**Naan Mudhalvan**

**IBM project**

**Artificial Intelligence(Phase 3- Development)**

## Topic- House Price Prediction using Machine

## Learning in python

**TABELS OF CONTENTS**

**SI.NO: TOPICS**

**3.1 Dataset and its detail explanation**

**Implementation**

**3.1.1 Basic Libraries**

**3.1.2 Implementation**

**3.2 Begin building the project by load the**

**dataset**

**3.2.1 Import libraries**

**3.2.2 Dataset**

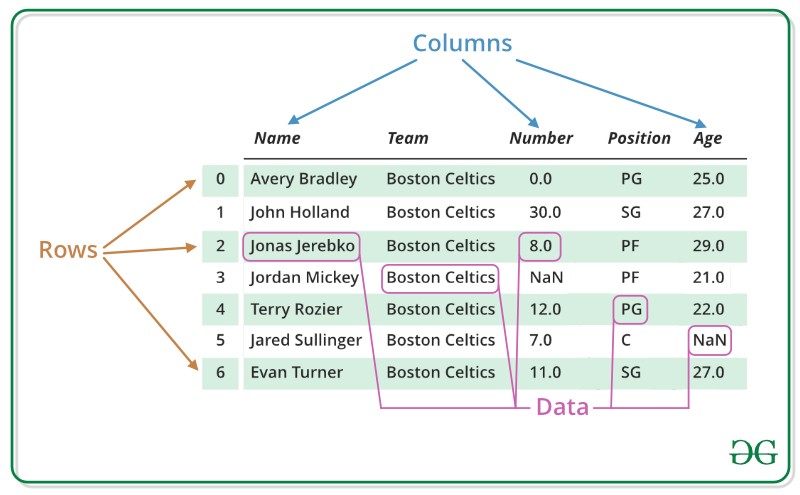
**3.2.3 Data Visualization**

**3.1 Dataset and its detail explanation implementation:**

**3.1.1 Basic Libraries:**

1. **Pandas – To load the Dataframe**
2. **Matplotlib – To visualize the data features i.e. barplot**
3. **Seaborn – To see the correlation between features using heatmap.**
4. **Pandas**

**Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.**



**Basic operation which can be performed on Pandas DataFrame :**

* **Creating a DataFrame**
* **Dealing with Rows and Columns**
* **Indexing and Selecting Data**
* **Working with Missing Data**
* **Iterating over rows and columns**

**2. Matplotlib**

**Pyplot is a Matplotlib module that provides a MATLAB-like interface. Pyplot provides functions that interact with the figure i.e. creates a figure, decorates the plot with labels, and creates a plotting area in a figure.**

**Syntax:**

**matplotlib.pyplot.plot(\*args, scalex=True, scaley=True, data=None, \*\*kwargs)**

***Example:***

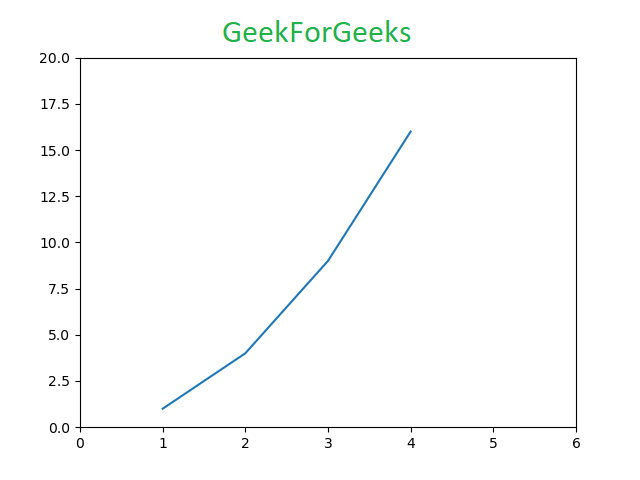
**# Python program to show pyplot module**

**import matplotlib.pyplot as plt**

**plt.plot([1, 2, 3, 4], [1, 4, 9, 16])**

**plt.axis([0, 6, 0, 20])**

**plt.show()**



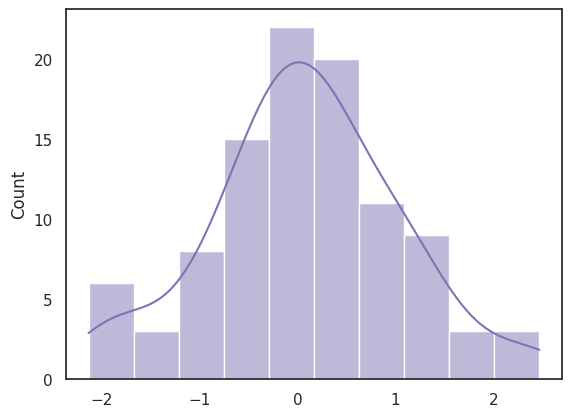
**3.Seaborn**

**Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on top matplotlib library and is also closely integrated with the data structures from pandas.**

**Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs so that we can switch between different visual representations for the same variables for a better understanding of the dataset.**

**Installation of Seaborn Library**

**pip install seaborn**



**3.1.2 Implementation:**

**import pandas as pd**

**import matplotlib.pyplot as plt**

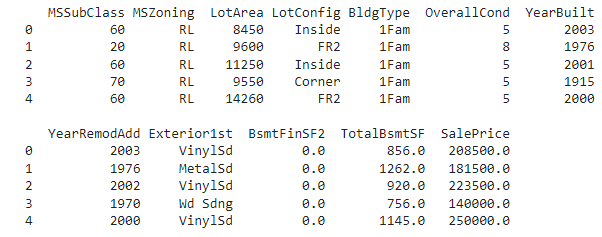
**import seaborn as sns**

**dataset = pd.read\_excel("HousePricePrediction.xlsx")**

**# Printing first 5 records of the dataset**

**print(dataset.head(5))**

**Output:**



**3.2 Begin building the project by load the dataset:**

**3.2.1 Import libraries:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import mean\_absolute\_error

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

from sklearn import svm

from sklearn.svm import SVC

from sklearn.metrics import mean\_absolute\_percentage\_error

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear\_model import LinearRegression

from catboost import CatBoostRegressor

**3.2.2 Dataset:**

dataset.drop(['Id'],

axis=1,

inplace=True)

dataset['SalePrice'] = dataset['SalePrice'].filna(

dataset['SalePrice'].mean())

new\_dataset = dataset.dropna()

new\_dataset.isnull().sum()

**Output for dataset:**



## Data Visualization:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.axes as ax

data = pd.read\_csv('data\_for\_lr.csv')

# Drop the missing values

data = data.dropna()

# training dataset and labels

train\_input = np.array(data.x[0:500]).reshape(500,1)

train\_output = np.array(data.y[0:500]).reshape(500,1)

# valid dataset and labels

test\_input = np.array(data.x[500:700]).reshape(199,1)

test\_output = np.array(data.y[500:700]).reshape(199,1)

class LinearRegression:

def \_\_init\_\_(self):

self.parameters = {}

def forward\_propagation(self, train\_input):

m = self.parameters['m']

c = self.parameters['c']

predictions = np.multiply(m, train\_input) + c

return predictions

def cost\_function(self, predictions, train\_output):

cost = np.mean((train\_output - predictions) \*\* 2)

return cost

def backward\_propagation(self, train\_input, train\_output, predictions):

derivatives = {}

df = (train\_output - predictions) \* -1

dm = np.mean(np.multiply(train\_input, df))

dc = np.mean(df)

derivatives['dm'] = dm

derivatives['dc'] = dc

return derivatives

def update\_parameters(self, derivatives, learning\_rate):

self.parameters['m'] = self.parameters['m'] - learning\_rate \* derivatives['dm']

self.parameters['c'] = self.parameters['c'] - learning\_rate \* derivatives['dc']

def train(self, train\_input, train\_output, learning\_rate, iters):

#initialize random parameters

self.parameters['m'] = np.random.uniform(0,1) \* -1

self.parameters['c'] = np.random.uniform(0,1) \* -1

#initialize loss

self.loss = []

#iterate

for i in range(iters):

#forward propagation

predictions = self.forward\_propagation(train\_input)

#cost function

cost = self.cost\_function(predictions, train\_output)

#append loss and print

self.loss.append(cost)

print("Iteration = {}, Loss = {}".format(i+1, cost))

#back propagation

derivatives = self.backward\_propagation(train\_input, train\_output, predictions)

#update parameters

self.update\_parameters(derivatives, learning\_rate)

return self.parameters, self.loss

#Prediction on test data

y\_pred = test\_input\*parameters['m'] + parameters['c']

# Plot the regression line with actual data pointa

plt.plot(test\_input, test\_output, '+', label='Actual values')

plt.plot(test\_input, y\_pred, label='Predicted values')

plt.xlabel('Test input')

plt.ylabel('Test Output or Predicted output')

plt.legend()

plt.show()

## Best fit Linear regression line with actual values - Geeksforgeeks