House Price Prediction

Using machine learning on KC Housing Dataset

BY:

Ahmad Alshawabkeh

Hassan Alnobani

Toqa Ghneimat

Chapters:

1) Data-set.
2) Preparing the dataset.
3) Model experiments.
4) Function include the class.
5) Saving the best trained model.
6) Creating User friendly inter-face.
7) Results.

1) KC Housing Dataset:

1.1) Dataset Source:

KC housing price dataset that was sourced from 'Kaggle'.

This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015. It has many characteristics of learning, and the dataset can be downloaded from here.

1.2) Dataset Description:

The data set contains (21613) observations with (19) features plus house price.

The names of the columns are given below.

- id
- date: Date house was sold(String)
- price: Price of the sold house
- bedrooms: Number of Bedrooms
- bathrooms: Number of bathrooms
- sqft_living: Square footage of the living room
- sqrt_log: Square footage of the log
- floors: Total floors in the house
- waterfront: Whether the house has a view a waterfront(1: yes, 0: not)
- view: The view from the house (0:the worst – 4:the best)
- condition: The condition of the hose (1:the worst - 5:the best)
- grade: The grade given to the house(1:the worst - 13:the best)
- sqft_above: Square footage of house apart from basement
- sqft_basement: Square footage of the basement
- yr_built: Built year
- yr_renovated: Year when the house was renovated
- zipcode: zipcode of the house
- lat: Latitude coordinate
- long: Longitude coordinate
- sqft_living15: Living room area in 2015(implies some renovations)
- sqrt_lot15: Lot area in 2015(implies some renovations)

1.3) More on KC housing dataset explanation:

Every single house on the surface of earth can be specified by the *latitude coordinates (lat) and longitude coordinates (long).*

Figure (1-1)
Longitude and latitude coordinates for King County

Figure (1-1) is to help understand the nature of King County, Location, water-front, views.

2) Preparing the dataset for models experiments (pre-processing):

pre-processing is an important part of building a state of the art model it is also a big part with many steps to take to get a well prepared dataset for model experiments:

2.1) Data Cleaning and Data Reduction:

- **performing data type conversions on columns in a Data-Frame using the Pandas library in Python:**
    - Converting multiple columns ("condition," "grade," "view," and "waterfront") in the Data-Frame from their current data types to the "category" data type.
    - Converting the "zipcode" column in the Data-Frame from its current data type to the "object" data type. Converting it to an object type usually means treating it as a generic text-based column rather than a numeric or categorical variable.

```
df["date"]= pd.to_datetime(df["date"]) # ... cahnging the date column from object to datetime
df[["condition","grade","view","waterfront"]]=df[["condition","grade","view","waterfront"]].astype("category")#... they have a finite set of element
df["zipcode"]=df["zipcode"].astype("object")
```

```
df.dtypes # .... check that apply the change
```

```
id                        int64
date             datetime64[ns]
price                   float64
bedrooms                  int64
bathrooms               float64
sqft_living               int64
sqft_lot                  int64
floors                  float64
waterfront             category
view                   category
condition              category
grade                  category
sqft_above                int64
sqft_basement             int64
yr_built                  int64
yr_renovated              int64
zipcode                  object
lat                     float64
long                    float64
sqft_living15             int64
sqft_lot15                int64
dtype: object
```

Figure (2-1)
Converting data-frame columns and showing data-types

- **Checking missing values and duplicated value:**
  1. Check for missing values in each column of the Data-Frame.
  2. Identify and investigate duplicated values in a Data-Frame based on a specific subset of columns.

```
df.isnull().sum()#.... to check missing values in each column
# there is no missing values
```

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

```
dup =df.duplicated(subset= ["id"] ,keep=False)#...creating false /true series to see duplicated values
dup_df=df[dup] #.. converting the series to df
dup_df#. ..There are duplicated values to be investigated
```

Figure (2-2)
Checking missing values and duplicated value

- **checking inconsistency in categorical/discrete columns:**
  1. By using .value.counts() and bar plot, creating a bar plot to visualize the distribution of values in the "bedrooms" column of the Data-Frame.
  2. We can see that 33 is so far from other dataset.

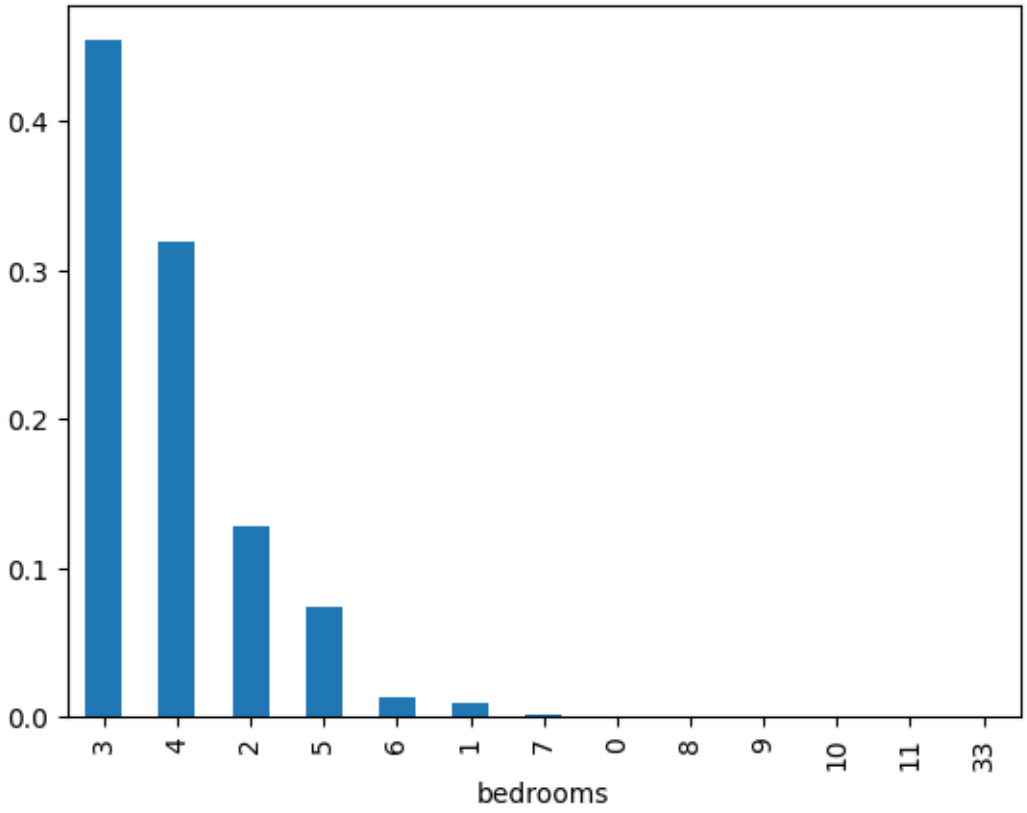

Figure (2-3)
The count of each value in the "bedrooms" column as bars in the plot.

- Observation:
  1. There are some houses that are sold twice in different date or price.
  2. Number of houses that are sold twice is (176 HOUSE)

2.2) Feature Engineering:

- **feature extraction:**
  - Converts zip codes to city names in a Data-Frame using the 'uszipcode' library. Imports the necessary components from the 'uszipcode' library. Applies the 'zco' function to the 'zipcode' column in the Data-Frame using the 'apply' method, creating a new 'city' column with the corresponding city names. The result is that the Data-Frame now contains a 'city' column with city names based on the zip codes in the 'zipcode' column.

```
#converting zip code to city name
from uszipcode import SearchEngine, SimpleZipcode
search = SearchEngine()
def zco(x):
    city = search.by_zipcode(x).major_city
    return city if city else 'None'

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

city
Seattle
Seattle
Kenmore
Seattle
Sammamish
...
Seattle
Seattle
Seattle
Issaquah
Seattle

Figure (2-4)
Column with city names based on the zip codes

- Creating a new column named "is_renovated" in the Data-Frame. It initializes all values in this column to 0. Then, it checks the "yr_renovated" column, and if the value is greater than 0 (indicating that a renovation year is provided), it sets the corresponding "is_renovated" value to 1, effectively indicating whether a property has been renovated (1) or not (0).

Feature selection:

- Converting selected columns ("yr_built," "is_renovated," "year," and "month") in the Data-Frame ' to the "category" data type. This is done for columns with a finite set of discrete values to optimize memory usage and potentially improve certain types of analysis.
    - Creating a new Data-Frame 'df_before_dummies' as a copy of the original Data-Frame before any further data transformation, such as one-hot encoding (dummies). It's done to keep an unmodified copy of the data for reference or to preserve the original data state.

```
# based on understanding our data and feature engineering, dropping non-sense columns
df[["yr_built","is_renovated" ,"year" ,"month"]]=df[["yr_built","is_renovated" ,"year" ,"month"]].astype("category")
#yr_untill_purchase= yr_purshase(2014/2015) - yr_build 1980
df_before_dummies=df
```

```
#one hot encoder for backward elimination and modeling
df_city=pd.get_dummies(df["city"],drop_first=True)
df_dummies=pd.concat([df,df_city] ,axis=1)
df_dummies = df_dummies.drop(columns=["city"])
df_lat=df_dummies
df_dummies=df_dummies.drop(["id","zipcode","lat","long","yr_renovated","date"],axis=1)
```

Figure (2-4)
One-hot encoding

- **BACK WORD ELIMINATION:**
  - Preparing the data for some machine learning model, after feature selection and data preprocessing.
  - Converts 'x' and 'y' into arrays.
  - Appends a column of ones to the beginning of 'x', presumably to represent the intercept term in a linear regression model.
  - Creates a new DataFrame 'df_after_extract_features' by dropping the target variable column from 'df_dummies' to retain only the feature columns.
  - Subsets the columns of 'df_after_extract_features' based on a specific set of column indices, likely as part of a feature selection process.
  - Concatenates the selected feature columns from 'df_after_extract_features' with the 'y' target variable into a new DataFrame 'df_new'.
  - Converts certain columns in 'df_new' (presumably categorical variables) to integers.

```
# x ,y
x=df_dummies.drop("price" ,axis=1)
y=df_dummies["price"]
x=x.values # converted to 2d array
y=y.values # converted to 1d array
y.shape
x.shape

x=np.append(arr=np.ones((21596,1)).astype("int"),values=x,axis=1) # adding one to the first column of the array
# b0 in  statsmodels is not defined
x.shape

df_after_extract_features=df_dummies.drop("price" ,axis=1) # dropping the column of target to only take features
df_after_extract_features=df_after_extract_features.iloc[: , [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,23,24,25,26,28,29,30,31,32,33,34,35,36,37,39]] #subsetting the index we got from the backword elimination
df_after_extract_features.columns # print columns name

y=df_dummies["price"]# our target Series
df_new=pd.concat([df_after_extract_features ,y] ,axis=1) # features dataFrame that resulted from the backword elimination
df_new[["waterfront","view","condition","grade","is_renovated","year","month"]]=df_new[["waterfront","view","condition","grade","is_renovated","year","month"]].astype(int) #converting data type to int
```

Figure (2-5)
BACK WORD ELIMINATION

3) Model experiments:

3.1) Experimenting on different Regression models:

Model used	R-squared (R2) Score
Linear Regression	0.5855556190833169
Support-Vector-Regression	0.3281779628507424
K-Neighbors- Regression	0.3951305235662549
XGB- Regression	0.7763372691322048
Sequential	0.5749963131978348
Decision-Tree- Regression	0.6438331847658667

3.2) Handling And Identify Outliers:

- **Handling Outliers:** defining a function handle_outliers that uses RobustScaler from scikit-learn to scale numeric columns in a DataFrame. This helps in handling outliers by making the data less sensitive to extreme values.
- **Identify Outliers:** defining a function identify_outliers to identify outliers in a specific column ("bedrooms") of the Data-Frame using the interquartile range (IQR) method. It creates a mask (outlier_mask) that marks the rows containing outliers in the specified column.

3.3) Regression_models ()

This class for random forest and gradient boosting algorithms, which contains functions that help automating any data to choose the best model.

This class has an input variable which is:

• Data (panda Data-Frame): Our data after analysis, feature selection and extraction

• Target (Pandas Series, NumPy array): the target variable that we want to predict which is in our case “price”.

• Algorithm (class in sklearn): Random Forest or Gradient boosting using scikti-learn to call one of it.

3.4) model function:

Description:

This function splitting our features (X) from our target (y) and also splitting our data to train and test data to get reliable result on unseen data.

Input: only get its input from class.

Output (tuple):

- X_train: which are the features on the training data.
- Y_ train: which is the target variable on the train data
- X_test: which are the features on the test data
- Y_test: which is the target variable on the test data

3.5) Train function:

Description:

Training the model to get the coefficients by fitting the X_Train and y_train on it which is now the return values from model function

Input: get its input from our class.

Output (model): training our model in X_train and y_train.

3.6) Score function:

Description:

Creating folds for validation data, then calculate the score for cross validation of data, training, and testing the data

Input:

• number_of_fold (int): number of folds want to split our train data to create CV.

• get its input from our class variable, train function and model function in our class.

Output (tuple):

• return score for cross validation of data, training, and testing the data.

3.7) Enhance_rf_model:

Description:

Hyper-parameter tuning for the random forest model, to get best hyper-parameter.

Using Randomized-Search-CV in sklearn which takes arguments (our model, a dict of params and its possible range, cross_validation and n_iters)

Which is the combination that it will take.

Input:

• n_estimators (list/array/int) = the number of trees in the forest. Default

Value =100

• max_depth (int /array/list) = the maximum depth of the tree, default

Value=8

• min_sample_leaf (int /array/list) = the minimum number of samples

Required to split an internal node, default value =3

• number_of_fold (int) = number of k fold, default value=5

Output (dictionary):

Return best hyper-parameters after grid search

3.8) Enhance_boost_model:

Description:

Hyper-parameter tuning for the gradient boosting model, to get best trees number (n_estimators) at a certain learning rate which is in our case 0.2. Using Randomized-Search-CV in sk-learn.

Input:

▪ n_estimators (int) = the number of boosting stages to perform.

▪ number_of_fold (int) = number of k fold, default value=5

Output:

- return best n_estimators at a certain learning rate after grid search

3.9) Enhance_boost_model_second

Description:

Hyper-parameter tuning for the gradient boosting model at a certain learning-rate and n_estimators, to get best hyper-parameter, related to the trees using Randomized-Search-CV in sk-learn

Input:

▪ max_depth: The maximum depth of the tree, default value=8.

▪ min_sample_leaf: The minimum number of samples required to split an

Internal node, default value =10.

Output:

Return best max_depth and min_sample_leaf at a certain learning rate and

n_estimators.

3.10) Learn curve:

Description:

Plot curves show us how well the model is performing as the data grows.

Which contains two curves:

• Training curve: The curve calculated from the training data; used to inform

How well a model is learning?

• Validation curve: The curve calculated from the validation data; used to

Inform of how well the model is generalizing to unseen instances.

Input (array/list):

Training size: the absolute numbers of training examples that will be used

To generate the learning curve; the values we are using are completely

Random. (X-axis of our curve)

Output (plot):

Two lines plot:

a) Train size vs train error

b) Train size vs test error

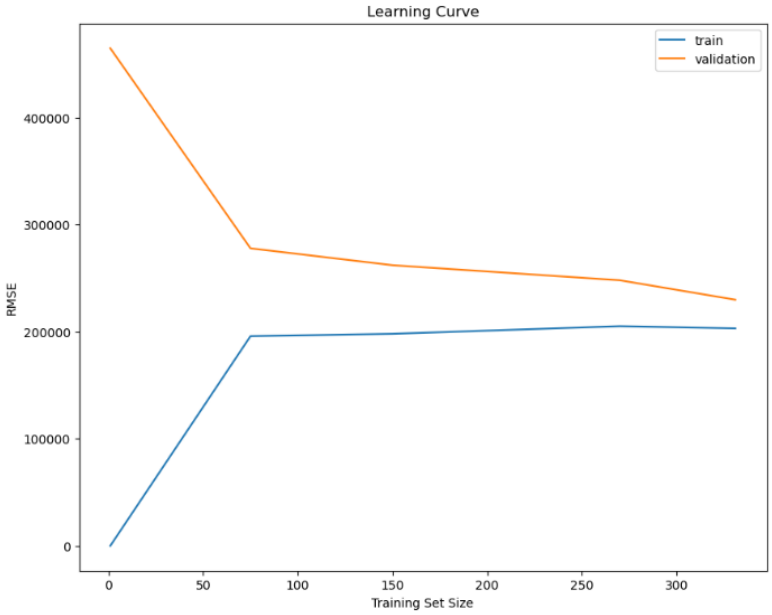

Figure (3-1)
Learning Curve

4) Function include the class:

4.1) Random_forest_method:

Description:

Most important features, depending on random forest selection.

▪ splitting our dataset into training and testing set

▪ Inistantiate Random Forest class and train it

▪ create a series with our purity using feature importance and names of

Features and sort it with descending order

▪ filter our data to create data frame with only features with high than 0.05

Input:

Get its input from our class

Output (tuple):

▪ Data-Frame that contains only the most importance features (index 0).

▪ Bar plot to show visually the most important feature (index 1).

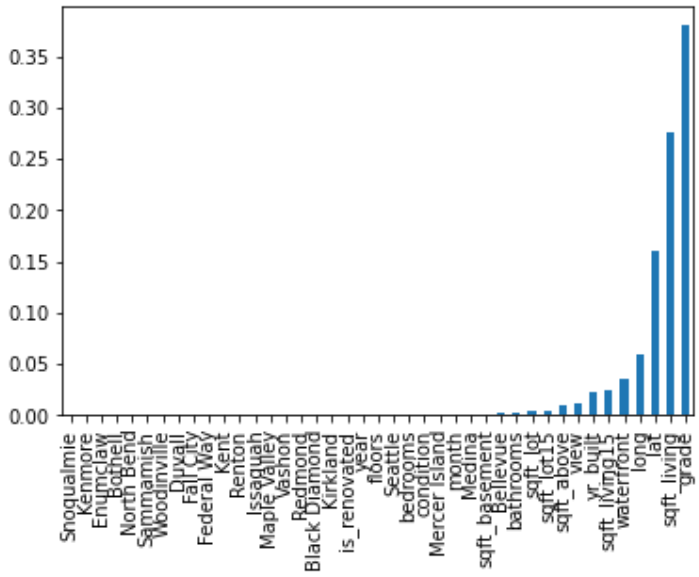

Figure (4-1)
Selected features based on Random Forest

4.2) Lasso function:

Description:

Most important features, depending on lasso coefficients (ignoring the coefficient which shrink to zero)

▪ splitting our dataset into training and testing set

▪ create a pipeline that contains Lasso class from sk-learn and Standard Scaler in order to instantiate the model and doing the preprocessing at the same time.

▪ using Grid-Search-CV in order to pick the best alpha

▪ find the coefficients using the best model (best_estimator) that result from

Grid-Search model and taking the absolute value from it.

▪ Create a Series with coefficient of the model and name of features

▪ filter this data to take only coefficient with higher than 0 (only important Feature)

Input:

k_fold (int): number of folds want to split our train data to create CV.

Output (tuple):

▪ DataFrame which contains the most important feature (index 0)

▪ Bar plot to show visually the most important feature (index 1)

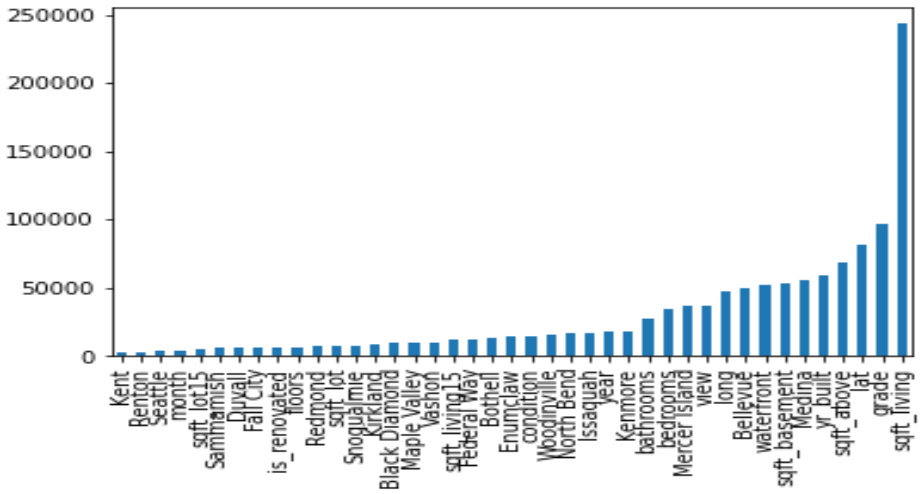

Figure (4-2)
Selected features based on Lasso

5) Saving the best trained model:

After creating two models and doing enhancement on it, it has been observed that Gradiant Boost (GB) has score (0.9) higher than Random Forest (0.86) so it will be taken.

Then, we save our best model using pickle library in order to load the model in a user notebook.

6) Creating User friendly interface:

In this part we show the steps in user notebook.

1. The file that has the best model has been loaded into user notebook.

2. After this the function “full_input_data “has been created:

Description:

It will take any input for information that related to the house that user want and give him the price of the house.

• create data frame which contains input that doesn’t related to one hot

Encoding column.

• Crate data frame for the one hot encoding column (city)

• concatenate first step with the second step to give us the whole data frame.

That will enter to our model (OUR X_TEST)

Input:

▪ data_before_dm (DataFrame): our data before performing one hot encoding.

▪ city_name (string): the city of the house. Default = “Seattle”.

▪ input_data (list): input of features that doesn’t related to one hot encoder

Output (1d array):

Return our prediction (y_pred)

6.2) Creating web page using FLASK library and (HTML/CSS):

Description:

A web-based House Price Predictor application. It consists of a web page where users can input various features of a house, and upon submitting the form, the application uses a pre-trained machine learning model to predict the house's price. The web page also displays the predicted price.

Input:

Users input various features of a house in the form fields. These include the number of bedrooms, bathrooms, square footage of living space, square footage of the lot, number of floors, waterfront status, view rating, condition rating, grade rating, square footage above ground, square footage of the basement, year built, latitude, and longitude.

Output:

If the user submits the form, the application will process the input data and make a prediction using the loaded machine learning model. The predicted house price will be displayed on the web page.

HTML Page Elements:

The HTML page consists of input fields for the house features, a "Predict" button to submit the form, and a section to display the predicted price.

CSS Styles:

The CSS styles provided in the code snippet are used to format and style the HTML elements on the page, including setting the background image, styling the form and input fields, and defining the appearance of the submit button.

Python Backend:

The Python code uses the Flask framework to create a web application. It loads a pre-trained machine learning model (GB) from a file named "trained_model.sav."

The webpage interface:

Figure (6-1)
Empty interface

The pic below after fill the input features:

Figure (6-2)
fillin interface

The result after click on predict button:

Figure (6-3)
Result of the deployment app

7) Results:

In this part we look in the results for the steps we through in this project:

Result 1:

From feature extraction the result is: try as much as possible to extract the most valuable info from our data.

Result 2:

From feature selection the result is: having two dataframe resulting after backward elimination:

• df_new (without latitude and longitude)

• df_lat (with latitude and longitude)

So, we need to test them in our model to see which one will give us the higher score.

Result 3:

From Regression_model class the result is:

This class can create Random-Forest and Gradiant-Boosting-Regressor for any data, calculate the accuracy, enhance models and checking it generalize our data or not by using learning curve**.**

Result 4:

From Feature_selection class:

Create this class to show if it gives us a different result than back word elimination so maybe if there any different it will enhance the model but there is no different in our case.

Result 5:

This result to show that Gradiant-Boosting-Regressor using df_lat has the higher score than the rest of models. Also, it has a good Generalization Error (over-fitting and under-fitting issue has been solved using hypermeter tuning) and making user interface for it.

7.2) GB model best results:

- Validation_score:0.8787369203393073.
- Train score: 0.9225414026860724.
- Test score: 0.9004212617550983.
- MSE: 13999158142.513332.
- R2 Score: 0.9004212617550983.
- Root mean square error: 118318.03811132659.