22AIE205

INTRODUCTION TO PYTHON

END SEM PROJECT

- BY TEAM 11 BATCH - A

Try Pitch

Quarter 3

PYTHON PROJECT

HOUSE PRICE PREDICTION USING NON-LINEAR REGRESSION

Try Pitch

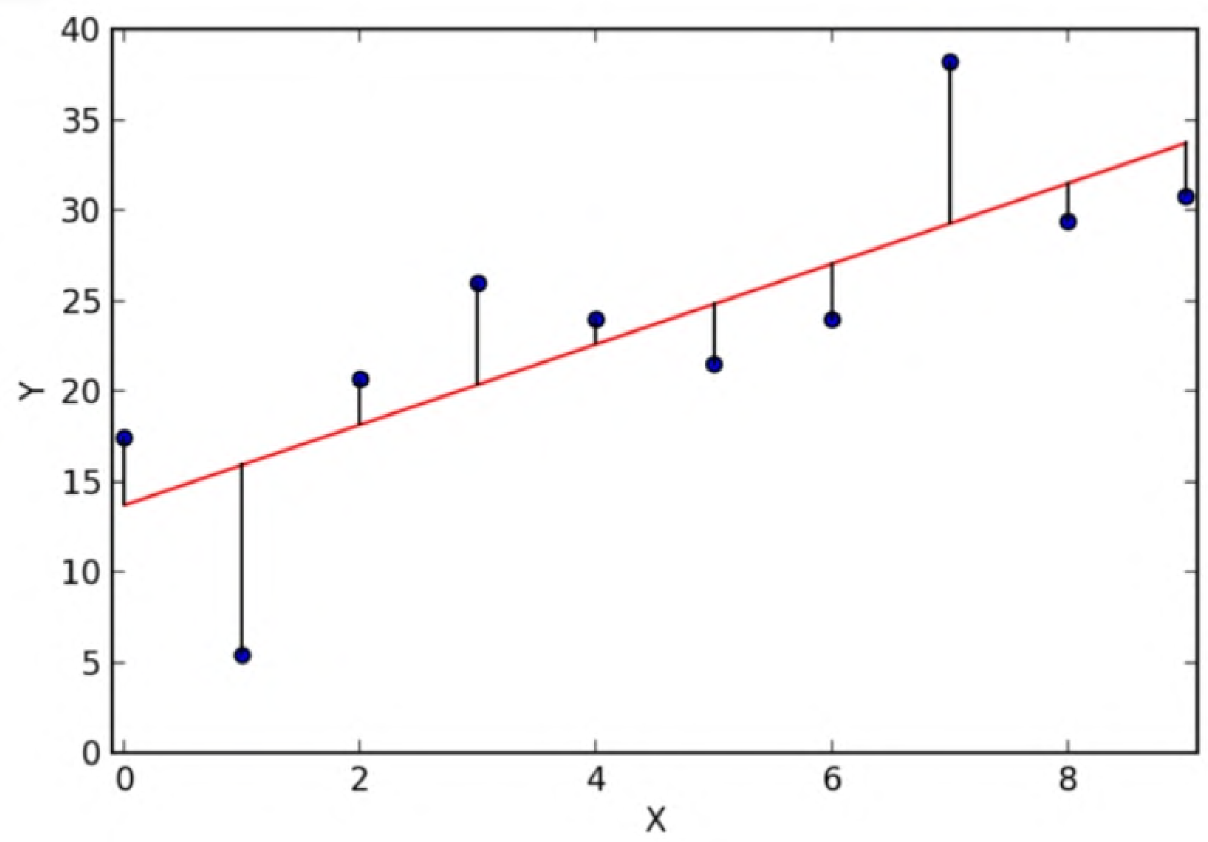
AGENDA

- ☐ Regression
- ☐ Types of Regression
- ☐ Gradient Descent
- ☐ Logic of Code
- ☐ Working of Code

Try Pitch

REGRESSION

- *Regression is a statistical technique used for modeling and analyzing the relationships between a dependent variable (target) and one or more independent variables (features).*
- *Regression is widely used in various fields, including economics, finance, biology, engineering, and machine learning. It's employed for tasks such as predicting stock prices, housing prices, sales, and many others.*



Regression Line Formula: $y = a + bx + u$

Multiple Regression Line Formula: $y = a + b_1x_1 + b_2x_2 + b_3x_3 + \ldots + b_tx_t + u$

Try Pitch

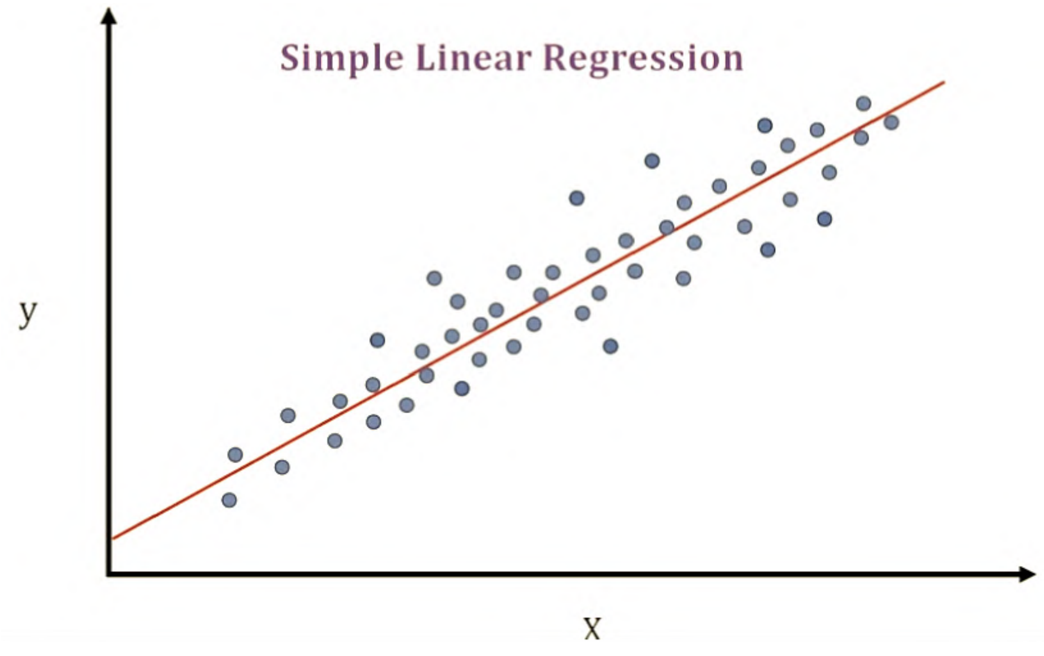
TYPES OF REGRESSION

1. LINEAR REGRESSION

Linear regression is a statistical method used to model the relationship between a dependent variable (also known as the response variable) and one or more independent variables (predictor variables or features)

The general form of a simple linear regression equation with one independent variable is:

$Y=\beta 0+\beta 1*X+\epsilon$



Try Pitch

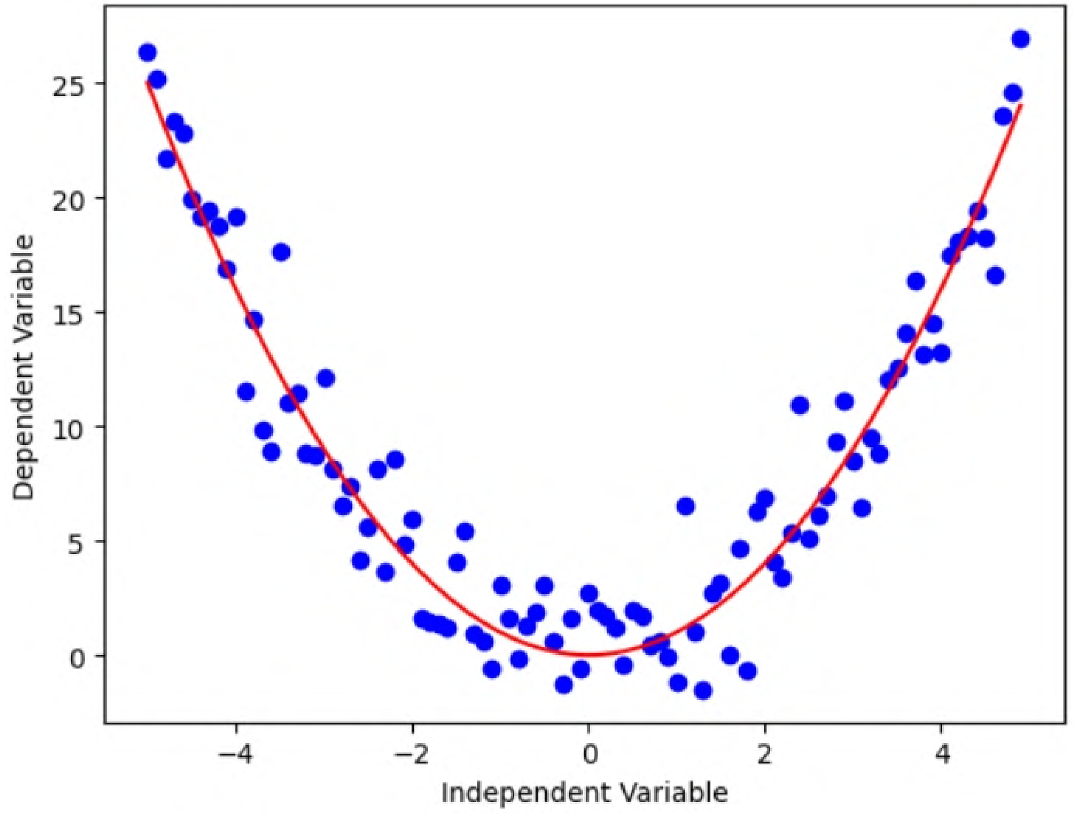
TYPES OF REGRESSION

2. NON - LINEAR REGRESSION

Non-linear regression is a statistical modeling technique used when the relationship between the dependent variable and the independent variable(s) cannot be adequately represented by a linear equation. In non-linear regression, the functional form of the model is more flexible, allowing for more complex relationships, including curves, exponentials, logarithms, and other non-linear patterns.

The general form of a simple Non - linear regression equation with one independent variable is:

$$Y=f(\beta0+\beta1*X1+\beta2*X2+...+\beta n*Xn)+\epsilon$$



Try Pitch

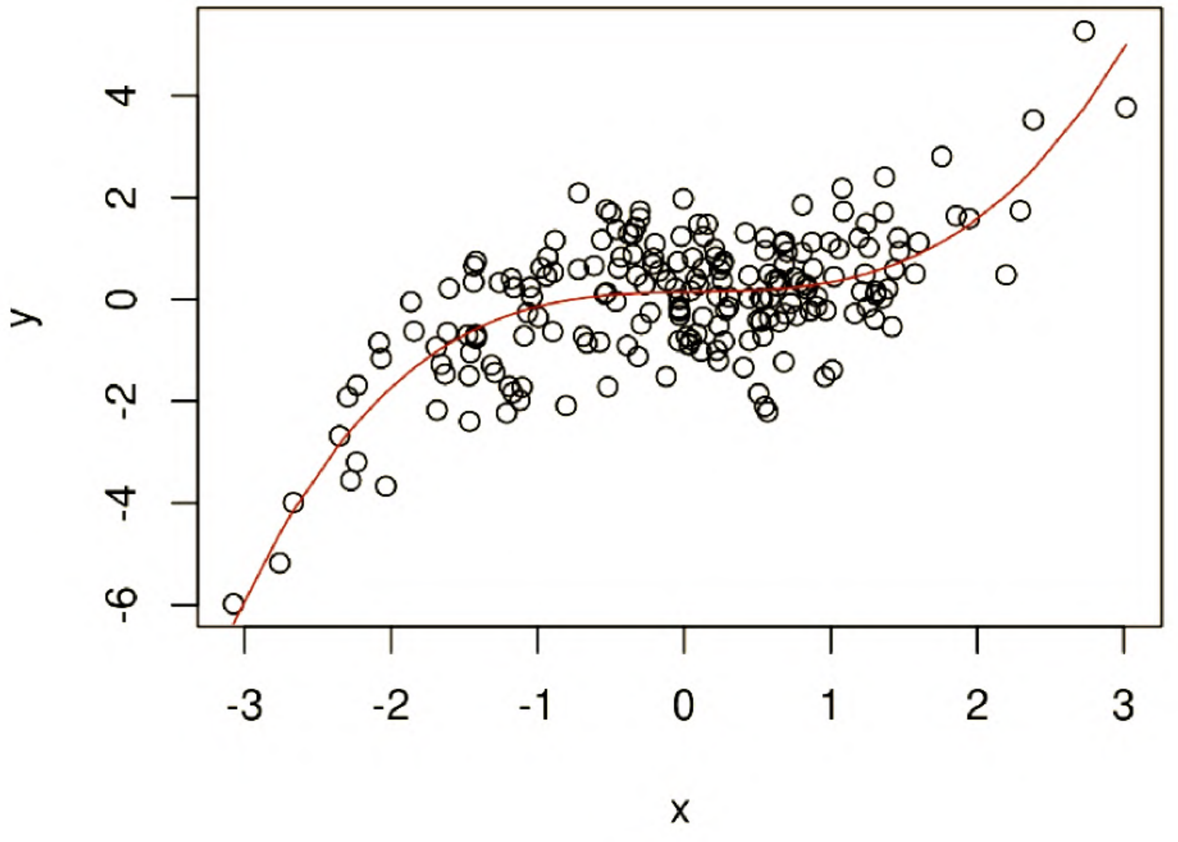
TYPES OF REGRESSION

3. POLYNOMIAL REGRESSION

Polynomial regression is a type of non-linear regression that extends the concept of linear regression by allowing the relationship between the independent variable(s) and the dependent variable to be modeled as an nth-degree polynomial. In polynomial regression, the regression equation takes the form:

The general form of a simple Polynomial regression equation with one independent variable is:

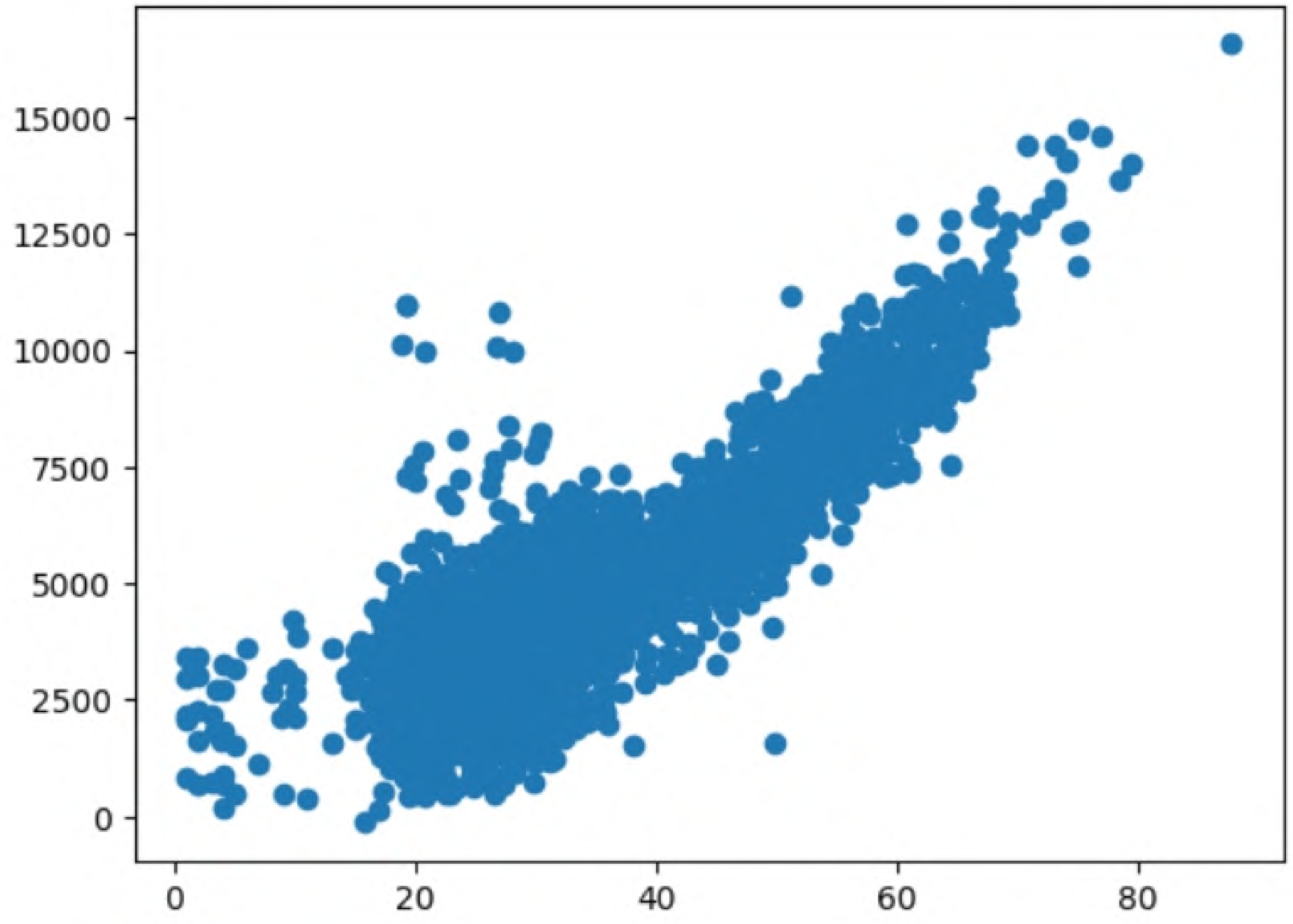
Y=β0+β1X^n+βX^(n-1)+…………….βX+ϵ



Try Pitch

PLOTTING OF OUR TRAINING DATA

WE WILL BE PLOTTING THE PRICES OF HOUSES WITH RESPECT TO SQUARE METERS OF THAT HOUSE



Try Pitch

COST FUNCTION

-
- For our Regression model, this function will find each and every error that occoured by our regression line with the original training set

Cost Function

$$J(\Theta_0, \Theta_1) = \frac{1}{2m} \sum_{i=1}^{m} [\underbrace{h_\Theta(x_i)}_{\text{Predicted Value}} - \underbrace{y_i}_{\text{True Value}}]^2$$

Try Pitch

GRADIENT DESCENT

- Gradient descent is an optimization algorithm commonly used in machine learning and mathematical optimization. Its primary purpose is to minimize a function iteratively by adjusting its parameters. It's widely employed in training machine learning models, including linear regression, neural networks, and other optimization problems.

- In this Regression training model we will be intent to reduce the Cost function So for that we will take the partial derivative of that Cost function with respective each parameter that we want to update

Try Pitch

WORKING OF GRADIENT DESCENT

- Calculate the gradient of the cost function with respect to each parameter. The gradient is a vector that points in the direction of the steepest increase in the cost function.

Cost Function

$$J\left(\Theta_0, \Theta_1\right) = \frac{1}{2m}\sum_{i=1}^{m}[\underbrace{h_\Theta(x_i)}_{\text{Predicted Value}} - \underbrace{y_i}_{\text{True Value}}]^2$$

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$\theta_0, \theta_1$$

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

Derivatives:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

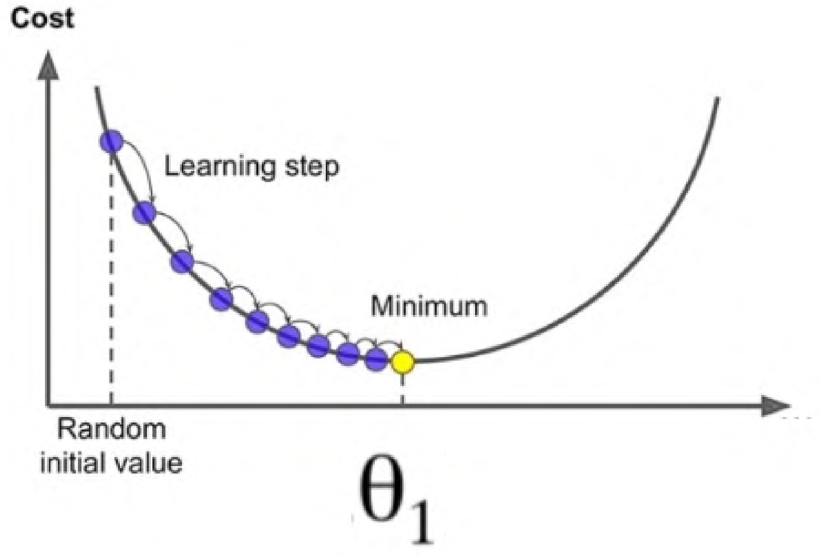
Try Pitch

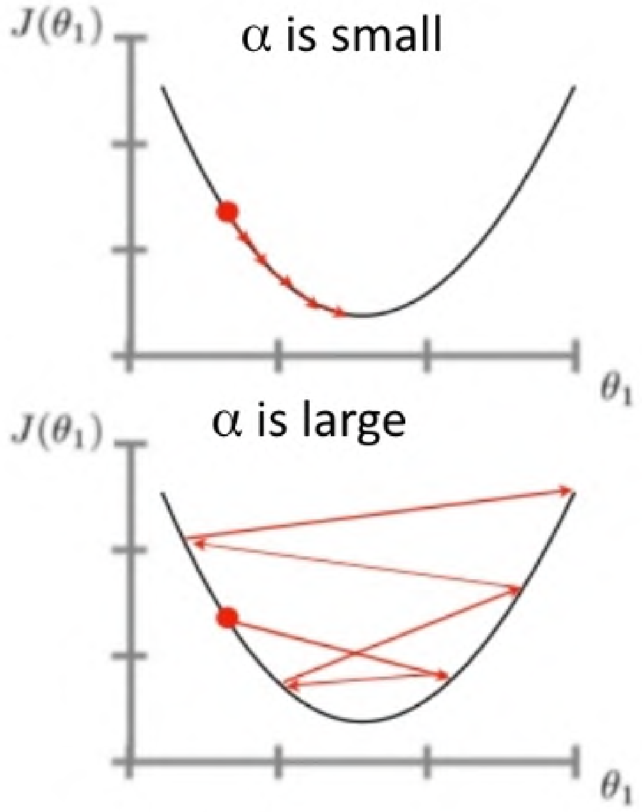
WORKING OF GRADIENT DESCENT

- Each coefficient is updated in the opposite direction of the gradient. The update rule for each coefficient

$$
\begin{aligned}
&\text{repeat until convergence } \{\\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)\\
&\qquad (\text{for } j = 1 \text{ and } j = 0)\\
&\}
\end{aligned}
$$

Here, α is the learning rate, controlling the size of the step taken during each iteration.





Try Pitch

OUR MAIN OJECTIVE

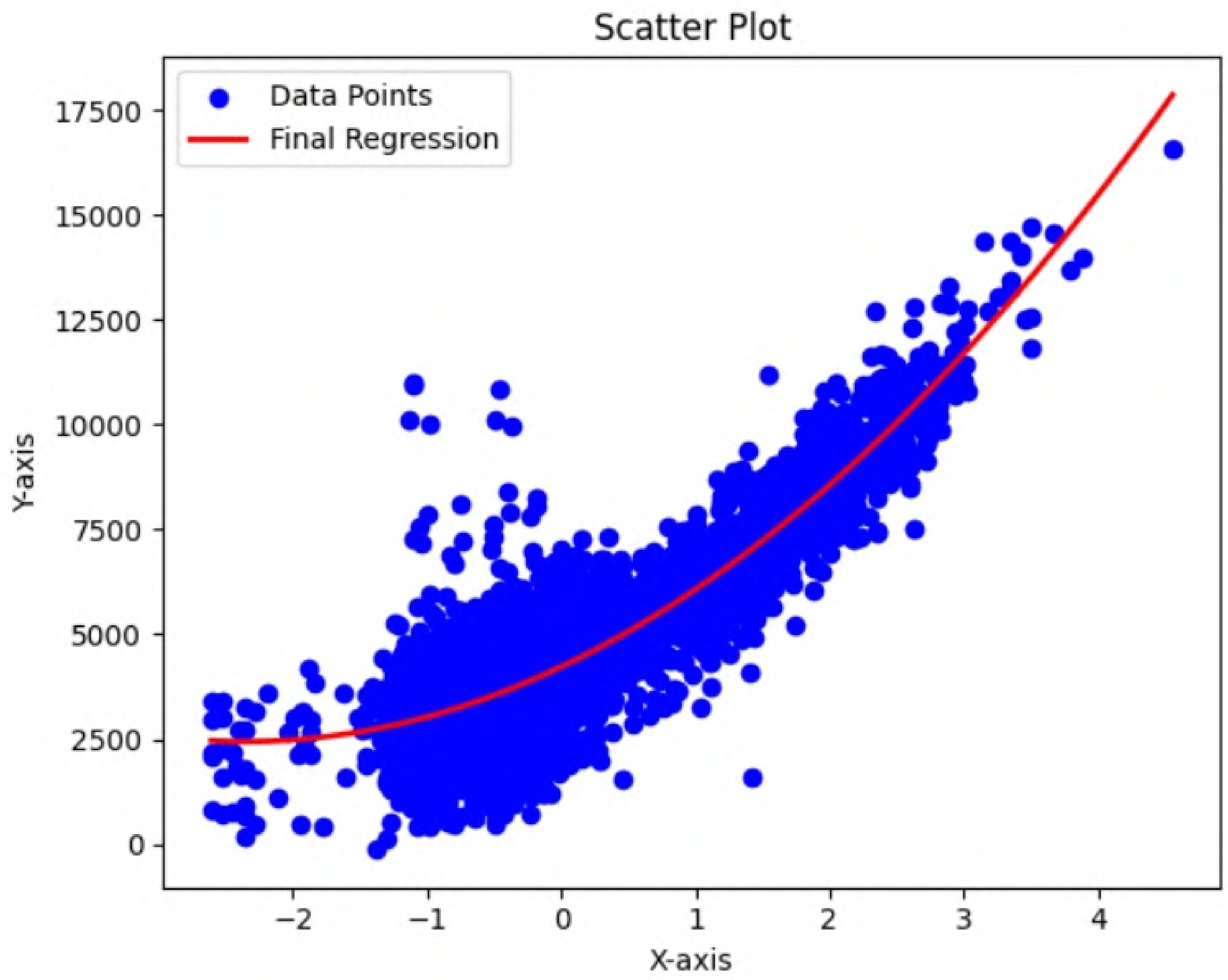
THE PRIMARY OBJECTIVE OF THIS PROJECT IS TO IMPLEMENT AND EXPLORE NON-LINEAR REGRESSION FOR PREDICTING HOUSE PRICES

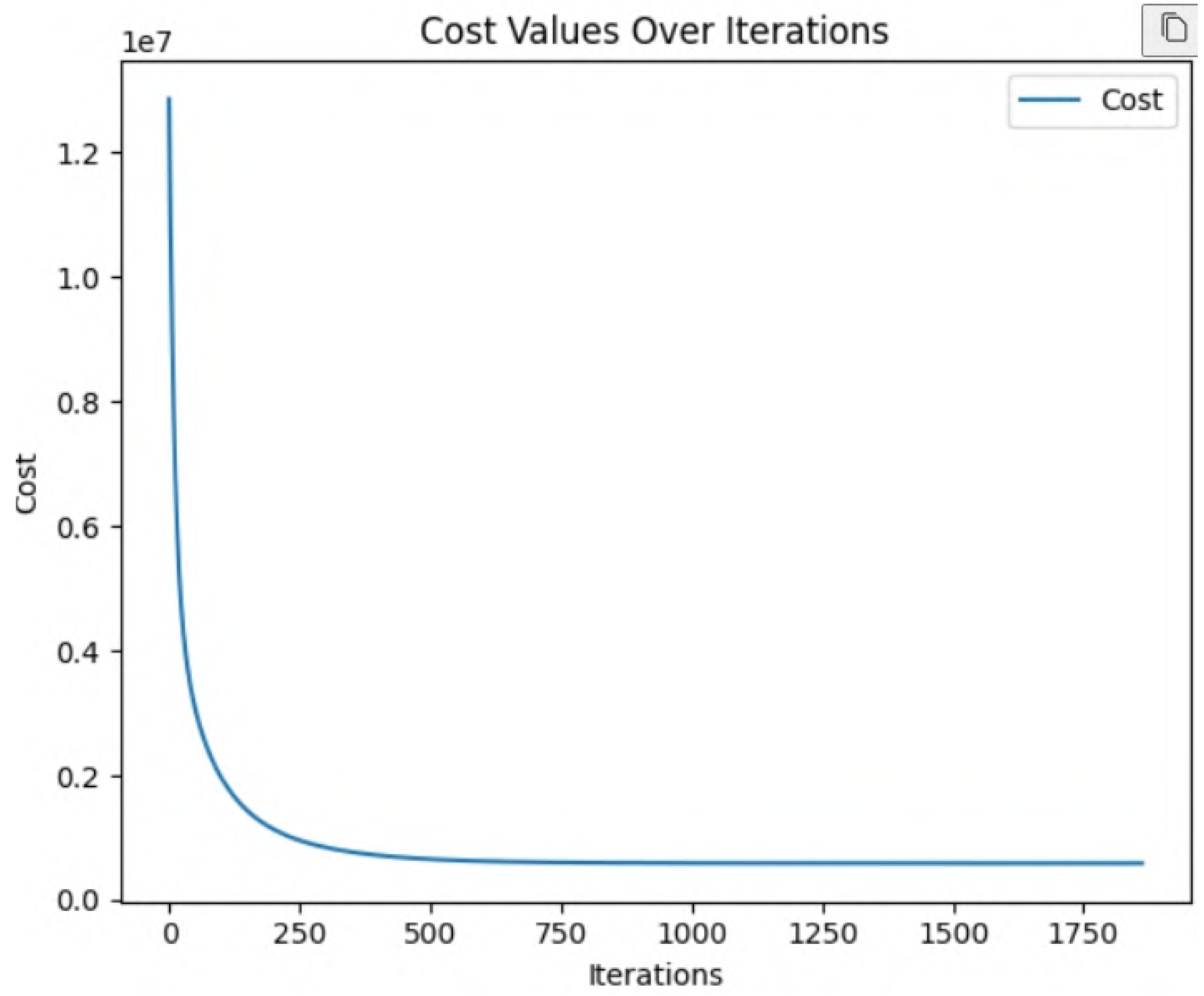
- We are importing data from a CSV file where we have the housing data of over 4000 houses in Australia and the required data of those houses like, It's area in sq. mt, no of bathrooms and bedrooms and beds and we have the price of that house in that area
- Here we will take the Area of the house and no of bathrooms and no of bedrooms, as deciding variable and with those we will train our regression model with given prices of training data
- Here we are using Non - linear Polynomial regression in our case

Try Pitch

DESIRED OUTPUT

DECREASE IN COST FUNCTION OVER ITTERATION





Try Pitch

WORKING OF CODE

First, we are importing the CSV file which has the housing data

We will copy the data in CSV in a variable data by using Pandas

No we will sort the data -

1 Remove all null values in the column which we are using for training the model (House area, No of bathrooms and No of Bedrooms so that we won't get errors

2 Change the all Null values in State column to the name of State which is maximum

We will take that filtered data and extract the Columns of square meters, price, no of bathrooms and no of bedrooms into variable x, y, z and w

Try Pitch

WORKING OF CODE

We will divide the data into training and testing

For Training purposes, we are going to make a suitable formula for y with the variables x, z, and w so that we can make the predictions more accurately

The formula we are using for the making of regression line is -

$$y = A \cdot x^2 + B \cdot x + C \cdot z + D \cdot w + E$$

```
y is the predicted house price.
x represents the square meters of the house.
z represents the number of bathrooms.
w represents the number of bedrooms.
A,B,C,D and E are the coefficients.
```

Try Pitch

WORKING OF CODE

First, we will initialize the coefficients A, B, C, D, and E as all zero or some constants

We will use the Gradient descent algorithm to change the coeffients itteratively and converge to the optimal line which produce less error

For that we will cost function **J(A, B, C, D, E)** as the mean squared error of the each with respect to our regression line

$$J(A,B,C,D,E) = \frac{1}{2m}\sum_{i=1}^{m}(h_i - y_i)^2$$

where h is the predicted value and

y is the value made by our equation

Try Pitch

WORKING OF CODE

Since our goal is to reduce the Cost function we will take the Partial derivative of the Cost function with respect to all coefficients those are directions we want to make the coefficients converge

$$\frac{\partial J}{\partial A} = \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) \cdot x_i^2$$

$$\frac{\partial J}{\partial B} = \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) \cdot x_i$$

$$\frac{\partial J}{\partial C} = \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) \cdot z_i$$

$$\frac{\partial J}{\partial D} = \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) \cdot w_i$$

$$\frac{\partial J}{\partial E} = \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i)$$

Try Pitch

WORKING OF CODE

Code :-

```
def compute_cost(self):
    m = len(self.x)
    predictions = self.A * self.x ** 2 + self.B * self.x + self.C * self.z + self.D * self.w + self.E
    squared_errors = (predictions - self.y) ** 2
    cost = 1 / (2 * m) * np.sum(squared_errors)
    return cost

def compute_gradient(self):
    predictions = self.A * self.x ** 2 + self.B * self.x + self.C * self.z + self.D * self.w + self.E
    errors = predictions - self.y
    dj_da = np.sum(errors * self.x ** 2) / len(self.x)
    dj_db = np.sum(errors * self.x) / len(self.x)
    dj_dc = np.sum(errors * self.z) / len(self.x)
    dj_dd = np.sum(errors * self.w) / len(self.x)
    dj_de = np.sum(errors) / len(self.x)
    return dj_da, dj_db, dj_dc, dj_dd, dj_de
```

Try Pitch

WORKING OF CODE

Now we have the gradients of each coefficient we will update each coefficients with these gradients like -

$$A := A - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) \cdot x_i^2$$

$$B := B - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) \cdot x_i$$

$$C := C - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) \cdot z_i$$

$$D := D - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) \cdot w_i$$

$$E := E - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i)$$

Make Learning rate α as 0.4 for correct solution

Like that we will do for every coefficients in every iteration untill the cost became nearer to zero

Try Pitch

WORKING OF CODE

Code :-

```
def fit(self, x, y, z, w, test_size=0.2):
    self.x, self.z, self.w, x_stats, z_stats, w_stats = self.normalize_data(x, z, w)
    self.y = y
    x_train, y_train, z_train, w_train, x_test, y_test, z_test, w_test = self.train_test_split(self.x, y, self.z, self.w, train_size=3000)
    cost = math.inf
    while np.abs(cost - self.compute_cost()) > 1e-10:
        self.cost_values.append(cost)
        dj_da, dj_db, dj_dc, dj_dd, dj_de = self.compute_gradient()
        self.A -= dj_da * self.alpha
        self.B -= dj_db * self.alpha
        self.C -= dj_dc * self.alpha
        self.D -= dj_dd * self.alpha
        self.E -= dj_de * self.alpha
```

Try Pitch

WORKING OF CODE

At last we will get the Optimal coeffients of the Regression line

We that line we can find the accuracy of the Testing data by the following formula -

$$\text{Accuracy} = 1 - \frac{\text{Mean Prediction value - Mean Actual value}}{\text{Mean Actual value}}$$

Try Pitch

WORKING OF CODE

Code :-

```python
def calculate_accuracy(self, predictions, actual):
    a = 1 - np.abs((predictions - actual) / actual)
    accuracy = np.mean(a)
    return accuracy

# Print accuracy and other information
train_predictions = self.A * x_train ** 2 + self.B * x_train + self.C * z_train + self.D * w_train + self.E
test_predictions = self.A * x_test ** 2 + self.B * x_test + self.C * z_test + self.D * w_test + self.E

train_accuracy = self.calculate_accuracy(train_predictions, y_train)
test_accuracy = self.calculate_accuracy(test_predictions, y_test)

print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

Try Pitch

WORKING OF CODE

Now we can predict the new Housing data given square meters, no of bathrooms and no of bedrooms

```
def plot_predicted_prices(self, x_values_set, z_values_set, w_values_set):
    plt.figure(figsize=(8, 5))

    all_predicted_prices = []

    for i in range(len(x_values_set)):
        x_values = x_values_set[i]
        z_value = z_values_set[i]
        w_value = w_values_set[i]

        x_normalized = (x_values - self.x_stats[0]) / self.x_stats[1]
        z_normalized = (z_value - self.z_stats[0]) / self.z_stats[1]
        w_normalized = (w_value - self.w_stats[0]) / self.w_stats[1]

        predicted_prices = self.A * x_normalized ** 2 + self.B * x_normalized + self.C * z_normalized + self.D * w_normalized + self.E
        all_predicted_prices.extend(predicted_prices)

        x_values_2 = np.linspace(min(self.x), max(self.x), 100).reshape(-1, 1)
        z_values_2 = np.mean(self.z)
        w_values_2 = np.mean(self.w)
        y_values = self.A * x_values_2 ** 2 + self.B * x_values_2 + self.C * z_values_2 + self.D * w_values_2 + self.E

        plt.scatter(x_normalized, predicted_prices, label=f'Set {i + 1}', marker='o')
```

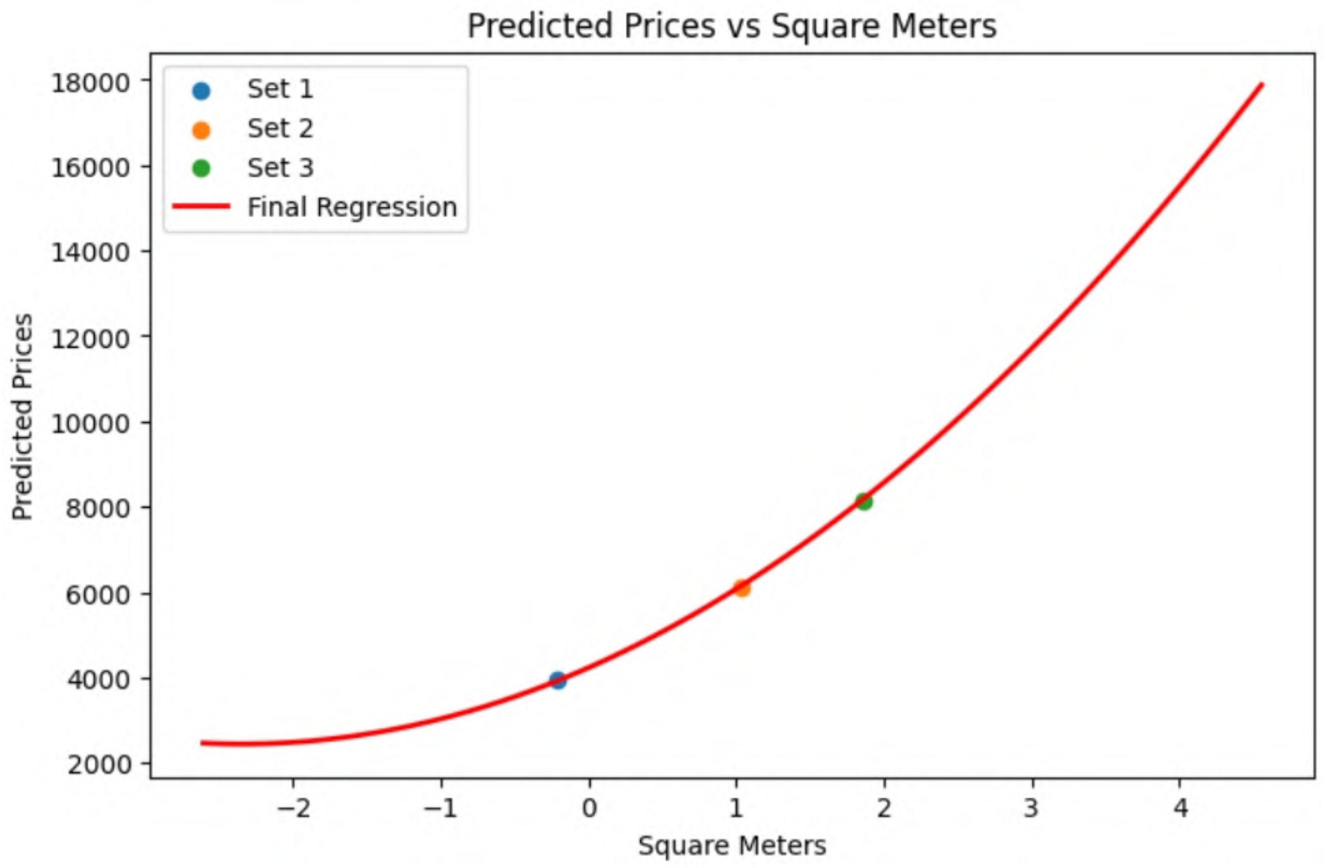
Try Pitch

WORKING OF CODE

Now we can predict the new Housing data given square meters, no of bathrooms and no of bedrooms

```
x_values_to_predict = np.array([30, 45,55]).reshape(-1, 1)
z_value_to_predict = 2,1,3
w_value_to_predict = 3,1,1

regression_model.plot_predicted_prices(x_values_to_predict, z_value_to_predict, w_value_to_predict)
regression_model.display_predicted_prices(x_values_to_predict, z_value_to_predict, w_value_to_predict)
```



Try Pitch

THE END

Thank you

Try Pitch

Pitch

Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

Create a presentation (It's free)