in comparison to the structured neural networks model towards the common goal of predicting the real estate prices most accurately and compare the efficiency of each with the other. The project will be implemented as a web application using python machine learning libraries.

Output: The predicted cost of a given property in certain conditions and accuracy when compared to the original cost using recurrent neural networks and then using structured neural networks.

# Motivation for the work

- Intended for use by architects, investors and government policy makers.
- Investor : This will give an idea about how profitable a given investment would be, based on it's worth to the user.
- Public policy makers : Using this, will be able to decide how valuable a given piece of land is, and how valuable it will be in the future.
- In general, we are using ML to decide whether a project is worth investing in and if it is, how much ?

- **Paper 1 : Reza Ghodsi et. al. compared Fuzzy Regression and ANN**

**Merits:** Fuzzy Regression performs better than conventional linear regression, by reason of the fact that the aforementioned functional relationships could also be determined even when there are no crisp values for either of the dependent or independent variables (or both of them) or even when they are in the form of intervals or fuzzy numbers.

**Shortcomings:** The MAPE for both the models were compared and the ANN model performed relatively well with respect to Fuzzy Regression.

- **Paper 2: Byeonghwa Park, Jae Kwon Bae compared a classical rule-based classifier RIPPER to various other algorithms like Naive Bayes, AdaBoost.**

## **Merits:**

- RIPPER is able to handle imbalanced class distribution and more robust to noisy data.
- Accuracy better than Naive Bayes by 16%
- Accuracy better than AdaBoost by 7%

## **Shortcomings:**

- It was implemented in a certain country hence it failed to cover the overall pricing across the globe.
- Performance evaluation was done only on the basis of certain chosen classifiers which might not prove the basis of the research paper.

- **Paper 3: Wan Teng Lim et al. compared ANN vs. ARMA**

## **Merits:**

- Better results compared to Linear or Multivariate Regression.
- Not all the data is linear. Hence, regression may fail.

## **Shortcomings:**

- One of the possible limitations is that the size of dataset, which may not be sufficiently large. Hence, the performance of the ANN is not optimized due to the lack of training data set and insufficient verification output data.
- Moreover, some of the data for the independent variables such as the population and annual inflation rate are converted from yearly into quarterly basis; the accuracy of the predicted outputs will be affected.

- **Paper 4: Junchi Bin compared Recurrent NN to some well known algorithms like SVR, LASSO.**

## **Merits:**

- Recurrent Neural Networks (RNNs) are a family of neural networks for processing sequential data
- Essentially ANNs in which output is given back into input in the next iteration/layer along with new input. Hence, input can be variable size
- Used for speech, text etc

## **Shortcomings:**

- The amount of training required which is comparatively much more than other kinds of networks
- Second, it cannot be stacked into very deep models. This is mostly because of the saturated activation function used in RNN models, making the gradient decay over layers.

Solution : LSTM

- **Paper 5: Tom Kauko et. al discusses about application of Neural Network**

## **Merits:**

- Provides an accuracy much better than conventional models like regression and tree based algorithms.
- It is robust to noise.
- It comes handy when dealing with non-linear data.

## **Shortcomings:**

- Due to the black box nature of the method it will remain unclear which functional relations account for the variation in results.
- The interpretation of results relies heavily on the expert knowledge of the researcher.
- It is clear that the neural network approach is not apt in testing predefined hypotheses and that even linking outcomes to existing theory is problematic.



- **Paper 6: Haiping Xu and Amol Gade compared Structured DNN to other algorithms like Regression etc.**

## **Merits:**

- Neurons in a structured DNN are structurally connected, which makes the network time and space efficient.
- a structured DNN with a reduced number of links and neurons requires much less computational resources, but would still lead to satisfactory or even better results
- The structured DNN model has been designed to learn from the most recently captured data points; Therefore, it allows the model to adapt to the latest market trends.

## **Shortcomings:**

- Although a DNN works more efficient than a shallow one, a typical DNN for solving a practical problem may still require a large number of hidden neurons.
- Traditionally, a DNN is considered a black-box approach, where the semantics of the hidden neurons are unknown, and there is no clear way to determine a suitable size of the network.

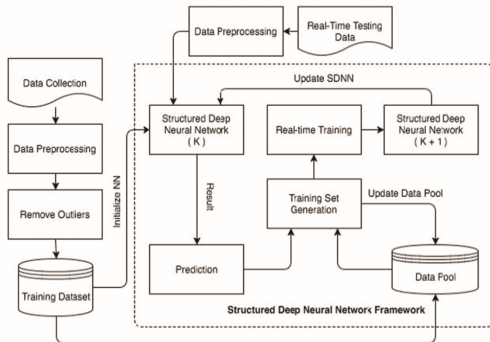
## Hardware Requirements

- Computer with 64-bit and 1.5GHz or faster processor
- 2GB RAM
- 5GB of available Hard-Disk Space

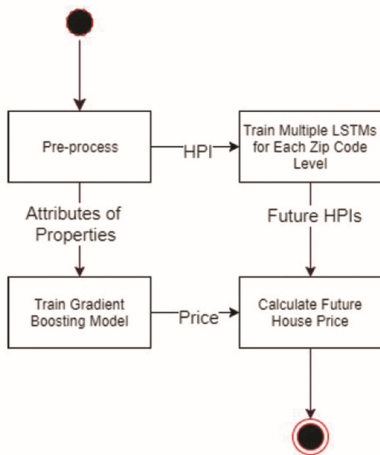
## Software Requirements

- Operating System - Linux or Windows
- Programming Language - Python/R
- Web Application Framework - Flask

# System Architecture(SNN)



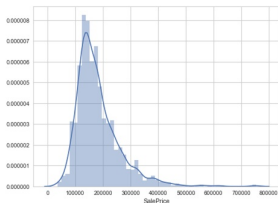
# System Architecture(RNN)



## Detailed Data Analysis

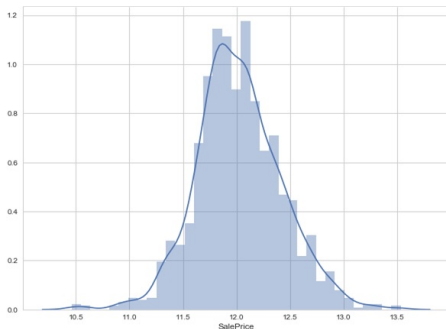
LotArea	LotShape	LotConfig	Neighborhood	BldgType	HouseStyle	OverallQual	OverallCond
8450	Reg	Inside	CollgCr	1Fam	2Story	7	5
9600	Reg	FR2	Veenker	1Fam	1Story	6	8
11250	IR1	Inside	CollgCr	1Fam	2Story	7	5
9550	IR1	Corner	Crawfor	1Fam	2Story	7	5
14260	IR1	FR2	NoRidge	1Fam	2Story	8	5

- The dataset has 1460 rows and 67 columns.
- The Data looks right-skewed and has the following structure:



- The skewness in the Sale Price: 1.88287

- We apply Log transformation to fix the skewed distribution and make it approximately symmetrical about the mean.



- After Log Transformation, the skewness reduces to 0.12133506

- Now we look into the dataset more closely and decide upon the correlation of the input attributes to the SalePrice
- A numerical analysis in Python gives us the following result:

SalePrice	1.000000
OverallQual	0.790982
GrLivArea	0.708624
GarageCars	0.640409
GarageArea	0.623431
TotalBsmtSF	0.613581

## Implementation(Partial)

```
Epoch      lrate      Error
(3, ' ', 0.5, ' ', 0.00065243080468423338)
('Output :', 0.4993475691953157)
('Weight :', array([[ 2.0580997 ],
                    [-2.46881534],
                    [ 2.62774925],
                    [-1.45831412],
                    [-0.87738697],
                    [ 1.66090992],
                    [-2.93097424],
                    [-1.02070211],
                    [-2.70513715],
                    [-0.38852469]]))
('delta: ', 3.6789407511199269e-06)
Epoch      lrate      Error
(4, ' ', 0.5, ' ', 0.00056730966614307028)
('Output :', 0.49943269033385695)
('Weight :', array([[ 2.11204121],
                    [-2.50925816],
                    [ 2.69774479],
                    [-1.48429184],
                    [-0.88145463],
                    [ 1.69706035],
                    [-2.9763666 ],
                    [-1.03951804],
                    [-2.74908225],
```



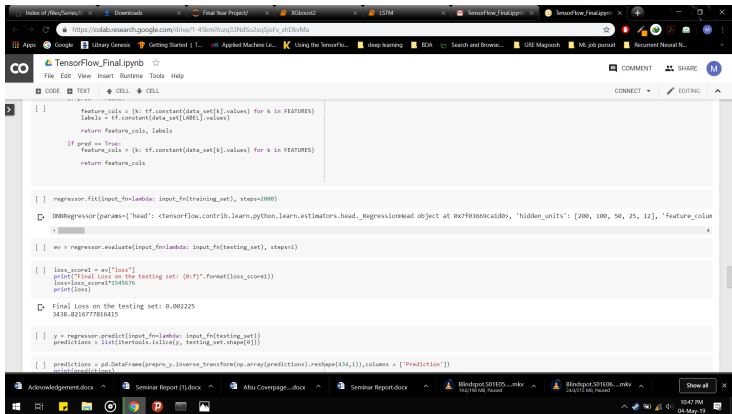
```
Epoch      lrate      Error
(0, ' ', 0.5, ' ', 0.3020480875549259)
('Output :', 0.1979519124450741)
('Weight :', array([[ 0.01049218],
                    [-0.86942699],
                    [-0.14375534],
                    [-0.80693817],
                    [-0.74568006],
                    [ 0.19349062],
                    [-0.547976 ],
                    [-0.78610863],
                    [-0.55938759],
                    [-0.30034743]]))
('delta: ', 0.059166820853662282)
Epoch      lrate      Error
(1, ' ', 0.5, ' ', 0.0011136127141371116)
('Output :', 0.49888638728586293)
('Weight :', array([[ 1.85006492],
                    [-2.30673839],
                    [ 2.35126386],
                    [-1.35643285],
                    [-0.86124928],
                    [ 1.51978843],
                    [-2.747437 ],
                    [-0.94684172],
                    [-2.52778862],
                    [-0.38318216]]))
('delta: ', 1.1546432742839009e-05)
```

- Successfully constructed XGBoost.
- Successfully constructed Structured Neural Network.
- Finished constructing the Recurrent Neural Network (LSTM) and almost all of the gradient boosting tree.
- Finished with the front end implementation of the project.

## Things done after Implementation:

- Testing has been done to get accuracy of each model and certifying the best out of the two.
- Connected the whole of the back end to the respective front end (using Flask framework)

# Implementation using Structured Neural Network



The screenshot shows a Jupyter Notebook titled 'TensorFlow\_Final.ipynb' in a web browser. The code is written in Python and uses TensorFlow for a housing price prediction task. It includes feature engineering, model training, evaluation, and prediction steps.

```
[ ] feature_cols = [k: tf.constant(data_set[k].values) for k in FEATURES]
labels = tf.constant(data_set[LABELS].values)
return feature_cols, labels

If pred == True:
    feature_cols = [k: tf.constant(data_set[k].values) for k in FEATURES]
    return feature_cols

[ ] regressor.fit(input_fn=lambda: input_fn(training_set), steps=2000)

DNNRegressor(params={'head': tensorflow.contrib.learn.python.learn.estimators.head.RegressionHead object at 0x7f0369ca0db0, 'hidden_units': [200, 100, 50, 25, 12], 'feature_column
x

[ ] sv = regressor.evaluate(input_fn=lambda: input_fn(testing_set), steps=1)

[ ] loss_score1 = sv["loss"]
print("Final Loss on the testing set: {0:f}".format(loss_score1))
loss=loss_score1*1545076
print(1094)

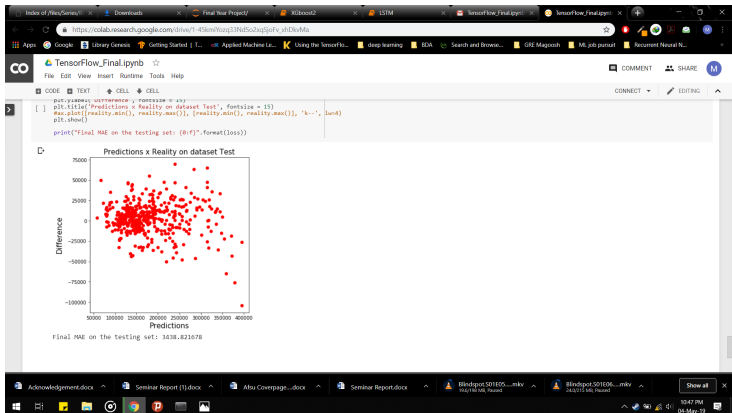
DNNRegressor(params={'head': tensorflow.contrib.learn.python.learn.estimators.head.RegressionHead object at 0x7f0369ca0db0, 'hidden_units': [200, 100, 50, 25, 12], 'feature_column
x

[ ] y = regressor.predict(input_fn=lambda: input_fn(testing_set))
predictions = list(itertools.islice(y, testing_set.shape[0]))

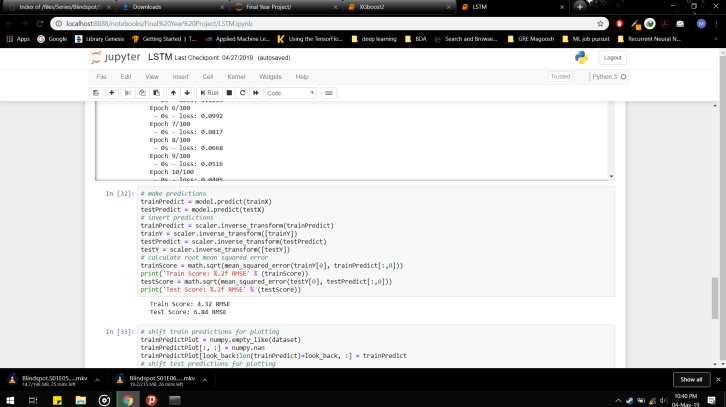
[ ] predictions = pd.DataFrame(prepro_y.inverse_transform(np.array(predictions).reshape(434,1)), columns = ['Prediction'])
return(predictions)
```

The output of the notebook shows the final loss on the testing set: 0.002225, and the final loss on the testing set: 3438.8216777816415.

# Implementation using Structured Neural Network(Output)



# Implementation using Recurrent Neural Network-LSTM



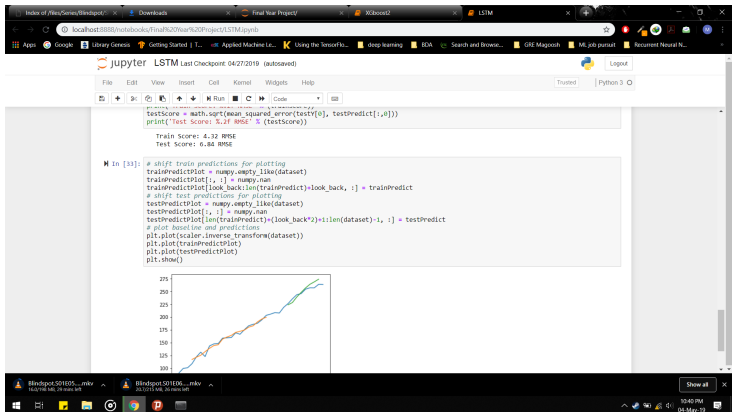
The screenshot shows a Jupyter Notebook titled 'LSTM Last Checkpoint: 04/27/2019 (autosaved)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a code editor. The code is written in Python and is divided into two main sections, In [32] and In [33].

```
Epoch 6/100  
- loss: 0.0992  
Epoch 7/100  
- loss: 0.0817  
Epoch 8/100  
- loss: 0.0668  
Epoch 9/100  
- loss: 0.0516  
Epoch 10/100  
- loss: 0.0406
```

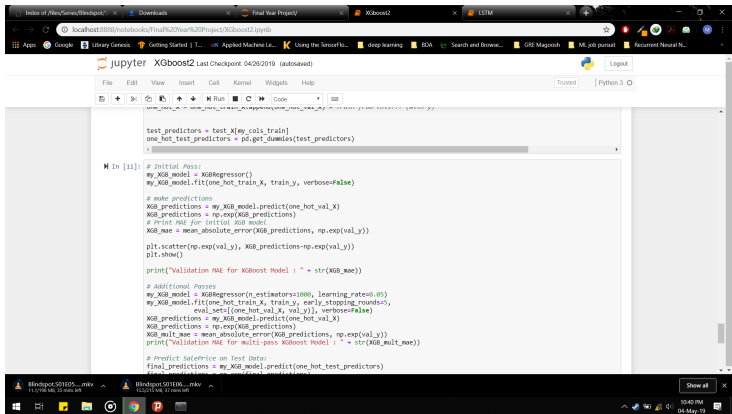
```
In [32]: # make predictions  
trainPredict = model.predict(trainX)  
testPredict = model.predict(testX)  
# invert predictions  
trainPredict = scaler.inverse_transform(trainPredict)  
trainY = scaler.inverse_transform([trainY])  
testPredict = scaler.inverse_transform(testPredict)  
testY = scaler.inverse_transform([testY])  
# calculate root mean squared error  
trainScore = math.sqrt(mean_squared_error(train[0], trainPredict[:,0]))  
print('Train Score: %.2f RMSE' % (trainScore))  
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))  
print('Test Score: %.2f RMSE' % (testScore))  
  
Train Score: 4.32 RMSE  
Test Score: 0.84 RMSE
```

```
In [33]: # shift train predictions for plotting  
trainPredictPlot = numpy.empty_like(dataset)  
trainPredictPlot[:, :] = numpy.nan  
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict  
# shift test predictions for plotting
```

# Implementation using Recurrent Neural Network-LSTM(Output)



# Implementation using XGBoost



The screenshot shows a Jupyter Notebook interface with the title 'XGboost2 Last Checkpoint: 04/26/2019 (autosaved)'. The notebook is running Python 3 code. The code defines test predictors, fits an XGBRegressor model, and makes predictions. It includes comments for initial and additional passes, and prints the validation MAE for both. The final prediction is also printed.

```
test_predictors = test_X[my_cols_train]
one_hot_test_predictors = pd.get_dummies(test_predictors)

# Initial Pass:
my_XGB_model = XGBRegressor()
my_XGB_model.fit(one_hot_train_X, train_y, verbose=False)

# make predictions
XGB_predictions = my_XGB_model.predict(one_hot_val_X)
XGB_predictions = np.exp(XGB_predictions)
# Print MAE for initial XGB model
XGB_mae = mean_absolute_error(XGB_predictions, np.exp(val_y))

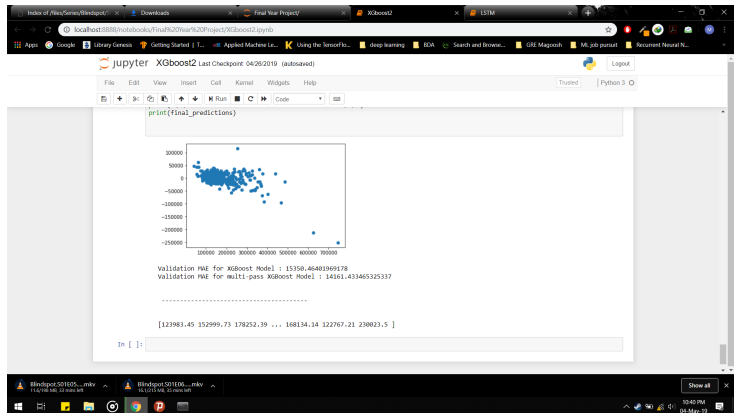
plt.scatter(np.exp(val_y), XGB_predictions - np.exp(val_y))
plt.show()

print("Validation MAE for XGBoost Model : " + str(XGB_mae))

# Additional Passes
my_XGB_model = XGBRegressor(n_estimators=1000, learning_rate=0.05)
my_XGB_model.fit(one_hot_train_X, train_y, early_stopping_rounds=5,
                eval_set=[(one_hot_val_X, val_y)], verbose=False)
XGB_predictions = my_XGB_model.predict(one_hot_val_X)
XGB_predictions = np.exp(XGB_predictions)
XGB_mult_mae = mean_absolute_error(XGB_predictions, np.exp(val_y))
print("Validation MAE for multi-pass XGBoost Model : " + str(XGB_mult_mae))

# Predict SalePrice on Test Data:
final_predictions = my_XGB_model.predict(one_hot_test_predictors)
```

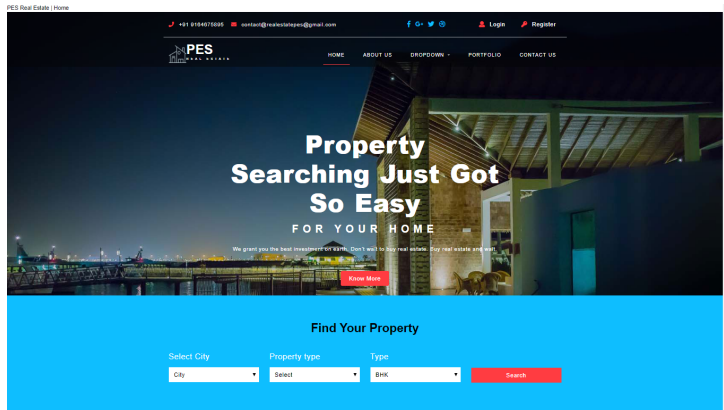
# Implementation using XGBoost(Output)



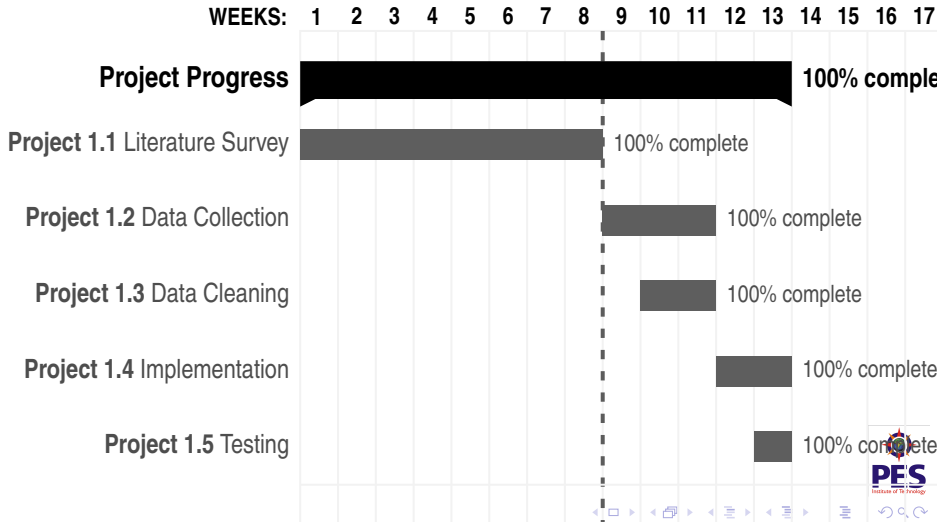


# Implementation Front End(Screenshots)

## Home Page [UI]



# Time line of completion of project from August 2018-April 2019(Gantt Charts).



# References



Haiping Xu and Amol Gade

Smart Real Estate Assessments



Reza Ghodsi, Abtin Boostani, Farshid Faghihi

Fuzzy Regression and Artificial Neural Network

*2010 Fourth Asia International Conference on Mathematical.*



Wan Teng Lim, Lipo Wang, Yaoli Wang, and Qing Chang

Housing Price Prediction Using Neural Networks

*2016 12th International Conference on Natural Computation.*



Junchi Bin, Shiyuan Tang, Yihao Liu, Gang Wang, Bryan Gardiner, Zheng Liu

Regression Model for Appraisal of Real Estate Using Recurrent Neural Network and Boosting Tree

*2017 2nd IEEE International Conference on Computational Intelligence and Applications.*



TOM KAUKO,PIETER HOOIMEIJER,JACCO HAKFOORT

Capturing Housing Market Segmentation: An Alternative Approach based on Neural Network Modelling

*TOM KAUKO,PIETER HOOIMEIJER,JACCO HAKFOORT (2002).*



Elsevier

Using machine learning algorithms for housing price prediction:The case of Fairfax County, Virginia housing data.

*Thank You*