FLIP ROBO

House Price Prediction Project

Submitted by:

Ganta Ganesh

ACKNOWLEDGMENT

Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world’s economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company.

INTRODUCTION:

Growing unaffordability of housing has become one of the major challenges for metropolitan cities around the world. In order to gain a better understanding of the commercialized housing market we are currently facing, we want to figure out what are the top influential factors of the housing price. Apart from the more obvious driving forces such as the inflation and the scarcity of land, there are also a number of variables that are worth looking into. Therefore, we choose to study the house prices predicting problem, which enables us to dig into the variables in depth and to provide a model that could more accurately estimate home prices.

Our object is to discuss the major factors that affect housing price and make precise predictions for it. We use 80 explanatory variables including almost every aspect of residential homes. Methods of both statistical regression models and machine learning regression models are applied and further compared according to their performance to better estimate the final price of each house. The model provides price prediction based on similar comparables of people's dream houses, which allows both buyers and sellers to better negotiate home prices according to market trend.

Source of Data:

The study is based on secondary data collected through various internet web sites.

Data Analysis:

Analysis of data and the information collected from the secondary sources were made keeping the objectives of the study in mind.

Project Definition :

To create a model that predicts the price of houses with the available independent variables.

Hardware and Software Requirements :

SYSTEM SPECIFICATION

The hardware and the software specifications of the projects are

1) Hardware Requirements

Processor: Intel I 3
Ram: 4 GB
Hard disk Driver: 50 GB
Monitor: 15” Colour monitor
2) Software requirements
OS: Linux/ Windows/ MAC
Language: Python
Libraries: Jupyter notebook, Python, Matplot lib, Pandas, Numpy.

PROJECT DESCRIPTION:

Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world’s economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company. A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the CSV file below. The company is looking at prospective properties to buy houses to enter the market. You are required to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not. For this company wants to know:

• Which variables are important to predict the price of variable?

• How do these variables describe the price of the house?

Idea

You are required to model the price of houses with the available independent variables. This model will then be used by the management to understand how exactly the prices vary with the variables. They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns. Further, the model will be a good way for the management to understand the pricing dynamics of a new market.

Solution

It is our job to predict the sales price for each house. For each Id in the test set, we must predict the value of the SalePrice variable.

Data Analysis:

Firstly, we do some EDAs to gain a general understanding of our data, and detecting some important metrics and trends that may come helpful for our further analysis and model building.

We will now understand about the features in our dataset.

MSSubClass: Identifies the type of dwelling involved in the sale.

MSZoning: Identifies the general zoning classification of the sale.

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

Alley: Type of alley access to property

LotShape: General shape of property

LandContour: Flatness of the property

Utilities: Type of utilities available

LotConfig: Lot configuration

LandSlope: Slope of property

Neighborhood: Physical locations within Ames city limits

Condition1: Proximity to various conditions

Condition2: Proximity to various conditions (if more than one is present)

BldgType: Type of dwelling

HouseStyle: Style of dwelling

OverallQual: Rates the overall material and finish of the house

OverallCond: Rates the overall condition of the house

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

RoofMatl: Roof material

Exterior1st: Exterior covering on house

Exterior2nd: Exterior covering on house (if more than one material)

MasVnrType: Masonry veneer type

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

ExterCond: Evaluates the present condition of the material on the exterior

Foundation: Type of foundation

BsmtQual: Evaluates the height of the basement

BsmtCond: Evaluates the general condition of the basement

BsmtExposure: Refers to walkout or garden level walls

BsmtFinType1: Rating of basement finished area

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

HeatingQC: Heating quality and condition

CentralAir: Central air conditioning

Electrical: Electrical system

1stFlrSF: First Floor square feet

2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors)

GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade

KitchenQual: Kitchen quality

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

GarageType: Garage location

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

GarageCond: Garage condition

PavedDrive: Paved driveway

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

Fence: Fence quality

MiscFeature: Miscellaneous feature not covered in other categories

MiscVal: $Value of miscellaneous feature

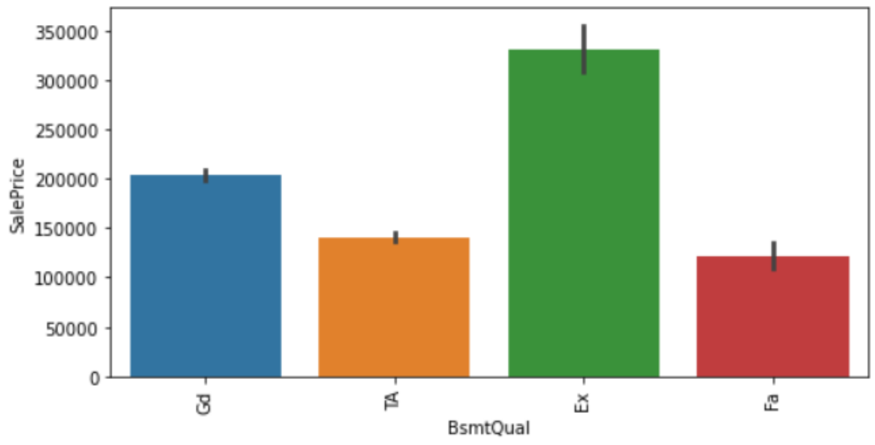
MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

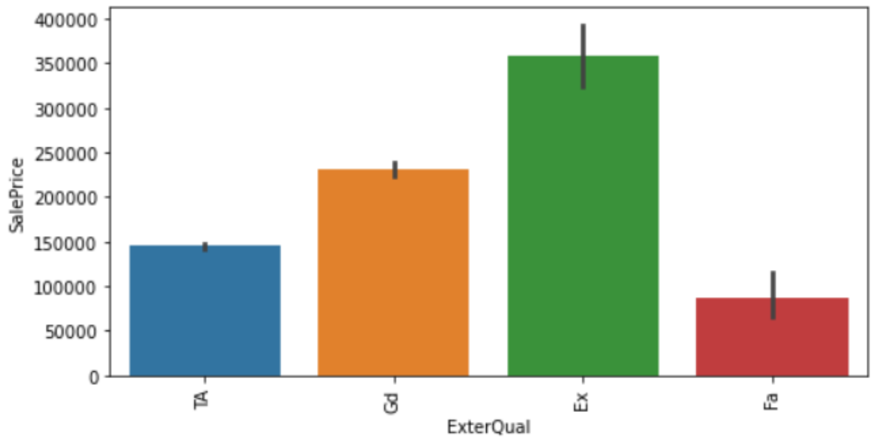
SaleCondition: Condition of sale

```
plt.figure(figsize=[8,4])
sns.barplot(x='BsmtQual',y='SalePrice',data=train)
plt.xticks(rotation = 90)
plt.show()
```



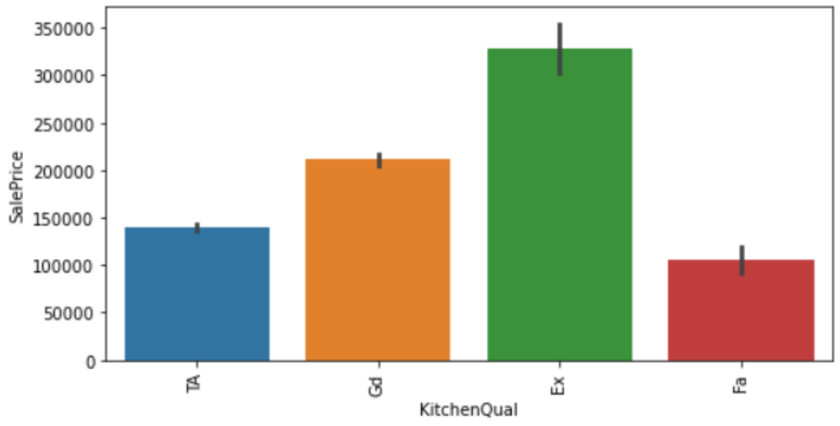
We can see that, for houses whose basement height is more than 100 inches the prices are high.

```
plt.figure(figsize=[8,4])
sns.barplot(x='ExterQual',y='SalePrice',data=train)
plt.xticks(rotation = 90)
plt.show()
```



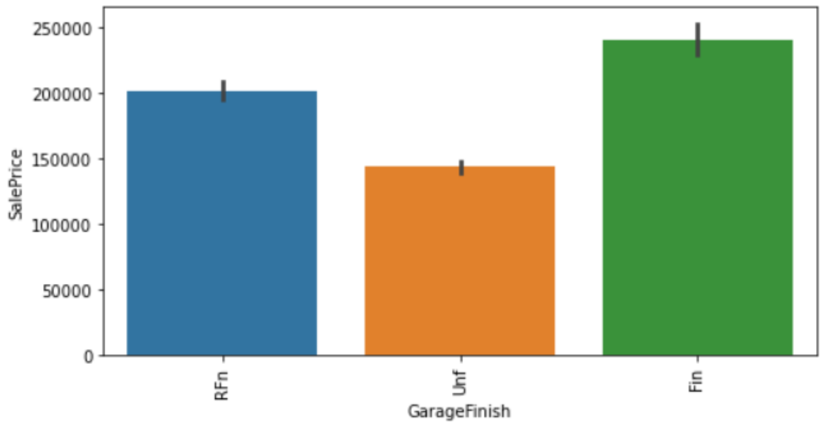
We can see that sale price is higher for Average/Typical type of material on exterior.

```
plt.figure(figsize=[8,4])
sns.barplot(x='KitchenQual',y='SalePrice',data=train)
plt.xticks(rotation = 90)
plt.show()
```



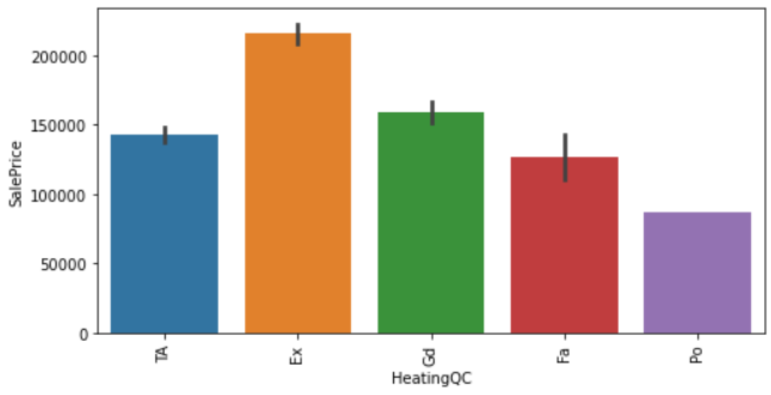
Kitchen Quality has negative relation with SalePrice and we can see that kitchen with excellent quality are more priced.

```
plt.figure(figsize=[8,4])
sns.barplot(x='GarageFinish',y='SalePrice',data=train)
plt.xticks(rotation = 90)
plt.show()
```



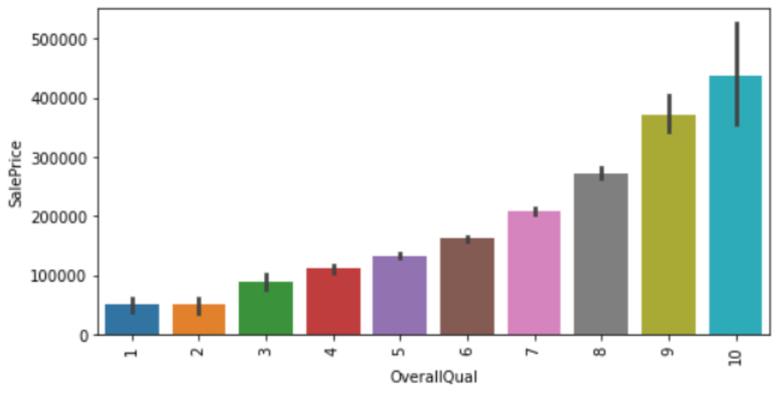
We can see that the house with finished Garage id having more price.

```
plt.figure(figsize=[8,4])
sns.barplot(x='HeatingQC',y='SalePrice',data=train)
plt.xticks(rotation = 90)
plt.show()
```



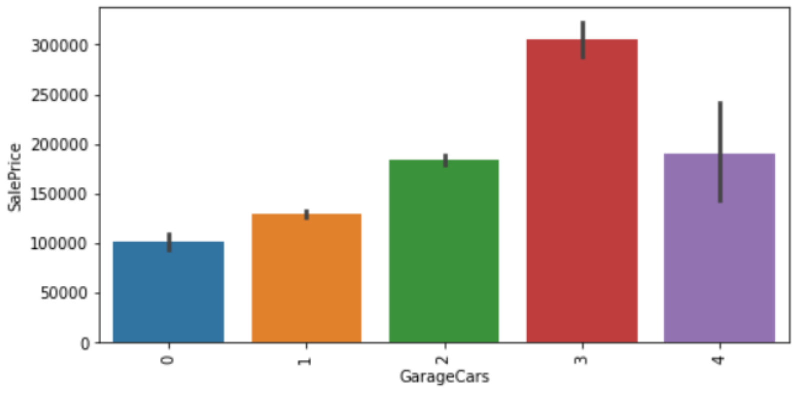
As the heating quality and condition is excellent, the price of the house will be more.

```
plt.figure(figsize=[8,4])
sns.barplot(x='OverallQual',y='SalePrice',data=train)
plt.xticks(rotation = 90)
plt.show()
```



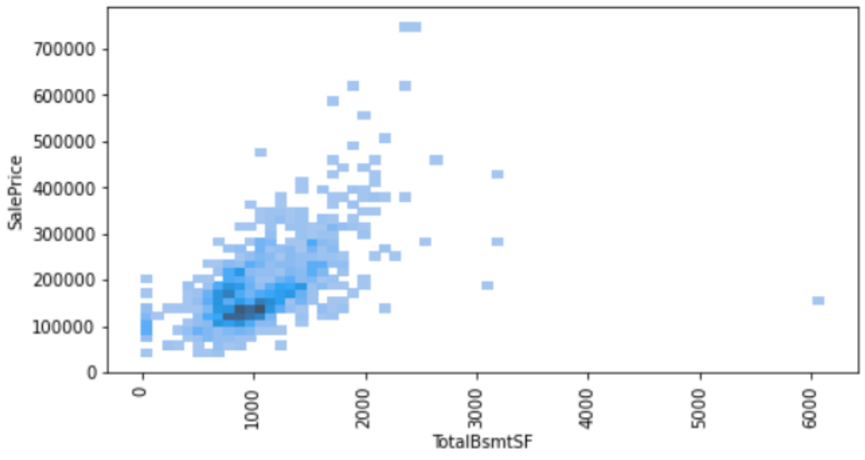
As overall quality of house increases, price is also getting increased.

```python
plt.figure(figsize=[8,4])
sns.barplot(x='GarageCars',y='SalePrice',data=train)
plt.xticks(rotation = 90)
plt.show()
```



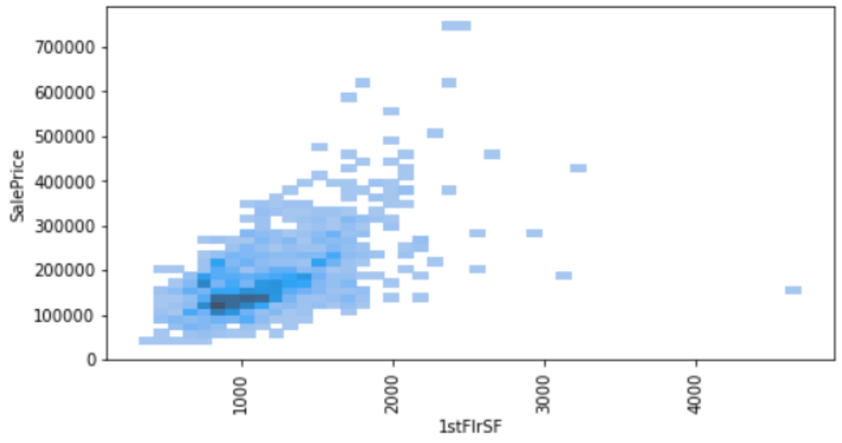
From above plot we can see that house with garage having 3 cars is having more price.

```python
plt.figure(figsize=[8,4])
sns.histplot(x='TotalBsmtSF',y='SalePrice',data=train)
plt.xticks(rotation = 90)
plt.show()
```



From above plot we observe that as basement size is increasing the price is also increased.

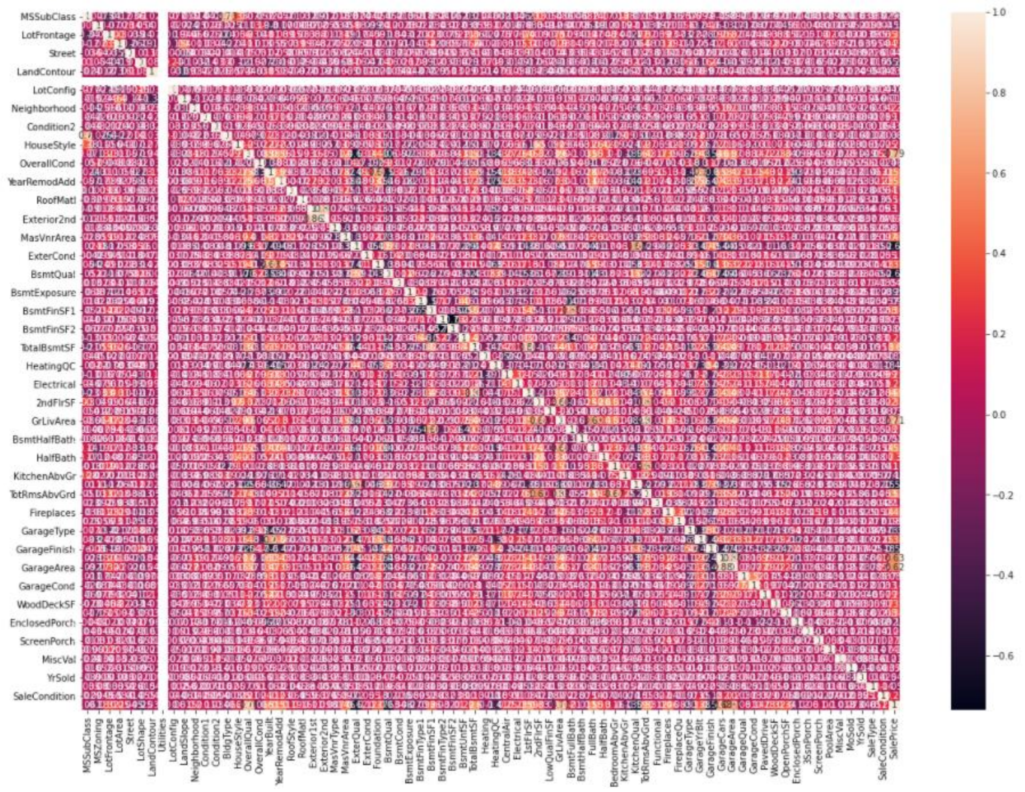
```python
plt.figure(figsize=[8,4])
sns.histplot(x='1stFlrSF',y='SalePrice',data=train)
plt.xticks(rotation = 90)
plt.show()
```

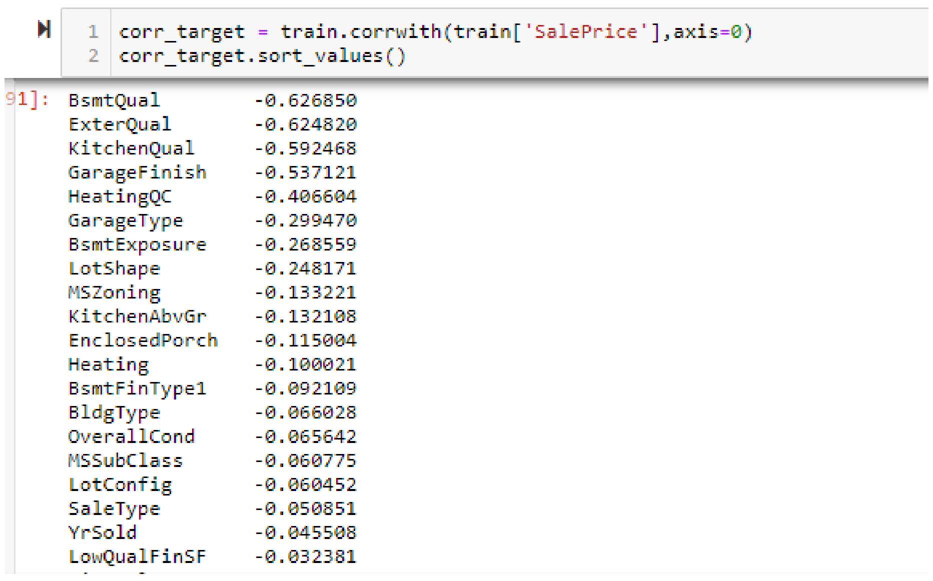


From above plot we observe that as first floor square feet size is increasing the price is also increased.

We examined the correlations between variables and correlations with our outcome of interest: SalePrice.

```python
plt.figure(figsize=[20,14])
sns.heatmap(corr_mat,annot=True)
plt.title = ('Correlation Matrix')
plt.show()
```



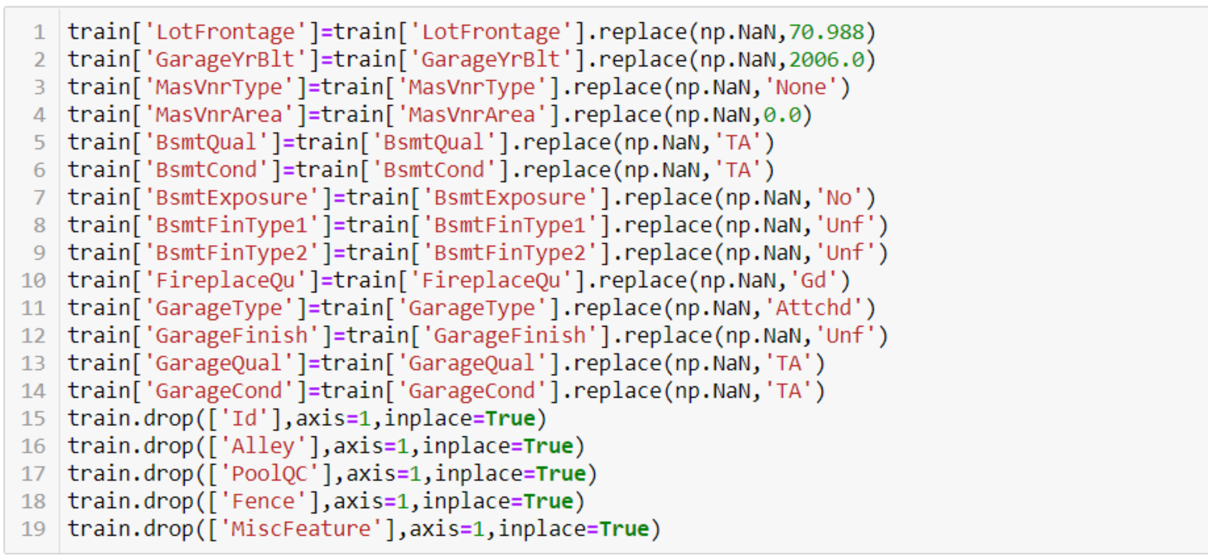


```python
corr_target = train.corrwith(train['SalePrice'],axis=0)
corr_target.sort_values()
```

```
BsmtQual        -0.626850
ExterQual       -0.624820
KitchenQual     -0.592468
GarageFinish    -0.537121
HeatingQC       -0.406604
GarageType      -0.299470
BsmtExposure    -0.268559
LotShape        -0.248171
MSZoning        -0.133221
KitchenAbvGr    -0.132108
EnclosedPorch   -0.115004
Heating         -0.100021
BsmtFinType1    -0.092109
BldgType        -0.066028
OverallCond     -0.065642
MSSubClass      -0.060775
LotConfig       -0.060452
SaleType        -0.050851
YrSold          -0.045508
LowQualFinSF    -0.032381
```

Data Cleaning and Feature Engineering

Before we rush into regression and machine learning prediction, it is very important to get our data “cleaned” enough. This process usually take 80% of time in a real-world data problem. In fact, in our project, we spend about 60% of our time cleaning the data ourselves!



```python
train['LotFrontage']=train['LotFrontage'].replace(np.NaN,70.988)
train['GarageYrBlt']=train['GarageYrBlt'].replace(np.NaN,2006.0)
train['MasVnrType']=train['MasVnrType'].replace(np.NaN,'None')
train['MasVnrArea']=train['MasVnrArea'].replace(np.NaN,0.0)
train['BsmtQual']=train['BsmtQual'].replace(np.NaN,'TA')
train['BsmtCond']=train['BsmtCond'].replace(np.NaN,'TA')
train['BsmtExposure']=train['BsmtExposure'].replace(np.NaN,'No')
train['BsmtFinType1']=train['BsmtFinType1'].replace(np.NaN,'Unf')
train['BsmtFinType2']=train['BsmtFinType2'].replace(np.NaN,'Unf')
train['FireplaceQu']=train['FireplaceQu'].replace(np.NaN,'Gd')
train['GarageType']=train['GarageType'].replace(np.NaN,'Attchd')
train['GarageFinish']=train['GarageFinish'].replace(np.NaN,'Unf')
train['GarageQual']=train['GarageQual'].replace(np.NaN,'TA')
train['GarageCond']=train['GarageCond'].replace(np.NaN,'TA')
train.drop(['Id'],axis=1,inplace=True)
train.drop(['Alley'],axis=1,inplace=True)
train.drop(['PoolQC'],axis=1,inplace=True)
train.drop(['Fence'],axis=1,inplace=True)
train.drop(['MiscFeature'],axis=1,inplace=True)
```

```
test['LotFrontage']=test['LotFrontage'].replace(np.NaN,70.988)
test['GarageYrBlt']=test['GarageYrBlt'].replace(np.NaN,2006.0)
test['MasVnrType']=test['MasVnrType'].replace(np.NaN,'None')
test['MasVnrArea']=test['MasVnrArea'].replace(np.NaN,0.0)
test['Electrical']=test['Electrical'].replace(np.NaN,'SBrkr')
test['BsmtQual']=test['BsmtQual'].replace(np.NaN,'TA')
test['BsmtCond']=test['BsmtCond'].replace(np.NaN,'TA')
test['BsmtExposure']=test['BsmtExposure'].replace(np.NaN,'No')
test['BsmtFinType1']=test['BsmtFinType1'].replace(np.NaN,'Unf')
test['BsmtFinType2']=test['BsmtFinType2'].replace(np.NaN,'Unf')
test['FireplaceQu']=test['FireplaceQu'].replace(np.NaN,'Gd')
test['GarageType']=test['GarageType'].replace(np.NaN,'Attchd')
test['GarageFinish']=test['GarageFinish'].replace(np.NaN,'Unf')
test['GarageQual']=test['GarageQual'].replace(np.NaN,'TA')
test['GarageCond']=test['GarageCond'].replace(np.NaN,'TA')
test.drop(['Id'],axis=1,inplace=True)
test.drop(['Alley'],axis=1,inplace=True)
test.drop(['PoolQC'],axis=1,inplace=True)
test.drop(['Fence'],axis=1,inplace=True)
test.drop(['MiscFeature'],axis=1,inplace=True)
```

There are, indeed, a lot of NAs in our oringinal dataset, which we need to clean the dataset and fill in the NA with appropriate value to make our prediction. Then, we try to fill the NAs by using their properties according to the value in those columns.

For the columns contain large percentage of NAs, we may remove the columns or combine them with other columns and we fill in the missing value with “none”.

<u>Label Encoding</u>

Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()


train['MSZoning']=le.fit_transform(train['MSZoning'])
train['Street']=le.fit_transform(train['Street'])
train['LotShape']=le.fit_transform(train['LotShape'])
train['LandContour']=le.fit_transform(train['LandContour'])
train['Utilities']=le.fit_transform(train['Utilities'])
train['LotConfig']=le.fit_transform(train['LotConfig'])
train['LandSlope']=le.fit_transform(train['LandSlope'])
train['Neighborhood']=le.fit_transform(train['Neighborhood'])
train['Condition1']=le.fit_transform(train['Condition1'])
train['Condition2']=le.fit_transform(train['Condition2'])
train['BldgType']=le.fit_transform(train['BldgType'])
train['HouseStyle']=le.fit_transform(train['HouseStyle'])
train['RoofStyle']=le.fit_transform(train['RoofStyle'])
train['RoofMatl']=le.fit_transform(train['RoofMatl'])
train['Exterior1st']=le.fit_transform(train['Exterior1st'])
train['Exterior2nd']=le.fit_transform(train['Exterior2nd'])
train['MasVnrType']=le.fit_transform(train['MasVnrType'])
train['ExterQual']=le.fit_transform(train['ExterQual'])
train['ExterCond']=le.fit_transform(train['ExterCond'])
train['Foundation']=le.fit_transform(train['Foundation'])
train['BsmtQual']=le.fit_transform(train['BsmtQual'])
train['BsmtCond']=le.fit_transform(train['BsmtCond'])
train['BsmtExposure']=le.fit_transform(train['BsmtExposure'])
train['BsmtFinType1']=le.fit_transform(train['BsmtFinType1'])
train['BsmtFinType2']=le.fit_transform(train['BsmtFinType2'])
train['Heating']=le.fit_transform(train['Heating'])
train['HeatingQC']=le.fit_transform(train['HeatingQC'])
train['CentralAir']=le.fit_transform(train['CentralAir'])
train['Electrical']=le.fit_transform(train['Electrical'])
train['KitchenQual']=le.fit_transform(train['KitchenQual'])
train['Functional']=le.fit_transform(train['Functional'])
train['FireplaceQu']=le.fit_transform(train['FireplaceQu'])
train['GarageType']=le.fit_transform(train['GarageType'])
train['GarageFinish']=le.fit_transform(train['GarageFinish'])
train['GarageQual']=le.fit_transform(train['GarageQual'])
train['GarageCond']=le.fit_transform(train['GarageCond'])
train['PavedDrive']=le.fit_transform(train['PavedDrive'])
train['SaleType']=le.fit_transform(train['SaleType'])
train['SaleCondition']=le.fit_transform(train['SaleCondition'])
```

```
test['MSZoning']=le.fit_transform(test['MSZoning'])
test['Street']=le.fit_transform(test['Street'])
test['LotShape']=le.fit_transform(test['LotShape'])
test['LandContour']=le.fit_transform(test['LandContour'])
test['Utilities']=le.fit_transform(test['Utilities'])
test['LotConfig']=le.fit_transform(test['LotConfig'])
test['LandSlope']=le.fit_transform(test['LandSlope'])
test['Neighborhood']=le.fit_transform(test['Neighborhood'])
test['Condition1']=le.fit_transform(test['Condition1'])
test['Condition2']=le.fit_transform(test['Condition2'])
test['BldgType']=le.fit_transform(test['BldgType'])
test['HouseStyle']=le.fit_transform(test['HouseStyle'])
test['RoofStyle']=le.fit_transform(test['RoofStyle'])
test['RoofMatl']=le.fit_transform(test['RoofMatl'])
test['Exterior1st']=le.fit_transform(test['Exterior1st'])
test['Exterior2nd']=le.fit_transform(test['Exterior2nd'])
test['MasVnrType']=le.fit_transform(test['MasVnrType'])
test['ExterQual']=le.fit_transform(test['ExterQual'])
test['ExterCond']=le.fit_transform(test['ExterCond'])
test['Foundation']=le.fit_transform(test['Foundation'])
test['BsmtQual']=le.fit_transform(test['BsmtQual'])
test['BsmtCond']=le.fit_transform(test['BsmtCond'])
test['BsmtExposure']=le.fit_transform(test['BsmtExposure'])
test['BsmtFinType1']=le.fit_transform(test['BsmtFinType1'])
test['BsmtFinType2']=le.fit_transform(test['BsmtFinType2'])
test['Heating']=le.fit_transform(test['Heating'])
test['HeatingQC']=le.fit_transform(test['HeatingQC'])
test['CentralAir']=le.fit_transform(test['CentralAir'])
test['Electrical']=le.fit_transform(test['Electrical'])
test['KitchenQual']=le.fit_transform(test['KitchenQual'])
test['Functional']=le.fit_transform(test['Functional'])
test['FireplaceQu']=le.fit_transform(test['FireplaceQu'])
test['GarageType']=le.fit_transform(test['GarageType'])
test['GarageFinish']=le.fit_transform(test['GarageFinish'])
test['GarageQual']=le.fit_transform(test['GarageQual'])
test['GarageCond']=le.fit_transform(test['GarageCond'])
test['PavedDrive']=le.fit_transform(test['PavedDrive'])
test['SaleType']=le.fit_transform(test['SaleType'])
test['SaleCondition']=le.fit_transform(test['SaleCondition'])
```

Standardization

The idea behind StandardScaler is that it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1. In case of multivariate data, this is done feature-wise (in other words independently for each column of the data).

Standardization:

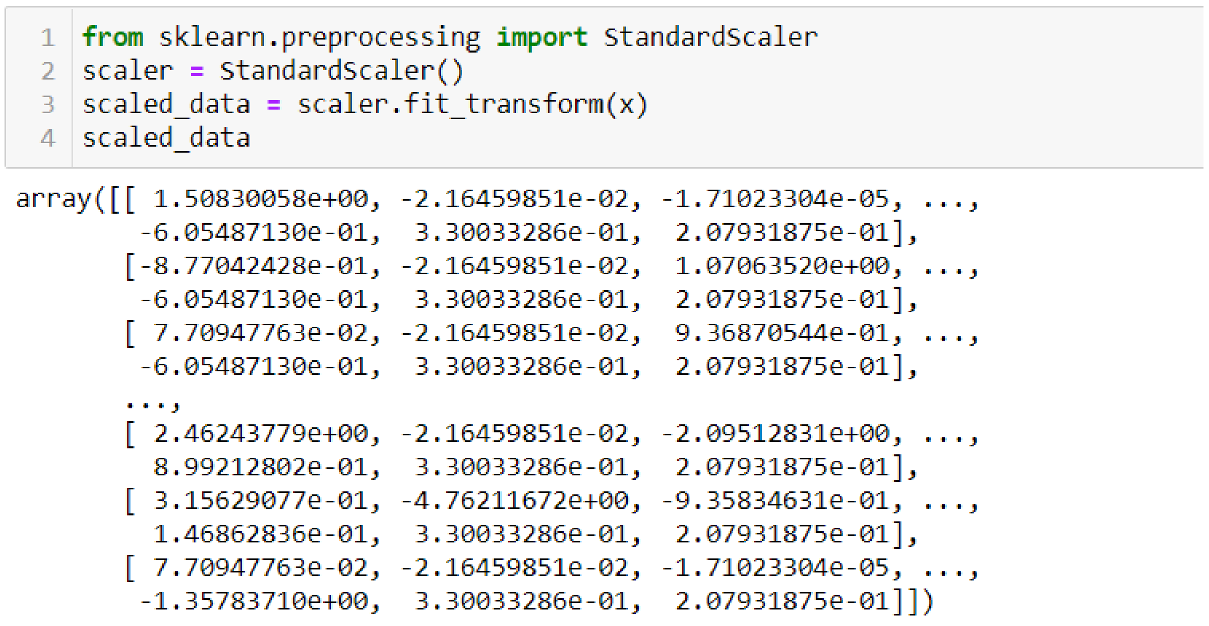
$$z = \frac{x-\mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

and standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}$$



```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(x)
scaled_data
```

```
array([[ 1.50830058e+00, -2.16459851e-02, -1.71023304e-05, ...,
        -6.05487130e-01,  3.30033286e-01,  2.07931875e-01],
       [-8.77042428e-01, -2.16459851e-02,  1.07063520e+00, ...,
        -6.05487130e-01,  3.30033286e-01,  2.07931875e-01],
       [ 7.70947763e-02, -2.16459851e-02,  9.36870544e-01, ...,
        -6.05487130e-01,  3.30033286e-01,  2.07931875e-01],
       ...,
       [ 2.46243779e+00, -2.16459851e-02, -2.09512831e+00, ...,
         8.99212802e-01,  3.30033286e-01,  2.07931875e-01],
       [ 3.15629077e-01, -4.76211672e+00, -9.35834631e-01, ...,
         1.46862836e-01,  3.30033286e-01,  2.07931875e-01],
       [ 7.70947763e-02, -2.16459851e-02, -1.71023304e-05, ...,
        -1.35783710e+00,  3.30033286e-01,  2.07931875e-01]])
```

Principal component analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modelling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are *image processing, movie recommendation system, optimizing the power allocation in various communication channels.* It is a feature extraction technique, so it contains the important variables and drops the least important variable.

The PCA algorithm is based on some mathematical concepts such as:

- Variance and Covariance
- Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that variable are not correlated to each other, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Principal Components in PCA

As described above, the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:

- The principal component must be the linear combination of the original features.
- These components are orthogonal, i.e., the correlation between a pair of variables is zero.
- The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

Steps for PCA algorithm

1. **Getting the dataset**
   Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.
2. **Representing data into a structure**
   Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

3. **Standardizing the data**
   In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.
   If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.
4. **Calculating the Covariance of Z**
   To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.
5. **Calculating the Eigen Values and Eigen Vectors**
   Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.
6. **Sorting the Eigen Vectors**
   In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.
7. **Calculating the new features Or Principal Components**
   Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.
8. **Remove less or unimportant features from the new dataset.**
   The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

<u>Applications of Principal Component Analysis</u>

- PCA is mainly used as the dimensionality reduction technique in various AI applications such **as computer vision, image compression, etc.**
- It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit_transform(scaled_data)
```

```
array([[-0.49620117, -1.5321365 ],
       [ 3.11123285,  0.40231369],
       [ 2.83698913,  0.14239518],
       ...,
       [-0.9155599 , -0.30645974],
       [-6.24647145,  3.49678523],
       [ 2.1545968 , -0.16542859]])
```

```
print(pca.explained_variance_ratio_)
```

```
[0.1378045  0.05504751]
```

```
pca.fit(scaled_data)
```

```
PCA(n_components=2)
```

```
x_pca = pca.transform(scaled_data)
```

```
scaled_data.shape
```

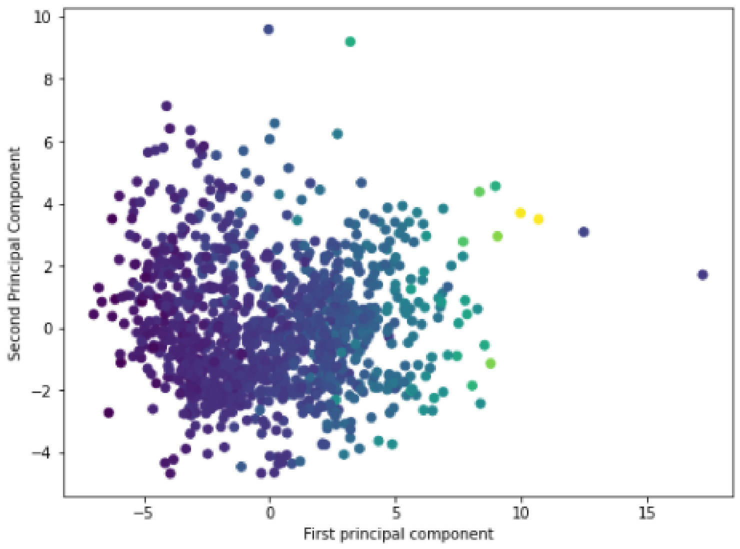
```
(1168, 75)
```

```
x_pca.shape
```

```
(1168, 2)
```

```
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=train['SalePrice'])
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')
```

```
Text(0, 0.5, 'Second Principal Component')
```

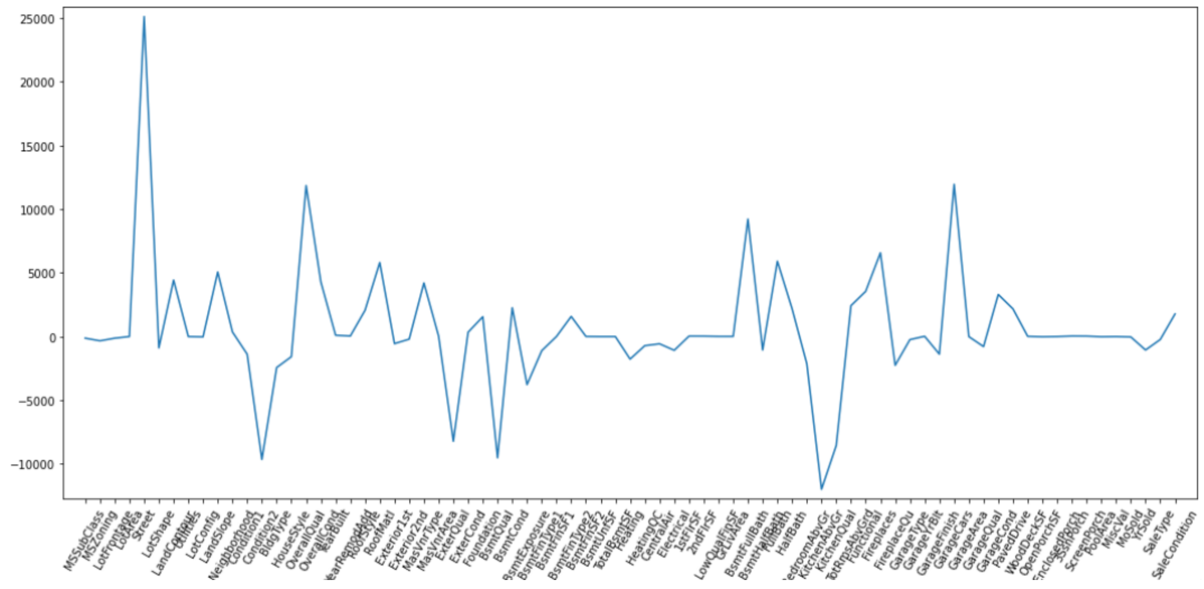


Lasso Regularization :

Lasso (Least Absolute Shrinkage and Selection Operator) regression is a regularized linear regression. It uses L1 norm to constrain the coefficients of the fitting model.

Usually, some coefficients will be set to 0 under the constrain. Therefore, the lasso regression is more robust compared to ordinary linear regression.

```python
colnames = x.columns
plt.figure(figsize=[18,8])
plt.plot(range(len(colnames)), lasso_coef)
plt.xticks(range(len(colnames)), colnames.values, rotation=60)
plt.margins(0.02)
plt.show()
```



Model Building

Linear Regression Model:

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

```python
from sklearn.linear_model import LinearRegression
from sklearn import metrics

reg = LinearRegression()
reg.fit(x_train,y_train)
y_pred = reg.predict(x_test)
print('R^2:',metrics.r2_score(y_test, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_pred))*(len(y_test)-1)/(len(y_test)-x_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
R^2: 0.7096823688392628
Adjusted R^2: 0.7076732502845173
MAE: 26849.952438791308
MSE: 1651336577.9951224
RMSE: 40636.64083059921
```

```python
from sklearn.linear_model import LinearRegression

reg_test = LinearRegression()
reg_test.fit(x_test,y_test)
y_test_pred = reg_test.predict(x_test)
print('R^2:', metrics.r2_score(y_test, y_test_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.7174665831120369
Adjusted R^2: 0.7155113345522586
MAE: 25918.05183379488
MSE: 1607059701.9811175
RMSE: 40088.14914636391
```

Random Forest Regression:

Random Forest Regression is a supervised learning algorithm that uses **ensemble learning** method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model. A Random Forest operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees.

```python
from sklearn.ensemble import RandomForestRegressor

RFR = RandomForestRegressor(n_estimators=10,random_state = 0 )
RFR.fit(x_train, y_train)
y_pred=RFR.predict(x_test)
print('R^2:',metrics.r2_score(y_test, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_pred))*(len(y_test)-1)/(len(y_test)-x_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

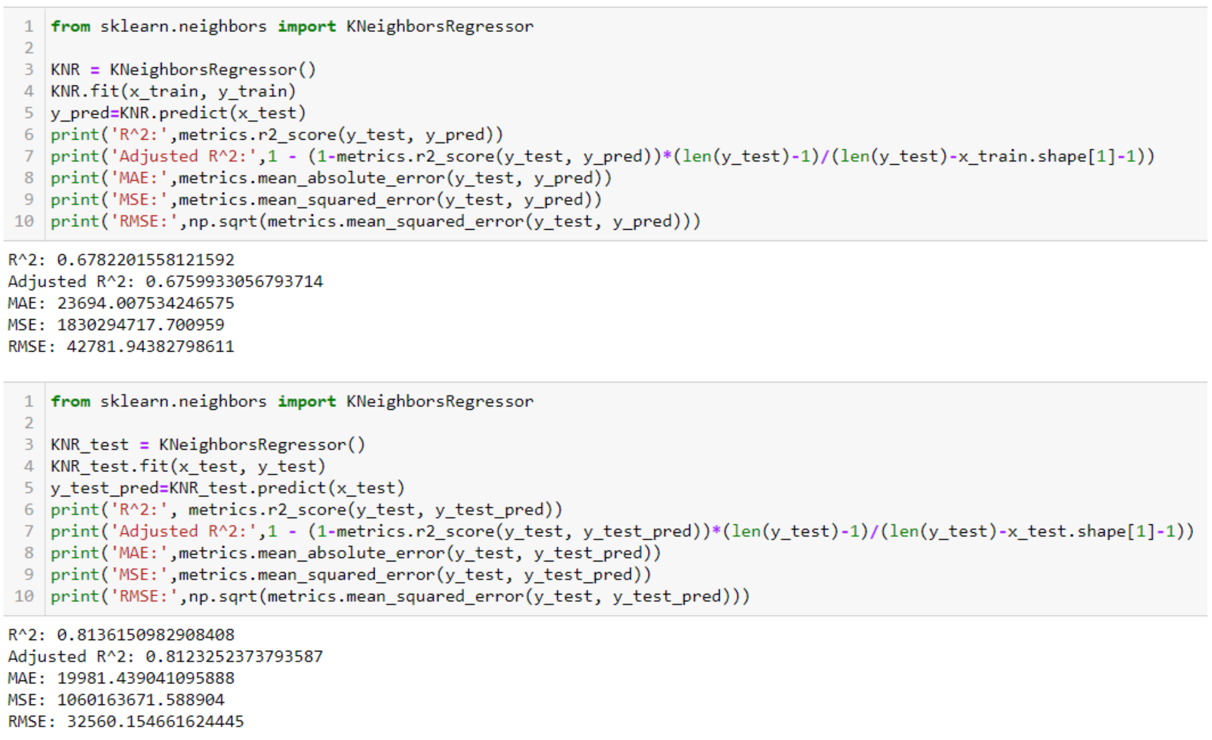
```
R^2: 0.6255195052597082
Adjusted R^2: 0.6229279447424744
MAE: 24886.59109589041
MSE: 2130057813.7053423
RMSE: 46152.54937384654
```

```python
from sklearn.ensemble import RandomForestRegressor

RFR_test = RandomForestRegressor(n_estimators=10,random_state = 0 )
RFR_test.fit(x_test, y_test)
y_test_pred=RFR_test.predict(x_test)
print('R^2:', metrics.r2_score(y_test, y_test_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))

```

```
R^2: 0.9575351413311959
Adjusted R^2: 0.9572412668767405
MAE: 10129.47397260274
MSE: 241541563.00746578
RMSE: 15541.60747823293
```

k-Nearest Neighbors (KNN)

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.



```python
from sklearn.neighbors import KNeighborsRegressor

KNR = KNeighborsRegressor()
KNR.fit(x_train, y_train)
y_pred=KNR.predict(x_test)
print('R^2:',metrics.r2_score(y_test, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_pred))*(len(y_test)-1)/(len(y_test)-x_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

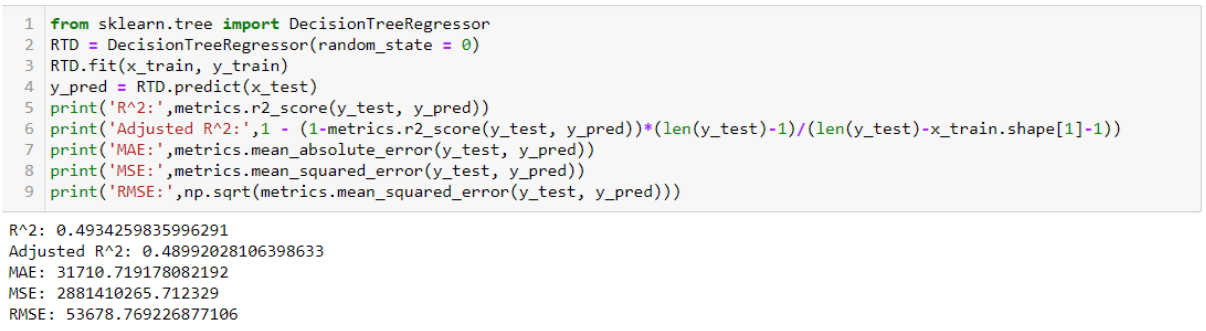
```
R^2: 0.6782201558121592
Adjusted R^2: 0.6759933056793714
MAE: 23694.007534246575
MSE: 1830294717.700959
RMSE: 42781.94382798611
```

```python
from sklearn.neighbors import KNeighborsRegressor

KNR_test = KNeighborsRegressor()
KNR_test.fit(x_test, y_test)
y_test_pred=KNR_test.predict(x_test)
print('R^2:', metrics.r2_score(y_test, y_test_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.8136150982908408
Adjusted R^2: 0.8123252373793587
MAE: 19981.439041095888
MSE: 1060163671.588904
RMSE: 32560.154661624445
```

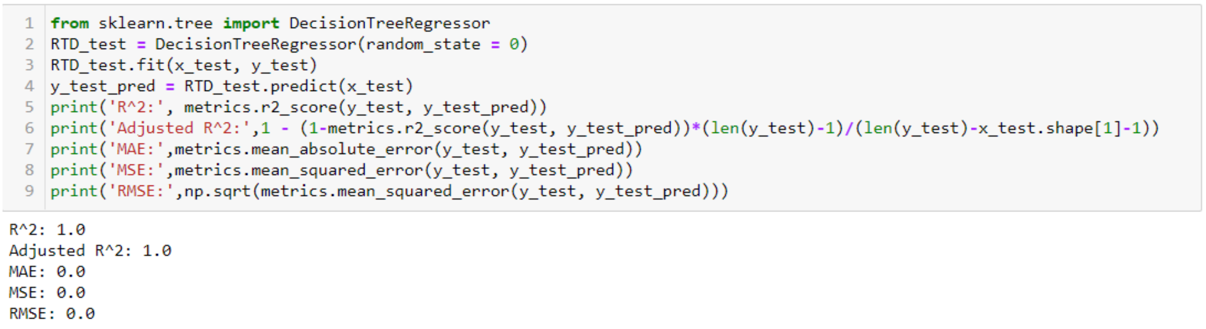
<u>Decision Tree Regressor:</u>

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.



```python
from sklearn.tree import DecisionTreeRegressor
RTD = DecisionTreeRegressor(random_state = 0)
RTD.fit(x_train, y_train)
y_pred = RTD.predict(x_test)
print('R^2:',metrics.r2_score(y_test, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_pred))*(len(y_test)-1)/(len(y_test)-x_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
R^2: 0.4934259835996291
Adjusted R^2: 0.48992028106398633
MAE: 31710.719178082192
MSE: 2881410265.712329
RMSE: 53678.769226877106
```



```python
from sklearn.tree import DecisionTreeRegressor
RTD_test = DecisionTreeRegressor(random_state = 0)
RTD_test.fit(x_test, y_test)
y_test_pred = RTD_test.predict(x_test)
print('R^2:', metrics.r2_score(y_test, y_test_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 1.0
Adjusted R^2: 1.0
MAE: 0.0
MSE: 0.0
RMSE: 0.0
```

Cross Validation:

The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).

```python
scr=cross_val_score(reg, x_pca, y, cv=5)
print('Cross validation score of Linear Regression : ',scr.mean())

scr=cross_val_score(RFR, x_pca, y, cv=5)
print('Cross validation score of RFR : ',scr.mean())

scr=cross_val_score(KNR, x_pca, y, cv=5)
print('Cross validation score of KNR : ',scr.mean())

scr=cross_val_score(RTD, x_pca, y, cv=5)
print('Cross validation score of SVR : ',scr.mean())
```

```
Cross validation score of Linear Regression :  0.7322342198781245
Cross validation score of RFR :  0.7827006462384858
Cross validation score of KNR :  0.7800649449256765
Cross validation score of SVR :  0.7088039586526416
```

```python
scr=cross_val_score(reg_test, x_pca, y, cv=5)
print('Cross validation score of Linear Regression : ',scr.mean())

scr=cross_val_score(RFR_test, x_pca, y, cv=5)
print('Cross validation score of RFR : ',scr.mean())

scr=cross_val_score(KNR_test, x_pca, y, cv=5)
print('Cross validation score of KNR : ',scr.mean())

scr=cross_val_score(RTD_test, x_pca, y, cv=5)
print('Cross validation score of RTD : ',scr.mean())
```

```
Cross validation score of Linear Regression :  0.7322342198781245
Cross validation score of RFR :  0.7827006462384858
Cross validation score of KNR :  0.7800649449256765
Cross validation score of RTD :  0.7088039586526416
```

Linear Regression Model is having least difference between cross validation score and r2 score in both train and test data, so we can choose Linear Regression as best model with r2 score of 70.96% for train data.

<u>Deployment</u>

Deployment is the method by which you integrate a machine learning model into an existing production environment to make practical business decisions based on data.

```python
import joblib
joblib.dump(reg,'House_Price.pkl')
```

```
['House_Price.pkl']
```