Linear Regression

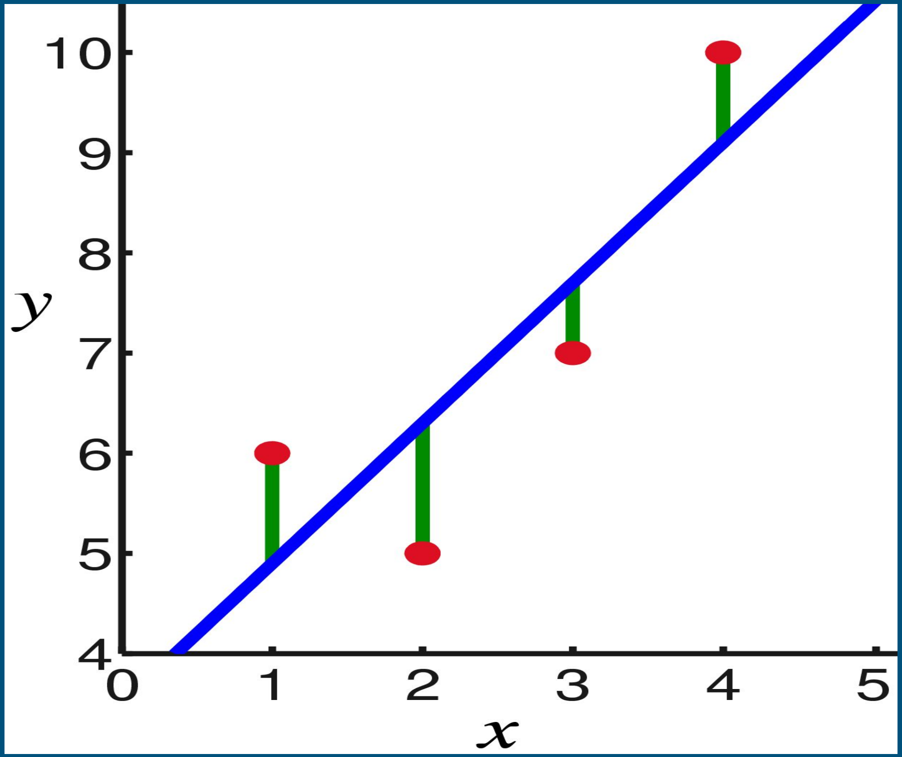
Colin Bennie, Caleb Hamblen, Sissi Shen, Justin Yang

How to fit a Simple Linear Model

- Goal is to estimate:

  yi = Bo + B1xi + εi

- Want to minimize (yi - yi_hat)



Simple Linear Model Assumptions

- ei random so that the following are true:
    - $E(\varepsilon i) = 0$
    - $var(\varepsilon i) = \boldsymbol{\sigma}^2$
    - $Cov(\varepsilon i, \varepsilon j) = 0$
    - $Ei \sim N(0, \boldsymbol{\sigma}^2)$
- Y needs to be linear to Bo and B1

Results

- $E(yi) = Bo+B1xi$: ***this is the regression function***
- $var(yi) = \boldsymbol{\sigma}$^2
- Yi|xi=x ~N(Bo+B1xi,$\boldsymbol{\sigma}$^2)
- Bo_hat = y_bar - B1_hat(x_bar)
- B1_hat = ∑(xi - x_bar)∑(yi - y_bar) / ∑(xi - x_bar)^2
- Yi_hat = Bo_hat + B1_hat(xi)

How to implement in Python: Sample Data

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
```

In [2]:
```
df = pd.read_csv('Automobile_data.csv')
df.head()
```

Out[2]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.4	10.0	102
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.4	8.0	115

5 rows × 26 columns

In [3]:
```
df.columns
```

Out[3]:
```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
```

How to implement in Python: Model Fitting

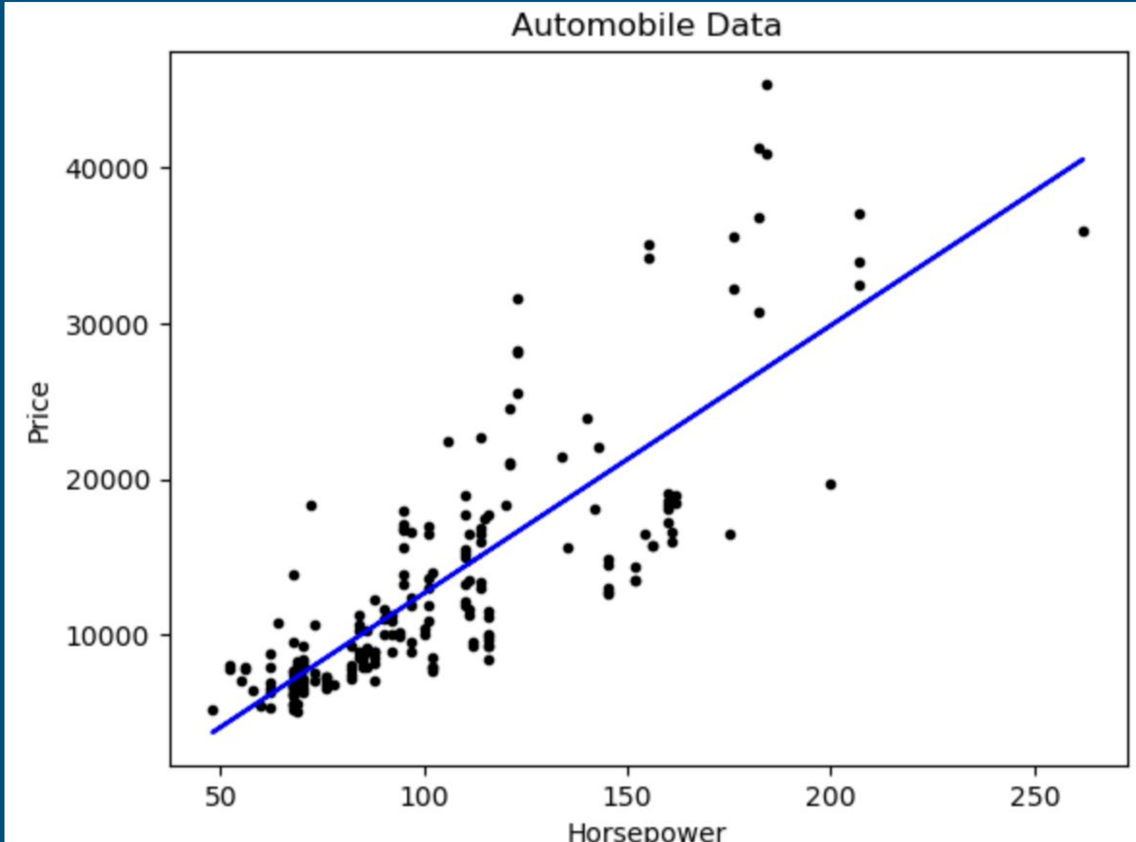
```
In [8]:
import statsmodels.formula.api as smf

model = smf.ols("price ~  horsepower", data = dfnew).fit()
b = model.params['Intercept']
m = model.params['horsepower']
print(f'intercept={b}, slope={m}')
```

```
intercept=-4562.174995667492, slope=172.20625117310604
```

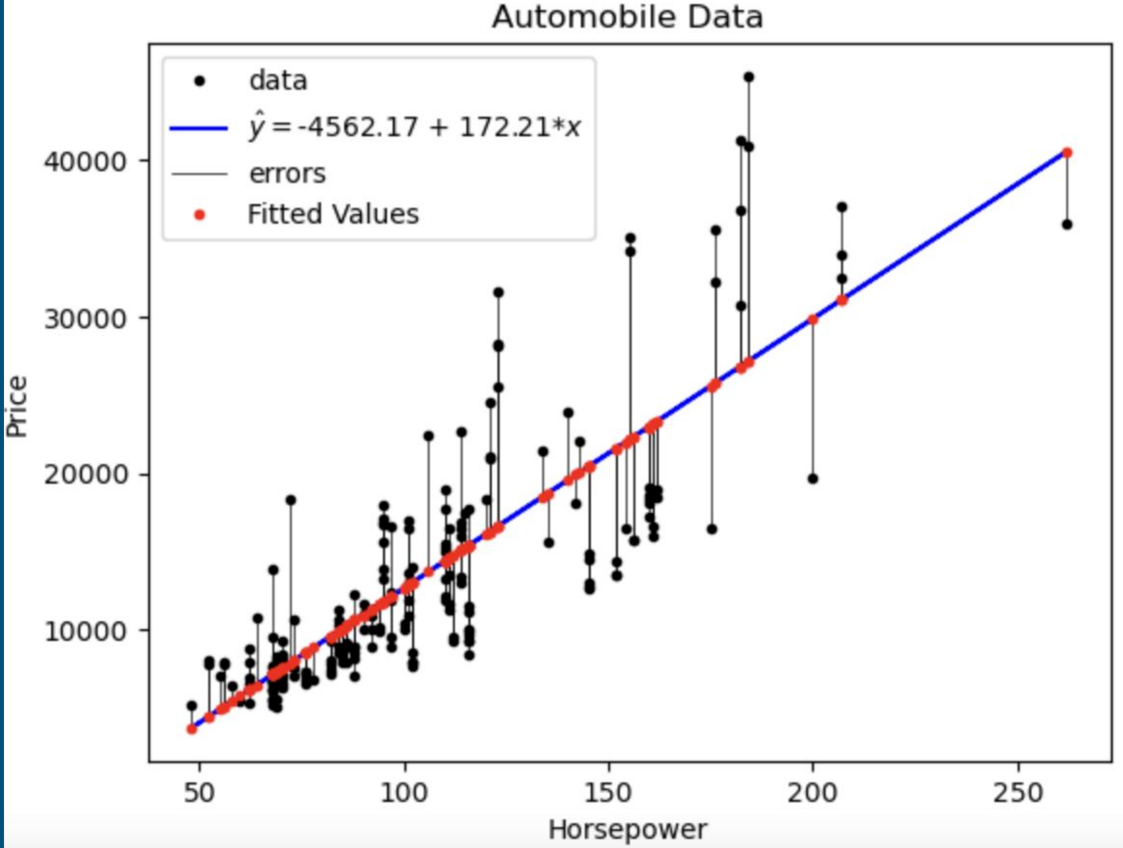
```
In [7]:
from sklearn import linear_model

model = linear_model.LinearRegression()
model.fit(dfnew[['horsepower']], dfnew['price'])
b = model.intercept_
w = model.coef_[0]
print(f'intercept={b}, slope={w}')
```

```
intercept=-4562.1749956674885, slope=172.20625117310607
```

How the model looks



How the model looks: residuals



Predicting using Model

In [11]:

```python
# Sklearn
X = dfnew[['horsepower']]
y = dfnew['price']
model = linear_model.LinearRegression()
model.fit(X, y)

XX = [[1000]] # Horsepower of a Formula 1 Car
model.predict(XX)[0]
```

Out[11]: 167644.0761774386

In [12]:

```python
# Statsmodels
model = smf.ols("price ~ horsepower", data = dfnew).fit()

XX = np.array([1000])# Horsepower of a Formula 1 Car
prediction = model.params[0] + model.params[1] * XX
prediction[0]
```

Out[12]: 167644.07617743855

Model Diagnosis

In [14]:
```
model.summary()
```

Out[14]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.657
Model:	OLS	**Adj. R-squared:**	0.655
Method:	Least Squares	**F-statistic:**	377.3
Date:	Fri, 29 Sep 2023	**Prob (F-statistic):**	1.19e-47
Time:	16:40:42	**Log-Likelihood:**	-1963.3
No. Observations:	199	**AIC:**	3931.
Df Residuals:	197	**BIC:**	3937.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-4562.1750	974.995	-4.679	0.000	-6484.943	-2639.407
horsepower	172.2063	8.866	19.424	0.000	154.722	189.690

Omnibus:	38.494	**Durbin-Watson:**	0.749
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	64.496
Skew:	1.013	**Prob(JB):**	9.89e-15
Kurtosis:	4.916	**Cond. No.**	323.

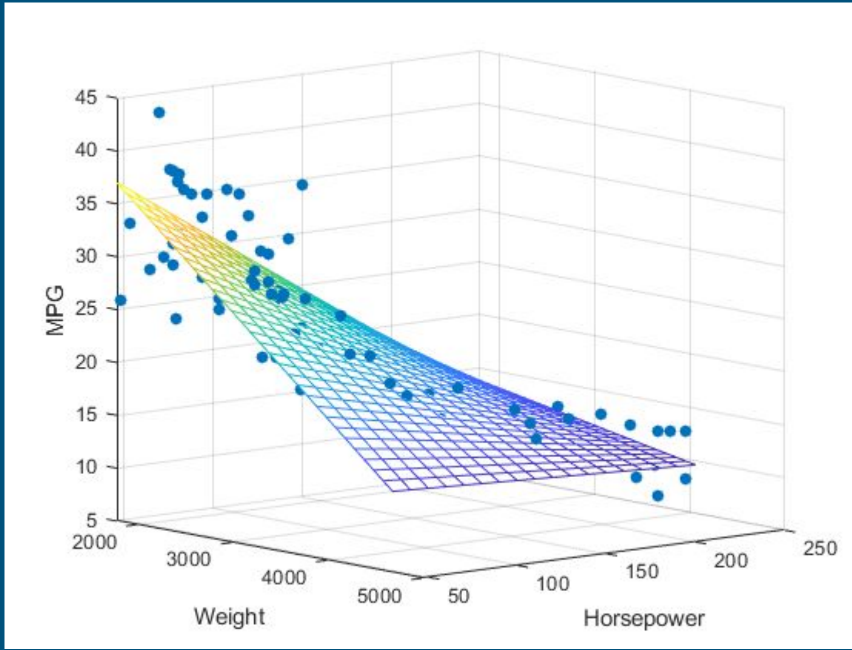
In [15]:
```
import statsmodels.api as sm

aov_table = sm.stats.anova_lm(model, typ=1)
aov_table
```

Out[15]:

	df	sum_sq	mean_sq	F	PR(>F)
horsepower	1.0	8.280790e+09	8.280790e+09	377.283543	1.189128e-47
Residual	197.0	4.323845e+09	2.194845e+07	NaN	NaN

How to fit a Multiple Linear Regression

- Uses two or more independent variables
- Fitting a plane instead of a line
- Y = b0 + b1X1 + b2X2 + ... + bnXn + ε
- Still want to minimize (Yi - Yi_hat)



Multiple Linear Model Assumptions

- **Linearity**: The relationship between the dependent and independent variables is linear.
- **Independence and Normality of Errors**: The error terms (ε) are independent of each other and follow a normal distribution.
- **Homoscedasticity**: The variance of the errors is constant across all levels of independent variables.
- **No Multicollinearity**: The independent variables are not highly correlated with each other.

Results

- Regression Function: E(Yi) = bo + b1Xi1 + b2Xi2 + ... + bn*Xin
- Coefficient Estimates:
  - Bo_hat = y_bar - B1_hat(x_bar1) - B2_hat(x_bar2) - ... - Bn_hat(x_barn)
  - Bn_hat = ∑(xi_n - x_bar_n)∑(yi - y_bar) / ∑(xi_n - x_bar_n)^2
  - B1_hat = ∑(xi1 - x_bar1)∑(yi - y_bar) / ∑(xi1 - x_bar1)^2
- Fitted Values using estimated coefficients:
  - Yi_hat = Bo_hat + B1_hatX1i + B2_hatX2i + ... + Bn_hat*Xni

Multiple Linear Regression: Python

```python
# Load in data for regression analysis
mlr_example = pd.read_csv('Automobile_data.csv')

# Identify target variable and potential predictors
print(mlr_example.columns)
```

✓ 0.0s Python

```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
```

```python
# Create regression model using smf.ols
model = smf.ols('price ~ engine_size + horsepower + highway_mpg + fuel_type', data=model_data).fit()
model.summary()
```

✓ 0.0s Python

MLR: Model Diagnosis

- R^2 and Adj-R^2
- Predictors all have their own coefficients and t test results

OLS Regression Results

Dep. Variable:	price	R-squared:	0.812
Model:	OLS	Adj. R-squared:	0.808
Method:	Least Squares	F-statistic:	209.7
Date:	Sat, 30 Sep 2023	Prob (F-statistic):	2.84e-69
Time:	15:26:07	Log-Likelihood:	-1903.4
No. Observations:	199	AIC:	3817.
Df Residuals:	194	BIC:	3833.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	3071.2513	3175.980	0.967	0.335	-3192.632	9335.134
fuel_type[T.gas]	-3882.1745	910.199	-4.265	0.000	-5677.331	-2087.018
engine_size	102.6287	11.343	9.048	0.000	80.258	124.999
horsepower	58.0126	14.923	3.888	0.000	28.581	87.444
highway_mpg	-174.3590	61.706	-2.826	0.005	-296.059	-52.659

Omnibus:	11.078	Durbin-Watson:	0.841
Prob(Omnibus):	0.004	Jarque-Bera (JB):	20.221
Skew:	0.243	Prob(JB):	4.07e-05
Kurtosis:	4.484	Cond. No.	2.25e+03

MLR: Model Diagnosis

- Sequential vs. Partial F Tests

```
1  # Sequential F Test on regression model
2
3  sequential_aov_table = sm.stats.anova_lm(model, typ=1)
4  sequential_aov_table
```

✓ 0.0s

	df	sum_sq	mean_sq	F	PR(>F)
fuel_type	1.0	1.496960e+08	1.496960e+08	12.267655	5.720085e-04
engine_size	1.0	9.503851e+09	9.503851e+09	778.844709	7.616789e-70
horsepower	1.0	4.863750e+08	4.863750e+08	39.858640	1.818820e-09
highway_mpg	1.0	9.742880e+07	9.742880e+07	7.984332	5.212118e-03
Residual	194.0	2.367285e+09	1.220250e+07	NaN	NaN

Reduced Model: ***price ~ fuel_type + engine_size***

Full Model: ***price ~ fuel_type + engine_size + horsepower***

```
1  # Partial F Test on regression model
2
3  partial_aov_table = sm.stats.anova_lm(model, typ=2)
4  partial_aov_table
```

[43] ✓ 0.0s

...

	sum_sq	df	F	PR(>F)
fuel_type	2.219861e+08	1.0	18.191857	3.118291e-05
engine_size	9.989917e+08	1.0	81.867806	1.530876e-16
horsepower	1.844180e+08	1.0	15.113133	1.389653e-04
highway_mpg	9.742880e+07	1.0	7.984332	5.212118e-03
Residual	2.367285e+09	194.0	NaN	NaN

Reduced Model: ***price ~ fuel_type + engine_size + highway_mpg***

Full Model: ***price ~ fuel_type + engine_size + highway_mpg + horsepower***

MLR: Multicollinearity and VIF Check

```python
# additional imports needed to check for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
from patsy import dmatrices

# Calculate the VIF for each predictor variable included in model
vif_y, vif_x = dmatrices('price ~ engine_size + horsepower + highway_mpg + fuel_type',
                         data=model_data, return_type='dataframe')

vif = pd.DataFrame()
vif['VIF Factor'] = [variance_inflation_factor(vif_x.values, i) for i in range(vif_x.shape[1])]
vif['Features'] = vif_x.columns
print(vif)
```

✓ 0.0s

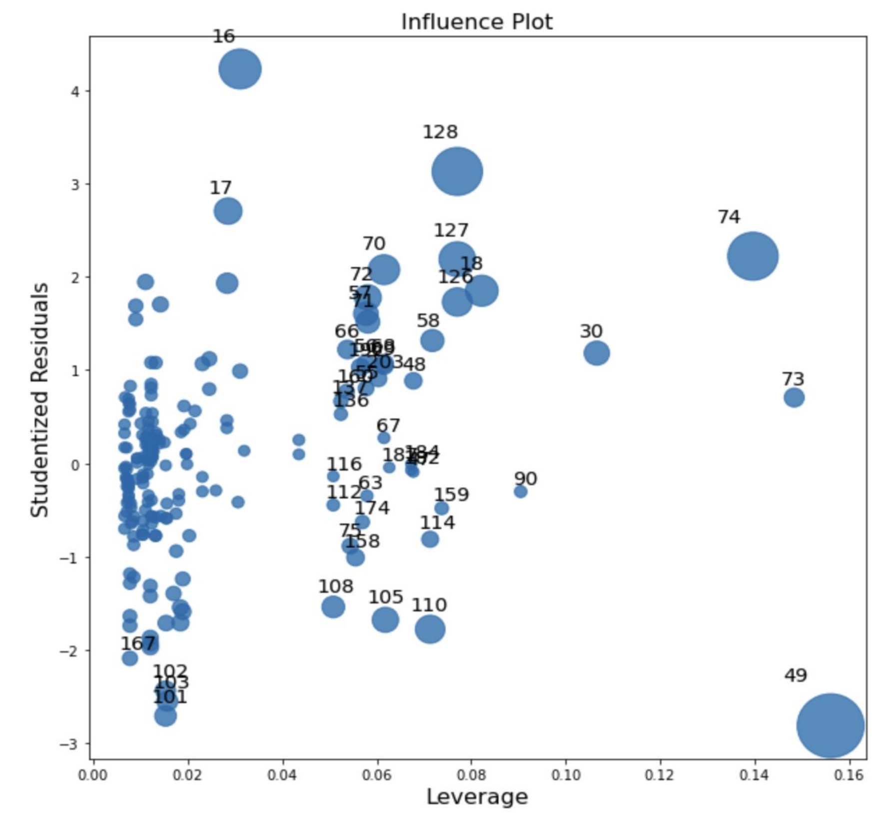
```
   VIF Factor          Features
0  164.497754         Intercept
1    1.221390  fuel_type[T.gas]
2    3.639274       engine_size
3    5.095827        horsepower
4    2.898497       highway_mpg
```

Link to Github with Code

https://github.com/cwbennie/comms_code_demo23

MLR: Influential Points

- Leverage
- Standardized Residuals
- Cook’s Distance



How the model looks: Code to plot

In [9]:

```
import matplotlib.pyplot as plt

X = dfnew[['horsepower']]
y = dfnew['price']
model = linear_model.LinearRegression()
model.fit(X, y)
b = model.intercept_
w = model.coef_[0]

x = dfnew.horsepower # we need a 1D array for plotting (and a 2D array for .fit() above)
plt.plot(x, y, '.', color='black', label='data')
plt.title('Automobile Data')
plt.xlabel('Horsepower')
plt.ylabel('Price')

y_hat = model.predict(X) # equivalent to y_hat = w * X[:, 0] + b
plt.plot(x, y_hat, color='red',
         label=f'$\\hat{{y}}=${round(b, 2)} + ({round(w, 2)})$x$')

plt.show()
```