Name : Neeraj Rijhwani Roll No. 64 Class:D15C

Aim: To implement the regression algorithm using Python.

Regression Algorithm:

Regression algorithms predict the output values based on input features from the data fed in the system. The go-to methodology is the algorithm builds a model on the features of training data and uses the model to predict the value for new data.

Regression has a wide range of real-life applications. It is essential for any machine learning problem that involves continuous numbers – this includes, but is not limited to, a host of examples, including:

- Financial forecasting (like house price estimates, or stock prices)
- Sales and promotions forecasting
- Testing automobiles
- Weather analysis and prediction
- Time series forecasting

Some of the most used regression algorithms are given below

1. **Simple Linear Regression model:**
    a. Simple linear regression is a statistical method that enables users to summarize and study relationships between two continuous (quantitative) variables.
    b. Linear regression is a linear model wherein a model that assumes a linear relationship between the input variables (x) and the single output variable (y).
    c. Here the "y" can be calculated from a linear combination of the input variables (x).
    d. When there is a single input variable (x), the method is called a simple linear regression. When there are multiple input variables, the procedure is referred to as multiple linear regression.

    Application: some of the most popular applications of Linear regression algorithms are in financial portfolio prediction, salary forecasting, real estate predictions and in traffic arriving at ETAs.

2. **Lasso Regression:**
    a. LASSO stands for Least Absolute Selection Shrinkage Operator wherein shrinkage is defined as a constraint on parameters.
    b. The goal of lasso regression is to obtain the subset of predictors that minimize prediction error for a quantitative response variable. The algorithm operates by imposing a constraint on the model parameters that causes regression coefficients for some variables to shrink toward a zero.

c. Variables with a regression coefficient equal to zero after the shrinkage process are excluded from the model. Variables with non-zero regression coefficient variables are most strongly associated with the response variable.

d. Explanatory variables can be either quantitative, categorical or both. This lasso regression analysis is basically a shrinkage and variable selection method and it helps analysts to determine which of the predictors are most important.

Application: Lasso regression algorithms have been widely used in financial networks and economics. In finance, its application is seen in forecasting probabilities of default and Lasso-based forecasting models are used in assessing enterprise wide risk framework. Lasso-type regressions are also used to perform stress test platforms to analyze multiple stress scenarios.

3. **Logistic regression:**

a. One of the most commonly used regression techniques in the industry which are extensively applied across fraud detection, credit card scoring and clinical trials, wherever the response is binary has a major advantage.

b. One of the major upsides of this popular algorithm is that one can include more than one dependent variable which can be continuous or dichotomous. The other major advantage of this supervised machine learning algorithm is that it provides a quantified value to measure the strength of association according to the rest of variables.

c. Despite its popularity, researchers have drawn out its limitations, citing a lack of robust technique and also a great model dependency.

Application: Today enterprises deploy Logistic Regression to predict house values in real estate business, customer lifetime value in the insurance sector and are leveraged to produce a continuous outcome such as whether a customer can buy/will buy scenario.

4. **Multivariate Regression algorithm:**

a. This technique is used when there is more than one predictor variable in a multivariate regression model and the model is called a multivariate multiple regression.

b. Termed as one of the simplest supervised machine learning algorithms by researchers, this regression algorithm is used to predict the response variable for a set of explanatory variables.

c. This regression technique can be implemented efficiently with the help of matrix operations and in Python, it can be implemented via the “numpy” library which contains definitions and operations for matrix objects.

Application: Industry application of Multivariate Regression algorithm is seen heavily in the retail sector where customers make a choice on a number of variables such as brand, price and product. The multivariate analysis helps decision makers to find the best combination of factors to increase footfalls in the store.

5. **Multiple Regression Algorithm:**
    a. This regression algorithm has several applications across the industry for product pricing, real estate pricing, marketing departments to find out the impact of campaigns.
    b. Unlike linear regression technique, multiple regression, is a broader class of regressions that encompasses linear and nonlinear regressions with multiple explanatory variables.

Application: Some of the business applications of multiple regression algorithms in the industry are in social science research, behavioral analysis and even in the insurance industry to determine claim worthiness.

Python Library Function Used:

Python library used for linear regression is Scikit-learn
sklearn.linear_model.LinearRegression

LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Methods for model.LinearRegression are given below

1. **fit(X, y[, sample_weight]):** Fit linear model.
2. **get_params([deep]):** Get parameters for this estimator.
3. **predict(X):** Predict using the linear model.
4. **score(X, y[, sample_weight]):** Return the coefficient of determination of the prediction.
5. **set_params(**params):** Set the parameters of this estimator.

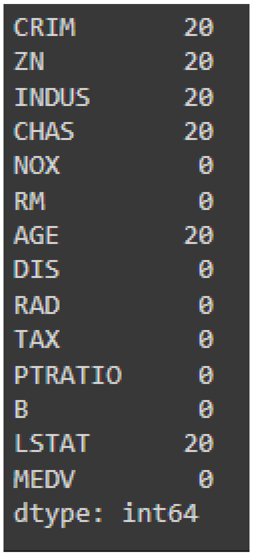
Data Modeling and Analysis

Dataset: Boston Housing Dataset

Preprocessing:

Handling missing values -

```
df.isnull().sum()
```

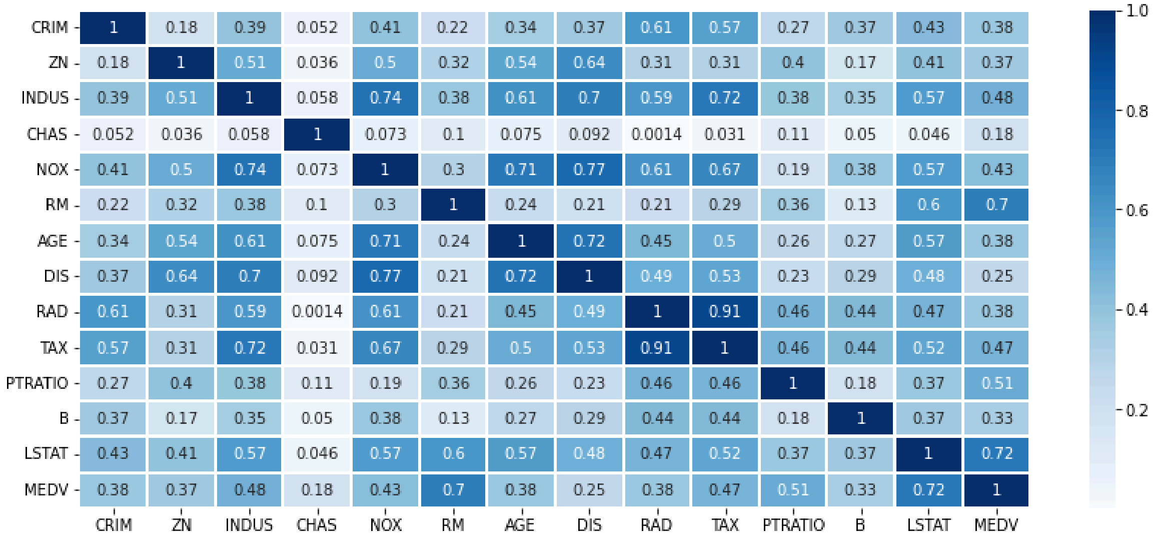


```
CRIM        20
ZN          20
INDUS       20
CHAS        20
NOX          0
RM           0
AGE         20
DIS          0
RAD          0
TAX          0
PTRATIO      0
B            0
LSTAT       20
MEDV         0
dtype: int64
```

```
df["CRIM"].fillna(df["CRIM"].mean(),inplace=True)
df["ZN"].fillna(df["ZN"].mean(),inplace=True)
df["INDUS"].fillna(df["INDUS"].mean(),inplace=True)
df["CHAS"].fillna(df["CHAS"].mean(),inplace=True)
df["AGE"].fillna(df["AGE"].mean(),inplace=True)
df["LSTAT"].fillna(df["LSTAT"].mean(),inplace=True)
```

Correlation Analysis:

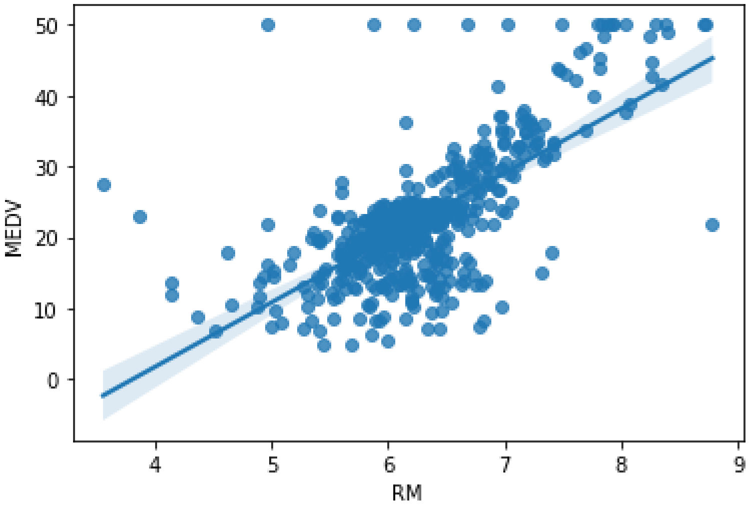
```
plt.figure(figsize=(14,6))
corr=abs(df.corr())
sns.heatmap(corr,annot=True,linewidth=1,cmap="Blues")
plt.show()
```



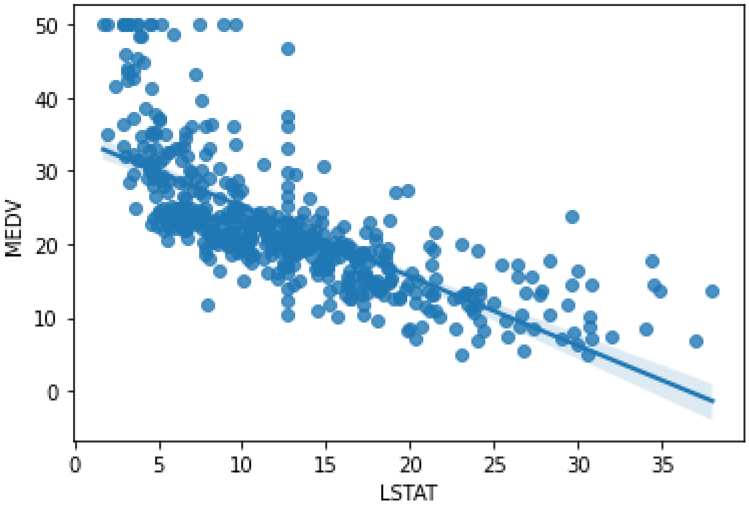
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
CRIM	1	0.18	0.39	0.052	0.41	0.22	0.34	0.37	0.61	0.57	0.27	0.37	0.43	0.38
ZN	0.18	1	0.51	0.036	0.5	0.32	0.54	0.64	0.31	0.31	0.4	0.17	0.41	0.37
INDUS	0.39	0.51	1	0.058	0.74	0.38	0.61	0.7	0.59	0.72	0.38	0.35	0.57	0.48
CHAS	0.052	0.036	0.058	1	0.073	0.1	0.075	0.092	0.0014	0.031	0.11	0.05	0.046	0.18
NOX	0.41	0.5	0.74	0.073	1	0.3	0.71	0.77	0.61	0.67	0.19	0.38	0.57	0.43
RM	0.22	0.32	0.38	0.1	0.3	1	0.24	0.21	0.21	0.29	0.36	0.13	0.6	0.7
AGE	0.34	0.54	0.61	0.075	0.71	0.24	1	0.72	0.45	0.5	0.26	0.27	0.57	0.38
DIS	0.37	0.64	0.7	0.092	0.77	0.21	0.72	1	0.49	0.53	0.23	0.29	0.48	0.25
RAD	0.61	0.31	0.59	0.0014	0.61	0.21	0.45	0.49	1	0.91	0.46	0.44	0.47	0.38
TAX	0.57	0.31	0.72	0.031	0.67	0.29	0.5	0.53	0.91	1	0.46	0.44	0.52	0.47
PTRATIO	0.27	0.4	0.38	0.11	0.19	0.36	0.26	0.23	0.46	0.46	1	0.18	0.37	0.51
B	0.37	0.17	0.35	0.05	0.38	0.13	0.27	0.29	0.44	0.44	0.18	1	0.37	0.33
LSTAT	0.43	0.41	0.57	0.046	0.57	0.6	0.57	0.48	0.47	0.52	0.37	0.37	1	0.72
MEDV	0.38	0.37	0.48	0.18	0.43	0.7	0.38	0.25	0.38	0.47	0.51	0.33	0.72	1

Data Visualization for Regression:

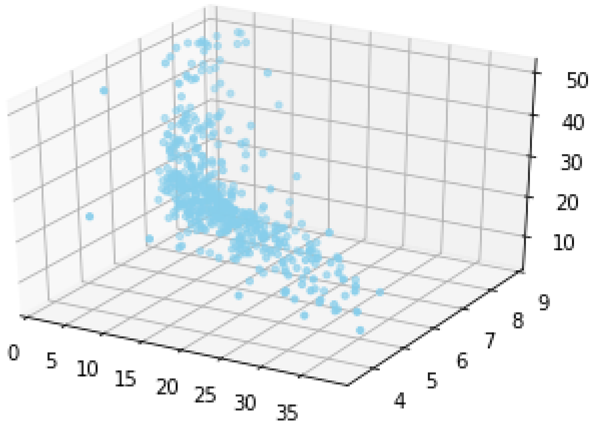
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.regplot(x="RM", y="MEDV", data=df)
```



```
sns.regplot(x="LSTAT", y="MEDV", data=df)
```



```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df['LSTAT'], df['RM'], df['MEDV'], c='skyblue', s=10)
plt.show()
```



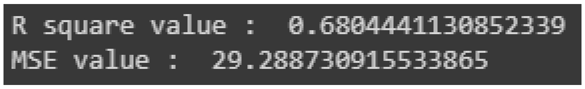
Train Test Split:

```
X = df.drop(columns=['MEDV'])
y = df['MEDV']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3,random_state=1)
```

Code and Observation:

Using Python Library:

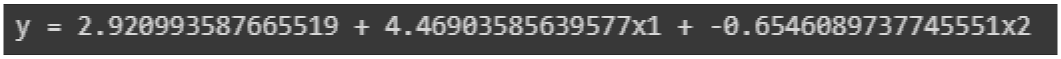
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train[["RM","LSTAT"]], y_train)
y_predicted = lr.predict(X_test[["RM","LSTAT"]])

from sklearn.metrics import mean_squared_error, r2_score
print("R square value : ",r2_score(y_test, y_predicted))
print("MSE value : ",mean_squared_error(y_test, y_predicted))
```



```
R square value :  0.6804441130852339
MSE value :  29.288730915533865
```

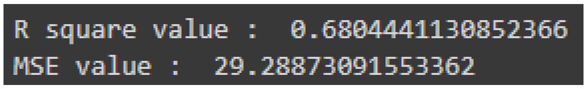
Using User Defined Function:

```
from itertools import zip_longest
from sklearn.metrics import mean_squared_error, r2_score
class CustomLinearRegression():

  def __init__(self, X1, X2, Y, N):
    self.X1 = X1
    self.X2 = X2
    self.Y = Y
    self.N = N
    self.a = 0
    self.b1 = 0
    self.b2 = 0

  def fit(self):
    sum_X1 = np.sum(self.X1)
    sum_X2 = np.sum(self.X2)
    sum_Y = np.sum(self.Y)
    X1_Mean = np.mean(self.X1)
    X2_Mean = np.mean(self.X2)
    Y_Mean = np.mean(self.Y)

    sum_X1_sq = np.sum(self.X1 ** 2)
    sum_X2_sq = np.sum(self.X2 ** 2)

    sum_X1_Y = np.sum(self.X1 * self.Y)
    sum_X2_Y = np.sum(self.X2 * self.Y)
    sum_X1_X2 = np.sum(self.X1 * self.X2)

    x12 = sum_X1_sq - ((sum_X1 ** 2)/self.N)
    x22 = sum_X2_sq - ((sum_X2 ** 2)/self.N)

    x1y = sum_X1_Y - ((sum_X1 * sum_Y)/self.N)
    x2y = sum_X2_Y - ((sum_X2 * sum_Y)/self.N)
    x1x2 = sum_X1_X2 - ((sum_X1 * sum_X2)/self.N)


    self.b1 =  ((x22 * x1y) - (x1x2*x2y)) / ((x12*x22) - (x1x2 ** 2))
    self.b2 =  ((x12 * x2y) - (x1x2*x1y)) / ((x12*x22) - (x1x2 ** 2))
    self.a = Y_Mean - self.b1* X1_Mean - self.b2* X2_Mean

  def print_equation(self):
    if self.a !=0 and self.b1 != 0 and self.b2 != 0:
      print(f"y = {self.a} + {self.b1}x1 + {self.b2}x2")
    else:
      print("You need to use fit method first.")

  def predict(self, x1_test, x2_test):
    y_pred = [self.a + self.b1*x1 + self.b2*x2 for x1,x2 in
zip_longest(x1_test, x2_test)]
    return y_pred

  def show_metrics(self, y_test, y_pred):
    print("R square value : ",r2_score(y_test, y_pred))
    print("MSE value : ",mean_squared_error(y_test, y_pred))

X1 = X_train["RM"]
X2 = X_train["LSTAT"]
Y = y_train
lin_reg = CustomLinearRegression(X1, X2, Y, N=len(Y))

lin_reg.fit()
lin_reg.print_equation()
```



```
y = 2.920993587665519 + 4.46903585639577x1 + -0.6546089737745551x2
```

```
x1_test = X_test["RM"]
x2_test = X_test["LSTAT"]
final_y_predicted = lin_reg.predict(x1_test, x2_test)
lin_reg.show_metrics(y_test, final_y_predicted)
```



```
R square value :  0.6804441130852366
MSE value :  29.28873091553362
```

Comparing Metrics:

	R Squared	MSE Value
Python Libraries	0.6804441130852339	29.288730915533865
User Defined Function	0.6804441130852366	29.28873091553362

Thus, the Linear Regression using sklearn Libraries gives us almost identical results as compared to our own defined function.

Conclusion:

Thus we have learnt about regression and its various types. We also used the scikit Python library for our own regression model, and compared it with a function of our own.