Sales Prediction

(Simple Linear Regression)

Reading and Understanding the Data

```
# Supress Warnings

import warnings
warnings.filterwarnings('ignore')

# Import the numpy and pandas package

import numpy as np
import pandas as pd

# Data Visualisation
import matplotlib.pyplot as plt
import seaborn as sns


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

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

Data Inspection

```
advertising.shape
```

```
(200, 5)
```

```
advertising.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  200 non-null    int64  
 1   TV          200 non-null    float64
 2   Radio       200 non-null    float64
 3   Newspaper   200 non-null    float64
 4   Sales       200 non-null    float64
dtypes: float64(4), int64(1)
memory usage: 7.9 KB
```

```
advertising.describe()
```

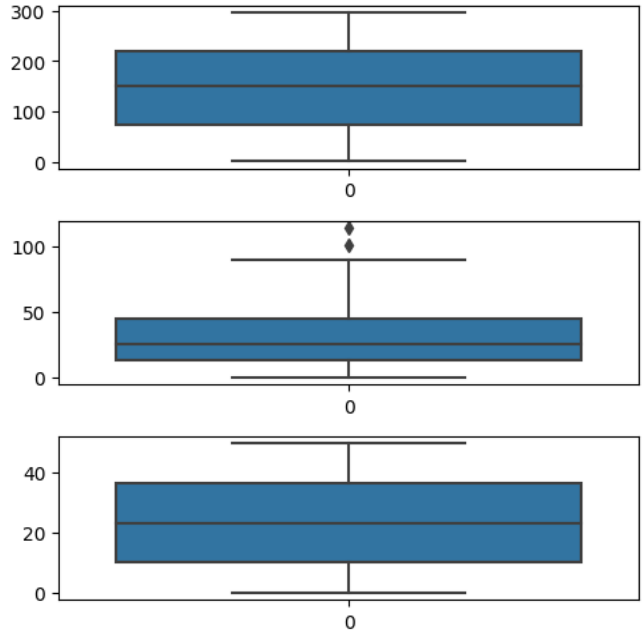
	Unnamed: 0	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000	200.000000
mean	100.500000	147.042500	23.264000	30.554000	14.022500
std	57.879185	85.854236	14.846809	21.778621	5.217457
min	1.000000	0.700000	0.000000	0.300000	1.600000
25%	50.750000	74.375000	9.975000	12.750000	10.375000
50%	100.500000	149.750000	22.900000	25.750000	12.900000
75%	150.250000	218.825000	36.525000	45.100000	17.400000
max	200.000000	296.400000	49.600000	114.000000	27.000000

Data Cleaning

```
# Checking Null values
advertising.isnull().sum()*100/advertising.shape[0]
# There are no NULL values in the dataset, hence it is clean.
```

```
Unnamed: 0    0.0
TV            0.0
Radio         0.0
Newspaper     0.0
Sales         0.0
dtype: float64
```

```
# Outlier Analysis
fig, axs = plt.subplots(3, figsize = (5,5))
plt1 = sns.boxplot(advertising['TV'], ax = axs[0])
plt2 = sns.boxplot(advertising['Newspaper'], ax = axs[1])
plt3 = sns.boxplot(advertising['Radio'], ax = axs[2])
plt.tight_layout()
```



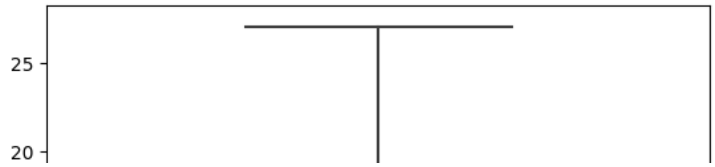

```
# There are no considerable outliers present in the data.
```

Exploratory Data Analysis

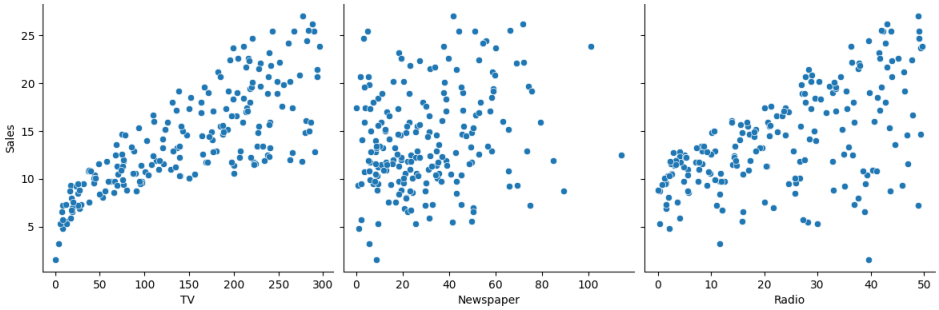
Univariate Analysis

Sales (Target Variable)

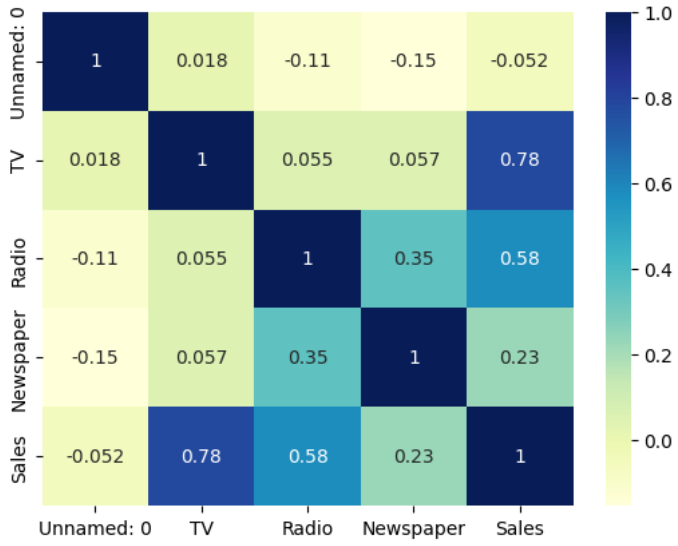
```
sns.boxplot(advertising['Sales'])
plt.show()
```




```
# Let's see how Sales are related with other variables using scatter plot.
sns.pairplot(advertising, x_vars=['TV', 'Newspaper', 'Radio'], y_vars='Sales', height=4, aspect=1, kind='scatter')
plt.show()
```




```
# Let's see the correlation between different variables.
sns.heatmap(advertising.corr(), cmap="YlGnBu", annot = True)
plt.show()
```



As is visible from the pairplot and the heatmap, the variable `TV` seems to be most correlated with `Sales`. So let's go ahead and perform simple linear regression using `TV` as our feature variable.

Model Building

Performing Simple Linear Regression

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

Train-Test Split

You now 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)


# Let's now take a look at the train dataset

X_train.head()
```

```
74     213.4
3      151.5
185    205.0
26     142.9
90     134.3
Name: TV, dtype: float64
```

```
y_train.head()
```

```
74     17.0
3      18.5
185    22.6
26     15.0
90     11.2
Name: Sales, dtype: float64
```

Building a Linear Model

You first need to import the `statsmodel.api` library using which you'll perform the 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) 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()


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

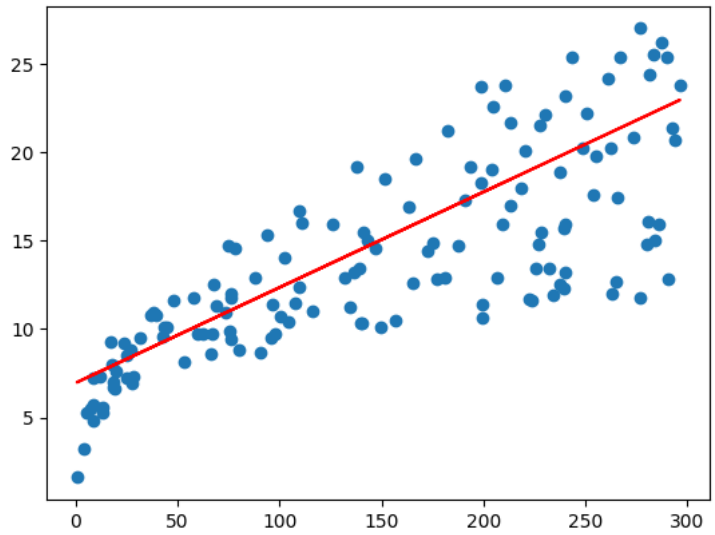
```
const    6.989666
TV       0.046497
dtype: float64
```

```
# 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:                       0.613
Model:                            OLS   Adj. R-squared:                  0.611
Method:                 Least Squares   F-statistic:                     219.0
Date:                Sat, 22 Apr 2023   Prob (F-statistic):           2.84e-30
Time:                        16:57:28   Log-Likelihood:                -370.62
No. Observations:                 140   AIC:                             745.2
Df Residuals:                     138   BIC:                             751.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.9897      0.548     12.762      0.000       5.907       8.073
TV             0.0465      0.003     14.798      0.000       0.040       0.053
==============================================================================
Omnibus:                        0.995   Durbin-Watson:                   1.983
Prob(Omnibus):                  0.608   Jarque-Bera (JB):                0.970
Skew:                          -0.008   Prob(JB):                        0.616
Kurtosis:                       2.593   Cond. No.                         328.
==============================================================================

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

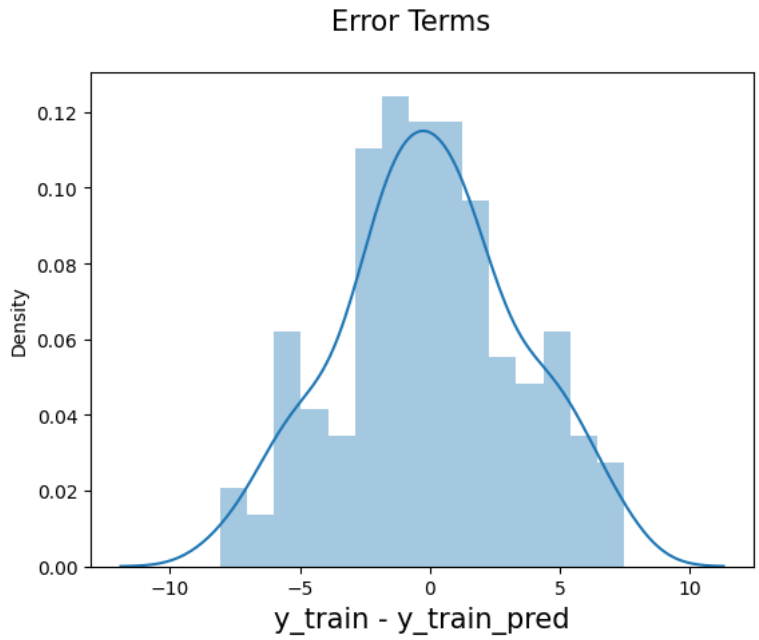
```
plt.scatter(X_train, y_train)
plt.plot(X_train, 6.948 + 0.054*X_train, 'r')
plt.show()
```



Model Evaluation

```
y_train_pred = lr.predict(X_train_sm)
res = (y_train - y_train_pred)
```

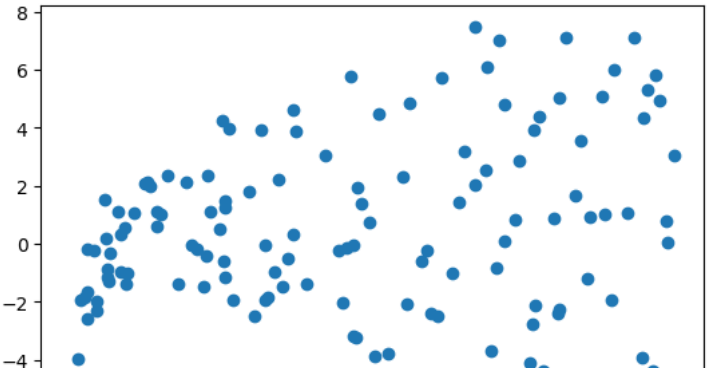
```
fig = plt.figure()
sns.distplot(res, bins = 15)
fig.suptitle('Error Terms', fontsize = 15)                  # Plot heading
plt.xlabel('y_train - y_train_pred', fontsize = 15)         # X-label
plt.show()
```



The residuals are following the normally distributed with a mean 0. All good!

Looking for patterns in the residuals

```
plt.scatter(X_train,res)
plt.show()
```



Predictions on the Test Set

Now that you have fitted a regression line on your train dataset, it's 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` 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)
```

```
y_pred.head()
```

```
126     7.352345
104    18.065337
99     13.276109
92     17.112141
111    18.228077
dtype: float64
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

Looking at the RMSE

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

```
2.8241456288327003
```

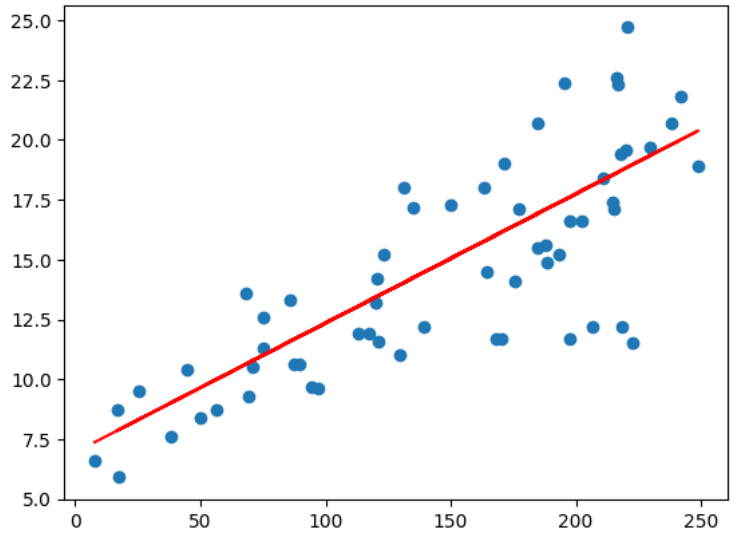
Checking the R-squared on the test set

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

```
0.5942987267783303
```

Visualizing the fit on the test set

```
plt.scatter(X_test, y_test)
plt.plot(X_test, 6.948 + 0.054 * X_test, 'r')
plt.show()
```



✓ 0s completed at 10:28 PM