Predicting houseprices using

Machine learning

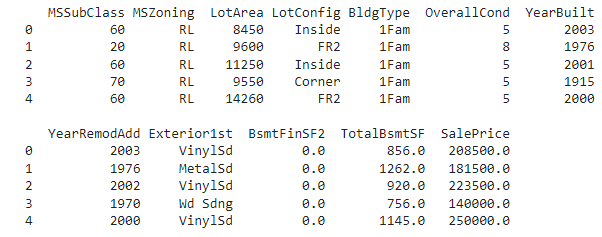
# Datasource:

Here we are using

* [**Pandas**](https://www.geeksforgeeks.org/python-pandas-dataframe/)**–** To load the Dataframe
* [**Matplotlib**](https://www.geeksforgeeks.org/matplotlib-tutorial/)**–** To visualize the data features i.e. barplot
* [**Seaborn**](https://www.geeksforgeeks.org/introduction-to-seaborn-python/)**–** To see the correlation between features using heatmap

|  |
| --- |
| **import** pandas as pd  **import** matplotlib.pyplot as plt  **import** seaborn as sns    dataset **=** pd.read\_excel("HousePricePrediction.xlsx")    # Printing first 5 records of the dataset  print(dataset.head(5)) |

**Output:**



# Datapreprocessing:

Now, we categorize the features depending on their datatype (int, float, object) and then calculate the number of them.

|  |
| --- |
| obj **=** (dataset.dtypes **==** 'object')  object\_cols **=** list(obj[obj].index)  **print**("Categorical variables:",len(object\_cols))    int\_ **=** (dataset.dtypes **==** 'int')  num\_cols **=** list(int\_[int\_].index)  print("Integer variables:",len(num\_cols))    fl **=** (dataset.dtypes **==** 'float')  fl\_cols **=** list(fl[fl].index)  print("Float variables:",len(fl\_cols)) |

## 

## Output:

Categorical variables : 4

Integer variables : 6

Float variables : 3

# Exploratory Data Analysis:

[EDA](https://www.geeksforgeeks.org/what-is-exploratory-data-analysis/) refers to the deep analysis of data so as to discover different patterns and spot anomalies. Before making inferences from data it is essential to examine all your variables.

So here let’s make a [heatmap](https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlation-heatmap-in-python/) using seaborn library.

plt.figure(figsize**=**(12, 6))

sns.heatmap(dataset.corr(),

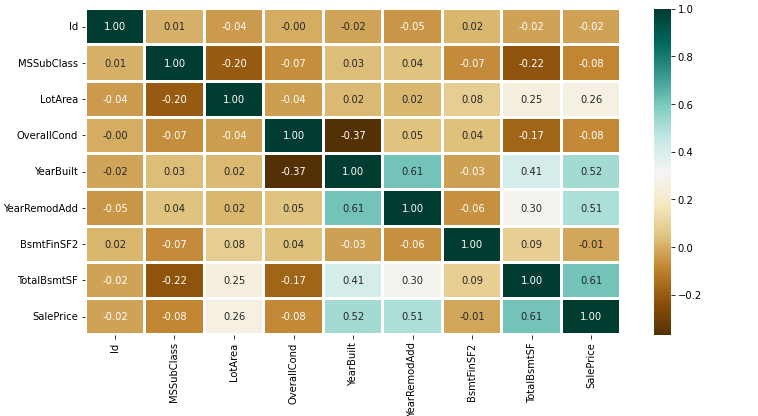
            cmap **=** 'BrBG',

            fmt **=** '.2f',

            linewidths **=** 2,

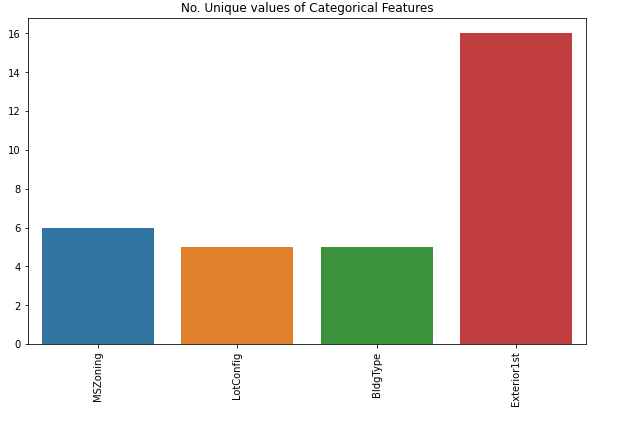
            annot **=** True)

**Output:**



To analyze the different categorical features. Let’s draw the [barplot](https://www.geeksforgeeks.org/bar-plot-in-matplotlib/)

|  |
| --- |
| unique\_values **=** []  **for** col **in** object\_cols:    unique\_values.append(dataset[col].unique().size)  plt.figure(figsize**=**(10,6))  plt.title('No. Unique values of Categorical Features')  plt.xticks(rotation**=**90)  sns.barplot(x**=**object\_cols,y**=**unique\_values) |

**Output:** 

## Data Cleaning

[Data Cleaning](https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/) is the way to improvise the data or remove incorrect, corrupted or irrelevant data.

As in our dataset, there are some columns that are not important and irrelevant for the model training. So, we can drop that column before training. There are 2 approaches to dealing with empty/null values

* We can easily delete the column/row (if the feature or record is not much important).
* Filling the empty slots with mean/mode/0/NA/etc. (depending on the dataset requirement).

As Id Column will not be participating in any prediction. So we can Drop it.

|  |
| --- |
| dataset.drop(['Id'],               axis**=**1,               inplace**=**True) |

Replacing SalePrice empty values with their mean values to make the data distribution symmetric.

|  |
| --- |
| dataset['SalePrice'] **=** dataset['SalePrice'].fillna(    dataset['SalePrice'].mean()) |

Drop records with null values (as the empty records are very less).

|  |
| --- |
| new\_dataset **=** dataset.dropna() |

Checking features which have null values in the new dataframe (if there are still any).

|  |
| --- |
| new\_dataset.isnull().sum() |

**Output**



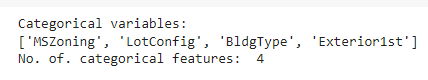
# 3.feature collection:

# OneHotEncoder – For Label categorical features:

One hot Encoding is the best way to convert categorical data into binary vectors. This maps the values to integer values. By using [OneHotEncoder](https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/), we can easily convert object data into int. So for that, firstly we have to collect all the features which have the object datatype. To do so, we will make a loop.

|  |
| --- |
| **from** sklearn.preprocessing **import** OneHotEncoder    s **=** (new\_dataset.dtypes **==** 'object')  object\_cols **=** list(s[s].index)  **print**("Categorical variables:")  print(object\_cols)  print('No. of. categorical features: ',        len(object\_cols)) |

**Output:**



Then once we have a list of all the features. We can apply OneHotEncoding to the whole list.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OH\_encoder **=** OneHotEncoder(sparse**=**False)  OH\_cols **=** pd.DataFrame(OH\_encoder.fit\_transform(new\_dataset[object\_cols]))  OH\_cols.index **=** new\_dataset.index  OH\_cols.columns **=** OH\_encoder.get\_feature\_names()  df\_final **=** new\_dataset.drop(object\_cols, axis**=**1)  df\_final **=** pd.concat([df\_final, OH\_cols], axis**=**1) 4.model selection: As we have to train the model to determine the continuous values, so we will be using these regression models.   * SVM-Support Vector Machine * Random Forest Regressor * Linear Regressor   And To calculate loss we will be using the [mean\_absolute\_percentage\_error](https://www.geeksforgeeks.org/how-to-calculate-mape-in-python/) module. It can easily be imported by using sklearn library. The formula for Mean Absolute Error :     **SVM – Support vector Machine:** SVM can be used for both regression and classification model. It finds the hyperplane in the n-dimensional plane. To read more about svm   |  | | --- | | **from** sklearn **import** svm  **from** sklearn.svm **import** SVC  **from** sklearn.metrics **import** mean\_absolute\_percentage\_error    model\_SVR **=** svm.SVR()  model\_SVR.fit(X\_train,Y\_train)  Y\_pred **=** model\_SVR.predict(X\_valid)    print(mean\_absolute\_percentage\_error(Y\_valid, Y\_pred)) |   **Output :**  0.18705129 **Random Forest Regression:** Random Forest is an ensemble technique that uses multiple of decision trees and can be used for both regression and classification tasks. To read more about random forests   |  | | --- | | **from** sklearn.ensemble **import** RandomForestRegressor    model\_RFR **=** RandomForestRegressor(n\_estimators**=**10)  model\_RFR.fit(X\_train, Y\_train)  Y\_pred **=** model\_RFR.predict(X\_valid)    mean\_absolute\_percentage\_error(Y\_valid, Y\_pred) |   **Output :**  0.1929469 **Linear Regression** Linear Regression predicts the final output-dependent value based on the given independent features. Like, here we have to predict SalePrice depending on features like MSSubClass, YearBuilt, BldgType, Exterior1st etc. To read more about Linear Regression   |  | | --- | | **from** sklearn.linear\_model **import** LinearRegression    model\_LR **=** LinearRegression()  model\_LR.fit(X\_train, Y\_train)  Y\_pred **=** model\_LR.predict(X\_valid)    print(mean\_absolute\_percentage\_error(Y\_valid, Y\_pred)) |   **Output :**  0.187416838 **CatBoostClassifier:** CatBoost is a machine learning algorithm implemented by Yandex and is open-source. It is simple to interface with deep learning frameworks such as Apple’s Core ML and Google’s TensorFlow. Performance, ease-of-use, and robustness are the main advantages of the CatBoost library. To read more about CatBoost refer [this](https://catboost.ai/en/docs/concepts/python-reference_catboostclassifier).   |  | | --- | | # This code is contributed by @amartajisce  **from** catboost **import** CatBoostRegressor  cb\_model **=** CatBoostRegressor()  cb\_model.fit(X\_train, y\_train)  preds **=** cb\_model.predict(X\_valid)    cb\_r2\_score**=**r2\_score(Y\_valid, preds)  cb\_r2\_score output: |   0.893643437976127 |