**Ordinary Least Squares (OLS):**

[**https://www.youtube.com/watch?v=ORyfPJypKuU&t=2s**](https://www.youtube.com/watch?v=ORyfPJypKuU&t=2s)

[**https://www.geeksforgeeks.org/ordinary-least-squares-ols-using-statsmodels/**](https://www.geeksforgeeks.org/ordinary-least-squares-ols-using-statsmodels/)

[**https://labex.io/tutorials/python-ordinary-least-squares-in-python-300247**](https://labex.io/tutorials/python-ordinary-least-squares-in-python-300247)

[**https://www.geeksforgeeks.org/ordinary-least-squares-and-ridge-regression-variance-in-scikit-learn/**](https://www.geeksforgeeks.org/ordinary-least-squares-and-ridge-regression-variance-in-scikit-learn/)

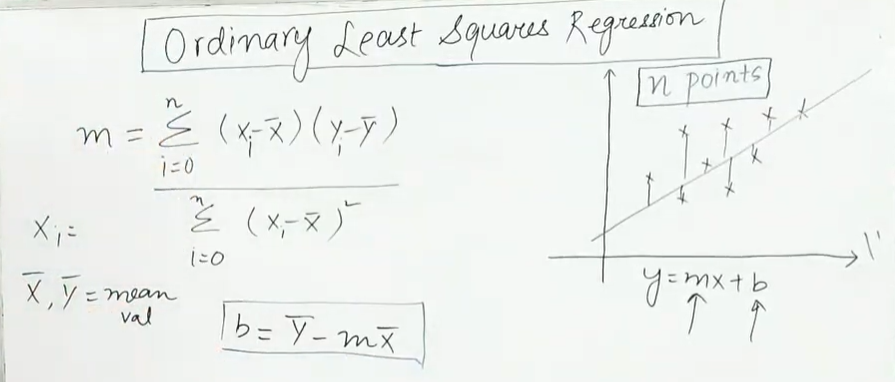
Ordinary Least Squares (OLS) is a widely used statistical method for estimating the parameters of a linear regression model. It minimizes the sum of squared residuals between observed and predicted values.

A[**linear regression model**](https://www.geeksforgeeks.org/ml-linear-regression/)establishes the relationship between a dependent variable (*y*) and one or more independent variables (*x*):

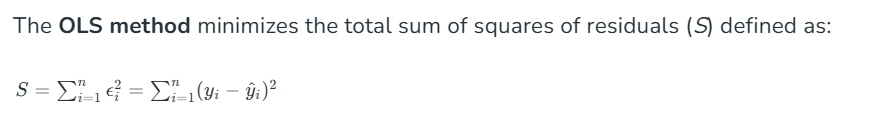
y^ = θ1x + θ0​

Where:

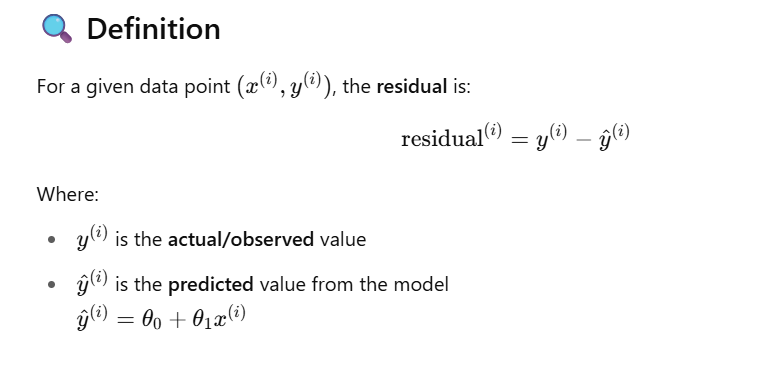
* y^: Predicted value of *y*
* θ1​: Slope of the line (coefficient of *x*)
* θ0​: Intercept (value of *y* when *x*=0)

****

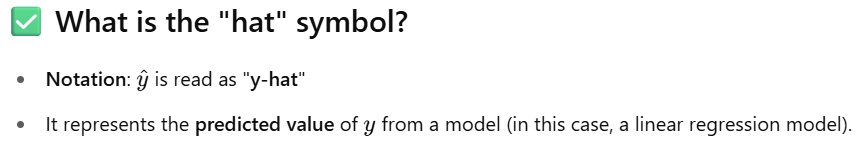
**M = (covariance of x and y) / (variance of x)**

****

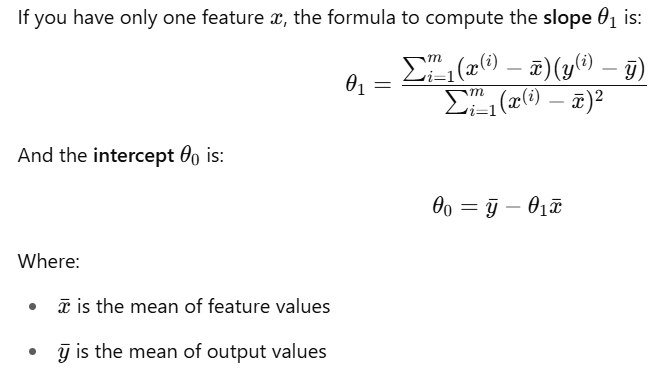
**What is Residual means: -** residual represents the difference between the actual value Y and the predicted value Y^ for a data point.







In scikit-learn, the **Linear Regression** model computes parameters θ0​ (intercept) and θ1​ (coefficient) using the **Ordinary Least Squares (OLS)** closed-form solution—not gradient descent by default.



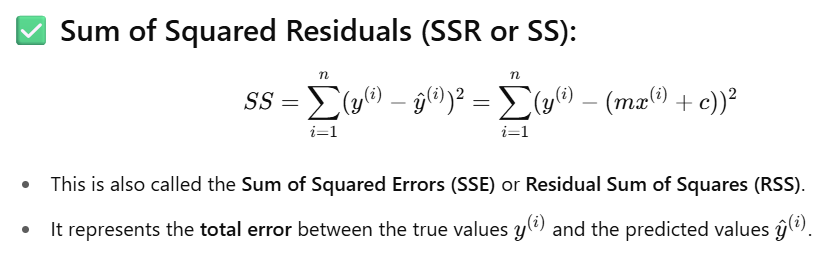
from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(X, y)

θ1= model.coef\_[0] # slope (coefficient)

θ0= model.intercept\_ # intercept



**Goal of Linear Regression:**

To **find the values of m and c** (or θ1​ and θ0​) such that the **sum of squared errors (SS)** is **minimized**.

This best-fitting line minimizes the squared vertical distance (residuals) between the data points and the line.

The **best fit line** is the (y^​ = θ0 ​+ θ1​x) **for which the sum of squared residuals is minimum**.

Linear regression code example using OLS actually sklearn by default uses OLS not gradient decent

And it won’t support p-value for result analysis

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score

# 1. Load California housing dataset

data = fetch\_california\_housing()

df = pd.DataFrame(data=data.data, columns=data.feature\_names)

df['MedHouseVal'] = data.target

# 2. Use only 1 feature: Median Income

X = df[['MedInc']] # Independent variable (feature)

y = df['MedHouseVal'] # Dependent variable (target)

# Show initial data

print("Sample Data:")

print(df[['MedInc', 'MedHouseVal']].head())

# 3. Split into Train and Test sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 4. Create and Train Linear Regression model

model = LinearRegression() # Uses OLS by default

# 3. Train the model (fit OLS line)

model.fit(X\_train, y\_train)

# 5. Predict on Test set

y\_pred = model.predict(X\_test)

# 6. Model Parameters

print("\nModel Parameters:")

print(f"Intercept (θ₀): {model.intercept\_:.4f}")

print(f"Coefficient (θ₁): {model.coef\_[0]:.4f}")

# 7. Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("\nModel Evaluation:")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Root Mean Squared Error (RMSE): {np.sqrt(mse):.4f}")

print(f"R² Score: {r2:.4f}")

# 8. Visualization

plt.figure(figsize=(10, 6))

plt.scatter(X\_test, y\_test, color='blue', label='Actual Values', alpha=0.4)

plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Predicted Regression Line')

plt.xlabel('Median Income')

plt.ylabel('Median House Value')

plt.title('OLS Linear Regression: Median Income vs House Value')

plt.legend()

plt.grid(True)

plt.show()

📈 Sample Output (You’ll See Something Like This):

yaml

Copy

Edit

Sample Data:

MedInc MedHouseVal

0 8.3252 4.526

1 8.3014 3.585

2 7.2574 3.521

3 5.6431 3.413

4 3.8462 3.422

Model Parameters:

Intercept (θ₀): 0.4189

Coefficient (θ₁): 0.4263

Model Evaluation:

Mean Squared Error (MSE): 0.5336

Root Mean Squared Error (RMSE): 0.7302

R² Score: 0.4767

🔍 Analysis

✔️ Coefficient Interpretation:

For every 1 unit increase in Median Income, the predicted House Value increases by about 0.426 (in $100,000s).

So: $10,000 income increase ⇒ ~$42,600 increase in house value.

✔️ Intercept:

When Median Income = 0, the model predicts a baseline house value of $41,890.

✔️ R² Score:

R² ≈ 0.48 → About 48% of the variation in house prices is explained by median income alone.

This is decent for a single-feature model but suggests we can improve accuracy with more features.

✔️ RMSE:

RMSE ≈ 0.73 → On average, predictions deviate from actual values by about $73,000.

**Step-by-Step OLS Linear Regression using statsmodels**

We’ll use the **California Housing dataset** with **MedInc** as the only predictor (single feature).

**✅ Code Example**

python

CopyEdit

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

import statsmodels.api as sm

# 1. Load dataset

data = fetch\_california\_housing()

df = pd.DataFrame(data.data, columns=data.feature\_names)

df['MedHouseVal'] = data.target

# 2. Choose one feature for regression: Median Income

X = df[['MedInc']]

y = df['MedHouseVal']

print("Sample Data:")

print(df[['MedInc', 'MedHouseVal']].head())

# 3. Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 4. Add constant term for intercept in statsmodels

X\_train\_const = sm.add\_constant(X\_train)

X\_test\_const = sm.add\_constant(X\_test)

# 5. Fit OLS model

model = sm.OLS(y\_train, X\_train\_const)

results = model.fit()

# 6. Print full statistical summary

print("\nRegression Summary:")

print(results.summary())

# 7. Make predictions

y\_pred = results.predict(X\_test\_const)

# 8. Evaluate model

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = results.rsquared

print("\nModel Evaluation:")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print(f"R² Score on Train Set (from statsmodels): {r2:.4f}")

# 9. Plot predictions vs actual

plt.figure(figsize=(10, 6))

sns.scatterplot(x=X\_test['MedInc'], y=y\_test, label='Actual', alpha=0.5)

sns.lineplot(x=X\_test['MedInc'], y=y\_pred, color='red', label='Predicted')

plt.xlabel("Median Income")

plt.ylabel("Median House Value")

plt.title("OLS Regression Line: MedInc vs MedHouseVal")

plt.grid(True)

plt.legend()

plt.show()

**📘 Sample Output Explanation**

**✅ results.summary() includes:**

* **R-squared** and **Adj. R-squared**: How well model explains the variance.
* **coef**: Intercept and slope values.
* **P>|t|**: p-values indicating statistical significance.
* **[0.025, 0.975]**: 95% confidence interval for coefficients.
* **F-statistic & Prob(F-statistic)**: Overall model significance.
* **Durbin-Watson**: Test for autocorrelation in residuals.

Example:

markdown

CopyEdit

==============================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------

const 0.4171 0.014 29.343 0.000 0.390 0.444

MedInc 0.4256 0.005 81.096 0.000 0.416 0.435

==============================================================================

**✅ Interpretation**

| **Term** | **Meaning** |
| --- | --- |
| **Intercept (θ₀)** | When MedInc = 0, predicted value is ~$41,710 |
| **MedInc coef (θ₁)** | Each unit increase in income (~$10k) raises home value by ~$42,560 |
| \*\*p-value (`P> | t |
| **R² Score** | ~0.48 means 48% of variance in house value is explained by income |
| **RMSE** | ~0.73 → avg prediction error is ~$73,000 |
| **F-statistic** | Tests if model explains a significant portion of the variation |

**In multidimensional data (target is depends on multiple features like x1, x2, x3) so equation will be y = ax1+bx2+cx3+d**

**So here a,b,c represents the weightage of feature how much % it has contribution to get target y**