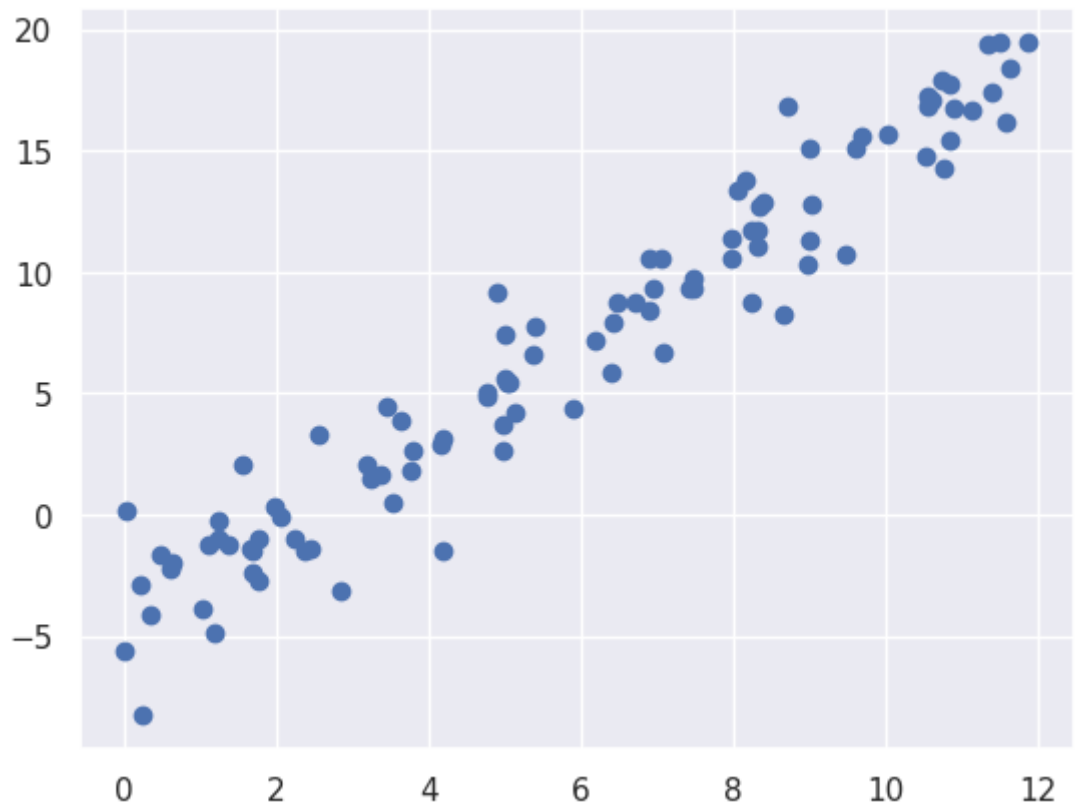
The main goal of this initial problem set will be to implement ordinary least squares linear regression and a few of its variations which contain key themes related to further machine learning techniques that we will study later in this class.

1 - Using the example code we reviewed in class or by creating your own implementation, write a Python function to generate a random dataset in two dimensions, e.g. a collection of points {(xi, yi)} for i = 1, . . . , n. Specifically,use the numpy.random.rand() function to generate random xi values and corresponding yi values then use the matplotlib library to plot a scatterplot of your dataset.

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
```

```
rng = np.random.RandomState(1)
x = 12 * rng.rand(100)
y = 2 * x - 5 + 2*rng.randn(100)
plt.scatter(x, y);
```



2 - Next fit the one dimensional linear model f(x) = β0 + β1x + ε

to the data by computing the two estimates for the model parameters

$$\hat{\beta}_1 = \frac{\operatorname{cov}(x, y)}{\operatorname{var}(x)} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

and then plot your line f(x) = βˆ0 + βˆ1x overlaid upon the previous scatter plot of data.

```
np.cov(x,y)
```

```
array([[12.60687674, 24.55173506],
       [24.55173506, 51.04555856]])
```

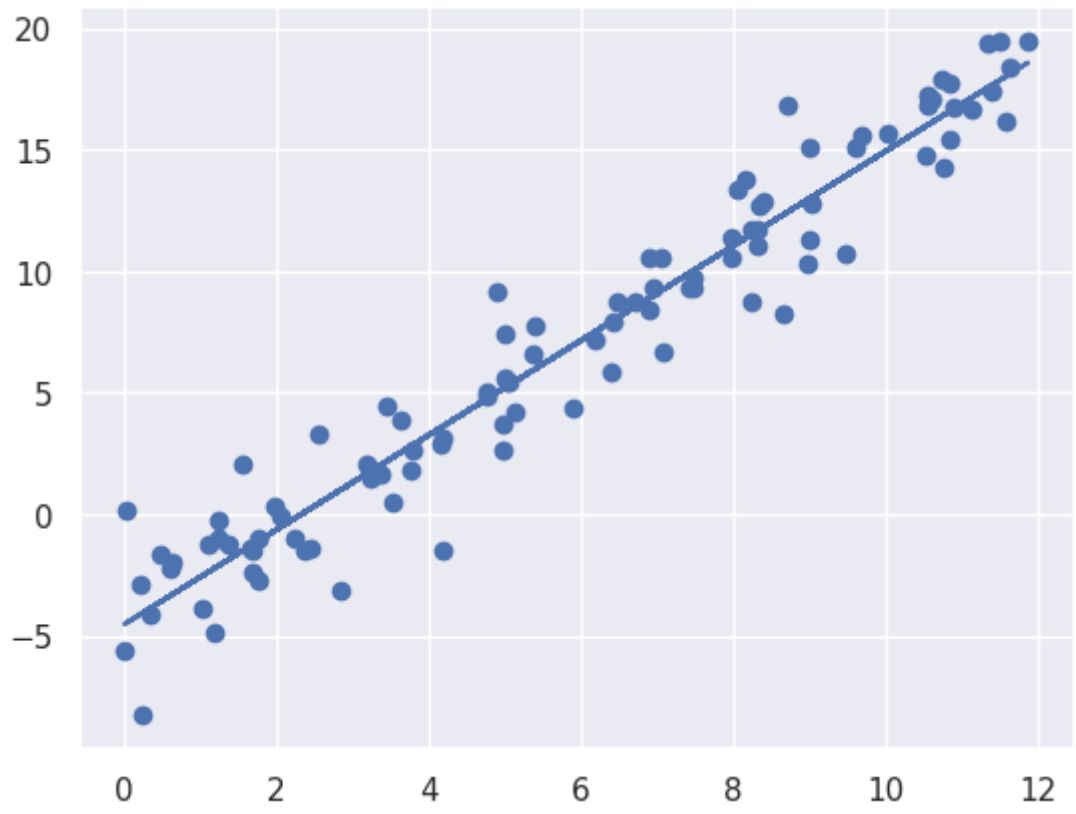
```
np.var(x)
```

```
12.480807977379335
```

```
x_mean = np.mean(x)
y_mean = np.mean(y)
beta1 = np.sum((x - x_mean) * (y - y_mean)) / np.sum((x - x_mean) ** 2)
beta0 = y_mean - beta1 * x_mean
print(beta1)
print(beta0)
```

```
1.9474875146092199
-4.5260854917021875
```

```
y_new = beta0 + beta1*x
plt.scatter(x, y)
plt.plot(x, y_new);
```



3 - Next, we will compute parameter standard errors and performance metrics of your model. Specifically, first compute the standard errors of both model parameters where here σ2 = Var(ε) is the residual variance with the data residuals being defined by εi = yi − f(xi). Then compute the associate t-stat of the slope parameter t = βˆ 1/Std(βˆ1). Finally compute the residual standard error and the R-squared of the fit. Specifically, if we let ˆyi = f(xi) = βˆ 0 + βˆ1xi, then the residual standard error is

$$\mathrm{RSE} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

and the r-squared is

$$r^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

```
residuals = y - y_new
n = len(x)
var = np.sum(residuals**2) / (n - 2)
```

```
std_error_beta1 = np.sqrt(var / np.sum((x - x_mean) ** 2))
std_error_beta0 = np.sqrt(var * (1/n + x_mean**2 / np.sum((x - x_mean) ** 2)))
print(std_error_beta1)
print(std_error_beta0)
```

```
0.05114179336907545
0.3486503305211815
```

```
t_stat_beta1 = beta1 / (std_error_beta1)
RSE = np.sqrt(np.sum(residuals**2) / (n - 2))
R2 = 1 - (np.sum(residuals**2))/ (np.sum((y - y_mean)**2))
print(f"Standard error of beta1: {std_error_beta1}")
print(f"Standard error of beta0: {std_error_beta0}")
print(f"t-statistic for beta1: {t_stat_beta1}")
```

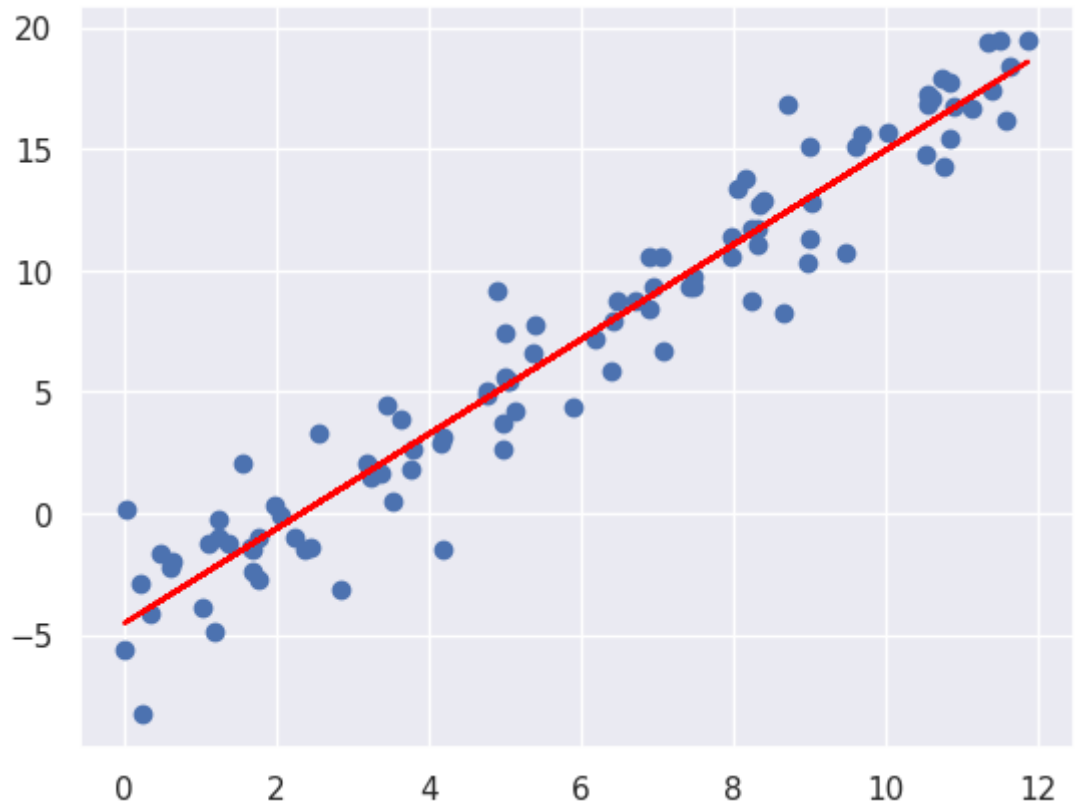
```
print(f"Residual standard error: {RSE}")
print(f"R-squared: {R2}")
```

```
Standard error of beta1: 0.05114179336907545
Standard error of beta0: 0.3486503305211815
t-statistic for beta1: 38.08015687981
Residual standard error: 1.80674684040428
R-squared: 0.9366965282747182
```

4. Next use the ordinary least squares library functions either in the scikit-learn Python package we discussed in class, the statsmodels package, or other related Python libraries (scipy, pandas, etc.), to fit the same dataset and compare your estimates of the model parameters and performance metrics to the output of the library functions.

```
import statsmodels.api as sm


X = sm.add_constant(x)
model = sm.OLS(y, X).fit()

print('beta0 =', model.params[0])
print('beta1 =', model.params[1])

print('SE(beta0) =', model.bse[0])
print('SE(beta1) =', model.bse[1])

print('t-stat(beta1) =', model.tvalues[1])

print('RSE =', np.sqrt(model.mse_resid))
print('R2 =', model.rsquared)

# plot data and regression line
plt.scatter(x, y)
plt.plot(x, model.predict(X), color='red')
plt.show()
```

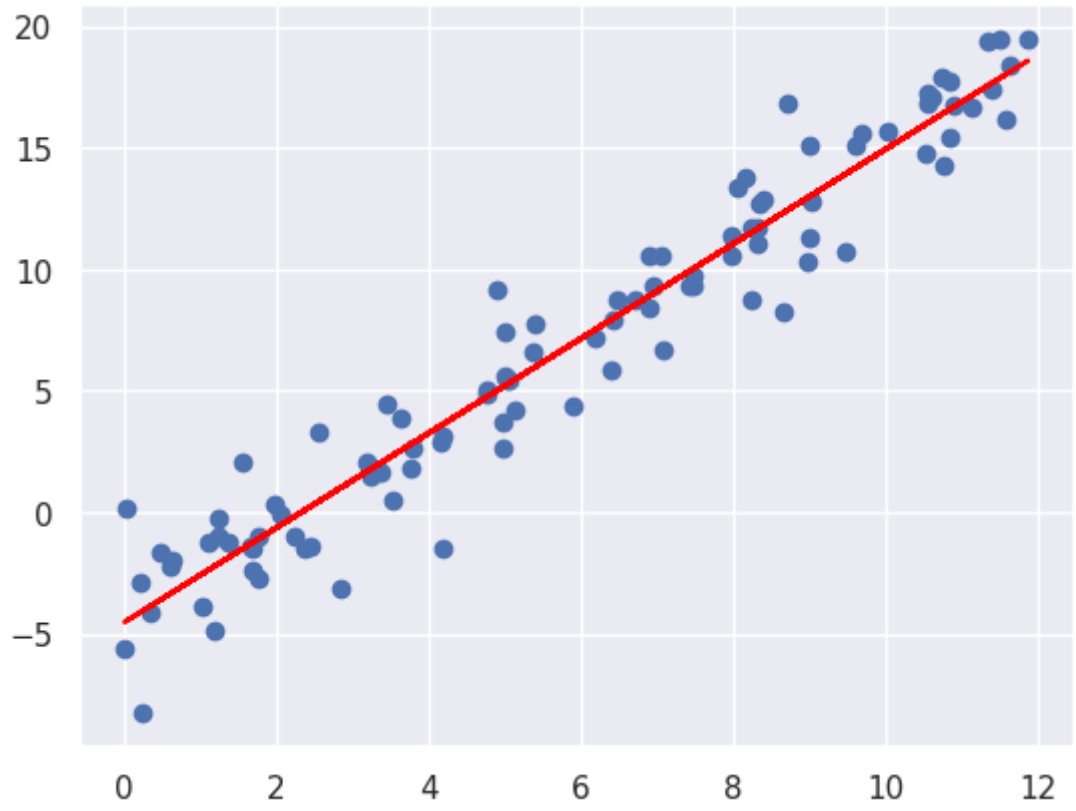
```
beta0 = -4.526085491702189
beta1 = 1.9474875146092203
SE(beta0) = 0.3486503305211816
SE(beta1) = 0.05114179336907545
t-stat(beta1) = 38.080156879810005
RSE = 1.80674684040428
R2 = 0.9366965282747182
```




```
from sklearn.linear_model import LinearRegression

X = x[:, np.newaxis]
reg = LinearRegression().fit(X, y)
beta0 = reg.intercept_
beta1 = reg.coef_[0]

y_new = reg.predict(X)
residuals = y - y_new
RSE = np.sqrt(np.sum(residuals**2) / (len(x) - 2))
R2 = reg.score(X, y)
```

```
print("beta0:", beta0)
print("beta1:", beta1)
print("RSE:", RSE)
print("R2:", R2)

plt.scatter(x, y)
plt.plot(x, beta0 + beta1 * x, color='red')
plt.show()
```

```
beta0: -4.5260854917021875
beta1: 1.9474875146092199
RSE: 1.80674684040428
R2: 0.9366965282747182
```



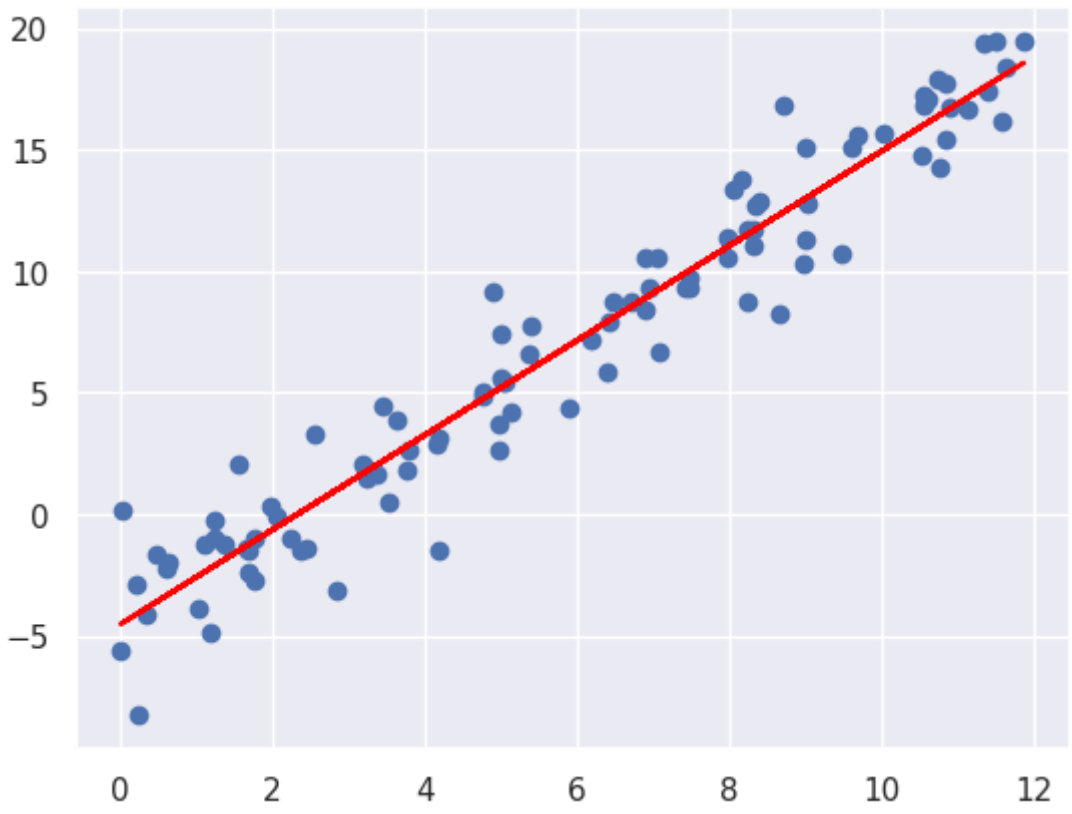
5. Next, we will implement an optimization based method to identify the best fit model parameters as opposed to directly estimate them from the data. In particular, consider the model error function

$$g(\beta_0, \beta_1) = \sum_{i=1}^{n} l(\epsilon_i) = \sum_{i=1}^{n} (y_i - f(x_i))^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \qquad (0.7)$$

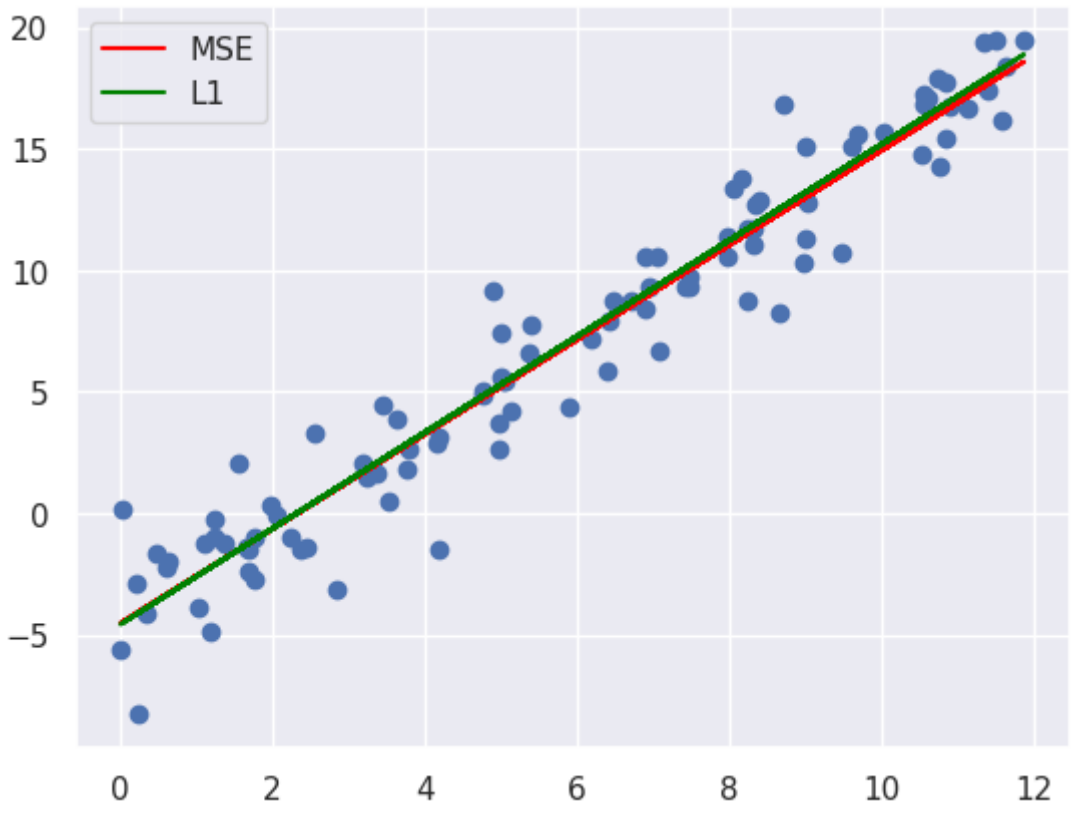
and use the Python function scipy.optimize.minimize to minimize $g(\beta_0, \beta_1)$ as a function of the two model parameters. How do the optimal points compare to the β_0 and β_1 estimates that you computed above? Next, extend your script to estimate these model parameters for other loss functions. In particular, consider the l_1 loss function that we describe in class $l(x) = |x|$ as well as define your own loss function. Overlay the best fit line with the $|x|$ loss function on the scatter plot of data and the mean squared error minimizing model that you previously displayed. Interpret the differences between these two models.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

def linear_func(beta, x):
    return beta[0] + beta[1] * x

def error_func(beta, x, y):
    y_pred = linear_func(beta, x)
    return np.sum((y - y_pred) ** 2)

res = minimize(error_func, [0, 0], args=(x, y))
beta0_opt, beta1_opt = res.x
```

```
plt.scatter(x, y)
plt.plot(x, beta0_opt + beta1_opt * x, color='red')
plt.show()
```




```
def l1_error_func(beta, x, y):
    y_pred = linear_func(beta, x)
    return np.sum(np.abs(y - y_pred))

res_l1 = minimize(l1_error_func, [0, 0], args=(x, y))
beta0_l1, beta1_l1 = res_l1.x

plt.scatter(x, y)
plt.plot(x, beta0_opt + beta1_opt * x, color='red', label='MSE')
plt.plot(x, beta0_l1 + beta1_l1 * x, color='green', label='L1')
plt.legend()
plt.show()
```



Implementation of this assignment on a house price dataset of one variable (the most important one) to see it real life)

```
import pandas as pd
data = pd.read_csv("House price dataset.csv")
data
```

	X	Y
0	7	208500
1	6	181500
2	7	223500
3	7	140000
4	8	250000
...	...	...
1455	6	175000
1456	6	210000
1457	7	266500
1458	5	142125
1459	5	147500

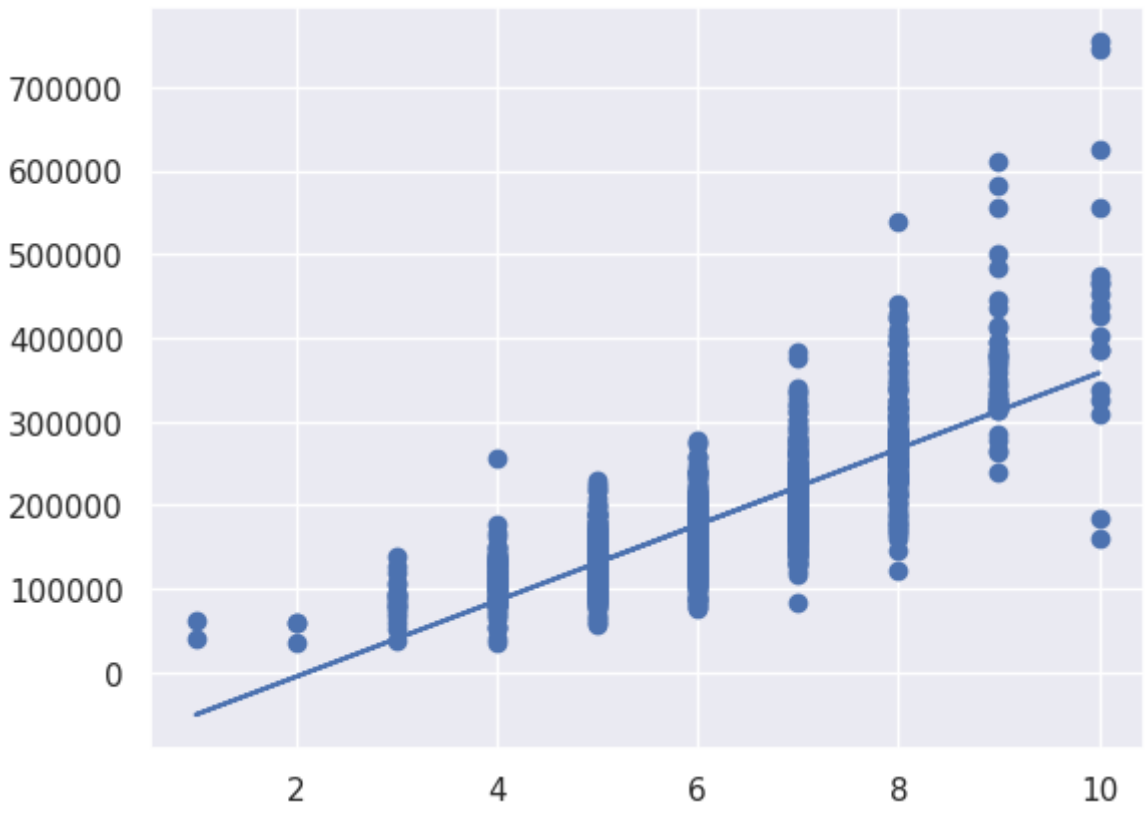
```
X = data["X"]
Y = data["Y"]
X = np.array(X)
Y = np.array(Y)
print(X)
print(Y)
```

```
[7 6 7 ... 7 5 5]
[208500 181500 223500 ... 266500 142125 147500]
```

```
X_mean = np.mean(X)
Y_mean = np.mean(Y)
beta1 = np.sum((X - X_mean) * (Y - Y_mean)) / np.sum((X - X_mean) ** 2)
beta0 = Y_mean - beta1 * X_mean
print(beta1)
print(beta0)
```

```
45435.8025930994
-96206.07951476038
```

```
Y_new = beta0 + beta1*X
plt.scatter(X, Y)
plt.plot(X, Y_new);
```




```
residuals = Y - Y_new
n = len(X)
var = np.sum(residuals**2) / (n - 2)
std_error_beta1 = np.sqrt(var / np.sum((X - X_mean) ** 2))
std_error_beta0 = np.sqrt(var * (1/n + X_mean**2 / np.sum((X - X_mean) ** 2)))
t_stat_beta1 = beta1 / (std_error_beta1)
RSE = np.sqrt(np.sum(residuals**2) / (n - 2))
R2 = 1 - (np.sum(residuals**2))/ (np.sum((Y - Y_mean)**2))
print(f"Standard error of beta1: {std_error_beta1}")
print(f"Standard error of beta0: {std_error_beta0}")
```

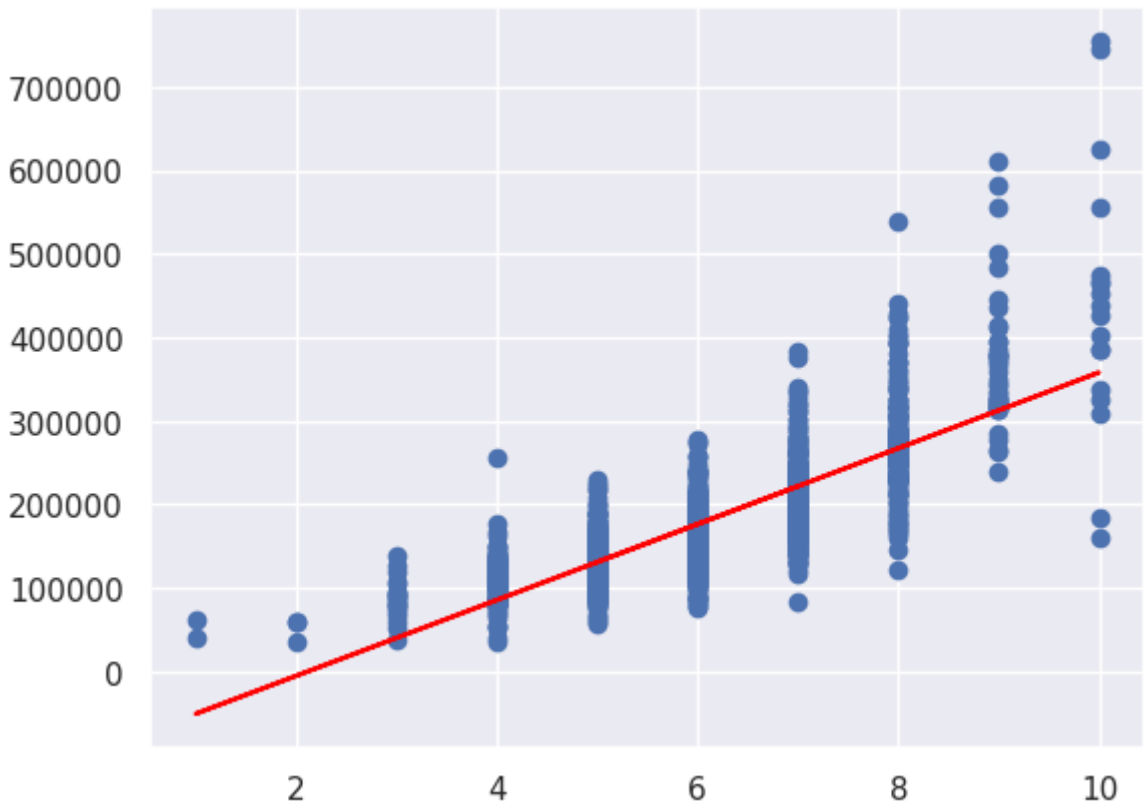
```
print(f"t-statistic for beta1: {t_stat_beta1}")
print(f"Residual standard error: {RSE}")
print(f"R-squared: {R2}")
```

```
Standard error of beta1: 920.4302464882172
Standard error of beta0: 5756.407385731069
t-statistic for beta1: 49.3636565795766
Residual standard error: 48622.7617959872
R-squared: 0.625651892462118
```

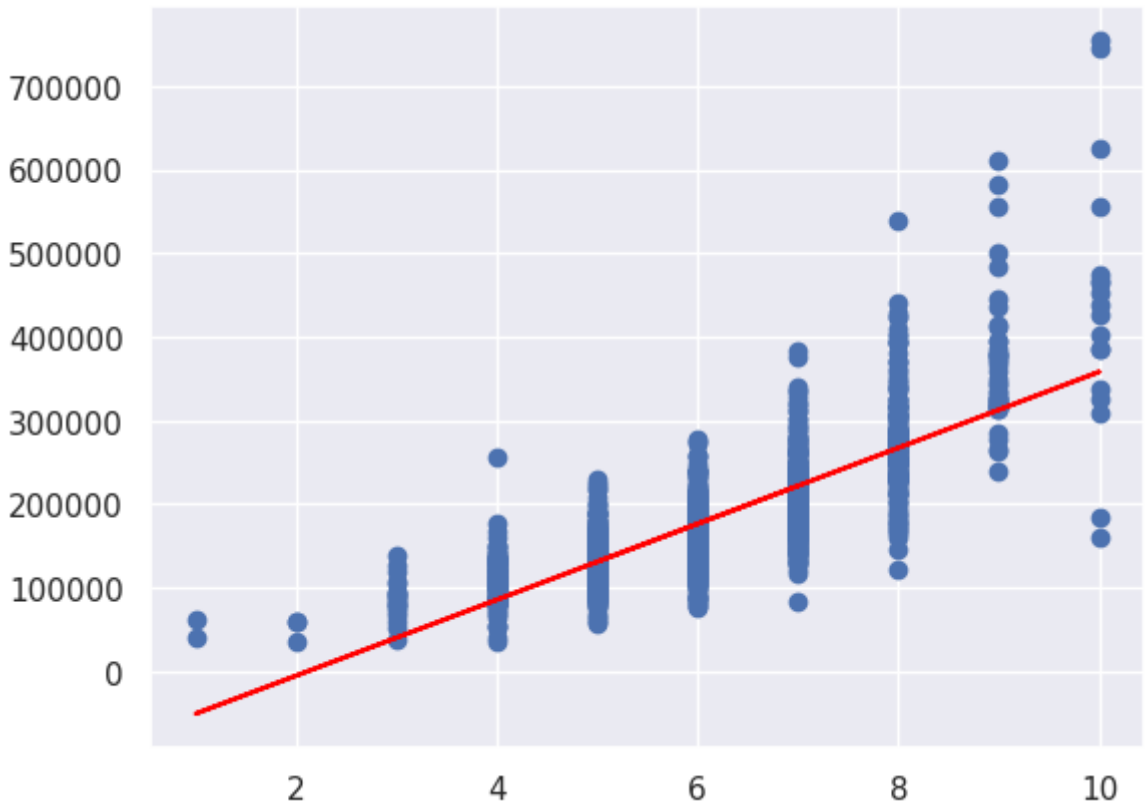
```
import statsmodels.api as sm


A = sm.add_constant(X)
model = sm.OLS(Y, A).fit()

print('beta0 =', model.params[0])
print('beta1 =', model.params[1])

print('SE(beta0) =', model.bse[0])
print('SE(beta1) =', model.bse[1])

print('t-stat(beta1) =', model.tvalues[1])

print('RSE =', np.sqrt(model.mse_resid))
print('R2 =', model.rsquared)
```

```
# plot data and regression line
plt.scatter(X, Y)
plt.plot(X, model.predict(A), color='red')
plt.show()
```

```
beta0 = -96206.07951475983
beta1 = 45435.8025930994
SE(beta0) = 5756.407385731059
SE(beta1) = 920.430246488218
t-stat(beta1) = 49.363656579576556
RSE = 48622.7617959872
R2 = 0.6256518924621179
```




```
from scipy.optimize import minimize

def linear_func(beta, X):
    return beta[0] + beta[1] * X

def error_func(beta, X, Y):
    Y_pred = linear_func(beta, X)
    return np.sum((Y - Y_pred) ** 2)

res = minimize(error_func, [0, 0], args=(X, Y))
beta0_opt, beta1_opt = res.x

plt.scatter(X, Y)
plt.plot(X, beta0_opt + beta1_opt * X, color='red')
plt.show()
```




```
def l1_error_func(beta, X, Y):
    Y_pred = linear_func(beta, X)
    return np.sum(np.abs(Y - Y_pred))

res_l1 = minimize(l1_error_func, [0, 0], args=(X, Y))
beta0_l1, beta1_l1 = res_l1.x

plt.scatter(X, Y)
plt.plot(X, beta0_opt + beta1_opt * X, color='red', label='MSE')
plt.plot(X, beta0_l1 + beta1_l1 * X, color='green', label='L1')
plt.legend()
plt.show()
```