Data Analysis Python using Regression

Dataset: A company's data where there is the amount spent on different types of advertisements and its subsequent sales.

Setup: Google Colab Notebooks.

Goal: To build a linear regression model in python, the steps are as follows:

1. Reading and understanding the data.
2. Visualizing the data.
3. Performing simple linear regression.
4. Residual analysis.
5. Predictions on the test set.

1. Reading and understanding the data:

a. Reading the data:

i. Import libraries: numpy and pandas. Before that suppress the warnings.

```
[1] import warnings
    warnings.filterwarnings('ignore')
```

```
[3] import numpy as np
    import pandas as pd
```

ii. Read the given CSV file using pandas.

```
[4] advertising = pd.read_csv("Company_data.csv")
    advertising
```

	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
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	14.0
197	177.0	9.3	6.4	14.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	18.4

200 rows × 4 columns

b. Understanding the data in the dataset:

i. Shape: The shape of the dataset.

```
[5] advertising.shape

(200, 4)
```

ii. Info: Using this, we can see whether there are any null values in the data. If yes, then do some data manipulation. But in our case, no null values are present in the data.

```
[6] advertising.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

iii. Describe: the values present in the columns are consistent throughout the data.

```
advertising.describe()
```

	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	15.130500
std	85.854236	14.846809	21.778621	5.283892
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	11.000000
50%	149.750000	22.900000	25.750000	16.000000
75%	218.825000	36.525000	45.100000	19.050000
max	296.400000	49.600000	114.000000	27.000000

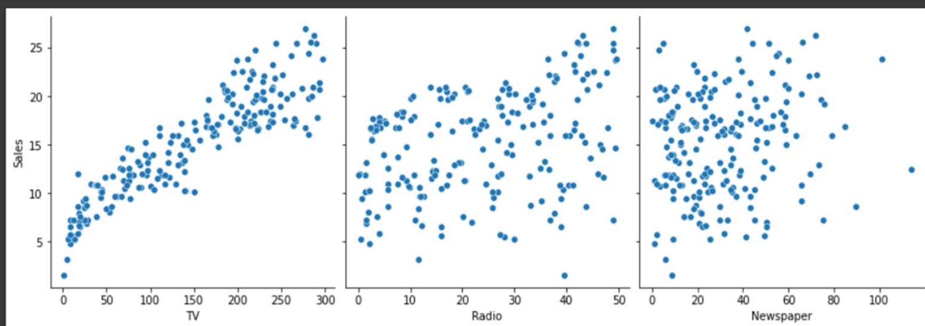
2. Visualizing the data:

a. Import matplotlib and seaborn libraries to make a pairplot of all the columns and see which columns are the most correlated to Sales.

```
[32] import matplotlib.pyplot as plt
     import seaborn as sns
```

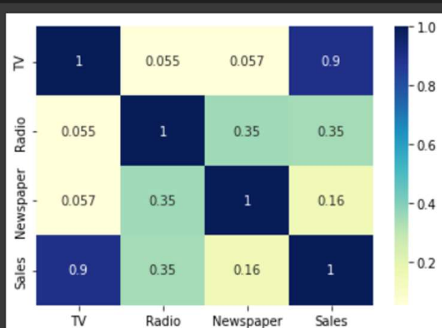
b. Using pairplot, visualize the data for correlation. Pairplot of each column w.r.t Sales column.

```
[33] sns.pairplot(advertising, x_vars=['TV', 'Radio', 'Newspaper'], y_vars='Sales', size=4, aspect=1, kind='scatter')
     plt.show()
```



c. If correlation cannot be determined by using pairplot then use the seaborn heatmap.

```
[34] sns.heatmap(advertising.corr(), cmap="YlGnBu", annot = True)
     plt.show()
```



From the above graphs, the TV column seems most correlated to Sales.

3. Performing Simple Linear Regression

Perform the simple linear regression using TV as a feature variable. The equation for simple linear regression: y = c+mx

For the case: y = c+m * TV
where m →model coefficients/model parameters.

Steps:

- Create X and Y
- Create Train and Test set
- Train your model
- Evaluate the model

a. Create X and Y:

X→ feature variable/ independent variable (TV)
y→ target variable (Sales)

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

b. Create Train and Test sets:

Split the variables into training and testing sets. Using the training set, build the model and perform the model test using the testing set.

- Split the data by importing train_test_split from the sklearn.model_selection library.

```
[36] 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)
```

- X _train data after splitting.

```
[37] X_train

74     213.4
3      151.5
185    205.0
26     142.9
90     134.3
       ...
87     110.7
103    187.9
67     139.3
24      62.3
8        8.6
Name: TV, Length: 140, dtype: float64
```

- Y_train data after splitting

```
[38] y_train

74     17.0
3      16.5
185    22.6
26     15.0
90     14.0
       ...
87     16.0
103    19.7
67     13.4
24      9.7
8       4.8
Name: Sales, Length: 140, dtype: float64
```

c. Building and training the model

A simple linear regression model can be built using 2 packages:

- statsmodel
- sklearn

a. Build the model using the statsmodel package.

i. Import statsmodel.api library from statsmodel package.

```
[39] import statsmodels.api as sm
```

ii. Add a constant to get an intercept.

```
[40] X_train_sm = sm.add_constant(X_train)
```

iii. Fit the regression line using OLS (Ordinary Least Square).

```
[41] lr = sm.OLS(y_train, X_train_sm).fit()
```

iv. Print the parameters i.e. c and m of the straight line.

```
[42] lr.params

const    6.948683
TV       0.054546
dtype: float64
```

v. Perform a summary to list out all the different parameters of the fitted regression line. The statistics for the regression line is shown below:

```
[43] 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:     Fri, 20 Jan 2023 Prob (F-statistic):   1.52e-52
           Time:     08:33:16          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.

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

vi. To determine whether the model is viable or not is decided by the below statistics:

- The coefficients and its p-value.
- R-squared value.
- F-statistic and its significance.

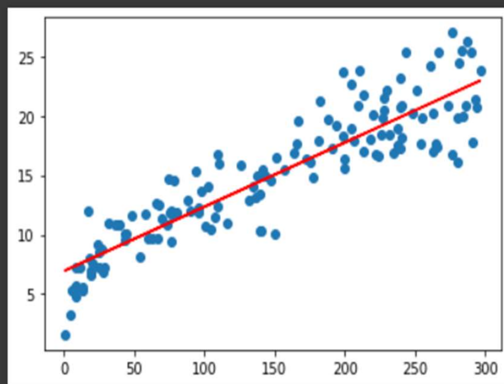

```
                    OLS Regression Results
   Dep. Variable:    Sales               R-squared:          0.816
          Model:     OLS              Adj. R-squared:        0.814
         Method:     Least Squares       F-statistic:        611.2
           Date:     Fri, 20 Jan 2023 Prob (F-statistic):   1.52e-52
           Time:     08:33:16          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.
```


- Coefficient for TV→0.05, p-value→very low, almost 0 i.e. coefficient is statistically significant.
- R-squared→0.816, 81.6% of the Sales variance can be explained by the TV column using this line.
- Prob F-statistic→very low p-value, practically 0, the model fit is statistically significant.

vii. Visualize the regression line to see how well the straight line fits the scatter plot between the Tv and Sales columns. From the parameters, the equation of the line is: Sales = 6.948+0.054*TV.

Best-fit regression line is shown below:

```
[44] plt.scatter(X_train, y_train)
     plt.plot(X_train, 6.948 + 0.054*X_train, 'r')
     plt.show()
```



4. Residual Analysis

After building a simple linear regression model using training data, evaluation is necessary. Before evaluation, we have to perform residual analysis.

Assumption of linear regression model: error terms are normally distributed.
Error = Actual y value – y predicted value

a. From the dataset, predict y_value using training dataset of X using predict attribute.

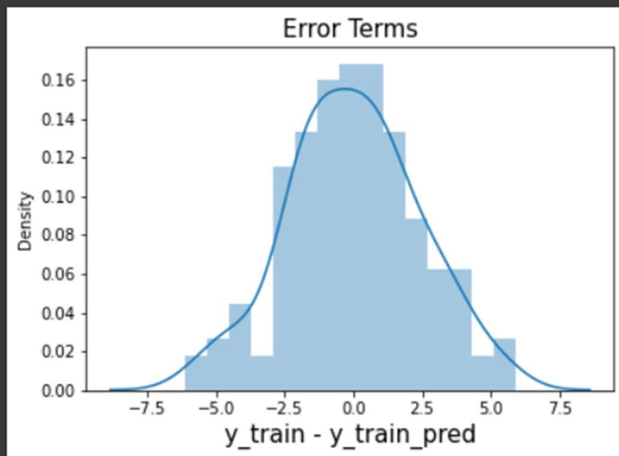
```
[45] y_train_pred = lr.predict(X_train_sm)
```

b. Create residuals from the y_train data and predicted y_data

```
[46] res = (y_train - y_train_pred)
```

c. Plot the histogram and see whether it looks like normal distribution or not. The histogram of residuals looks like as shown below. It follows the normal distribution with a mean 0.

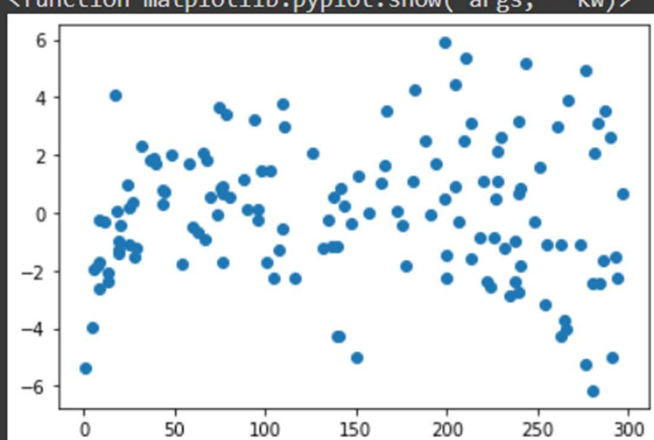
```python
[48] fig = plt.figure()
     sns.distplot(res, bins = 15)
     plt.title('Error Terms', fontsize = 15)
     plt.xlabel('y_train - y_train_pred', fontsize = 15)
     plt.show()
```



d. Look for patterns in the residuals to make sure they are not following any specific pattern. The scatter plot do not follow any specific pattern, now use the built linear regression model to evaluate test data.

```python
[49] plt.scatter(X_train,res)
     plt.show

<function matplotlib.pyplot.show(*args, **kw)>
```



5. Predictions on the Test data or evaluating the model

Make some predictions to the test data

a. Similar to the training dataset, add a constant to X_test

```python
[51] X_test_sm = sm.add_constant(X_test)
```

b. Predict the y values corresponding to X_test_sm using the predict attribute

```
[52] y_test_pred = lr.predict(X_test_sm)
```

c. Print the first 15 predicted values.

```
[57] y_test_pred.head(15)

     126     7.374140
     104    19.941482
     99     14.323269
     92     18.823294
     111    20.132392
     167    18.228745
     116    14.541452
     96     17.726924
     52     18.752384
     69     18.774202
     164    13.341445
     124    19.466933
     182    10.014155
     154    17.192376
     125    11.705073
     dtype: float64
```

d. To calculate R^2 value, import r2_score library from sklearn.metrics package.

```
[58] from sklearn.metrics import r2_score
```

e. Check the R-squared value.

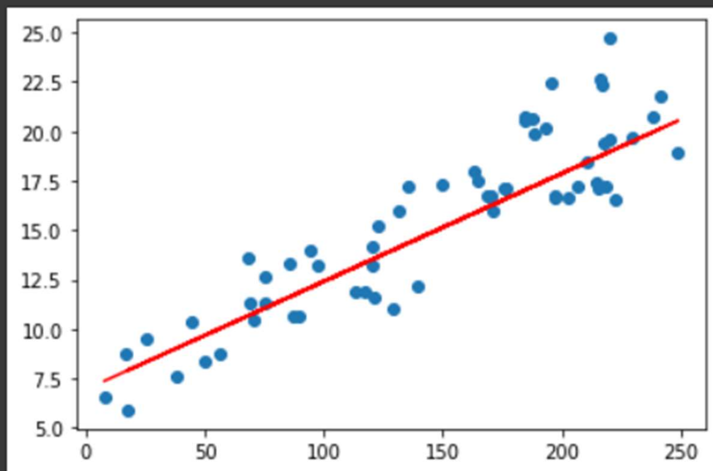
```
[59] r_squared = r2_score(y_test, y_test_pred)
     r_squared

     0.792103160124566
```

f. We have, R^2 value of test data = 0.792 and R^2 value of training data = 0.816 i.e. R^2 value of test data is within 5% of R^2 value of training data. Conclusion: The model is pretty stable i.e. what the model has learned on the training set can generalize on the unseen test data.

g. Visualize the line on the test data. The scatter-plot with the best for line looks like as shown below

```
[60] plt.scatter(X_test, y_test)
     plt.plot(X_test, y_test_pred, 'r')
     plt.show()
```



****Build a linear regression model using sklearn.**

a. Use the linear_model library from sklearn to make the model. Similar to statsmodel, split the data into train and test.

```
[64] from sklearn.model_selection import train_test_split
     X_train_lm, X_test_lm, y_train_lm, y_test_lm = train_test_split(X, y, train_size = 0.7, test_size = 0.3, random_state = 100)
```

b. For simple linear regression, add a column to perform the regression fit properly

i. The shape of the column before adding the column.

```
[65] X_train_lm.shape

     (140,)
```

ii. Add an additional column to train and test data.

```
[66] X_train_lm = X_train_lm.values.reshape(-1,1)
     X_test_lm = X_test_lm.values.reshape(-1,1)
```

iii. The shape of X for train and test data

```
[67] print(X_train_lm.shape)
     print(X_test_lm.shape)

(140, 1)
(60, 1)
```

c. Fit the line to the plot importing the LinearRegression library from the sklearn.linear_model.

```
[68] from sklearn.linear_model import LinearRegression
```

d. Create an object for LinearRegression

```
[69] lm = LinearRegression()
```

e. Fit the model using .fit() method

```
[71] lm.fit(X_train_lm, y_train_lm)

LinearRegression()
```

f. Find the coefficients of the model

i. Intercept value

```
[72] print("Intercept : ", lm.intercept_)

Intercept :  6.948683200001357
```

ii. Slope value

```
[73] print('Slope : ', lm.coef_)

Slope :  [0.05454575]
```

iii. Equation: Sales = 6.948 + 0.054*TV.

g. Make predictions with y_value

```
[74] y_train_pred = lm.predict(X_train_lm)
     y_test_pred = lm.predict(X_test_lm)
```

h. Compare the r2 value of both train and test data.

```
print(r2_score(y_train,y_train_pred))
print(r2_score(y_test,y_test_pred))

0.8157933136480389
0.7921031601245662
```

i. We have, R^2 value of test data = 0.792 and R^2 value of training data = 0.816 i.e. R^2 value of test data is within 5% of R^2 value of training data. Apply the model to the unseen test data in future. It is the same as the statsmodel.