SURYA GROUP OF INSTITUTION

VIKRAVANDI 605-652

PHASE 3 DEVELOPMENT PART 1

PREDICTION HOUSE PRICES USING MACHINE LEARNING

(Building your project by Loading and Preprocessing the Dataset.)

NAAN MUDHALVAN

PREPARED BY:

ANJUGAM C

REG NO :422221106002

ECE DEPARTMENT

3RD YEAR 5TH SEM

AI_ PHASE 3:

In this part you will begin building your project by loading and preprocessing the dataset.
Start building the house price prediction model by loading and preprocessing the dataset.
Load the housing dataset and preprocess the data.

Data Preprocessing in Machine learning

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

Why do we need Data Preprocessing?

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

- **Getting the dataset**
- **Importing libraries**
- **Importing datasets**
- **Finding Missing Data**
- **Encoding Categorical Data**
- **Splitting dataset into training and test set**
- **Feature scaling**

Data Preprocessing:

Data preprocessing is a predominant step in machine learning to yield highly accurate and insightful results. Greater the quality of data, greater is the reliance on the produced results. **Incomplete, noisy, and inconsistent data** are the properties of large real-world datasets. Data preprocessing helps in increasing the quality of data by filling in missing incomplete data, smoothing noise and resolving inconsistencies.

- **Incomplete data** can occur for a number of reasons. Attributes of interest may not always be available, such as customer information for sales transaction data. Relevant data may not be recorded due to a misunderstanding, or because of equipment malfunctions.
- There are many possible reasons for **noisy data** (having incorrect attribute values). The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. Errors in data transmission can also occur. Incorrect data may also result from inconsistencies in naming conventions or data codes used, or inconsistent formats for input fields, such as date.

There are a number of data preprocessing techniques available such as,

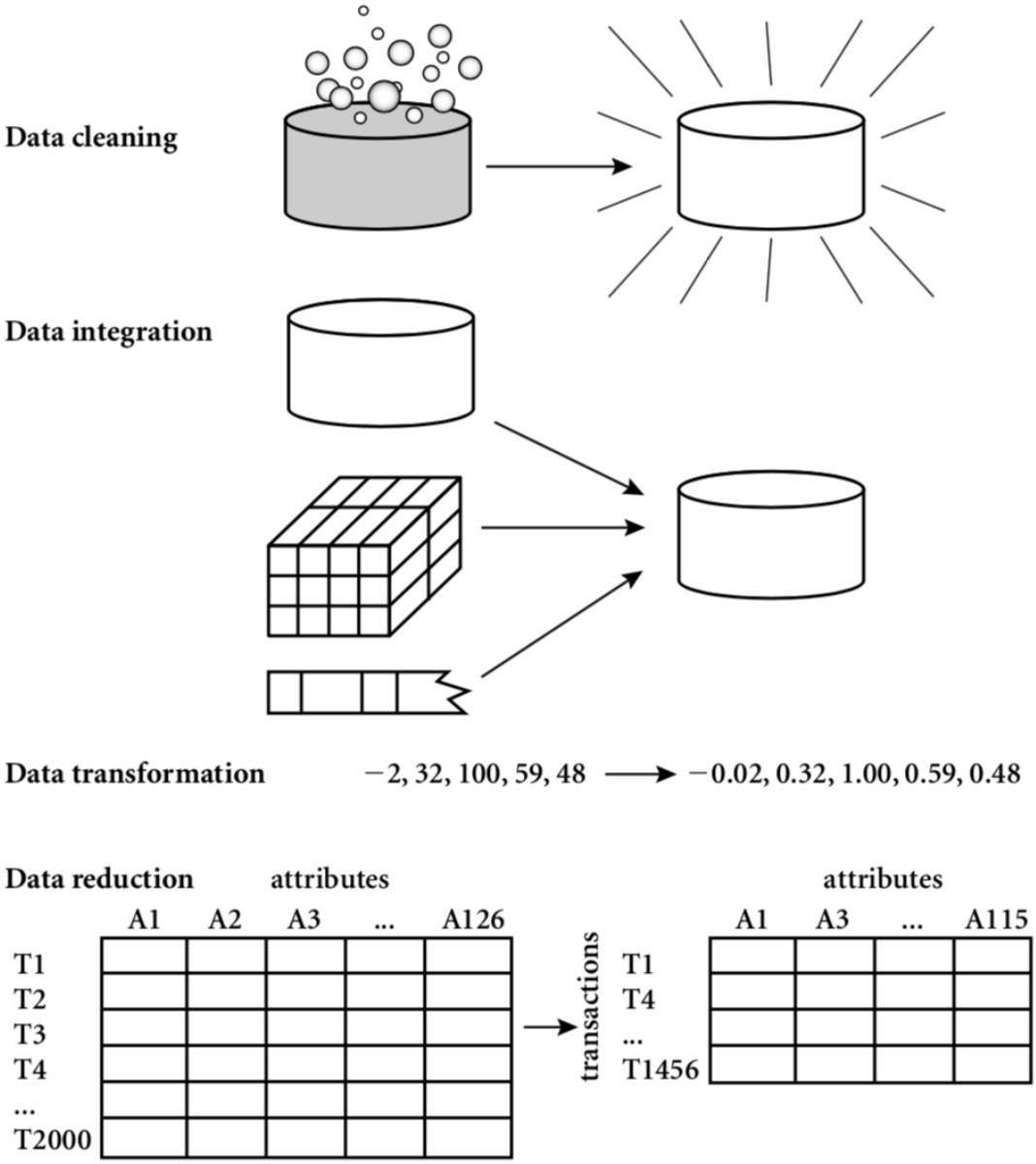
1. **Data Cleaning**
2. **Data Integration**
3. **Data Transformation**
4. **Data Reduction**

Data cleaning can be applied to filling in missing values, remove noise, resolving inconsistencies, identifying and removing outliers in the data.

Data integration merges data from multiple sources into a coherent data store, such as a data warehouse.

Data transformations, such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurements.

Data reduction can reduce the data size by eliminating redundant features, or clustering, for instance.



Import the required libraries

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from operator import itemgetter
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import OrdinalEncoder
from category_encoders.target_encoder import TargetEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import (GradientBoostingRegressor, GradientBoostingClass
ifier)
import xgboost
```

Load the dataset for training and testing

```
train = pd.read_csv('../input/house-prices-advanced-regression-techniques/train.csv')
test = pd.read_csv('../input/house-prices-advanced-regression-techniques/test.csv')
```

1. Data Cleaning

1.1 Find the missing percentage of each columns in training set.

```
def find_missing_percent(data):
    """
    Returns dataframe containing the total missing values and percentage of total
    missing values of a column.
    """
    miss_df = pd.DataFrame({'ColumnName':[],'TotalMissingVals':[],'PercentMissing':[]})
    for col in data.columns:
        sum_miss_val = data[col].isnull().sum()
        percent_miss_val = round((sum_miss_val/data.shape[0])*100,2)
        miss_df = miss_df.append(dict(zip(miss_df.columns,[col,sum_miss_val,percent_miss_val])),ignore_index=True)
    return miss_df
```

```
miss_df = find_missing_percent(train)
'''Displays columns with missing values'''
display(miss_df[miss_df['PercentMissing']>0.0])
print("\n")
print(f"Number of columns with missing values:{str(miss_df[miss_df['PercentMissing']>0.0].shape[0])}")
```

	ColumnName	TotalMissingVals	PercentMissing
3	LotFrontage	259.0	17.74
6	Alley	1369.0	93.77
25	MasVnrType	8.0	0.55
26	MasVnrArea	8.0	0.55
30	BsmtQual	37.0	2.53
31	BsmtCond	37.0	2.53
32	BsmtExposure	38.0	2.60
33	BsmtFinType1	37.0	2.53
35	BsmtFinType2	38.0	2.60
42	Electrical	1.0	0.07
57	FireplaceQu	690.0	47.26
58	GarageType	81.0	5.55
59	GarageYrBlt	81.0	5.55
60	GarageFinish	81.0	5.55
63	GarageQual	81.0	5.55
64	GarageCond	81.0	5.55
72	PoolQC	1453.0	99.52
73	Fence	1179.0	80.75
74	MiscFeature	1406.0	96.30

```
Number of columns with missing values:19
```

1.2 Drop the columns which have more than 70% of missing values

```
drop_cols = miss_df[miss_df['PercentMissing'] >70.0].ColumnName.tolist()
print(f"Number of columns with more than 70%: {len(drop_cols)}")
train = train.drop(drop_cols,axis=1)
test = test.drop(drop_cols,axis =1)

miss_df = miss_df[miss_df['ColumnName'].isin(train.columns)]
'''Columns to Impute'''
impute_cols = miss_df[miss_df['TotalMissingVals']>0.0].ColumnName.tolist()
miss_df[miss_df['TotalMissingVals']>0.0]
```

```
Number of columns with more than 70%: 4
```

Out[5]:

	ColumnName	TotalMissingVals	PercentMissing
3	LotFrontage	259.0	17.74
25	MasVnrType	8.0	0.55
26	MasVnrArea	8.0	0.55
30	BsmtQual	37.0	2.53
31	BsmtCond	37.0	2.53
32	BsmtExposure	38.0	2.60
33	BsmtFinType1	37.0	2.53
35	BsmtFinType2	38.0	2.60
42	Electrical	1.0	0.07
57	FireplaceQu	690.0	47.26
58	GarageType	81.0	5.55
59	GarageYrBlt	81.0	5.55
60	GarageFinish	81.0	5.55
63	GarageQual	81.0	5.55
64	GarageCond	81.0	5.55

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
```

2. Handling Missing data:

The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset.

Ways to handle missing data:

There are mainly two ways to handle missing data, which are:

By deleting the particular row: The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.

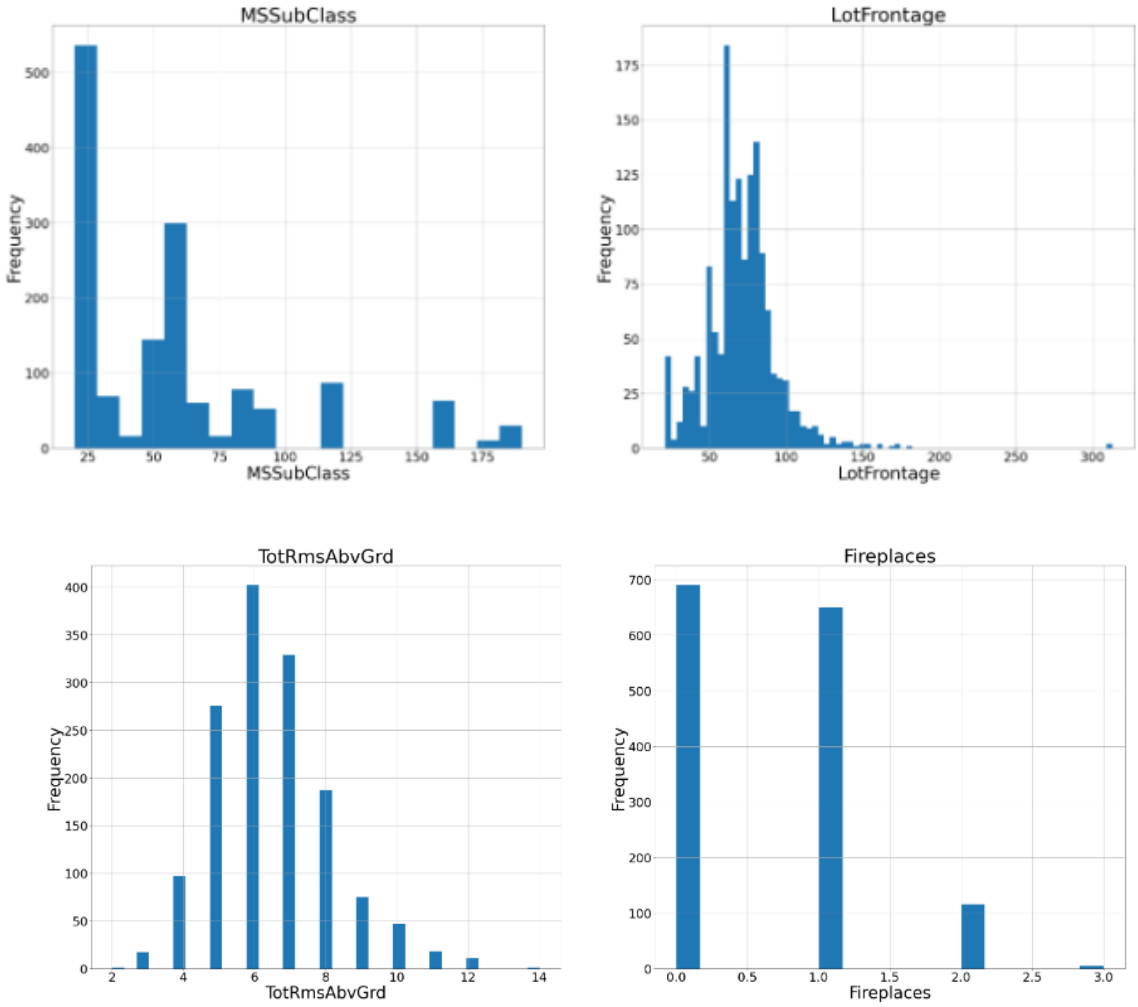
By calculating the mean: In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc. Here, we will use this approach.

To handle missing values, we will use **Scikit-learn** library in our code, which contains various libraries for building machine learning models. Here we will use **Imputer** class of **sklearn.preprocessing** library. Below is the code for it:

```
'''
These columns have IQR equal to zero. Freedman Diaconis Rule doesn't work sign
ificantly well for these columns.
Use sturges rule to find the optimal number of bins for the columns.
'''
cols_list = ['LowQualFinSF','BsmtFinSF2','BsmtHalfBath','KitchenAbvGr',
             'EnclosedPorch','3SsnPorch','ScreenPorch','PoolArea','MiscVal']

# Except ID
```

```
hist_cols = numeric_cols[1:]
for i in range(0,len(hist_cols),2):
    if(i == len(hist_cols)-1):
        plot_histogram(train,hist_cols[i],hist_cols[i],cols_list,True)
    else:
        plot_histogram(train,hist_cols[i],hist_cols[i+1],cols_list)
```



3. Encoding Categorical data:

Categorical data is data which has some categories such as, in our dataset; there are two categorical variable, **Country**, and **Purchased**.

Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

For Country variable:

Firstly, we will convert the country variables into categorical data. So to do this, we will use **LabelEncoder()** class from **preprocessing** library.

```
#Catgorical data
#for Country Variable
from sklearn.preprocessing import LabelEncoder
label_encoder_x= LabelEncoder()
x[:, 0]= label_encoder_x.fit_transform(x[:, 0])
```

Output:

```
Out[15]:
  array([[2, 38.0, 68000.0],
           [0, 43.0, 45000.0],
         [1, 30.0, 54000.0],
         [0, 48.0, 65000.0],
         [1, 40.0, 65222.22222222222],
         [2, 35.0, 58000.0],
         [1, 41.111111111111114, 53000.0],
         [0, 49.0, 79000.0],
         [2, 50.0, 88000.0],
        [0, 37.0, 77000.0]], dtype=object)
```

Explanation:

In above code, we have imported **LabelEncoder** class of **sklearn library**. This class has successfully encoded the variables into digits.

But in our case, there are three country variables, and as we can see in the above output, these variables are encoded into 0, 1, and 2. By these values, the machine learning model may assume that there is some correlation between these variables which will produce the wrong output. So to remove this issue, we will use **dummy encoding**.

Dummy Variables:

Dummy variables are those variables which have values 0 or 1. The 1 value gives the presence of that variable in a particular column, and rest variables become 0. With dummy encoding, we will have a number of columns equal to the number of categories.

In our dataset, we have 3 categories so it will produce three columns having 0 and 1 values. For Dummy Encoding, we will use **OneHotEncoder** class of **preprocessing** library.

CONCLUSION:

In the above code, we have included all the data preprocessing steps together. But there are some steps or lines of code which are not necessary for all machine learning models. So we can exclude them from our code to make it reusable for House price prediction model.