# SimpleLinearRegression_GradientDescent

February 4, 2024

## 1 Importing Necessary libraries and importing the dataset

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
df = pd.read_csv("datasets/tv_marketing/tvmarketing.csv")
df.head()
```

[ ]:

| | TV | Sales |
|---|---|---|
| 0 | 230.1 | 22.1 |
| 1 | 44.5 | 10.4 |
| 2 | 17.2 | 9.3 |
| 3 | 151.5 | 18.5 |
| 4 | 180.8 | 12.9 |

## 2 Info of dataset and Null checking

[ ]:
```
df.shape # Got around 200 records
```

[ ]: (200, 2)

[ ]:
```
df.describe()
```

[ ]:

| | TV | Sales |
|---|---|---|
| count | 200.000000 | 200.000000 |
| mean | 147.042500 | 14.022500 |
| std | 85.854236 | 5.217457 |
| min | 0.700000 | 1.600000 |
| 25% | 74.375000 | 10.375000 |
| 50% | 149.750000 | 12.900000 |
| 75% | 218.825000 | 17.400000 |
| max | 296.400000 | 27.000000 |

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
```

```
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   TV      200 non-null    float64
 1   Sales   200 non-null    float64
dtypes: float64(2)
memory usage: 3.3 KB
```

[ ]:
```
df.isnull().sum()
```

[ ]:
```
TV       0
Sales    0
dtype: int64
```

[ ]:
```
#No null values so no preprocessing for null values is required.
```

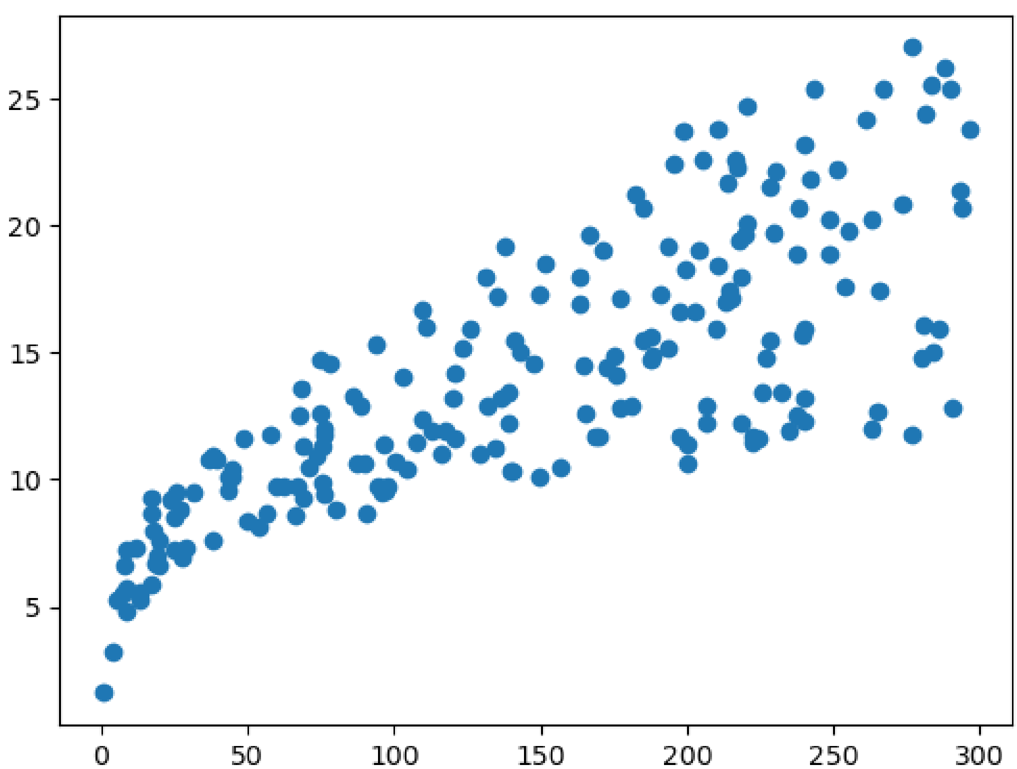
[ ]:
```
df.corr() # a correlation of 0.78 looks good so lets try linear regression
```

[ ]:

| | TV | Sales |
|---|---|---|
| TV | 1.000000 | 0.782224 |
| Sales | 0.782224 | 1.000000 |

[ ]:
```
plt.scatter(df["TV"],df["Sales"])
```

[ ]:
```
<matplotlib.collections.PathCollection at 0x7fe234290990>
```



```
[ ]: #based on the data visualization we can fit out a line => linear regression
```

```
[ ]: # There is no encoding required as there is no categorical columns
```

```
[ ]: # No scaling required as there is one variable /feature that we are using to␣
      ↪predict
```

```
[ ]: # Split the dataset into dependent and independent variables
```

```
[ ]: X = np.array(df["TV"],dtype=np.float64)
     y = np.array(df["Sales"],dtype=np.float64)
```

# 3 Test train split

```
[ ]: from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.8)
```

# 4 Train the Model

```
[ ]: def gradient_descent(x, y, m, c, alpha, num_iters):
         n = len(y)
         mse_history = np.zeros(num_iters)

         for i in range(num_iters):
             y_pred = m*x + c
             mse = np.sum((y - y_pred)**2) / n
             dm = -(2/n)*np.sum(x*(y - y_pred))
             dc = -(2/n)*np.sum(y - y_pred)
             m = m - alpha*dm
             c = c - alpha*dc

             mse_history[i] = mse

         return m, c, mse_history
```

```
[ ]: m, c, mse_history = gradient_descent(X_train, y_train,0,0, 0.0000001, 1000)
```

```
[ ]: print(m,c)
```

```
0.0832391961174241 0.0007796140101263338
```

```
[ ]: y_pred = m*X_test + c
     pd.DataFrame(y_pred, y_test).head()
```

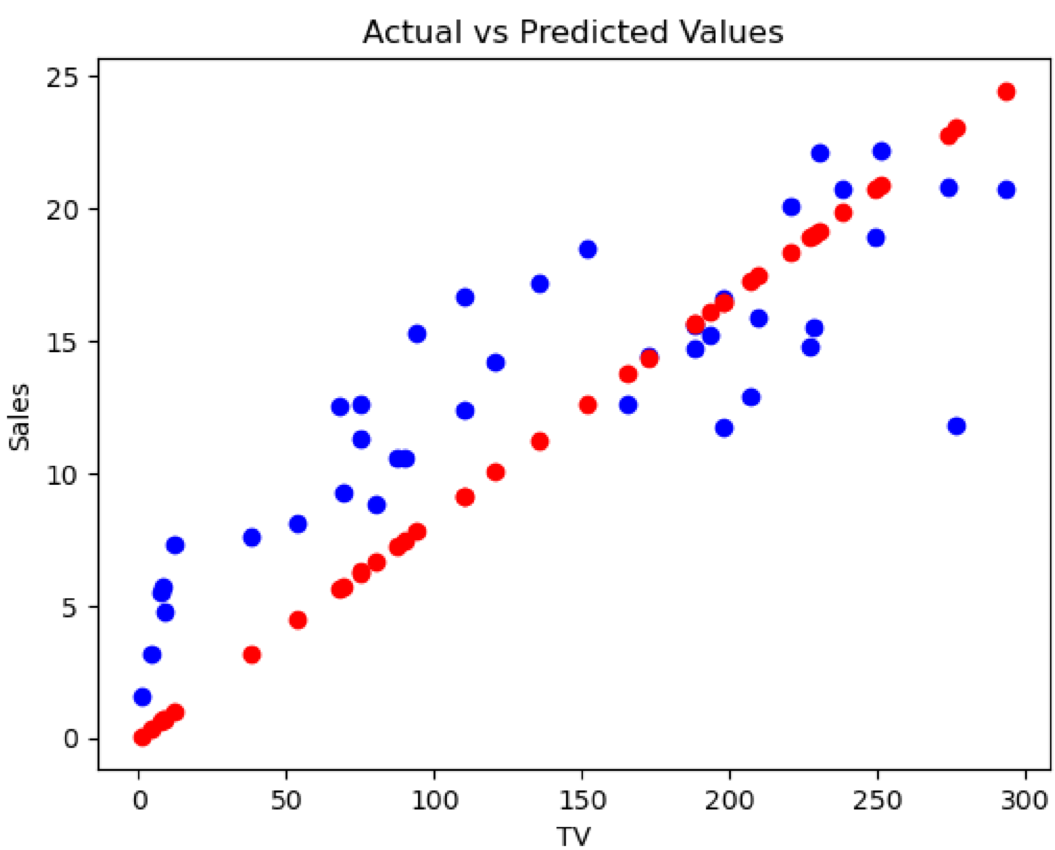
[ ]:
```
             0
12.6   6.252043
16.6  16.448845
11.8  23.033065
8.8    6.676563
15.3   7.816940
```

[ ]:
```
# Plotting the actual values
plt.scatter(X_test, y_test, color='blue', label='Actual')

# Plotting the predicted values
plt.scatter(X_test, y_pred, color='red', label='Predicted')

# Adding labels and title to the plot
plt.xlabel('TV')
plt.ylabel('Sales')
plt.title('Actual vs Predicted Values')
```

[ ]:
```
Text(0.5, 1.0, 'Actual vs Predicted Values')
```

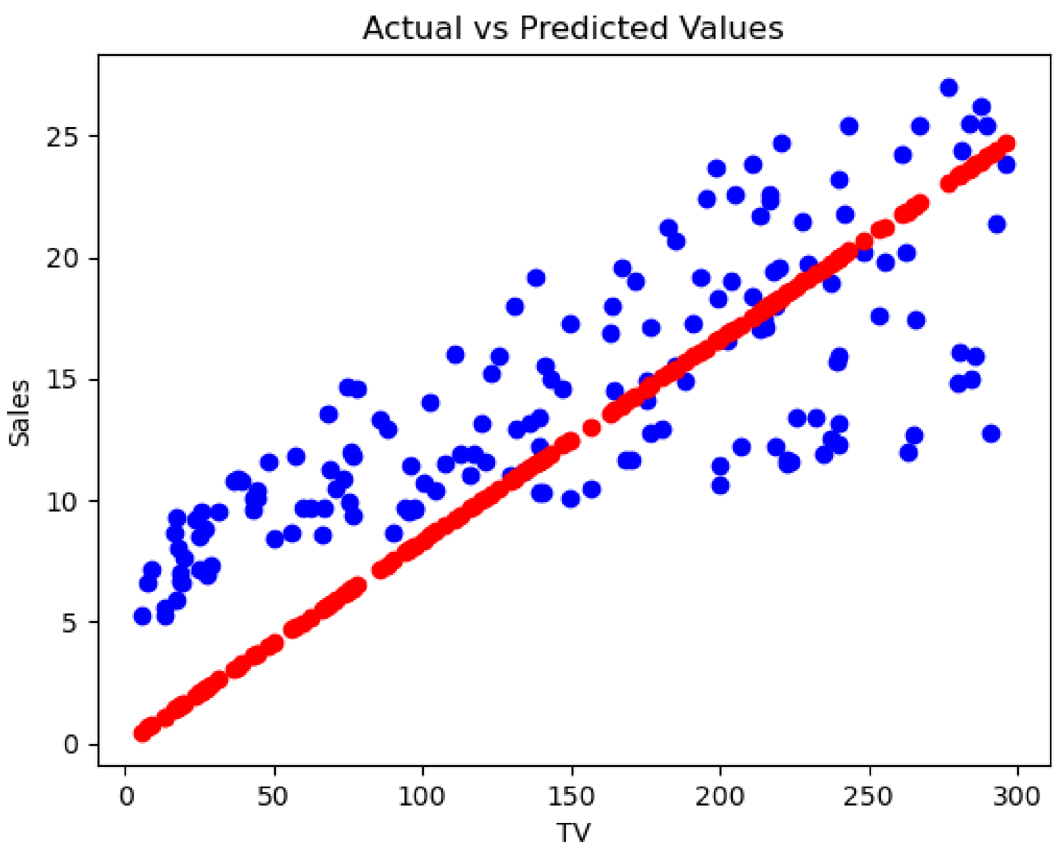


[ ]:
```
# Plotting the actual values
plt.scatter(X_train, y_train, color='blue', label='Actual')

# Plotting the predicted values
plt.scatter(X_train, m*X_train+c, color='red', label='Predicted')

# Adding labels and title to the plot
plt.xlabel('TV')
plt.ylabel('Sales')
plt.title('Actual vs Predicted Values')
```

[ ]: Text(0.5, 1.0, 'Actual vs Predicted Values')




## 5 Evaluate the model

[ ]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
print("The mean absolute error is: ",mean_absolute_error(y_test, m*X_test+c))
print("The mean squared error: ",mean_squared_error(y_test, m*X_test+c))
print(r2_score(y_pred,y_test))
```

```
The mean absolute error is:  3.614280924693047
The mean squared error:  18.70996462733873
0.6463383416172792
```

# 6 Conclusion

```
[ ]: print(f"The slope is: {m} and intercept is: {c}")
```

```
The slope is: 0.0832391961174241 and intercept is: 0.0007796140101263338
```

### 6.0.1 So what does slope and intercept mean ?

*Lets consider the scenario so every unit increase in tv there is a increase of 0.083 unit increase in sales. As for intercept, this means that when TV equals 0, the model predicts sales will be 0.077 units. This suggests there is some baseline level of sales even without any TV advertising spend*

As for conclusion based on metrics of the model , the MAE suggests the model's predictions are off from the actual values by around 2.6 units on average. This seems reasonably accurate.