Multiple Linear Regression

This lesson will introduce multiple linear regression and focus on how to perform it in Python.

WE'LL COVER THE FOLLOWING

In the last lesson, we performed simple linear regression. But it was a limited model because we only used one variable from our dataset in our predictions. Our linear model was based on the fact that there was some relationship between two variables. But the variable that we are trying to predict can have relationships with other variables too. Including other variables in our model may help us make better predictions.

Multiple Linear Regression

We can extend the simple linear regression model to a multiple linear regression model by adding more variables and parameters to the model equation. Then the prediction ($\hat{y}$) becomes:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + ... + \theta_m x_m$$

where $x_1, x_2, x_3, ..., x_n$ are different attributes in our data. The above linear equation takes multiples attributes, multiplies them with some weights to get a prediction. Weights can be obtained by minimizing a loss function using

gradient descent as we did for simple linear regression.

We will be using the USA Housing Dataset. Our goal is to predict the **price** of a house. Other variables in the dataset are listed below.

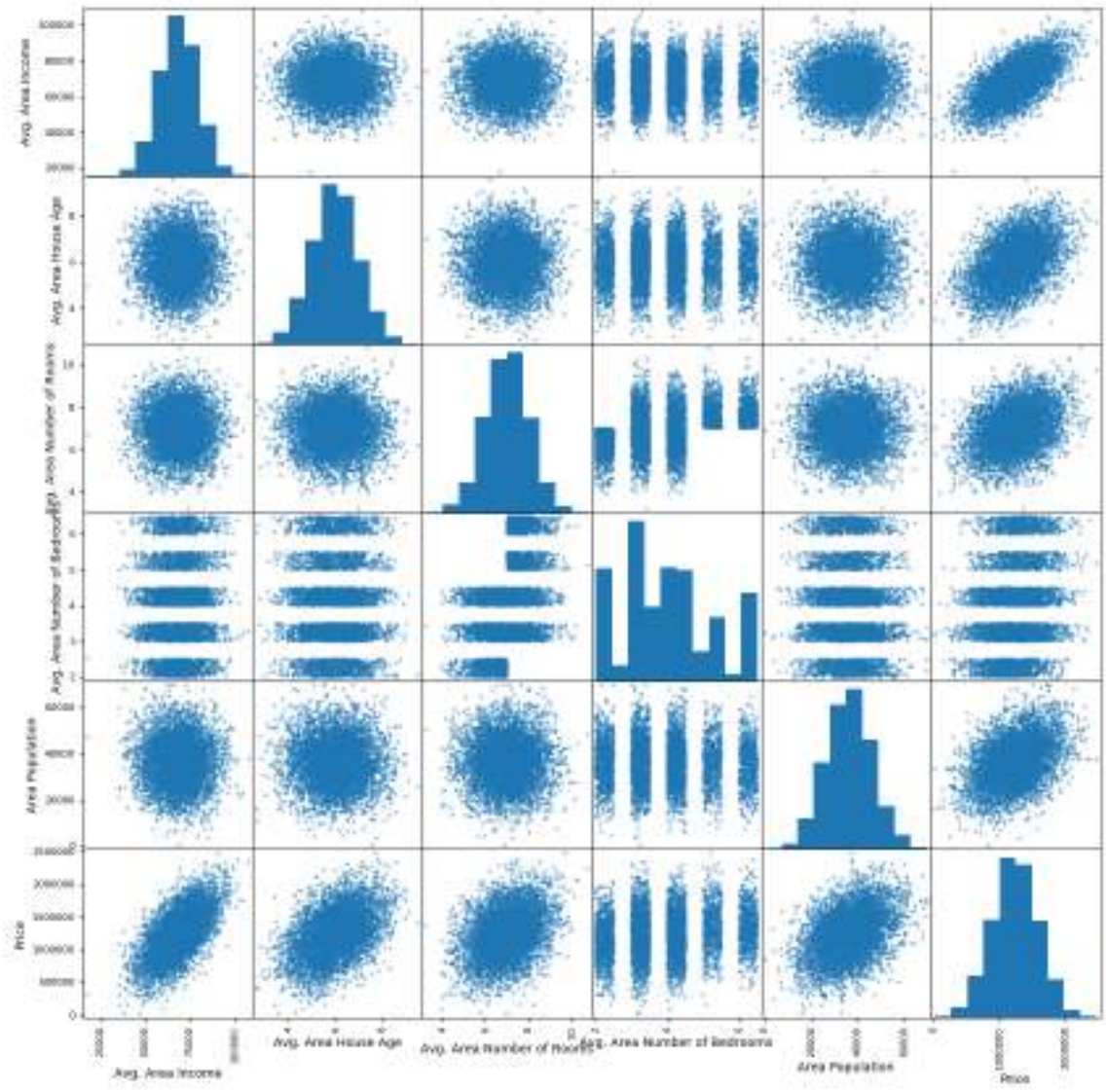
USA_Housing.csv

```
# USA HOUSING
# Avg. Area Income
# Avg. Area House Age
# Avg. Area Number of Rooms
# Avg. Area Number of Bedrooms
# Area Population
# Price
# Address
```

Now we need to decide what variables we need to include in our model. For that we can look at the relationships of all variables with `Price`.

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('USA_Housing.csv')
pd.scatter_matrix(df,figsize = (15,15))
print(df.head())
```

We have plotted scatter plots of all variables with each other using the function `scatter_matrix`. Note that this function was not called on the dataframe `df`, rather it was used with `pd` since it is a function available in *Pandas*. We gave it our dataframe `df` and the figure size.



We only need to focus on the last row of these scatter plots as `Price` is on the y-axis here. We can see that all five variables show some kind of pattern with `Price` which means we should include these variables in our model. So, the model equation will become:

$$\hat{y} = \theta_0 + \theta_1 * AvgAreaIncome + \theta_2 * AvgAreaHouseAge + \theta_3 * AvgAreaNumberRooms$$

$$+ \theta_4 * AvgAreaNumberBedrooms + \theta_5 * AreaPopulation$$

Here θ_0 is called the **intercept** and it is not multiplied with any variable. Now that we have our model decided we need to minimize the Mean Squared Error function.

Minimizing the Loss function

If we denote our predictions by $\hat{y}$ and the actual values by y, then loss function will be calculated as

$$L(\theta, X, Y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$L(\theta, X, Y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + ... + \theta_m x_m))^2$$

We will be minimizing this loss function with gradient descent. Recall that in gradient descent we:

- Start with random initial values of θ.
- Compute $\theta_t - \alpha \frac{\partial}{\partial \theta} L(\theta, X, Y)$ to update the value of all θ.
- Keep updating the value of all θ until they stop changing values. This can be the point where we have reached the minimum of the error function.

We need to do step 2 for all the parameters as we did for 2 parameters in simple linear regression in the previous lesson.

Model fitting in Python

In the previous lesson, we coded the entire gradient descent algorithm for simple linear regression ourselves. But extending that to multiple linear models can be messy. We have libraries in Python that will do that for us. We do not need to get into the messy details of mathematics. However, we do need to know what is going on behind the scenes when we call a library function, that is why we coded gradient descent ourselves in the previous lessons.

Python has a library called **scikit-learn** that has all kinds of functions that help make different kinds of predictive models and optimize them. It also has a lot of loss functions that we can use directly. Scikit-learn is written in code as `sklearn`. We will be importing two things from it.

`sklearn.metrics.mean_squared_error`

The metrics module, which can be imported as `sklearn.metrics`, has different

loss functions that we can use. We will import the mean squared error function from this.

`mean_squared_error`

It gives us the mean squared error loss. It expects the following arguments:

- `y_true`: Actual values of target variable
- `y_pred`: Predicted values of target variable

`sklearn.linear_model.LinearRegression`

The `linear_model` module has functions and classes to use different kinds of predictive models. A model like a linear regression model is implemented as a class. It has different attributes and functions that we can use. The two main functions are:

`fit`

It makes a model using the data provided and fits it to find the best model parameters. It expects two arguments:

- `X`: The data that will be used to predict
- `Y`: The actual values of the target variable

`predict`

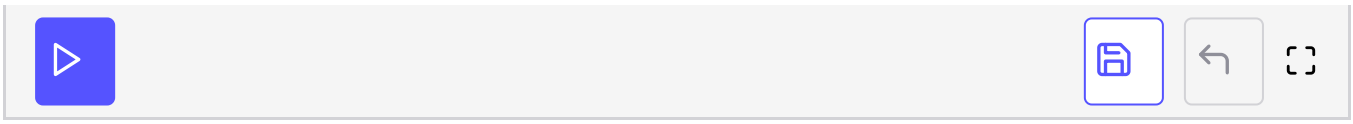
It gives us the predicted values using the equation $\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + ... + \theta_m x_m$. It expects the following arguments

- `X`: The data that will be used to predict. For a dataframe, it gives us a predicted value for each row.

Now that we know what functionalities we need from the library, let's use them below to predict house prices.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error


df = pd.read_csv('USA_Housing.csv')

# Split dataframe into Xs and Y
X = df.drop(columns = ['Address','Price'])
Y = df[['Price']]
```

```
# Linear Regression model fitting
lr = LinearRegression()
lr.fit(X,Y)

# Loss and predictions
predictions = lr.predict(X)
loss = mean_squared_error(y_true = Y,y_pred = predictions)
print('loss = :',loss)

# Print parameters
print('model parameters : ' ,lr.coef_)
print('intercept :',lr.intercept_)
```



In **lines 3 and 4**, we have imported the `LinearRegression` class and `mean_squared_error` function as discussed above. We read the data into a dataframe in **line** 7. Since we decided above that we will not be using any non-numeric variables for prediction, we drop `Price` and `Address` and form a new dataframe `X` in **line 10**. It has all the variables that we can use in prediction. In **line 11**, we separate the actual values of `Price` in a dataframe called `Y`.

In **line 14**, we initialize the `LinearRegression` class. We call the class object `lr`. We then use the `fit` function to fit our model in the next line. The `fit` function will find the best model for us and store the model parameters internally.

Now we get predictions using our fitted model in **line 18** using the `predict` function. We save them in `predictions`. Then, we find the mean squared error in the predictions using the `mean_squared_error` function. We print the loss in the next line. In the end, we print our model parameters which are stored as `coef_`. The intercept (θ_0) is stored as `intercept_`.

So, this is how easily we can make linear regression models in Python. In the next lesson, we will focus on the performance of these models.