

# **Kings County house prediction model**

## *Assignment #2*

### Group #1

- Rama AlOmari
- Soukaina Abugaoud
- Ibrahim Malkawi

# Contents

## **1.introduction**

1.1 Problem definition and importance of the project .....	3
1.2 Dataset .....	3
1.3 The goal .....	3

## **2.Data Pre-processing**

2.1 Data cleaning .....	4
2.2 Outliers handling .....	5
2.3 Feature extraction .....	7
2.4 Splitting the data .....	8
2.5 Feature selection .....	8

## **3.Model building**

3.1 Model testing .....	11
3.2 Choosing the appropriate model .....	11
3.3 Finding the Best Parameters for Enhanced Accuracy .....	11
3.4 LGBM model .....	13
3.5 Saving the model .....	14

## **4.Model implementation**

4.1 Model implementation.....	14
3.2 user manual.....	15

## **1.1: problem definition and importance of the project**

The problem at hand is to develop a model that accurately predicts the price of houses based on their features. The goal is to use the dataset to create a model that can accurately predict the price of a house given its features.

The importance of this problem lies in the fact that the housing market is a crucial aspect of the economy, and accurate price prediction can help individuals and businesses make informed decisions. For example, home buyers can use this model to make better decisions about which houses to purchase, while sellers can use it to determine the best price at which to list their homes. Additionally, investors and real estate agents can use this model to gain insights into the housing market and make informed decisions about their investments.

## **1.2: Dataset**

The dataset we have is for houses that are located in Kings County, WA.

It has about 20 thousand houses that have been sold between May of 2014 and May of 2015, each house has a unique set of features and price, these features were used to train the model to predict the selling price.

## **1.3: the goal**

To predict the house price accurately by giving the model the desired features either by the buyer or the seller.

## 2.Data pre-processing:

The dataset used for this project is a collection of housing prices and corresponding features such as location, number of bedrooms, square footage, and age of the property.

### 2.1 Data cleaning:

Before analyzing the data, several preprocessing steps were applied to ensure its quality and consistency. These steps included changing the date format to a more readable format, dropping houses with zero bedrooms or bathrooms, removing duplicates, checking for missing values. Outliers were also identified and removed.

Here are some screenshots for the code used to do so:

#### **change the date format**

```
df['date'] = pd.to_datetime(df['date'], dayfirst=True)
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day'] = df['date'].dt.day
df['Date'] = df[['day', 'month', 'year']].apply(lambda x: '/'.join(x.astype(str)), axis=1)
```

#### **Drop houses with zero bedrooms or bathrooms**

```
df=df.drop(df[df['bathrooms']==0].index, axis=0)
df=df.drop(df[df['bedrooms']==0].index, axis=0)
```

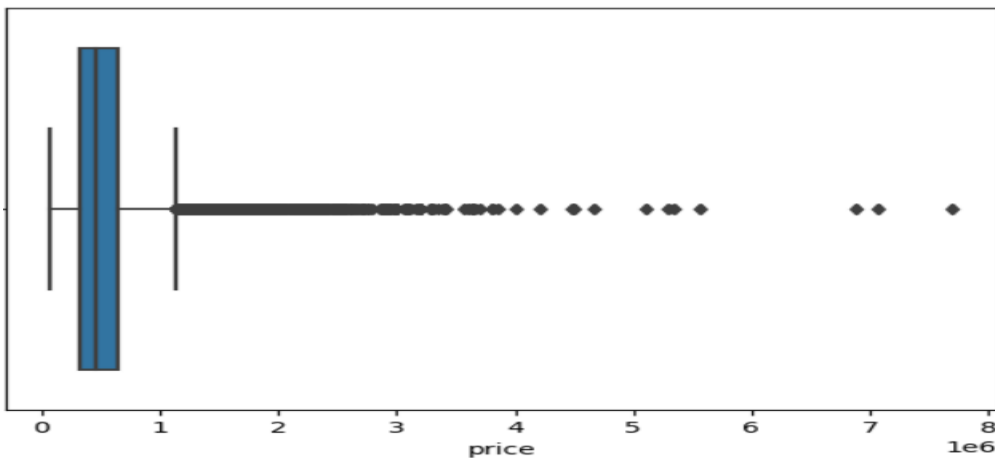
#### **Dropping duplicates**

```
df=df.drop_duplicates(subset=["id","price"])
```

Because some houses were sold twice in the dataset, we had to ensure that we did not drop them as duplicates. To do this, we added the sale prices, as the same house cannot be sold twice at the same price.

## 2.2 Outliers handling:

In addition to handling duplicates, we also addressed the issue of outliers in the dataset, as they have the potential to negatively impact the performance of the machine learning model. Outliers refer to data points that deviate significantly from the majority of the data, and can result from various factors such as mistakes while inputting the data. To deal with outliers, we first identified them using visualization techniques such as boxplots and scatterplots. We then applied various methods to handle them, including removing them, transforming them. By addressing outliers, we aimed to ensure that the model is trained on high-quality data that is representative of the underlying distribution and can yield accurate and reliable predictions.



### removing price outliers

```
#using upper limit and lower limit to remove price outliers
```

```
def iqr_method(df):  
    perc_75 = np.percentile(df, 75)  
    perc_25 = np.percentile(df, 25)  
    iqr_range = perc_75 - perc_25  
    iqr_upper = perc_75 + 1.5 * iqr_range  
    iqr_lower = perc_25 - 1.5 * iqr_range  
    return(iqr_lower,iqr_upper)
```

```
lower,upper=iqr_method(df['price'])  
print('upper limit for price  = ', upper )  
print('lower limit for price  = ', lower)  
df=df[(df['price']>lower)&(df['price']<upper)]
```

As shown in the box plot above there are some outliers that required handling, any data point that falls outside the upper and lower whiskers is considered an outlier, to handle

this issue, we employed the (`iqr_method`) function that takes a Pandas Data Frame as input and calculates the upper and lower limits for outliers using the interquartile range (IQR). It does this by first calculating the 75th and 25th percentiles of the values in the Data Frame using NumPy's percentile function. It then calculates the IQR by subtracting the 25th percentile from the 75th percentile.

Next, it calculates the upper and lower limits for outliers using the formulas

$Q3 + 1.5 * IQR$  and  $Q1 - 1.5 * IQR$ , where  $Q3$  is the 75th percentile,  $Q1$  is the 25th percentile, and  $IQR$  is the interquartile range. Finally, it returns the upper and lower limits for outliers. Then, we called the (`iqr_method`) function on the (`price`) column of the (`df`) Data Frame. Now, we can define the outliers are larger than the upper limit of price and smaller than the lower limit of price, and they were removed from the dataset, as these values are unlikely to be representative of the majority of the data and may negatively impact the performance of the machine learning model.

---

## Removing outliers for bedrooms , bathrooms and sqft\_living

```
#using z-score to remove the outliers

def remove_outliers(df, threshold=3):
    cols = ['bedrooms', 'bathrooms', 'sqft_living']
    df_clean = df.copy()
    # initialize a new dataframe to avoid modifying the original
    for col in cols:
        zscore = (df_clean[col] - df_clean[col].mean()) / df_clean[col].std()
        df_clean = df_clean[abs(zscore) <= threshold]
    return df_clean
```

We defined a function called (`remove_outliers`) that takes a pandas Data Frame as input, and removes outliers based on the z-score method for the columns (`bedrooms`), (`bathrooms`), and (`sqft_living`). The function creates a new Data Frame to avoid modifying the original. Then, the function calculates the z-score for each column in the `cols` list using the formula  $(x - \text{mean}) / \text{std}$ , where  $x$  is each value in the column, `mean` is the mean value of the column, and `std` is the standard deviation of the column. The z-score is a standardized value that measures how many standard deviations a data point is away from the mean. In this case, a threshold value of 3 which is a common value used is used to determine whether a data point is an outlier. If the absolute value of the z-score for a data point is greater than the threshold value, then that data point is considered an outlier and is removed from the data frame. Finally, the function returns the cleaned Data Frame. The purpose of this code is to remove any extreme data points from the Data Frame that may skew the analysis or models based on the data, and to preprocess the data.

before training the machine learning model, as it ensures that the model is trained on high-quality data that is representative of the underlying distribution.

## **2.3 Feature extraction:**

we added some features:

**House age:**

### **creating a new column named house age from yr\_built feature**

```
df['House_Age']=df['year']-df['yr_built']
```

This code snippet was implemented to add a new column to the dataset, which represents the age of each house. The "year built" feature from the original dataset was used to calculate the age, which was then stored in the newly created "House\_Age" column. This additional column can provide valuable insights into the age distribution of the houses in the dataset and help in making informed decisions based on the age of the property.

---

living\_to\_lot\_ratio :

### **Adding new feature**

```
df['living_to_lot_ratio']=df['sqft_living']/df['sqft_lot']
```

The above code calculates the living-to-lot ratio of each property and adds a new column called "living\_to\_lot\_ratio" to the existing dataset. The formula used to calculate the ratio is the total square footage of living space divided by the total square footage of the lot on which the property is built.

---

**City:**

```
search = SearchEngine()
def zco(x):
    city = search.by_zipcode(x).major_city
    return city if city else 'None'
```

```
df['city'] = df['zipcode'].apply(zco)
```

The above code utilizes the Python library "uszipcode" to retrieve the name of the major city associated with each zip code in a given dataset. The "SearchEngine()" class is first instantiated to create an instance of the search engine.

The "zco()" function is defined to take a single parameter, which is the zip code of a property. The function calls the "by\_zipcode( )" method of the search engine instance, which retrieves information about the zip code, including the name of the major city associated with it. If a major city is found, it is returned by the function, else it returns the string 'None'.

---

## **2.4 Splitting the data:**

```
x = df.drop(['price', 'Date', 'city', 'id'], axis=1)
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

Initially, the variables X and y were defined in the code to segregate the input data and target variable from the dataset. The input data was stored in X and the target variable in y. However, certain features, such as Date, city, and id, that were considered to be personally identifiable information and therefore excluded from the input data, were dropped from the input data.

Further, the data was split into two parts for training and testing purposes. The splitting was done using the "train\_test\_split ( )" method from the scikit-learn library. The testing dataset size was set to be 30% of the total dataset size, and the remaining 70% was used for training.

By splitting the dataset into training and testing data, we can train the machine learning model on the training data and evaluate its performance on the testing data, thus providing an estimate of the model's accuracy and generalization ability. Dropping certain features from the dataset ensures that the model doesn't overfit on these features and can make accurate predictions on new data.

---

## **2.5 feature selection:**

we used two methods for feature selection, correlation-based feature selection and RandomForestRegressor-based feature selection.



```
# Selection of best features based on correlations :
corr = df.corr()
corr_target = abs(corr["price"])

# select features with correlation coefficient > 0.1
best_features_corr= corr_target[corr_target > 0.1].index.tolist()
print('The best features based on correlations: ', best_features_corr[1:])
print('The number of features was selected based on correlations is : ',len( best_features_corr[1:]) )

The best features based on correlations: ['bedrooms', 'bathrooms', 'sqft_living', 'floors', 'view', 'grade', 'sqft_above', 'sqft_basement', 'lat', 'living_to_lot_ratio']
The number of features was selected based on correlations is : 10
```

The first function, "corr()", computes the correlation between the variables in the dataset using the Pearson correlation coefficient. The absolute correlation values for the "price" target variable are calculated and stored in "corr\_target". The next step involves selecting the features that have a correlation coefficient greater than 0.1, which are considered to be the most important features based on correlation. The resulting "best\_features\_corr" list contains the names of the selected features. This method can be used to identify the most important features based on linear correlation, although it may not be suitable for non-linear data like we have.

```
# Selection of best features Based on RandomForestRegressor :
rfr = RandomForestRegressor(n_estimators=100, random_state=42)
rfr.fit(X_train, y_train)

# select the best features
best_features_rfr = X_train.columns[rfr.feature_importances_ > 0.01].tolist()
print('The best feature based on RFR model :',best_features_rfr)
print('The number of features was selected based on RandomForestRegressor is: ',len(best_features_rfr) )

The best feature based on RFR model : ['sqft_living', 'sqft_lot', 'view', 'grade', 'sqft_above', 'lat', 'long', 'House_Age', 'living_to_lot_ratio']
The number of features was selected based on RandomForestRegressor is: 9
```

The second function involves fitting a RandomForestRegressor model to the training dataset and calculating the feature importance scores for each variable in the dataset using the "feature\_importances\_" attribute. The features with importance scores greater than 0.01 are selected, and their names are stored in "best\_features\_rfr". This method can be useful for identifying the most important features in a non-linear dataset.

And as we can see the outputs of each function are different from each other so we decided to define a model to compare the performance of each features set, using the LGBM algorithm.

```
#split the data but only using the selected features based on correlations
XX=df[['bedrooms', 'bathrooms', 'sqft_living', 'floors', 'view', 'grade', 'sqft_above', 'sqft_basement', 'lat', 'living_to_lot_ratio']]
|
yy=df['price']
XX_train, XX_test, yy_train, yy_test = train_test_split(XX, yy, test_size=0.3, random_state=42)
lgbm(XX_train, XX_test, yy_train, yy_test)

R^2: 0.82
RMSE: 87129.0
```

```
#split the data but only using the selected features based on RFR
XX1=df[['sqft_living', 'sqft_lot', 'view', 'grade', 'sqft_above', 'lat', 'long', 'House_Age', 'living_to_lot_ratio']]
yy1=df['price']
XX1_train, XX1_test, yy1_train, yy1_test = train_test_split(XX1, yy1, test_size=0.3, random_state=42)
lgbm(XX1_train, XX1_test, yy1_train, yy1_test)
```

R<sup>2</sup>: 0.868  
RMSE: 74610.73

The two functions for feature selection were compared based on their effectiveness in predicting the target variable, which is the "price" in this case. The correlation-based feature selection achieved an R2 score of 0.82 and an RMSE of 87129.0, whereas the RandomForestRegressor (RFR) feature selection achieved an R2 score of 0.868 and an RMSE of 74610.73. The R2 score represents the proportion of variance in the target variable explained by the features in the model, with higher values indicating better performance. The RMSE score represents the average deviation of the predicted values from the actual values, with lower values indicating better performance.

From the output, we can see that the RFR-based feature selection outperformed the correlation-based feature selection in terms of both R2 and RMSE metrics, with a significantly higher R2 score and lower RMSE score. Therefore, it is suggested that the RFR-based feature selection method is more effective in identifying the most important features for predicting the "price" target variable.

After our data analysis, and using LGBM algorithm and comparing the results, we have decided to choose the features was selected by RandomForestRegressor model, and we added "bedrooms" and "bathrooms" features to them, in order to use them in our app. The selected features can then be used to train a machine learning model with improved accuracy and generalization ability.

```
# if we try to add more features and compare the score:
XX2=df[['sqft_living', 'sqft_lot', 'view', 'grade', 'sqft_above', 'lat', 'long', 'House_Age', 'living_to_lot_ratio', 'bedrooms', 'bathrooms']]
yy2=df['price']
XX2_train, XX2_test, yy2_train, yy2_test = train_test_split(XX2, yy2, test_size=0.3, random_state=42)
lgbm(XX2_train, XX2_test, yy2_train, yy2_test)
```

R<sup>2</sup>: 0.869  
RMSE: 74377.72

Notice that the R2-score has increased 0.001 when we added "bedrooms" and "bathrooms" features to the features was selected by RandomForestRegressor model, and RMSE has decreased 233.01 when we added "bedrooms" and "bathrooms" features to the features was selected by RandomForestRegressor model.

### **3.Model building:**

#### **3.1 Model testing**

\_We tested multiple models so we can choose the appropriate one, here is the list of them:

- 1-Linear regression
- 2-Lasso regression
- 3-Decision Tree Regressor
- 4-Random Forest Regressor
- 5-Gradient Boosting Regressor
- 6-XGB Regressor
- 7-LGBM Regressor

#### **3.2 Choosing the appropriate model**

there is a table that shows 7 models that were used to predict house price, R2\_Score and RMSE for each of them.

	Model	R2 Score	RMSE
1	LinearRegression	0.696388	113189.454785
2	Lasso	0.696387	113189.579280
3	DecisionTreeRegressor	0.718951	108902.393992
4	RandomForestRegressor	0.857360	77583.042400
5	GradientBoostingRegressor	0.866119	75163.223609
6	XGB Regressor	0.859430	77018.098746
7	LGBM Regresso	0.868903	74377.724738

#### **3.3 Finding the Best Parameters for Enhanced Accuracy**

In order to further improve the accuracy of our top two models, we utilized the grid search method to identify the best combination of hyperparameters. Grid search is a

popular method for tuning hyperparameters by systematically searching through a predefined parameter space and evaluating the performance of each parameter combination. By using this method, we were able to find the best parameters set to enhance the accuracy of our models. After evaluating the performance of the models with the newly tuned parameters, we were able to determine which model achieved the highest accuracy and select it as the best performing model.

Based on the evaluation of the models using the newly tuned parameters, the LGBM model was found to have the highest R2- score and lowest Root Mean Squared Error and was selected as the best performing model.

The best parameter set for LGBM Regressor is:

```
Best parameters: {'learning_rate': 0.05, 'max_depth': -1, 'n_estimators': 500}
```

**Learning rate:** Learning rate determines how quickly the algorithm learns the underlying patterns in the data. It is a hyperparameter that controls the step size at each iteration when moving toward a minimum of a loss function. A low learning rate can result in slow convergence, while a high learning rate can cause the algorithm to overshoot the minimum.

**Max depth:** Max depth refers to the maximum depth of a decision tree. It is a hyperparameter that controls the complexity of the model. A larger max depth can result in a more complex model that may overfit the data, while a smaller max depth can result in a simpler model that may underfit the data.

**n\_estimators:** n\_estimators refer to the number of decision trees in the model. It is a hyperparameter that controls the complexity of the model and the number of iterations in the gradient boosting process. A larger number of n\_estimators can result in a more complex model that may overfit the data, while a smaller number of n\_estimators can result in a simpler model that may underfit the data.

```

param_grid = {
    'max_depth': [-1, 2, 3, 4, 5, 6, 7, 8],
    'learning_rate': [0.01, 0.1, 0.05, 1],
    'n_estimators': [100, 200, 500]
}
lgbm = LGBMRegressor()
# Initialize GridSearchCV object with the defined parameter grid
grid_search = GridSearchCV(estimator=lgbm, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X2, y2)
# Print the best parameters and score
print('Best parameters: ', grid_search.best_params_)
print('Best score: ', grid_search.best_score_)

Best parameters: {'learning_rate': 0.05, 'max_depth': -1, 'n_estimators': 500}
Best score: 0.8701295499461953

```

---

### **3.4 LGBM model:**

LGBM (Light Gradient Boosting Machine) is a machine learning algorithm that belongs to the family of gradient boosting models. It is designed to be a fast, accurate, and scalable algorithm, particularly for large-scale datasets. LGBM uses a tree-based model that builds decision trees in a gradient boosting framework. It works by iteratively adding decision trees to the model, with each tree learning from the errors of the previous tree. LGBM is particularly effective in handling high-dimensional data, and it uses a unique technique called "leaf-wise" growth that can improve its performance compared to other gradient boosting algorithms. In addition, LGBM supports categorical features and can handle missing values. Overall, LGBM is a powerful machine learning algorithm that can deliver high accuracy in a variety of applications, particularly for large-scale datasets.

```

lgbm1 = LGBMRegressor(n_estimators=500, learning_rate=0.05, max_depth=-1)
lgbm_model1 = lgbm1.fit(X2_train, yy2_train)
yy2_pred_lgbm1 = lgbm_model1.predict(X2_test, num_iteration = lgbm_model1.best_iteration_)
r21=r2_score(yy2_test,yy2_pred_lgbm1)
rmse1= mean_squared_error(yy2_test, yy2_pred_lgbm1, squared = False)
print('R^2:', r21.round(3) )
print('RMSE:',rmse1.round(2))

```

R^2: 0.87  
RMSE: 74040.01

### **3.5 Saving the model**

```

# save a model to a pikle file
pickle.dump(lgbm_model1, open('sk.pkl', 'wb'))

```

This code is using the Python's built-in pickle module to save the trained LGBM model object `lgbm_model1` into a file named (`sk.pkl`) in binary format ('**wb**' stands for "write"). By doing this, you can persist the trained model to disk and load it later on to make predictions on new data without having to retrain the model from scratch every time. It's a way to save time and resources in the long run, especially if you have a complex model that takes a long time to train.

### **4.Model implementation:**

The code is a script for creating a GUI application that predicts house prices. The application has an input section where users can input different features, and a button to trigger the prediction. After the prediction, the application displays the predicted price in a label, It was used the Tkinter library for creating the graphical user interface.

The script defines two functions, `main` and `on_predict`, which handle different aspects of the program.

```

def on_predict(self):
    # Get the input values from the user
    input_values = {}
    for feature, entry in self.inputs.items():
        input_values[feature] = float(entry.get())

    # Prepare the input data as a list of lists
    input_data = [[input_values[feature] for feature in features]]

    # Convert the input data to a numpy array
    input_data = np.array(input_data).astype('float32')

    # Ensure the input shape matches the expected shape of the model
    if input_data.shape != (1, len(features)):
        raise ValueError(f"Invalid input shape: expected {(1, len(features))}, got {input_data.shape}")

    # Use the loaded model to make the prediction
    predicted_price = model.predict(input_data)[0]

    # Update the predicted price label
    self.predicted_price_label.config(text=f"The estimated price of the house is ${predicted_price:,.2f}")

```

This code defines the `on_predict` method of a House Price Prediction App class, When the user clicks on the "Predict Price" button in the GUI.

Overall, the purpose of this code is to define a method that takes in user input, processes it, and uses a pre-trained model to make a prediction on the input data. The predicted value is then displayed to the user in the GUI.

```

def main():
    # Create a new Tkinter window
    window = tk.Tk()

    # Create the HousePricePredictionApp object
    app = HousePricePredictionApp(window)

    # Start the Tkinter event loop
    window.mainloop()

if __name__ == '__main__':
    main()

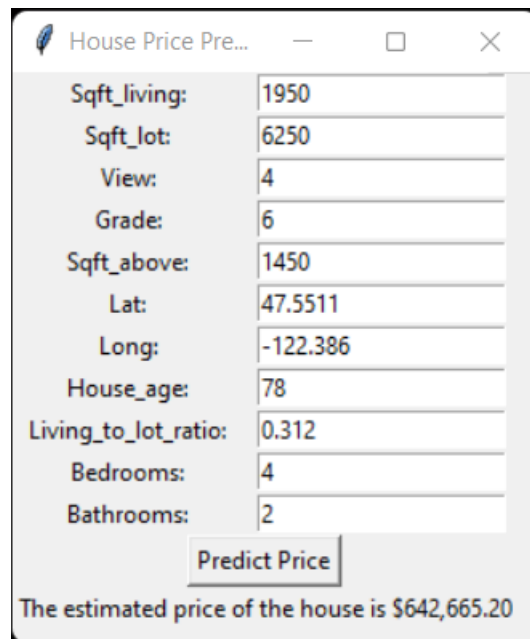
```

The main function creates a new Tkinter window, initializes the House Price Prediction App object, and starts the Tkinter event loop. Finally, the script checks whether it is being run as the main program and calls main function if true.

## **4.2 User manual:**

This graphical user interface is designed to help you predict the sale price of a house in Kings County based on various details about the house, such as the number of bedrooms,

bathrooms, and square footage. To use this tool, simply enter the relevant information into the form on the main screen and click the 'Predict Price' button. The tool will then calculate the predicted sale price of the house and display it.



Attribute	Value
Sqft_living:	1950
Sqft_lot:	6250
View:	4
Grade:	6
Sqft_above:	1450
Lat:	47.5511
Long:	-122.386
House_age:	78
Living_to_lot_ratio:	0.312
Bedrooms:	4
Bathrooms:	2

**Predict Price**

The estimated price of the house is \$642,665.20

This figure shows that the estimated price of the houses is \$642,665.20, after you filled the required information.