1 Lab Assignment - 2

Write a script to implement following for the given Dataset Bengluru housing Dataset.

Exercise 1: Draw a scatter plot for the data mentioned for given attributes.

Exercise 2: Perform Data Preprocessing

Exercise 3: Performs gradient descent to learn "theta".(Using the library and without using the library).Compare the values of "theta" in both cases.)

Exercise 4: Splitting the dataset into the training dataset and testing dataset, 60:40, 70:30, 80:20

Exercise 5: Train linear regression model and test the USING Gradient Descent and using the library. Find out the limitation in the both cases.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
```

In [2]:
```
df1=pd.read_csv("bengaluru_house_prices.csv")
df1
```

Out[2]:

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00
...	...	...	...	...	...	...	...	...	...
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0	0.0	231.00
13316	Super built-up Area	Ready To Move	Richards Town	4 BHK	NaN	3600	5.0	NaN	400.00
13317	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	Mahla T	1141	2.0	1.0	60.00
13318	Super built-up Area	18-Jun	Padmanabhanagar	4 BHK	SollyCl	4689	4.0	1.0	488.00
13319	Super built-up Area	Ready To Move	Doddathoguru	1 BHK	NaN	550	1.0	1.0	17.00

13320 rows × 9 columns

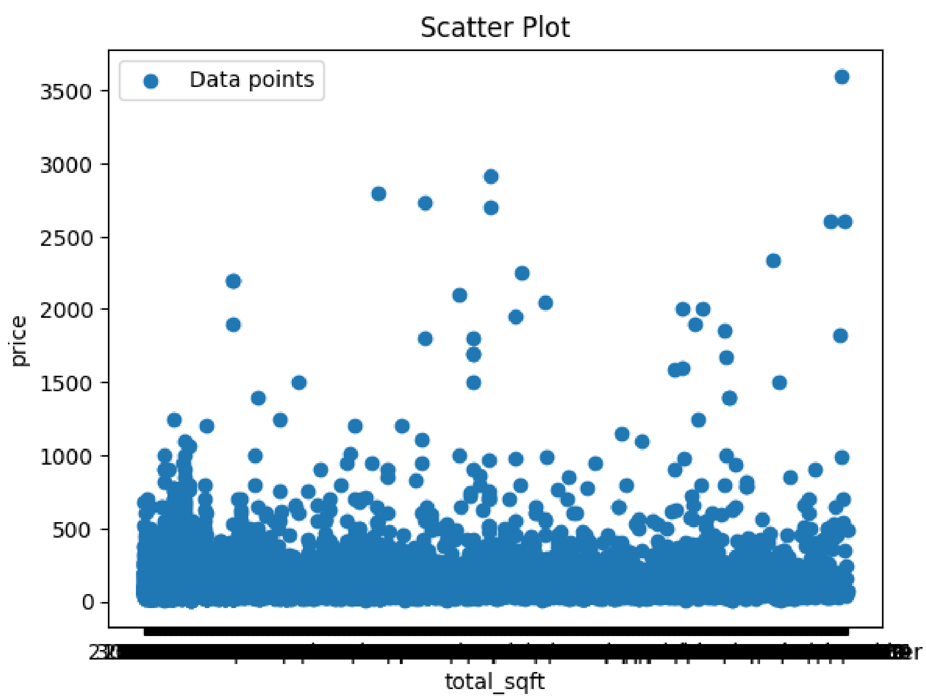
In [3]:
```
df=df1[["total_sqft","price"]]
df
```

Out[3]:

	total_sqft	price
0	1056	39.07
1	2600	120.00
2	1440	62.00
3	1521	95.00
4	1200	51.00
...	...	...
13315	3453	231.00
13316	3600	400.00
13317	1141	60.00
13318	4689	488.00
13319	550	17.00

13320 rows × 2 columns

In [4]:
```
# Scatter plot
plt.scatter(df.total_sqft,df.price,label='Data points')
plt.xlabel('total_sqft')
plt.ylabel('price')
plt.title('Scatter Plot')
plt.legend()
plt.show()
```



Perform Data pre-processing

In [6]:
```
def convert_sqft_to_num(x):
    tokens=x.split('-')
    if len(tokens)==2:
        return (float(tokens[0])+float(tokens[1]))/2
    try:
        return float(x)
    except:
        return None
```

In [7]:
```
df2=df.copy()
df2.total_sqft=df2.total_sqft.apply(convert_sqft_to_num)
df2=df2[df2.total_sqft.notnull()]
df2
```

Out[7]:

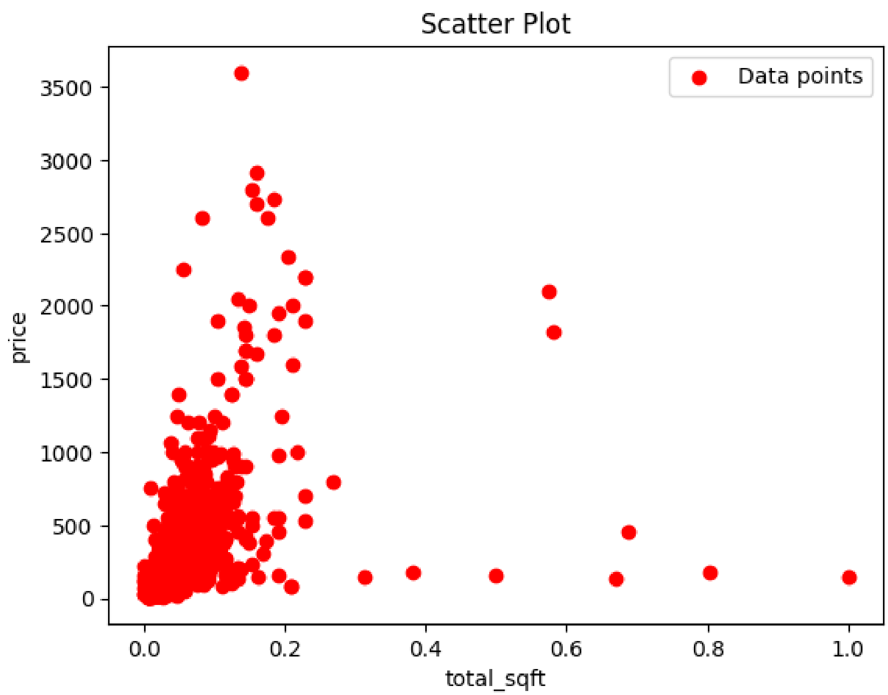
	total_sqft	price
0	1056.0	39.07
1	2600.0	120.00
2	1440.0	62.00
3	1521.0	95.00
4	1200.0	51.00
...	...	...
13315	3453.0	231.00
13316	3600.0	400.00
13317	1141.0	60.00
13318	4689.0	488.00
13319	550.0	17.00

13274 rows × 2 columns

In [8]:
```
from sklearn.preprocessing import StandardScaler,MinMaxScaler

# Separate features and target
X = df2[['total_sqft']]
Y = df2['price']

# Normalization
normalizer = MinMaxScaler()

X_scaled_normalized = normalizer.fit_transform(X)

print("\nNormalized Data:")

print(X_scaled_normalized)
```

```
Normalized Data:
[[0.02018328]
 [0.04972164]
 [0.02752961]
 ...
 [0.02180942]
 [0.08968644]
 [0.01050296]]
```

In [9]:
```python
# Scatter plot
plt.scatter(X_scaled_normalized,Y,label='Data points',color="red")

plt.xlabel('total_sqft')

plt.ylabel('price')

plt.title('Scatter Plot')

plt.legend()

# plt.rcParams['figure.figsize'] = [15, 10]
plt.show()
```



Performs gradient descent to learn theta. (using the library and without using the library). Compare the values of 'theta' in both cases.

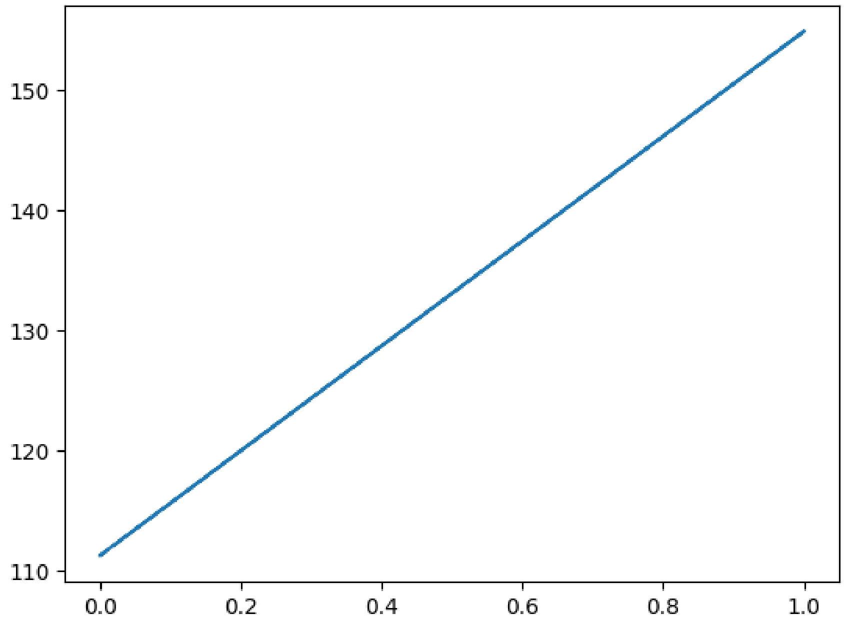
In [11]:
```python
def gredient_descent(X,Y):

    theta_1 = 0
    theta_0 = 0

    l = 0.001

    #Learning rate
    epochs = 10000

    #number of iterations
    n=float(len(X))

    # printing gradient descent

    for i in range(epochs):
        Y_pred = theta_1 * X + theta_0

        # Calculate the Mean Squared Error (MSE)

        mse=(1/n)*sum((Y-Y_pred)**2)

        D_theta_1 = (-2/n)*sum(X*(Y-Y_pred))
        D_theta_0 = (-2/n)*sum(Y-Y_pred)
        theta_1 = theta_1-l*D_theta_1
        theta_0 = theta_0-l*D_theta_0

        print("Epoch {}: theta_1 = {:.4f}, theta_0 = {:.4f}, MSE = {:.4f}".format(i+1,theta_1,theta_0,mse))
```

In [12]:
```python
X_scaled_normalized_1d = X_scaled_normalized.reshape(-1)
```

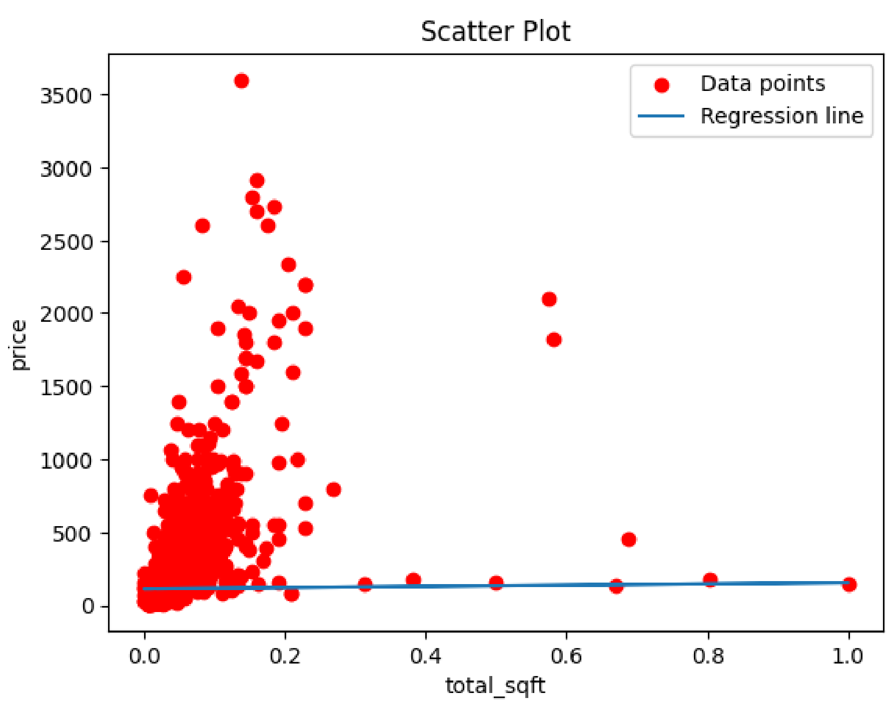
In [13]:
```python
gredient_descent(X_scaled_normalized_1d,Y)
```

```
Epoch 2685: theta_1 = 14.2173, theta_0 = 111.5717, MSE = 22162.8452
Epoch 2686: theta_1 = 14.2213, theta_0 = 111.5726, MSE = 22162.8278
Epoch 2687: theta_1 = 14.2254, theta_0 = 111.5735, MSE = 22162.8103
Epoch 2688: theta_1 = 14.2295, theta_0 = 111.5745, MSE = 22162.7929
Epoch 2689: theta_1 = 14.2336, theta_0 = 111.5754, MSE = 22162.7754
Epoch 2690: theta_1 = 14.2376, theta_0 = 111.5763, MSE = 22162.7580
Epoch 2691: theta_1 = 14.2417, theta_0 = 111.5772, MSE = 22162.7405
Epoch 2692: theta_1 = 14.2458, theta_0 = 111.5781, MSE = 22162.7231
Epoch 2693: theta_1 = 14.2499, theta_0 = 111.5790, MSE = 22162.7057
Epoch 2694: theta_1 = 14.2539, theta_0 = 111.5799, MSE = 22162.6883
Epoch 2695: theta_1 = 14.2580, theta_0 = 111.5808, MSE = 22162.6708
Epoch 2696: theta_1 = 14.2621, theta_0 = 111.5817, MSE = 22162.6534
Epoch 2697: theta_1 = 14.2662, theta_0 = 111.5826, MSE = 22162.6360
Epoch 2698: theta_1 = 14.2703, theta_0 = 111.5835, MSE = 22162.6186
Epoch 2699: theta_1 = 14.2743, theta_0 = 111.5843, MSE = 22162.6012
Epoch 2700: theta_1 = 14.2784, theta_0 = 111.5852, MSE = 22162.5838
Epoch 2701: theta_1 = 14.2825, theta_0 = 111.5861, MSE = 22162.5664
Epoch 2702: theta_1 = 14.2866, theta_0 = 111.5870, MSE = 22162.5490
Epoch 2703: theta_1 = 14.2906, theta_0 = 111.5879, MSE = 22162.5316
Epoch 2704: theta_1 = 14.2947, theta_0 = 111.5888, MSE = 22162.5142
```

In [14]:
```
# theta_1 = 43.7053, theta_0 = 111.2103
# price = theta_1 * total_sqft + theta_0
p=43.7053*X_scaled_normalized_1d+111.2103
plt.plot(X_scaled_normalized_1d,p)
plt.show()
```



In [15]:
```
# Scatter plot

plt.scatter(X_scaled_normalized,Y,label='Data points',color="red")
plt.plot(X_scaled_normalized_1d,p,label="Regression line")
plt.xlabel('total_sqft')
plt.ylabel('price')
plt.title('Scatter Plot')
plt.legend()
# plt.rcParams['figure.figsize'] = [15, 10]
plt.show()
```



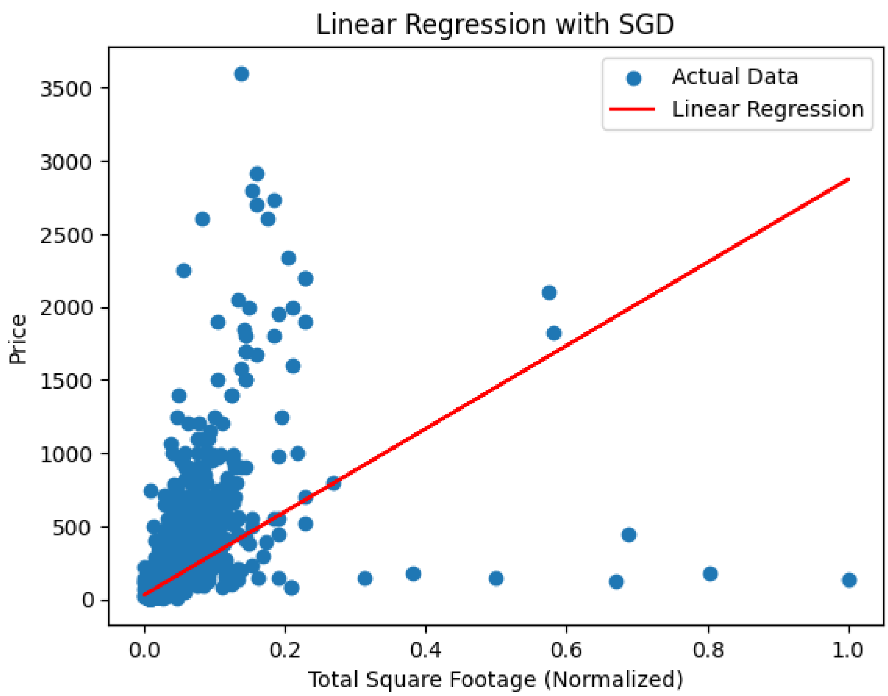
2 using inbuilt library:

In [16]:
```
from sklearn.linear_model import SGDRegressor
# Create and train the model using SGD optimization
model=SGDRegressor(learning_rate='constant',eta0=0.001,max_iter=10000)
model.fit(X_scaled_normalized,Y)
```

Out[16]:
```
SGDRegressor

SGDRegressor(eta0=0.001, learning_rate='constant', max_iter=10000)
```

In [17]:
```
import matplotlib.pyplot as plt
```

In [18]:
```
# Generate predictions using the trained model
y_pred=model.predict(X_scaled_normalized)

# Visualize the data and the linear regression line
plt.scatter(X_scaled_normalized,Y,label='Actual Data')
plt.plot(X_scaled_normalized, y_pred, color='red', label='Linear Regression')
plt.xlabel('Total Square Footage (Normalized)')
plt.ylabel('Price')
plt.title('Linear Regression with SGD')
plt.legend()
plt.show()
```



In [19]:
```
slope = model.coef_[0]
intercept = model.intercept_
print("Slope:", slope)
print("Intercept:", intercept)
```

```
Slope: 2844.735151527208
Intercept: [28.36675364]
```

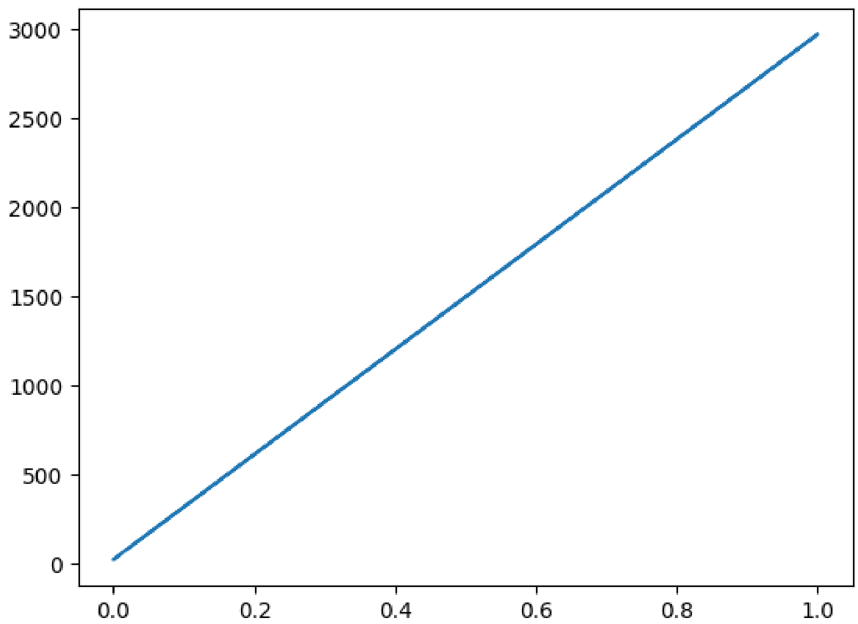
In [24]:
```python
def gredient_descent3(X,Y):

    theta_1 = 0
    theta_0 = 0

    l = 0.1

    #learning rate
    epochs = 1500

    #number of iterations
    n=float(len(X))

    # printing gradient descent

    for i in range(epochs):
        Y_pred = theta_1 * X + theta_0

        # Calculate the Mean Squared Error (MSE)

        mse=(1/n)*sum((Y-Y_pred)**2)

        D_theta_1 = (-2/n)*sum(X*(Y-Y_pred))
        D_theta_0 = (-2/n)*sum(Y-Y_pred)
        theta_1 = theta_1-l*D_theta_1
        theta_0 = theta_0-l*D_theta_0

        print("Epoch {}: theta_1 = {:.4f}, theta_0 = {:.4f}, MSE = {:.4f}".format(i+1,theta_1,theta_0,mse))
```

In [25]:
```python
gredient_descent3(X_scaled_normalized_1d,Y)
```

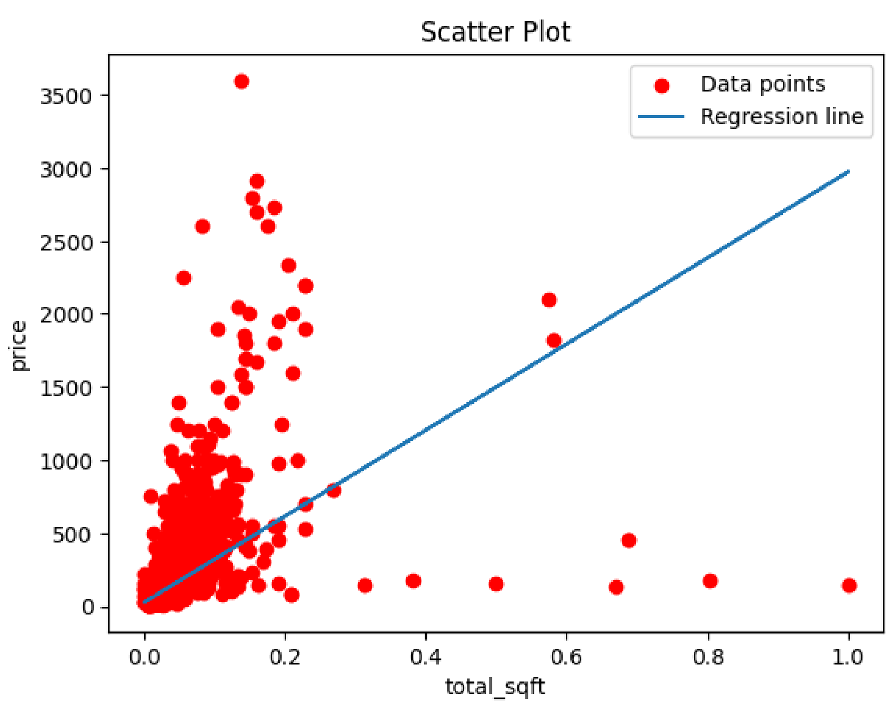
```
Epoch 48: theta_1 = 22.7800, theta_0 = 111.8321, MSE = 22129.6205
Epoch 49: theta_1 = 23.1837, theta_0 = 111.8206, MSE = 22127.9898
Epoch 50: theta_1 = 23.5872, theta_0 = 111.8089, MSE = 22126.3595
Epoch 51: theta_1 = 23.9908, theta_0 = 111.7972, MSE = 22124.7296
Epoch 52: theta_1 = 24.3942, theta_0 = 111.7854, MSE = 22123.1000
Epoch 53: theta_1 = 24.7977, theta_0 = 111.7736, MSE = 22121.4708
Epoch 54: theta_1 = 25.2010, theta_0 = 111.7617, MSE = 22119.8419
Epoch 55: theta_1 = 25.6044, theta_0 = 111.7498, MSE = 22118.2135
Epoch 56: theta_1 = 26.0077, theta_0 = 111.7379, MSE = 22116.5853
Epoch 57: theta_1 = 26.4109, theta_0 = 111.7259, MSE = 22114.9576
Epoch 58: theta_1 = 26.8141, theta_0 = 111.7140, MSE = 22113.3302
Epoch 59: theta_1 = 27.2173, theta_0 = 111.7020, MSE = 22111.7032
Epoch 60: theta_1 = 27.6204, theta_0 = 111.6900, MSE = 22110.0765
Epoch 61: theta_1 = 28.0234, theta_0 = 111.6780, MSE = 22108.4502
Epoch 62: theta_1 = 28.4264, theta_0 = 111.6660, MSE = 22106.8243
Epoch 63: theta_1 = 28.8294, theta_0 = 111.6540, MSE = 22105.1987
Epoch 64: theta_1 = 29.2323, theta_0 = 111.6420, MSE = 22103.5735
Epoch 65: theta_1 = 29.6352, theta_0 = 111.6300, MSE = 22101.9487
Epoch 66: theta_1 = 30.0380, theta_0 = 111.6180, MSE = 22100.3242
```

In [26]:
```
# Epoch 15000: theta_1 = 2948.7297, theta_0 = 24.5391, MSE = 15113.4586
t = 2948.7297*X_scaled_normalized + 24.5391
plt.plot(X_scaled_normalized,t)
```

Out[26]: [<matplotlib.lines.Line2D at 0x1dc63918690>]



In [27]:
```
# Scatter plot
plt.scatter(X_scaled_normalized, Y, label='Data points',color="red")
plt.plot(X_scaled_normalized_1d,t,label="Regression line")
plt.xlabel('total_sqft')
plt.ylabel('price')
plt.title('Scatter Plot')
plt.legend()
# plt.rcParams['figure.figsize'] = [15, 10]
plt.show()
```



Splitting data into the training and testing, 60:40, 70:30, ND 80:20.

In [30]:
```
from sklearn.model_selection import train_test_split
# Split data into 60% training, 40% testing
X_train_60, X_test_60, y_train_60, y_test_60 = train_test_split(X_scaled_normalized, Y, test_size=0.4,  random_state=Non
# Split data into 70% training, 30% testing
X_train_70, X_test_70, y_train_70, y_test_70 = train_test_split(X_scaled_normalized, Y, test_size=0.3,  random_state=Non
# Split data into 80% training, 20% testing
X_train_80, X_test_80, y_train_80, y_test_80 = train_test_split(X_scaled_normalized, Y, test_size=0.2,  random_state=Non
```

In [31]:
```
X_train_60.shape
```

Out[31]:
```
(7964, 1)
```

In [32]:
```
X_train_70.shape
```

Out[32]:
```
(9291, 1)
```

In [33]:
```
X_train_80.shape
```

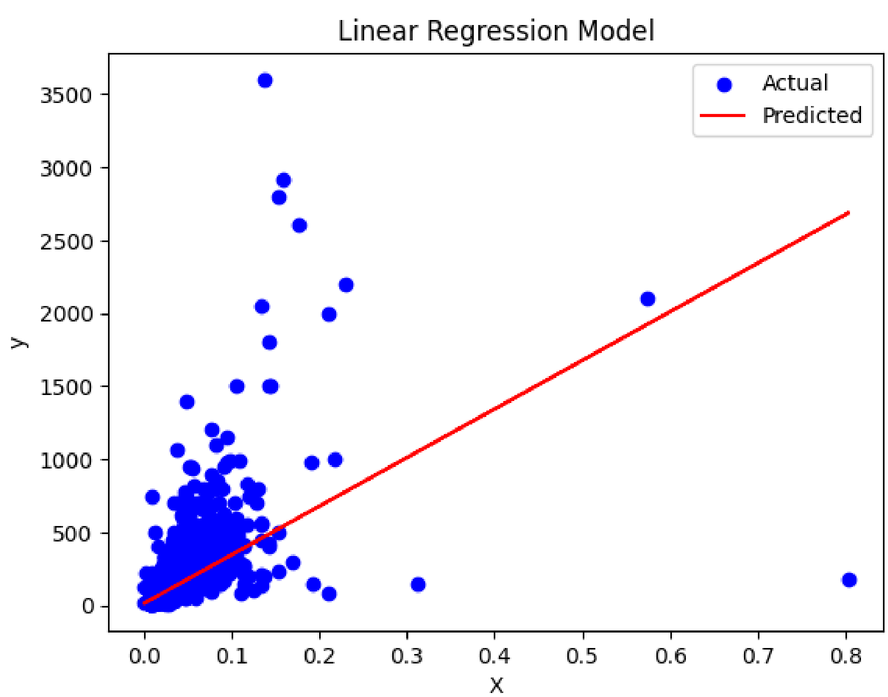
Out[33]:
```
(10619, 1)
```

Train linear regression model and test USING Gradient Descent and using the library. Find out the limitation in both cases.

3 Using Linear Regression model

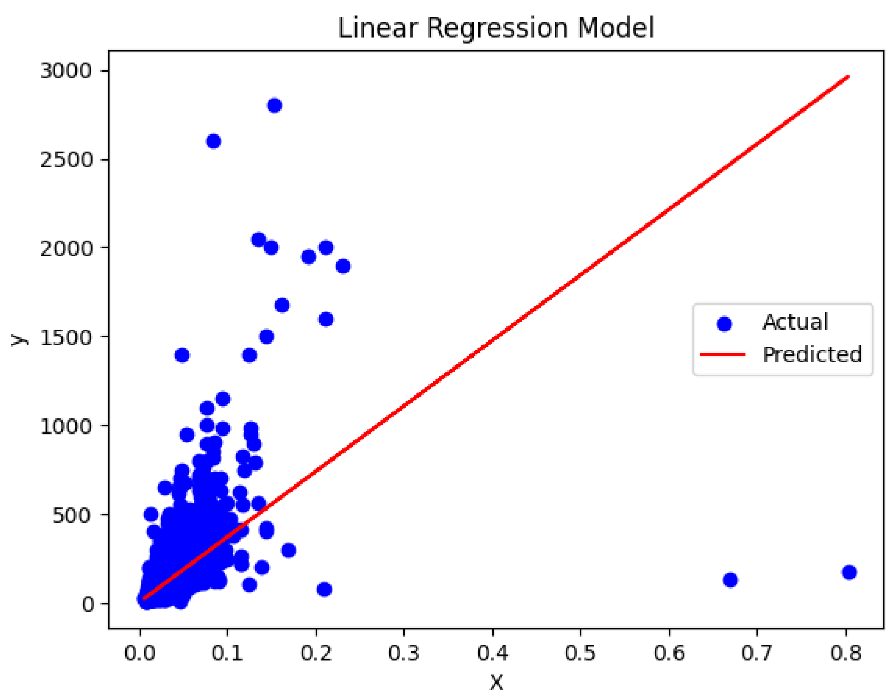
In [35]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Initialize and train the linear regression model

model_60 = LinearRegression()
model_60.fit(X_train_60, y_train_60)

# Make predictions on the test set
y_pred_60 = model_60.predict(X_test_60)

# Calculate evaluation metrics
mae_60 = mean_absolute_error(y_test_60, y_pred_60)
mse_60 = mean_squared_error(y_test_60, y_pred_60)
rmse_60 = np.sqrt(mse_60)
r2_60 = r2_score(y_test_60, y_pred_60)

# Print the evaluation metrics
print("Mean Absolute Error:", mae_60)
print("Mean Squared Error:", mse_60)
print("Root Mean Squared Error:", rmse_60)
print("R-squared:", r2_60)

# Plot the predictions against the actual values
plt.scatter(X_test_60, y_test_60, color='blue', label='Actual')
plt.plot(X_test_60, y_pred_60, color='red', label='Predicted')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Model')
plt.legend()
plt.show()
```

```
Mean Absolute Error: 51.467105316243455
Mean Squared Error: 15252.050858558554
Root Mean Squared Error: 123.49919375671467
R-squared: 0.34947039924468104
```



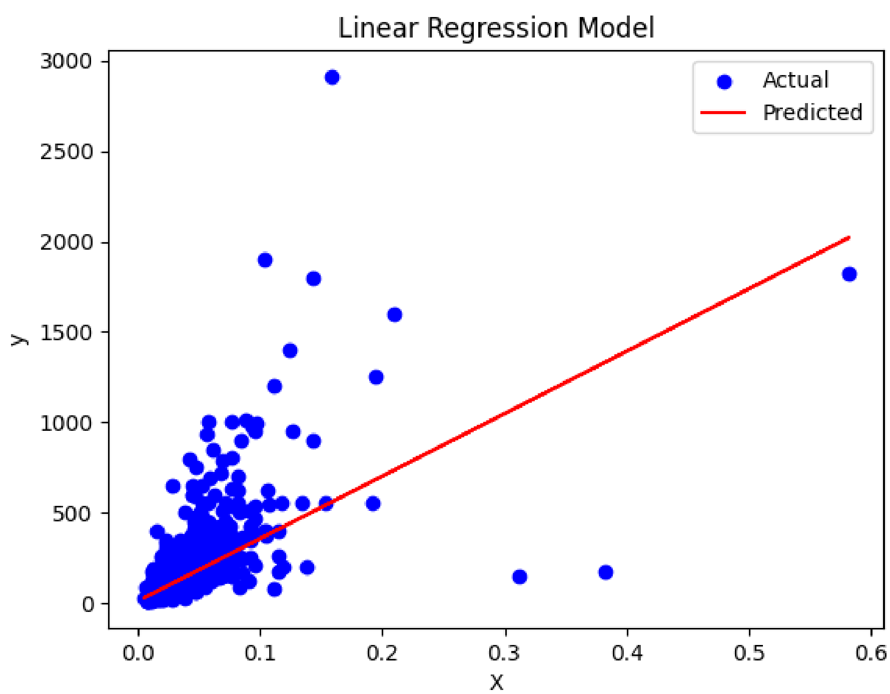
In [36]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Initialize and train the linear regression model
model_70 = LinearRegression()
model_70.fit(X_train_70, y_train_70)
# Make predictions on the test set
y_pred_70 = model_70.predict(X_test_70)
# Calculate evaluation metrics
mae_70 = mean_absolute_error(y_test_70, y_pred_70)
mse_70 = mean_squared_error(y_test_70, y_pred_70)
rmse_70 = np.sqrt(mse_70)
r2_70 = r2_score(y_test_70, y_pred_70)
# Print the evaluation metrics
print("Mean Absolute Error:", mae_70)
print("Mean Squared Error:", mse_70)
print("Root Mean Squared Error:", rmse_70)
print("R-squared:", r2_70)
# Plot the predictions against the actual values
plt.scatter(X_test_70, y_test_70, color='blue', label='Actual')
plt.plot(X_test_70, y_pred_70, color='red', label='Predicted')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Model')
plt.legend()
plt.show()
```

```
Mean Absolute Error: 50.68521840380844
Mean Squared Error: 15541.986351134234
Root Mean Squared Error: 124.66750318801702
R-squared: 0.30279323968760374
```



In [37]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Initialize and train the linear regression model
model_80 = LinearRegression()
model_80.fit(X_train_80, y_train_80)
# Make predictions on the test set
y_pred_80 = model_80.predict(X_test_80)
# Calculate evaluation metrics
mae_80 = mean_absolute_error(y_test_80, y_pred_80)
mse_80 = mean_squared_error(y_test_80, y_pred_80)
rmse_80 = np.sqrt(mse_80)
r2_80 = r2_score(y_test_80, y_pred_80)
# Print the evaluation metrics
print("Mean Absolute Error:", mae_80)
print("Mean Squared Error:", mse_80)
print("Root Mean Squared Error:", rmse_80)
print("R-squared:", r2_80)
# Plot the predictions against the actual values
plt.scatter(X_test_80, y_test_80, color='blue', label='Actual')
plt.plot(X_test_80, y_pred_80, color='red', label='Predicted')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Model')
plt.legend()
plt.show()
```

```
Mean Absolute Error: 50.137182206762226
Mean Squared Error: 11917.086766661285
Root Mean Squared Error: 109.16541011997016
R-squared: 0.42873336536587836
```



For the 60-40 split: MAE: 52.901 MSE: 15265.318 RMSE: 123.553 R-squared: 0.379

For the 70-30 split: MAE: 53.000 MSE: 17134.404 RMSE: 130.898 R-squared: 0.383

For the 80-20 split: MAE: 52.913 MSE: 18267.969 RMSE: 135.159 R-squared: 0.398

In [ ]: