# Housing Price Prediction Case Study

## **Multiple Linear Regression**

### **Problem Statement:**

Consider a real estate company that has a dataset containing the prices of properties in the Delhi region. It wishes to use the data to optimize the sale prices of the properties based on important factors such as area, bedrooms, parking, etc.

Essentially, the company wants —

* To identify the variables affecting house prices, e.g., area, number of rooms, bathrooms, etc.
* To create a linear model that quantitatively relates house prices with variables such as number of rooms, area, number of bathrooms, etc.
* To know the accuracy of the model, i.e., how well these variables can predict house prices.

### **Data**

Use housing dataset.

## **Reading and Understanding the Data**

*# Supress Warnings*

import warnings

warnings.filterwarnings('ignore')

*# Import the numpy and pandas package*

import numpy as np

import pandas as pd

*# Data Visualisation*

import matplotlib.pyplot as plt

import seaborn as sns

housing = pd.DataFrame(pd.read\_csv("../input/Housing.csv"))

*# Check the head of the dataset*

housing.head()

## **Data Inspection**

housing.shape

housing.info()

housing.describe()

## **Data Cleaning**

*# Checking Null values*

housing.isnull().sum()\*100/housing.shape[0]

*# There are no NULL values in the dataset, hence it is clean.*

*# Outlier Analysis*

fig, axs = plt.subplots(2,3, figsize = (10,5))

plt1 = sns.boxplot(housing['price'], ax = axs[0,0])

plt2 = sns.boxplot(housing['area'], ax = axs[0,1])

plt3 = sns.boxplot(housing['bedrooms'], ax = axs[0,2])

plt1 = sns.boxplot(housing['bathrooms'], ax = axs[1,0])

plt2 = sns.boxplot(housing['stories'], ax = axs[1,1])

plt3 = sns.boxplot(housing['parking'], ax = axs[1,2])

plt.tight\_layout()

*# Outlier Treatment*

*# Price and area have considerable outliers.*

*# We can drop the outliers as we have sufficient data.*

*# outlier treatment for price*

plt.boxplot(housing.price)

Q1 = housing.price.quantile(0.25)

Q3 = housing.price.quantile(0.75)

IQR = Q3 - Q1

housing = housing[(housing.price >= Q1 - 1.5\*IQR) & (housing.price <= Q3 + 1.5\*IQR)]

*# outlier treatment for area*

plt.boxplot(housing.area)

Q1 = housing.area.quantile(0.25)

Q3 = housing.area.quantile(0.75)

IQR = Q3 - Q1

housing = housing[(housing.area >= Q1 - 1.5\*IQR) & (housing.area <= Q3 + 1.5\*IQR)]

*# Outlier Analysis*

fig, axs = plt.subplots(2,3, figsize = (10,5))

plt1 = sns.boxplot(housing['price'], ax = axs[0,0])

plt2 = sns.boxplot(housing['area'], ax = axs[0,1])

plt3 = sns.boxplot(housing['bedrooms'], ax = axs[0,2])

plt1 = sns.boxplot(housing['bathrooms'], ax = axs[1,0])

plt2 = sns.boxplot(housing['stories'], ax = axs[1,1])

plt3 = sns.boxplot(housing['parking'], ax = axs[1,2])

plt.tight\_layout()

## **Exploratory Data Analytics**

Let's now spend some time doing what is arguably the most important step - **understanding the data**.

* If there is some obvious multicollinearity going on, this is the first place to catch it
* Here's where you'll also identify if some predictors directly have a strong association with the outcome variable

### **Visualising Numeric Variables**

Let's make a pairplot of all the numeric variables

sns.pairplot(housing)

plt.show()

#### **Visualising Categorical Variables**

As you might have noticed, there are a few categorical variables as well. Let's make a boxplot for some of these variables.

plt.figure(figsize=(20, 12))

plt.subplot(2,3,1)

sns.boxplot(x = 'mainroad', y = 'price', data = housing)

plt.subplot(2,3,2)

sns.boxplot(x = 'guestroom', y = 'price', data = housing)

plt.subplot(2,3,3)

sns.boxplot(x = 'basement', y = 'price', data = housing)

plt.subplot(2,3,4)

sns.boxplot(x = 'hotwaterheating', y = 'price', data = housing)

plt.subplot(2,3,5)

sns.boxplot(x = 'airconditioning', y = 'price', data = housing)

plt.subplot(2,3,6)

sns.boxplot(x = 'furnishingstatus', y = 'price', data = housing)

plt.show()

We can also visualise some of these categorical features parallely by using the hue argument. Below is the plot for furnishingstatus with airconditioning as the hue.

plt.figure(figsize = (10, 5))

sns.boxplot(x = 'furnishingstatus', y = 'price', hue = 'airconditioning', data = housing)

plt.show()

## **Data Preparation**

* You can see that your dataset has many columns with values as 'Yes' or 'No'.
* But in order to fit a regression line, we would need numerical values and not string. Hence, we need to convert them to 1s and 0s, where 1 is a 'Yes' and 0 is a 'No'.

*# List of variables to map*

varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']

*# Defining the map function*

def binary\_map(x):

return x.map({'yes': 1, "no": 0})

*# Applying the function to the housing list*

housing[varlist] = housing[varlist].apply(binary\_map)

*# Check the housing dataframe now*

housing.head()

### **Dummy Variables**

The variable furnishingstatus has three levels. We need to convert these levels into integer as well.

For this, we will use something called dummy variables.

*# Get the dummy variables for the feature 'furnishingstatus' and store it in a new variable - 'status'*

status = pd.get\_dummies(housing['furnishingstatus'])

*# Check what the dataset 'status' looks like*

status.head()

Now, you don't need three columns. You can drop the furnished column, as the type of furnishing can be identified with just the last two columns where —

* 00 will correspond to furnished
* 01 will correspond to unfurnished
* 10 will correspond to semi-furnished

*# Let's drop the first column from status df using 'drop\_first = True'*

status = pd.get\_dummies(housing['furnishingstatus'], drop\_first = True)

*# Add the results to the original housing dataframe*

housing = pd.concat([housing, status], axis = 1)

*# Now let's see the head of our dataframe.*

housing.head()

*# Drop 'furnishingstatus' as we have created the dummies for it*

housing.drop(['furnishingstatus'], axis = 1, inplace = True)

housing.head()

### **Splitting the Data into Training and Testing Sets**

from sklearn.model\_selection import train\_test\_split

*# We specify this so that the train and test data set always have the same rows, respectively*

np.random.seed(0)

df\_train, df\_test = train\_test\_split(housing, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

### **Rescaling the Features**

Here we can see that except for area, all the columns have small integer values. So, it is extremely important to rescale the variables so that they have a comparable scale. If we don't have comparable scales, then some of the coefficients as obtained by fitting the regression model might be very large or very small as compared to the other coefficients. This might become very annoying at the time of model evaluation. So, it is advised to use standardization or normalization so that the units of the coefficients obtained are all on the same scale. As you know, there are two common ways of rescaling:

1. Min-Max scaling
2. Standardisation (mean-0, sigma-1)

This time, we will use MinMax scaling.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

*# Apply scaler() to all the columns except the 'yes-no' and 'dummy' variables*

num\_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']

df\_train[num\_vars] = scaler.fit\_transform(df\_train[num\_vars])

df\_train.head()

df\_train.describe()

*# Let's check the correlation coefficients to see which variables are highly correlated*

plt.figure(figsize = (16, 10))

sns.heatmap(df\_train.corr(), annot = True, cmap="YlGnBu")

plt.show()

As you might have noticed, area seems to the correlated to price the most

### **Dividing into X and Y sets for the model building**

y\_train = df\_train.pop('price')

X\_train = df\_train

## **Model Building**

This time, we will be using the **LinearRegression function from SciKit Learn** for its compatibility with RFE (which is a utility from sklearn)

### **RFE**

Recursive feature elimination

*# Importing RFE and LinearRegression*

from sklearn.feature\_selection import RFE

from sklearn.linear\_model import LinearRegression

*# Running RFE with the output number of the variable equal to 10*

lm = LinearRegression()

lm.fit(X\_train, y\_train)

rfe = RFE(lm, 6) *# running RFE*

rfe = rfe.fit(X\_train, y\_train)

list(zip(X\_train.columns,rfe.support\_,rfe.ranking\_))

[('area', True, 1),

col = X\_train.columns[rfe.support\_]

col

X\_train.columns[~rfe.support\_]

### **Building model using statsmodel, for the detailed statistics**

*# Creating X\_test dataframe with RFE selected variables*

X\_train\_rfe = X\_train[col]

*# Adding a constant variable*

import statsmodels.api as sm

X\_train\_rfe = sm.add\_constant(X\_train\_rfe)

lm = sm.OLS(y\_train,X\_train\_rfe).fit() *# Running the linear model*

In [42]:

*#Let's see the summary of our linear model*

print(lm.summary())

*# Calculate the VIFs for the model*

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

vif = pd.DataFrame()

X = X\_train\_rfe

vif['Features'] = X.columns

vif['VIF'] = [variance\_inflation\_factor(X.values, i) for i **in** range(X.shape[1])]

vif['VIF'] = round(vif['VIF'], 2)

vif = vif.sort\_values(by = "VIF", ascending = False)

vif

## **Residual Analysis of the train data**

So, now to check if the error terms are also normally distributed (which is infact, one of the major assumptions of linear regression), let us plot the histogram of the error terms and see what it looks like.

y\_train\_price = lm.predict(X\_train\_rfe)

res = (y\_train\_price - y\_train)

*# Importing the required libraries for plots.*

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

*# Plot the histogram of the error terms*

fig = plt.figure()

sns.distplot((y\_train - y\_train\_price), bins = 20)

fig.suptitle('Error Terms', fontsize = 20) *# Plot heading*

plt.xlabel('Errors', fontsize = 18) *# X-label*

plt.scatter(y\_train,res)

plt.show()

*# There may be some relation in the error terms.*

## **Model Evaluation**

#### **Applying the scaling on the test sets**

num\_vars = ['area','stories', 'bathrooms', 'airconditioning', 'prefarea','parking','price']

df\_test[num\_vars] = scaler.fit\_transform(df\_test[num\_vars])

#### **Dividing into X\_test and y\_test**

y\_test = df\_test.pop('price')

X\_test = df\_test

*# Adding constant variable to test dataframe*

X\_test = sm.add\_constant(X\_test)

*# Now let's use our model to make predictions.*

*# Creating X\_test\_new dataframe by dropping variables from X\_test*

X\_test\_rfe = X\_test[X\_train\_rfe.columns]

*# Making predictions*

y\_pred = lm.predict(X\_test\_rfe)

from sklearn.metrics import r2\_score

r2\_score(y\_test, y\_pred)

*# Plotting y\_test and y\_pred to understand the spread.*

fig = plt.figure()

plt.scatter(y\_test,y\_pred)

fig.suptitle('y\_test vs y\_pred', fontsize=20) *# Plot heading*

plt.xlabel('y\_test', fontsize=18) *# X-label*

plt.ylabel('y\_pred', fontsize=16) *# Y-label*

,0.5,'y\_pred')

We can see that the equation of our best fitted line is:

price=0.35×area+0.20×bathrooms+0.19×stories+0.10×airconditioning+0.10×parking+0.11×prefarea

## **The subplot() Function**

The subplot() function takes three arguments that describes the layout of the figure.

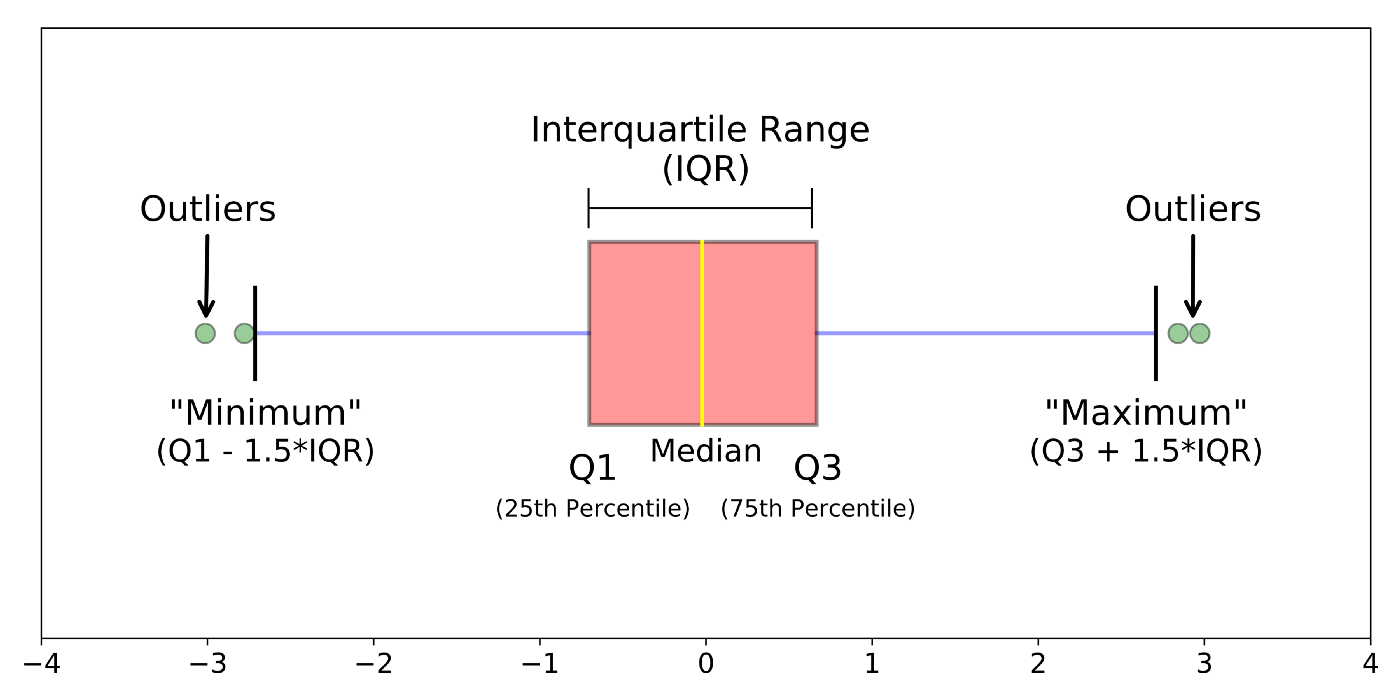
The layout is organized in rows and columns, which are represented by the first and second argument.

The third argument represents the index of the current plot.

plt.subplot(1, 2, 1)  
#the figure has 1 row, 2 columns, and this plot is the first plot.

plt.subplot(1, 2, 2)  
#the figure has 1 row, 2 columns, and this plot is the second plot.

# Understanding Boxplots

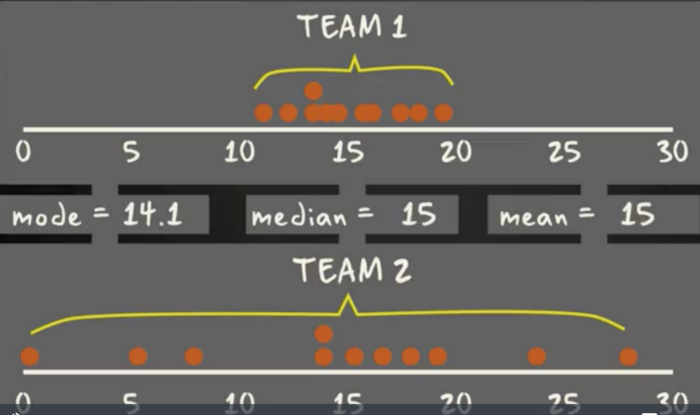


Different parts of a boxplot

The image above is a boxplot**.**A boxplot is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”). It can tell you about your outliers and what their values are. It can also tell you if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed.

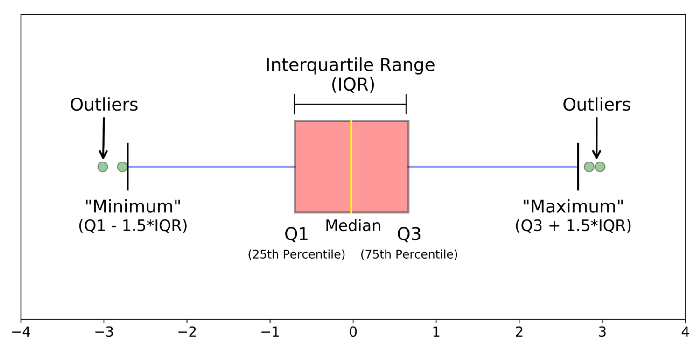
# What is a Boxplot?

For some distributions/datasets, you will find that you need more information than the measures of central tendency (median, mean, and mode).



There are times when mean, median, and mode aren’t enough to describe a dataset

You need to have information on the variability or dispersion of the data. A boxplot is a graph that gives you a good indication of how the values in the data are spread out. Although boxplots may seem primitive in comparison to a [histogram](https://datavizcatalogue.com/methods/histogram.html) or [density plot](https://datavizcatalogue.com/methods/density_plot.html), they have the advantage of taking up less space, which is useful when comparing distributions between many groups or datasets.



Different parts of a boxplot

Boxplots are a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”).

**median (Q2/50th Percentile)**: the middle value of the dataset.

**first quartile (Q1/25th Percentile)**: the middle number between the smallest number (not the “minimum”) and the median of the dataset.

**third quartile (Q3/75th Percentile)**: the middle value between the median and the highest value (not the “maximum”) of the dataset.

**interquartile range (IQR)**: 25th to the 75th percentile.

**whiskers (shown in blue)**

**outliers (shown as green circles)**

**“maximum”**: Q3 + 1.5\*IQR

**“minimum”**: Q1 -1.5\*IQR

## **What Is Multicollinearity?**

Multicollinearity is the occurrence of high intercorrelations among two or more independent variables in a multiple regression model. Multicollinearity can lead to skewed or misleading results when a researcher or analyst attempts to determine how well each independent variable can be used most effectively to predict or understand the dependent variable in a statistical model.

**Interpreting the Linear Regression Summary Output**

The top of our summary starts by giving us a few details we already know. Our **Dependent Variable** is ‘Price,’ we’ve using OLS known as Ordinary Least Squares, and the **Date**and **Time**we’ve created the **Model**. Next, it details our **Number of Observations** in the dataset. **Df Residuals** is another name for our Degrees of Freedom in our model. This is calculated in the form of ‘n-k-1’ or ‘number of observations-number of predicting variables-1.’ **Df Model** numbers our predicting variables.

Our **Covariance Type** is listed as nonrobust. Covariance is a measure of how two variables are linked in a positive or negative manner, and a robust covariance is one that is calculated in a way to minimize or eliminate variables, which is not the case here.

**R-squared** is possibly the most important measurement produced by this summary. R-squared is the measurement of how much of the independent variable is explained by changes in our dependent variables. In percentage terms, 0.611 would mean our model explains 61.1% of the change in our ‘Price’ variable. **Adjusted R-squared** is important for analyzing multiple dependent variables’ efficacy on the model. Linear regression has the quality that your model’s R-squared value will never go down with additional variables, only equal or higher. Therefore, your model could look more accurate with multiple variables even if they are poorly contributing. The adjusted R-squared penalizes the R-squared formula based on the number of variables, therefore a lower adjusted score may be telling you some variables are not contributing to your model’s R-squared properly.

The **F-statistic** in linear regression is comparing your produced linear model for your variables against a model that replaces your variables’ effect to 0, to find out if your group of variables are *statistically significant*. To interpret this number correctly, using a chosen alpha value and an F-table is necessary. **Prob (F-Statistic)** uses this number to tell you the accuracy of the null hypothesis, or whether it is accurate that your variables’ effect is 0. In this case, it is telling us 1.31e-69 chance of this. **Log-likelihood** is a numerical signifier of the likelihood that your produced model produced the given data. It is used to compare coefficient values for each variable in the process of creating the model. **AIC**and **BIC**are both used to compare the efficacy of models in the process of linear regression, using a penalty system for measuring multiple variables. These numbers are used for feature selection of variables.

Now we see the work of our model! Let’s break it down.

The Intercept is the result of our model if all variables were tuned to 0. In the classic ‘y = mx+b’ linear formula, it is our b, a constant added to explain a starting value for our line.

Beneath the intercept are our variables. Remember our formula? ‘Price ~ Area + Bathroom + Stories + Airconditioning + Parking + Prefarea’. Here we see our dependent variables represented.

Our first informative column is the coefficient. For our intercept, it is the value of the intercept. For each variable, it is the measurement of how change in that variable affects the independent variable. It is the ‘m’ in ‘y = mx + b’ One unit of change in the dependent variable will affect the variable’s coefficient’s worth of change in the independent variable. If the coefficient is negative, they have an *inverse* relationship. As one rises, the other falls.

Our std error is an estimate of the standard deviation of the coefficient, a measurement of the amount of variation in the coefficient throughout its data points. The t is related and is a measurement of the precision with which the coefficient was measured. A low std error compared to a high coefficient produces a high t statistic, which signifies a high significance for your coefficient.

P>|t| is one of the most important statistics in the summary. It uses the t statistic to produce the *p value*, a measurement of how likely your coefficient is measured through our model by chance. The p value of 0.000 for Area is saying there is a 0% chance the Area variable has no affect on the dependent variable, Price, and our results are produced by chance. Proper model analysis will compare the p value to a previously established *alpha value*, or a threshold with which we can apply significance to our coefficient. A common alpha is 0.05, which few of our variables pass in this instance.

[0.025 and 0.975] are both measurements of values of our coefficients within 95% of our data, or within two standard deviations. Outside of these values can generally be considered outliers.

**Omnibus**describes the normalcy of the distribution of our residuals using skew and kurtosis as measurements. A 0 would indicate perfect normalcy. **Prob(Omnibus)** is a statistical test measuring the probability the residuals are normally distributed. A 1 would indicate perfectly normal distribution. **Skew**is a measurement of symmetry in our data, with 0 being perfect symmetry. **Kurtosis**measures the peakedness of our data, or its concentration around 0 in a normal curve. Higher kurtosis implies fewer outliers.

**Durbin-Watson** is a measurement of homoscedasticity, or an even distribution of errors throughout our data. Heteroscedasticity would imply an uneven distribution, for example as the data point grows higher the relative error grows higher. Ideal homoscedasticity will lie between 1 and 2.**Jarque-Bera (JB)** and**Prob(JB)** are alternate methods of measuring the same value as Omnibus and Prob(Omnibus) using skewness and kurtosis. We use these values to confirm each other. **Condition number** is a measurement of the sensitivity of our model as compared to the size of changes in the data it is analyzing. Multicollinearity is strongly implied by a high condition number. Multicollinearity a term to describe two or more independent variables that are strongly related to each other and are falsely affecting our predicted variable by redundancy.

The Akaike information criterion (**AIC**) is an estimator of prediction error and thereby relative quality of **statistical** models for a given set of data.

In statistics, the Bayesian information criterion or Schwarz information criterion is a criterion for model selection among a finite set of models; the model with the lowest BIC is preferred. It is based, in part, on the likelihood function and it is closely related to the Akaike information criterion.