Lab #1 : Simple Linear Regression

<u>Objectives</u> : learn how to build a machine learning model in python using a simple linear regression algorithm

<u>Exercise #1</u>

Type this python code and tell what does it do.

```
import numpy as np
from sklearn.linear_model import LinearRegression
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])

# y = 1 * x_0 + 2 * x_1 + 3

y = np.dot(X, np.array([1, 2])) + 3
reg = LinearRegression().fit(X, y)
reg.score(X, y)
# it should return 1.0

reg.coef_
# it should return array([1., 2.])

reg.intercept_
# it should return 3.0...

reg.predict(np.array([[3, 5]]))
# it should return array([16.])
```

<u>Exercise #2</u>

Consider the Advertising sales channel prediction data**.**

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12.0
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9
8.7	48.9	75.0	7.2
57.5	32.8	23.5	11.8

‘Sales’ is the target variable that needs to be predicted. Now, based on this data, our objective is to create a predictive model, that predicts sales based on the money spent on different platforms for marketing.

Step 1: Importing Python Libraries

Here are the important libraries that we will be needing for this linear regression.

- **NumPy** (to perform certain mathematical operations)
- **pandas** (to store the data in a pandas DataFrames)
- **matplotlib.pyplot** (you will use matplotlib to plot the data)

In order to load these, just start with these few lines of codes in your first cell:

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

Step 2: Loading the Dataset

Let us now import data into a DataFrame. A DataFrame is a data type in Python. The simplest way to understand it would be that it stores all your data in tabular format.

```
advertising = pd.read_csv( "advertising.xls" )
advertising.head()
```

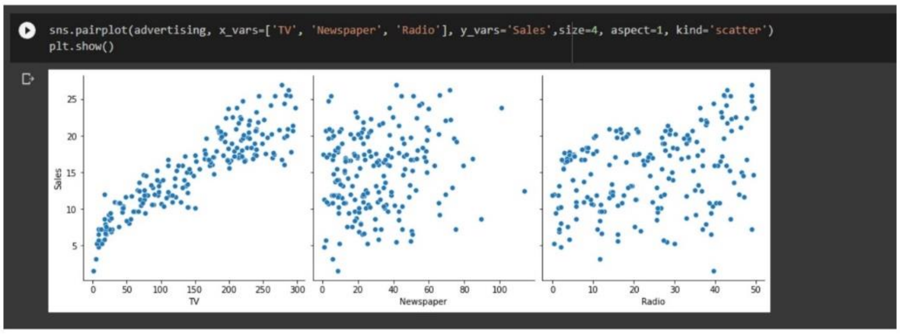
```
# Read the given CSV file, and view some sample records

advertising = pd.read_csv("advertising.csv")
advertising.head()
```

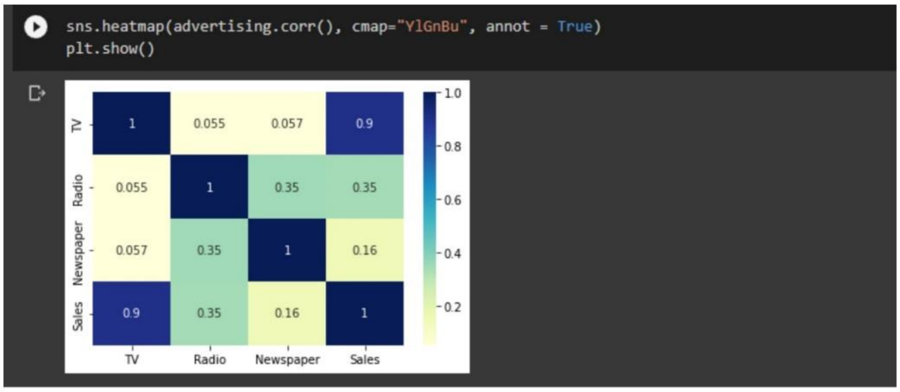
	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

Step 3: Visualization

Let us plot the scatter plot for target variable vs. predictor variables in a single plot to get the intuition. Also, plotting a heatmap for all the variables,

```
#Importing seaborn library for visualizations
import seaborn as sns
#to plot all the scatterplots in a single plot
sns.pairplot(advertising, x_vars=[ 'TV', 'Newspaper','Radio' ], y_vars =
'Sales', size = 4, kind = 'scatter' )
plt.show()
#To plot heatmap to find out correlations
sns.heatmap( advertising.corr(), annot = True )
plt.show()
```





From the scatterplot and the heatmap, we can observe that ‘Sales’ and ‘TV’ have a higher correlation as compared to others because it shows a linear pattern in the scatterplot as well as giving 0.9 correlation.

Step 4: Performing Simple Linear Regression

Here, as the TV and Sales have a higher correlation we will perform the simple linear regression for these variables.

We first assign the feature variable, `TV`, during this case, to the variable `X` and the response variable, `Sales`, to the variable `y`.

```
X = advertising[ 'TV' ]
y = advertising[ 'Sales' ]
```

And after assigning the variables you need to split our variable into training and testing sets. You’ll perform this by importing `train_test_split` from the `sklearn.model_selection` library. It is usually a good practice to keep 70% of the data in your train dataset and the rest 30% in your test dataset.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, train_size =
0.7, test_size = 0.3, random_state = 100 )
```

In this way, you can split the data into train and test sets.

One can check the shapes of train and test sets with the following code,

```
print( X_train.shape )
print( X_test.shape )
```

```
print( y_train.shape )
print( y_test.shape )
```

importing LinearRegression library from sklearn.linear_model to perform linear regression

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

There's one small step that we need to add, though. When there's only a single feature, we need to add an additional column in order for the linear regression fit to be performed successfully. Code is given below,

```
X_train_lm = X_train.values.reshape(-1,1)
X_test_lm = X_test.values.reshape(-1,1)
```

One can check the change in the shape of the above data frames.

```
print(X_train_lm.shape)
print(X_test_lm.shape)
```

Launch the training

```
model = LinearRegression().fit(X_train_lm, y_train)
```

Print the training score

```
coeff_train = model.score(X_train_lm, y_train)
print(f"Coefficient de détermination R² en train : {coeff_train:.2f}")
```

Print the test score

```
coeff_test = model.score(X_test_lm, y_test)
print(f"Coefficient de détermination R² en test : {coeff_test:.2f}")
```

You can get the intercept and slope values with sklearn using the following code,

```
#get intercept
print(model.intercept_ )
#get slope
print(model.coef_)
```

Visualising the Training set results

```
plt.scatter(X_train_lm,y_train,color='red')
plt.plot(X_train_lm,model.predict(X_train_lm),color='blue')
plt.title("Simple Linear Regression on Training Data")
plt.xlabel("TV")
plt.ylabel("Sales")
plt.show()
```

Apart from `sklearn`, there is another package namely `statsmodels` that can be used to perform linear regression. We will use the `statsmodels` library to build the model. Since we have already performed a train-test split, we don't need to do it again.

importing statmodels library to perform linear regression

```
import statsmodels.api as sm
```

By default, the `statsmodels` library fits a line on the dataset which passes through the origin. But in order to have an intercept, you need to manually use the `add_constant` attribute of `statsmodels`. And once you've added the constant to your `X_train` dataset, you can go ahead and fit a regression line using the `OLS` (Ordinary Least Squares) the attribute of `statsmodels` as shown below,

```
# Add a constant to get an intercept
X_train_sm = sm.add_constant(X_train)
# Fit the resgression line using 'OLS'
lr = sm.OLS(y_train, X_train_sm).fit()
```

One can see the values of betas using the following code,

```
# Print the parameters,i.e. intercept and slope of the regression line
obtained
lr.params
```

```
# Print the parameters, i.e. the intercept and the slope of the regression line fitted
lr.params

const    6.948683
TV       0.054546
dtype: float64
```

Here, 6.948 is the intercept, and 0.0545 is a slope for the variable TV.

Now, let's see the evaluation metrics for this linear regression operation. You can simply view the summary using the following code,

```
#Performing a summary operation lists out all different parameters of the
regression line fitted
print(lr.summary())
```

```
# Performing a summary operation lists out all the different parameters of the regression line fitted
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.816
Model:                            OLS   Adj. R-squared:                  0.814
Method:                 Least Squares   F-statistic:                     611.2
Date:                Sat, 25 Sep 2021   Prob (F-statistic):           1.52e-52
Time:                        08:42:05   Log-Likelihood:                -321.12
No. Observations:                 140   AIC:                             646.2
Df Residuals:                     138   BIC:                             652.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.9487      0.385     18.068      0.000       6.188       7.709
TV             0.0545      0.002     24.722      0.000       0.050       0.059
==============================================================================
Omnibus:                        0.027   Durbin-Watson:                   2.196
Prob(Omnibus):                  0.987   Jarque-Bera (JB):                0.150
Skew:                          -0.006   Prob(JB):                        0.928
Kurtosis:                       2.840   Cond. No.                         328.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

As you can see, this code gives you a brief summary of the linear regression. Here are some key statistics from the summary:

1. The **coefficient** for TV is 0.054, with a very low p-value. The coefficient is statistically significant. So the association is not purely by chance.
2. **R – squared** is 0.816 Meaning that 81.6% of the variance in `Sales` is explained by `TV`. This is a decent R-squared value.
3. **F-statistics** has a very low p-value(practically low). Meaning that the model fit is statistically significant, and the explained variance isn't purely by chance.

Step 5: Performing predictions on the test set

Now that you have simply fitted a regression line on your train dataset, it is time to make some predictions on the test data. For this, you first need to add a constant to the `X_test` data like you did for `X_train` and then you can simply go on and predict the y values corresponding to `X_test` using the `predict` the attribute of the fitted regression line.

```
# Add a constant to X_test
X_test_sm = sm.add_constant(X_test)
# Predict the y values corresponding to X_test_sm
y_pred = lr.predict(X_test_sm)
```

You can see the predicted values with the following code,

```
y_pred.head()
```

```
y_pred.head()
```

```
126     7.374140
104    19.941482
99     14.323269
92     18.823294
111    20.132392
dtype: float64
```

To check how well the values are predicted on the test data we will check some evaluation metrics using sklearn library.

```
#Imporitng libraries
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
#RMSE value
print( "RMSE: ",np.sqrt( mean_squared_error( y_test, y_pred ) ) )
#R-squared value
print( "R-squared: ",r2_score( y_test, y_pred ) )
```

Looking at the RMSE

```
[ ] #Returns the mean squared error; we'll take a square root
    np.sqrt(mean_squared_error(y_test, y_pred))
```

```
2.019296008966232
```

Checking the R-squared on the test set

```
r_squared = r2_score(y_test, y_pred)
r_squared
```

```
0.792103160124566
```

We are getting a decent score for both train and test sets.