ASSIGNMENT No: 01

Title:-

Implement a Linear Regression Model to predict house prices for regions in the USA using the provided dataset.

Mapping with Syllabus -

Unit 1

Objective -

Develop a model to estimate house prices based on relevant features using Linear Regression.

Outcome -

- Apply Linear Regression in a real-world scenario.
- Understand the implementation of a regression model for house price prediction.

Software Requirements -

- Python (3.x recommended)
- Jupyter Notebook or any Python IDE

Hardware Requirements -

A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

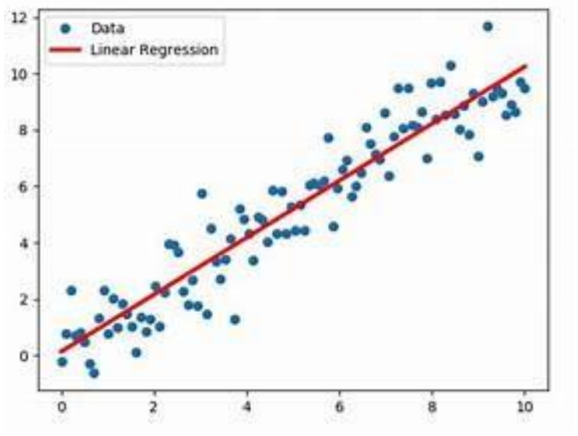
Prerequisites -

- Basic understanding of Python programming
- Familiarity with the concepts of supervised learning

Dataset -

https://github.com/huzaifsayed/Linear-Regression-Model-for-House-Price-Prediction/blob/master/USA_Housing.csv

LINEAR REGRESSION MODEL -



Libraries or Modules Used -

- NumPy
- pandas
- scikit-learn
- Matplotlib
- Seaborn

Theory -

Linear Regression –

WHAT IS LINEAR REGRESSION ?

When we see a relationship in a scatterplot, we can use a line to summarize the relationship in the data. We can also use that line to make predictions in the data. This process is called linear regression.Linear Regression is a supervised learning algorithm used for predicting a continuous outcome, typically represented by the target variable. In the context of this assignment, we aim to predict house prices based on various features such as area income, house age, number of rooms, and others.

Linear regression is used to study the linear relationship between a dependent variable Y (blood pressure) and one or more independent variables X (age, weight, sex).

The dependent variable Y must be continuous, while the independent variables may be either continuous (age), binary (sex), or categorical (social status). The initial judgment of a possible relationship between two continuous variables should always be made on the basis of a scatter plot (scatter graph). This type of plot will show whether the relationship is linear (figure 1) or nonlinear (figure 2).

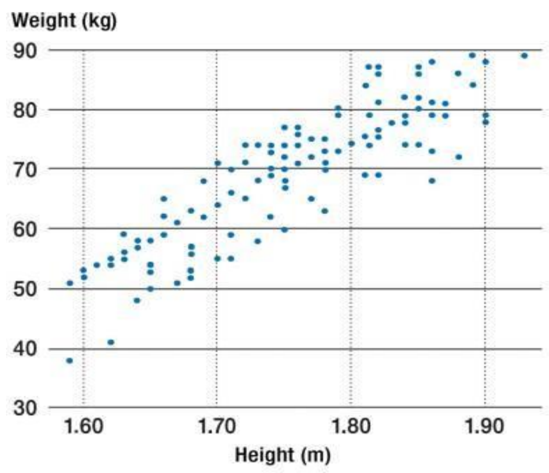


Figure 1

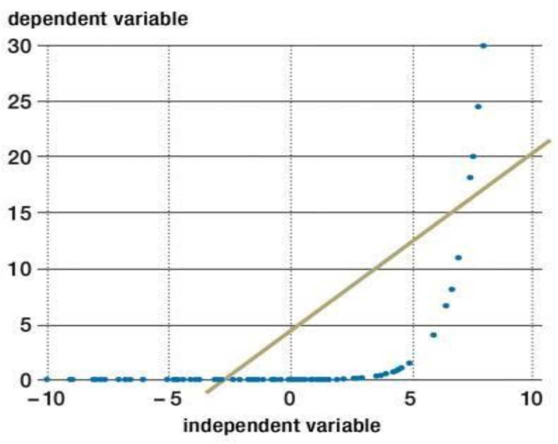


Figure 2

A scatter plot showing an exponential relationship. In this case, it would not be appropriate to compute a coefficient of

Simple linear regression formula-

The formula for a simple linear regression is:

$$y = \beta_0 + \beta_1 X + \epsilon$$

- **y** is the predicted value of the dependent variable (**y**) for any given value of the independent variable (**x**).

- B_0 is the **intercept**, the predicted value of **y** when the **x** is 0.
- B_1 is the regression coefficient – how much we expect **y** to change as **x** increases.
- **x** is the independent variable (the variable we expect is influencing **y**).
- **e** is the **error** of the estimate, or how much variation there is in our estimate of the regression coefficient.

Linear regression finds the line of best fit line through your data by searching for the regression coefficient (B_1) that minimizes the total error (e) of the model.

While you can perform a linear regression by hand, this is a tedious process, so most people use statistical programs to help them quickly analyze the data.

Model Training –

Training a regression model involves teaching the model to predict continuous values based on input features. Here's a brief explanation of the process, along with some images to illustrate key concepts.
Regression Model Training:

1. Data Collection: Gather a dataset with input features (independent variables) and corresponding target values (dependent variable).
2. Data Splitting: Split the dataset into training and testing sets. The training set is used to train the model, and the testing set is used to evaluate its performance.
3. Model Selection: Choose a regression model architecture. Common choices include linear regression, decision trees, or more complex models like neural networks.
4. Feature Scaling: Normalize or standardize the input features to ensure that they are on a similar scale. This helps the model converge faster during training.
5. Model Training: Feed the training data into the chosen model and adjust the model's parameters to minimize the difference between predicted and actual target values.
6. Loss Function: Use a loss function to measure the difference between predicted and actual values. The goal is to minimize this loss during training.
7. Gradient Descent: Use optimization algorithms like gradient descent to iteratively update the model parameters and reduce the loss.
8. Model Evaluation: Evaluate the trained model on the testing set to assess its performance on unseen data.
9. Prediction: Once satisfied with the model's performance, use it to make predictions on new, unseen data.

10 Model Deployment: If the model performs well, deploy it to production for making real-world predictions.

Evaluation Metrics -

The performance of the model will be assessed using various evaluation metrics, such as Mean Squared Error (MSE) and R-squared. These metrics provide insights into how well the model generalizes to unseen data.

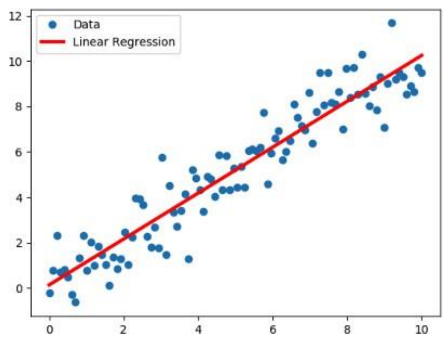


Figure: Linear Regression Model

Algorithm –

Simple Linear Regression Algorithm -

You ll start with the simplest case, which is simple linear regression. There are five basic steps when you re implementing linear regression:

1. Import the packages and classes that you need.
2. Provide data to work with, and eventually do appropriate transformations.
3. Create a regression model and fit it with existing data.
4. Check the results of model fitting to know whether the model is satisfactory.
5. Apply the model for predictions.

These steps are more or less general for most of the regression approaches and implementations. Throughout the rest of the tutorial, you'll learn how to do these steps for several different scenarios.

Step 1: Import packages and classes

The first step is to import the package numpy and the class LinearRegression from sklearn.linear_model:

Python
```python
>>> import numpy as np

>>> from sklearn.linear_model import LinearRegression
```

Now, you have all the functionalities that you need to implement linear regression.

The fundamental data type of NumPy is the array type called numpy.ndarray. The rest of this tutorial uses the term **array** to refer to instances of the type numpy.ndarray.

You'll use the class sklearn.linear_model.LinearRegression to perform linear and polynomial regression and make predictions accordingly.

Step 2: Provide data

The second step is defining data to work with. The inputs (regressors, *x*) and output

(response,*y*) should be arrays or similar objects. This is the simplest way of providing data for regression:

Python

```
>>> x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))

>>> y = np.array([5, 20, 14, 32, 22, 38])
```

Now, you have two arrays: the input, x, and the output, y. You should call .reshape() on x because this array must be **two-dimensional**, or more precisely, it must have **one column** and **as many rows as necessary**

. That's exactly what the argument (-1, 1) of .reshape() specifies.

This is how x and y look now:

Python

```
>>> x

array([[ 5],

[15],

[25],

[35],

[45],


>>> y

array([ 5, 20, 14, 32, 22, 38])
```

As you can see, x has two dimensions, and x.shape is (6, 1), while y has a single dimension, and y.shape is (6,).

Step 3: Create a model and fit it

The next step is to create a linear regression model and fit it using the existing data.

Create an instance of the class LinearRegression, which will represent the regression model:

Python

```
>>> model = LinearRegression()
```

This statement creates the variable model as an instance of LinearRegression. You can provide several optional parameters to LinearRegression:

- fit_intercept is a Boolean that, if True, decides to calculate the intercept b_0 or, if False, considers it equal to zero. It defaults to True.

- normalize is a Boolean that, if True, decides to normalize the input variables. It defaults to False, in which case it doesn't normalize the input variables.
- copy_X is a Boolean that decides whether to copy (True) or overwrite the input variables (False). It's True by default.
- n_jobs is either an integer or None. It represents the number of jobs used in parallel computation. It defaults to None, which usually means one job. -1 means to use all available processors.

Your model as defined above uses the default values of all parameters. It's time to start

using the model. First, you need to call .fit() on model:

Python

```
>>> model.fit(x, y) LinearRegression()
```

With fit(), you calculate the optimal values of the weights b_0 and b_1, using the existing input and output, x and y, as the arguments. In other words, .fit() **fits the model**. It returns self, which is the variable model itself. That's why you can replace the last two statements with this one:

```
Python
>>> model = LinearRegression().fit(x, y)
```

This statement does the same thing as the previous two. It's just shorter.

Step 4: Get results

Once you have your model fitted, you can get the results to check whether the model works satisfactorily and to interpret it.

You can obtain the coefficient of determination, R^2, with .score() called on model:

```
Python
>>> r_sq = model.score(x, y)

>>> print(f"coefficient of determination: {r_sq}")
```

When you're applying .score(), the arguments are also the predictor x and response y, and the

return value is R^2.

The attributes of model are .intercept_, which represents the coefficient b_0, and .coef_, which

represents b_1:

```
Python

>>> print(f"intercept: {model.intercept_}")

intercept: 5.633333333333329



>>> print(f"slope: {model.coef_}")

slope: [0.54]
```

The code above illustrates how to get b
while .coef_ is an array.

The value of b_0 is approximately 5.63. This illustrates that your model predicts the response

5.63 when is zero. The value b_1 = 0.54 means that the predicted response rises by 0.54 when *x* is increased by one.

You'll notice that you can provide y as a two-dimensional array as well. In this case, you'll get a similar result. This is how it might look:

```
Python

>>> new_model = LinearRegression().fit(x, y.reshape((-1, 1)))

>>> print(f"intercept: {new_model.intercept_}") intercept:
[5.63333333]



>>> print(f"slope: {new_model.coef_}")

slope: [[0.54]]
```

As you can see, this example is very similar to the previous one, but in this case, .intercept_ is a one-dimensional array with the single element b_0, and .coef_ is a two-dimensional array with the single element b_1.

Step 5: Predict response

Once you have a satisfactory model, then you can use it for predictions with either existing or new data. To obtain the predicted response, use .predict():

```
Python

>>> y_pred = model.predict(x)

>>> print(f"predicted response:\n{y_pred}")
```

```
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```

When applying .predict(), you pass the regressor as the argument and get the corresponding predicted response. This is a nearly identical way to predict the response:

Python

```
>>> y_pred = model.intercept_ + model.coef_ * x

>>> print(f"predicted response:\n{y_pred}") predicted
response:

[[ 8.33333333]

[13.73333333]

[19.13333333]
```

In this case, you multiply each element of x with model.coef_ and add model.intercept_ to the product.

The output here differs from the previous example only in dimensions. The predicted response is now a two-dimensional array, while in the previous case, it had one dimension.

If you reduce the number of dimensions of x to one, then these two approaches will yield the same result. You can do this by replacing x with x.reshape(-1), x.flatten(), or x.ravel() when multiplying it with model.coef_.

In practice, regression models are often applied for forecasts. This means that you can use fitted models to calculate the outputs based on new inputs:

Python

```
   >>> x_new = np.arange(5).reshape((-1, 1))

 >>> x_new
  array([[0],
       [1],

[2],

>>> y_new = model.predict(x_new)
>>> y_new
array([5.63333333, 6.17333333, 6.71333333, 7.25333333, 7.79333333])
```

Here .predict() is applied to the new regressor x_new and yields the response y_new. This example conveniently uses arange() from numpy to generate an array with the elements from 0, inclusive, up to but excluding 5 that is, 0, 1, 2, 3, and 4.

Application -

Real estate pricing strategies
1. Assisting clients in making informed decisions
2. Market analysis for regions in the USA
3. Predictive tool for estimating house prices

Inference -

The process involves exploring the data, data collection, model fitting, and interpretation of regression results. Practical skills in using statistical software for regression analysis are likely to be developed, enhancing students ability to analyze relationships between variables and make predictions based on linear model.

Code -

```
import pandas as pd import
numpy as np
from sklearn.model_selection import train_test_split from
sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score import
matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('USA_Housing.csv') #
Explore the dataset print(df.head())
print(df.info())
# Handle missing values if any df =
df.dropna()

# Select relevant features and target variable
X = df[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of
Bedrooms', 'Area Population']]
y = df['Price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Linear Regression model model =
LinearRegression()
```