

**Project Report**

Master of Computer Application

Semester – II

Machine Learning Theory and Practice

# Project title: Laptops Price Prediction

By

THANUJA M

Reg. No.:2411022250013

**Department of Computer Application**

**Alliance University**

**Chandapura - Anekal Main Road, Anekal**

**Bengaluru - 562 106 March 2025**

**Laptops Price Prediction**

**Introduction:**

This dataset offers a solid foundation for analyzing how these factors influence laptop prices, enabling you to build predictive models. The diverse range of specifications ensures that you can explore relationships between features and price, perform preprocessing tasks like handling missing data or encoding categorical variables, and apply machine learning techniques like regression.

# Data Preprocessing

In machine learning, data preprocessing refers to the techniques and steps used to transform raw data into a clean, structured format that can be effectively utilized by a machine learning model. It involves processes such as cleaning, organizing, and standardizing the data to improve its quality and ensure optimal performance during training and evaluation.

**Why Data Preprocessing?**

* **Quality Matters:** Models are only as good as the data fed into them. Clean, wellprepared data ensures better performance.
* **Handling Real-World Data:** Raw data can be messy, incomplete, or inconsistent. Preprocessing resolves these issues

**Data Preprocessing and Preparation for Laptops Price Prediction:**

* Importing Essential Libraries
* Loading the Dataset
* Identifying Missing Values
* Handling Missing Values with Mean Imputation
* Detecting Outliers in the Data
* Removing Outliers for Clean Data
* Applying Label Encoding to Categorical Features
* Analyzing Correlations Among Variables • Evaluating Outcome Proportionality
* Separating Features and Target Variable
* Normalizing and Standardizing the Features
* Building and Implementing Linear Regression Model

**1.Importing the necessary libraries**

import pandas as pd import

numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler from

sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt import seaborn as sns from

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

**Code Explanation**:

**pandas:** A powerful library for data manipulation and analysis. It helps handle tabular data using Data Frame objects.

**NumPy** : Essential for numerical computations. It provides support for arrays, mathematical functions.

**sklearn.preprocessing (**LabelEncoder**,** StandardScaler**)**: Tools for preparing data for machine learning. LabelEncoder converts categorical labels to numerical form, and StandardScaler normalizes features by scaling them. **sklearn.linear\_model (**LinearRegression**)**: Implements linear regression, a popular algorithm for predicting numeric values based on input features. **matplotlib.pyplot (**plt**)**: A plotting library for creating static, interactive, and animated visualizations in Python. **seaborn (**sns**)**: Built on matplotlib, it simplifies data visualization by providing a high-level interface for creating attractive plots.

**mean\_squared\_error** :It calculates the average of the squared differences between the actual values and the predicted values. **r2\_score** :It measures the proportion of variance in the target variable that is predictable from the features.

# 2.Load the Laptops Price Prediction Dataset

df=pd.read\_csv('/content/data.csv') df

**Code Explanation:**

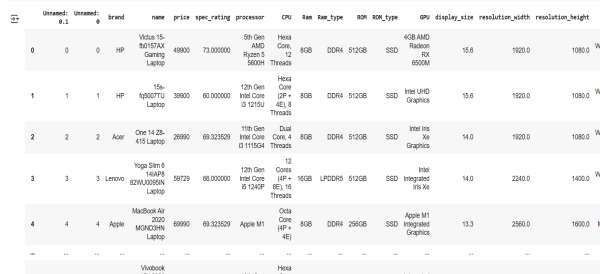
* pd. read\_csv('/content/data.csv'): It uses the pandas library to load data from a CSV file located at /content/data.csv into a Data Frame (df). A Data Frame is a tabular structure similar to a spreadsheet or SQL table.
* df: Displays the loaded Data Frame so you can visually inspect the data.



**Output**



**:**



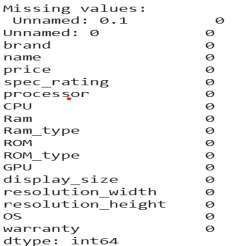
# 3. Finding missing values

print("Missing values:\n", df.isnull().sum())

**Explanation:**

* **df.isnull():** Creates a new DataFrame where each element is True if the corresponding value in df is missing (null), and False otherwise**.**
* **sum():** Sums up the True values (which are considered as 1) for each column, giving the total count of missing values in every column.

**Output:**



# 4. Replace missing values with the mean

df.fillna(df.mean(numeric\_only=True), inplace=True) df  **Code Explanation:**

* **df.mean(numeric\_only=True**): Calculates the mean (average) of each numeric column in the DataFrame. The numeric\_only=True parameter ensures that only numeric columns are considered for this calculation.
* **df.fillna:** Fills the missing (null) values in df with the calculated means for their respective columns.
* **inplace=True**: Updates the DataFrame df directly without creating a new copy.

**Output**:



1. **Check the outlier**

# Box plots for detecting outliers num\_cols = df.select\_dtypes(include=['number']).columns plt.figure(figsize=(12, 6)) sns.boxplot(data=df[num\_cols]) plt.xticks(rotation=90)

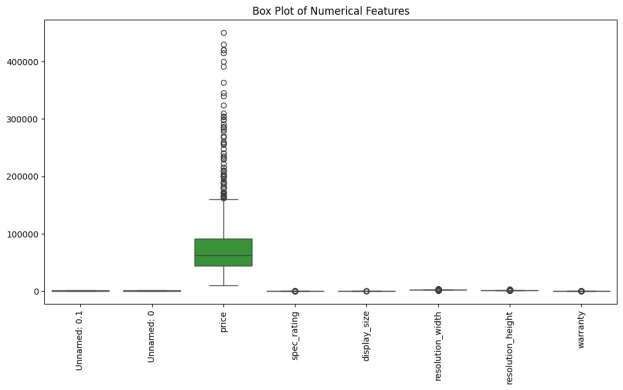
plt.title("Box Plot of Numerical Features") plt.show()

**Code Explanation:**

* + - **Identify Numerical Columns:** df.select\_dtypes(include=['number']).columns selects columns with numerical data types.
    - **Figure Setup**: plt.figure(figsize=(12,6))creates a plot of specified size.
    - **Create Boxplot**: generates a boxplot for the numerical columns. • **Formatting**:

plt.xticks(rotation=90) rotates the x-axis labels for better readability, and adds a title to the plot.

* + - **Display**: plt.show() renders the plot. **Output:**



1. **Drop the outliers** numerical\_df = df.select\_dtypes(include=np.number)

Q1 = numerical\_df.quantile(0.25)

Q3 = numerical\_df.quantile(0.75) IQR = Q3 - Q1

filtered\_numerical\_df = numerical\_df[ ~((numerical\_df < ) | (numerical\_df > (Q3 + 1.5 \* IQR))).any(axis=1)] df = df.loc[filtered\_numerical\_df.index] print(df) print("Outliers removed.")

**Explanation:**

* + - The first quartile (Q1) is the 25th percentile, indicating the value below which 25% of the data falls.
    - The third quartile (Q3) is the 75th percentile, indicating the value below which 75% of the data falls.
    - IQR (Interquartile Range) is calculated as IQR=Q3-Q1. It measures the spread of the middle 50% of the data.

**Outlier Detection**:

* + - Data points lying outside a defined range are considered outliers
    - The range is defined as (Q1 - 1.5 \* IQR, Q3 + 1.5 \* IQR)
    - Any value less than Q1 - 1.5 is considered a lower outlier. • Any value greater than Q3 +

1.5 is considered an upper outlier.

**Output:**

Unnamed: 0.1 Unnamed: 0 brand name price spec\_rating processor \

1. 0 0 6 427 49900 73.000000 79
2. 1 1 6 31 39900 60.000000 26

6 6 6 5 215 36790 60.000000 26

8 8 8 2 465 48990 64.000000 34 10

10 10 11 394 49990 69.323529 36

.. ... ... ... ... ... ... ...

869 907 996 5 225 63990 69.323529 43

872 910 999 6 421 51990 65.000000 79

875 913 1002 1 67 50990 65.000000 32

880 918 1007 6 423 59990 64.000000 79 888 926 1015 2 474 44990 69.323529 58

CPU Ram Ram\_type ROM ROM\_type GPU display\_size resolution\_width \

1. 13 4 1 3 1 7 15.6 1920.0
2. 12 4 1 3 1 81 15.6 1920.0

6 12 4 1 3 1 81 15.6 1920.0

8 3 4 1 3 1 76 15.6 1920.0

10 0 1 1 3 1 70 15.6 1920.0 .. ... ... ... ... ... ... ... ...

869 0 4 1 3 1 70 15.6 1920.0

872 13 4 1 3 1 6 15.6 1920.0

875 14 4 1 3 1 14 15.6 1920.0

880 13 4 1 3 1 20 15.6 1920.0 888 12 4 1 3 1 56 15.6 1920.0

resolution\_height OS warranty

1. 1080.0 5 1
2. 1080.0 5 1

6 1080.0 5 1

8 1080.0 5 1 10

1080.0 5 1

.. ... .. ...

869 1080.0 5 1

872 1080.0 5 1

875 1080.0 5 1

880 1080.0 5 1

888 1080.0 5 1

[203 rows x 18 columns] Outliers removed**.**

# 7. Perform label encoding

label\_encoder = LabelEncoder()

categorical\_columns = df.select\_dtypes(include=['object']).columns

for col in categorical\_columns: df[col] = label\_encoder.fit\_transform(df[col]) print(df) print("Label encoding applied.")

**Explanation:**

Initialize Label Encoder:

* **label\_encoder = LabelEncoder():**Creates an instance of LabelEncoder from sklearn , which converts categorical values into integer labels.
* **categorical\_columns = df.select\_dtypes(include=['object']).**columns Selects columns in the DataFrame () that have a data type of (usually indicating categorical data).
* for col in categorical\_columns: df[col] = label\_encoder.fit\_transform(df[col]) Iterates through each categorical column, transforms its unique values into numerical labels using , and replaces the original column values with these encoded labels.

**Output:**

Unnamed: 0.1 Unnamed: 0 brand name price spec\_rating processor \

1. 0 0 6 427 49900 73.000000 79
2. 1 1 6 31 39900 60.000000 26
3. 2 2 1 291 26990 69.323529 11
4. 5 5 1 105 39990 62.000000 31
5. 6 6 5 215 36790 60.000000 26

.. ... ... ... ... ... ... ...

1. 923 1012 5 110 125699 75.000000 54
2. 924 1013 1 77 49990 69.323529 42
3. 925 1014 1 64 56990 69.323529 43
4. 926 1015 2 474 44990 69.323529 58

891 929 1018 2 339 129990 73.000000 55

CPU Ram Ram\_type ROM ROM\_type GPU display\_size resolution\_width \

1. 13 4 1 3 1 7 15.6 1920.0
2. 12 4 1 3 1 81 15.6 1920.0
3. 11 4 1 3 1 78 14.0 1920.0
4. 3 4 1 3 1 78 14.0 1920.0
5. 12 4 1 3 1 81 15.6 1920.0 .. ... ... ... ... ... ... ... ...
6. 4 1 3 3 1 30 15.6 1920.0
7. 12 1 5 3 1 70 14.0 1920.0
8. 0 1 7 3 1 69 15.6 1920.0
9. 12 4 1 3 1 56 15.6 1920.0

891 5 1 1 3 1 30 15.6 1920.0

resolution\_height OS warranty

1. 1080.0 5 1
2. 1080.0 5 1
3. 1080.0 5 1
4. 1080.0 5 1
5. 1080.0 5 1 .. ... .. ...
6. 1080.0 5 1
7. 1080.0 6 1
8. 1080.0 5 1
9. 1080.0 5 1

891 1080.0 5 1

[608 rows x 18 columns] Label encoding applied.

# 8. Check correlation matrix

plt.figure(figsize=(12, 8)) sns.heatmap(df.corr(), annot=True, cmap="coolwarm",

fmt=".2f", linewidths=0.5) plt.title("Feature Correlation Heatmap") plt.show()

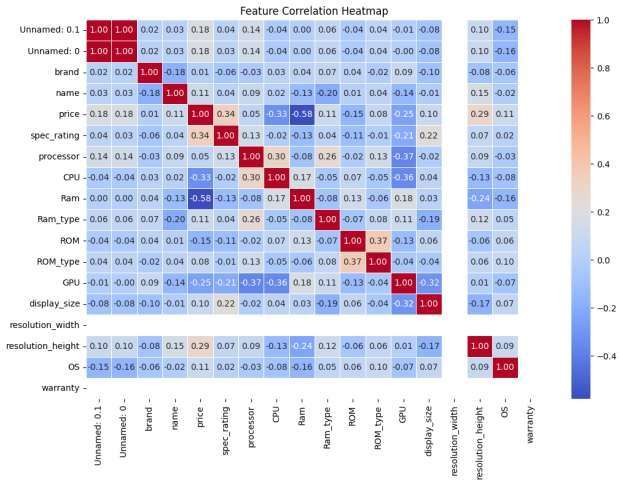
**Code Explanation:**

**plt.figure(figsize=(12, 8)):** Sets the figure size to 12x8 inches. **df.corr():** Computes the correlation matrix for numeric columns in the DataFrame. **sns.heatmap():**

Plots the heatmap.

**annot=True**: Displays the correlation values in the heatmap. **cmap="coolwarm":** Uses the "coolwarm" color palette. **fmt=".2f":** Formats correlation values to 2 decimal places. **linewidths=0.5**: Adds space between cells for better visibility. **plt.title():** Sets the title of the heatmap as "Feature Correlation Heatmap". **plt.show():** Displays the plot**.**

**Output:**



**9. Check outcome proportionality**

X\_column = 'spec\_rating' # Replace with your feature of interest (e.g., spec\_rating) y\_column = 'price' plt.figure(figsize=(10, 5))

sns.histplot(df["price"], bins=30, kde=True) plt.title("Distribution of Price")

plt.xlabel("Price") plt.ylabel("spec\_rating") plt.show()

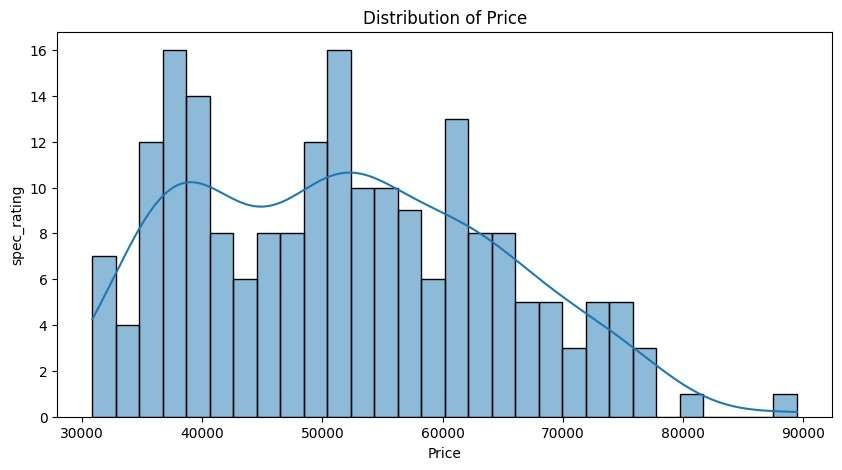
**Code Explanation:**

**X\_column = 'spec\_rating' and y\_column = 'price':** These lines define variables for the feature and target columns. However, they aren't directly used in the plot below.

**plt.figure(figsize=(10, 5)):**Sets the figure size to 10x5 inches for better visualization **sns.histplot(df["price"], bins=30, kde=True):**Createsa histogram for the price column. **bins=30**: Divides the data into 30 bins (intervals) for the histogram. **kde=True**: Adds a

smooth KDE curve over the histogram to represent the data's probability density.

**Output:**



**10. Separate independent and target variables**

X = df[['Unnamed: 0.1', 'Unnamed: 0', 'brand', 'name', 'spec\_rating', 'processor',

'CPU', 'Ram', 'Ram\_type', 'ROM', 'ROM\_type', 'GPU', 'display\_size',

'resolution\_width', 'resolution\_height', 'OS', 'warranty']] # Independent variables y = df['price'] # Target variable

# Display the results print("Independent Variables (X):") print(X.head()) print("\nTarget Variable (y):") print(y.head())

**Code Explanation:**

* X: Contains columns like brand, name, spec\_rating, etc., which are features used as inputs for modeling.
* **y:** Stores the price column, which is the target variable the model will predict.
* print(X.head()): Displays the first 5 rows of the independent variables (X).
* print(y.head()): Displays the first 5 rows of the target variable (y).

**Output:**

Independent Variables (X):

Unnamed: 0.1 Unnamed: 0 brand name \

1. 0 0 HP Victus 15-fb0157AX Gaming Laptop
2. 1 1 HP 15s-fq5007TU Laptop
3. 2 2 Acer One 14 Z8-415 Laptop
4. 3 3 Lenovo Yoga Slim 6 14IAP8 82WU0095IN Laptop 4 4 4 Apple MacBook Air 2020 MGND3HN Laptop

spec\_rating processor CPU \

1. 73.000000 5th Gen AMD Ryzen 5 5600H Hexa Core, 12 Threads
2. 60.000000 12th Gen Intel Core i3 1215U Hexa Core (2P + 4E), 8 Threads
3. 69.323529 11th Gen Intel Core i3 1115G4 Dual Core, 4 Threads
4. 66.000000 12th Gen Intel Core i5 1240P 12 Cores (4P + 8E), 16 Threads
5. 69.323529 Apple M1 Octa Core (4P + 4E)

Ram Ram\_type ROM ROM\_type GPU display\_size \

1. 8GB DDR4 512GB SSD 4GB AMD Radeon RX 6500M 15.6
2. 8GB DDR4 512GB SSD Intel UHD Graphics 15.6
3. 8GB DDR4 512GB SSD Intel Iris Xe Graphics 14.0
4. 16GB LPDDR5 512GB SSD Intel Integrated Iris Xe 14.0
5. 8GB DDR4 256GB SSD Apple M1 Integrated Graphics 13.3

resolution\_width resolution\_height OS warranty

1. 1920.0 1080.0 Windows 11 OS 1
2. 1920.0 1080.0 Windows 11 OS 1
3. 1920.0 1080.0 Windows 11 OS 1
4. 2240.0 1400.0 Windows 11 OS 1
5. 2560.0 1600.0 Mac OS 1

Target Variable (y):

1. 49900
2. 39900
3. 26990
4. 59729
5. 69990

Name: price, dtype: int64

# 11. Apply normalization and standardization

from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Normalization (Min-Max Scaling) normalizer

= MinMaxScaler() df\_normalized = pd.DataFrame(normalizer.fit\_transform(df), columns=df.columns)

print("\nNormalized Data (First 5 Rows):\n", df\_normalized.head())

# Standardization (Z-score Scaling) scaler

= StandardScaler() df\_standardized = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)

print("\nStandardized Data (First 5 Rows):\n", df\_standardized.head())

**Code Explanation:**

Normalization (Min-Max Scaling):

* **What it Does**: Rescales all feature values to fall within the range [0, 1].
* **How**:MinMaxScalar() computes each value as: $$ X\_{\text{scaled}} = \frac{X - X\_{\text{min}}}{X\_{\text{max}} - X\_{\text{min}}} $$.
* **Purpose**: Useful when features have different scales but you want them in the same range, e.g., for machine learning algorithms sensitive to data magnitude.

Standardization (Z-score Scaling):

* **What it Does**: Converts data to have a mean of 0 and a standard deviation of 1.
* **How**: StandardScaler() computes each value as: $$ Z = \frac{X - \mu}{\sigma} $$ where ( \mu ) is the mean, and ( \sigma ) is the standard deviation.
* **Purpose**: Ensures data is centered and scaled, which is critical for models that assume normally distributed input (e.g., logistic regression, k-means clustering).

**Output:**

Normalized Data (First 5 Rows):

Unnamed: 0.1 Unnamed: 0 brand name price spec\_rating \

1. 0.000000 0.000000 0.3125 0.775510 0.324820 1.000000
2. 0.001080 0.000985 0.3125 0.040816 0.154320 0.000000
3. 0.006479 0.005911 0.2500 0.382189 0.101294 0.000000
4. 0.008639 0.007882 0.0625 0.846011 0.309304 0.307692 4 0.010799 0.009852 0.6250 0.714286 0.326354

0.717195

processor CPU Ram Ram\_type ROM ROM\_type GPU display\_size \

1. 0.612613 0.722222 1.00 0.0 0.0 0.0 0.060976 0.0
2. 0.135135 0.666667 1.00 0.0 0.0 0.0 0.963415 0.0
3. 0.135135 0.666667 1.00 0.0 0.0 0.0 0.963415 0.0
4. 0.207207 0.166667 1.00 0.0 0.0 0.0 0.902439 0.0 4 0.225225 0.000000 0.25 0.0 0.0 0.0

0.829268 0.0

resolution\_width resolution\_height OS warranty

1. 0.0 0.0 0.0 0.0
2. 0.0 0.0 0.0 0.0
3. 0.0 0.0 0.0 0.0
4. 0.0 0.0 0.0 0.0 4 0.0 0.0 0.0 0.0 Standardized Data (First 5 Rows):

Unnamed: 0.1 Unnamed: 0 brand name price spec\_rating \

1. -1.401712 -1.405221 -0.022882 0.929133 -0.155123 1.814140
2. -1.398035 -1.401918 -0.022882 -1.445029 -0.954369 -1.944763
3. -1.379649 -1.385402 -0.280936 -0.341883 -1.202934 -1.944763
4. -1.372295 -1.378795 -1.055100 1.156956 -0.227854 -0.788177
5. -1.364941 -1.372189 1.267392 0.731286 -0.147930 0.751102

processor CPU Ram Ram\_type ROM ROM\_type GPU \ 0

1.236558 0.429152 0.756519 0.0 0.0 0.0 -1.537691

1. -0.712194 0.267823 0.756519 0.0 0.0 0.0 1.230337
2. -0.712194 0.267823 0.756519 0.0 0.0 0.0 1.230337
3. -0.418042 -1.184141 0.756519 0.0 0.0 0.0 1.043308
4. -0.344505 -1.668129 -1.309491 0.0 0.0 0.0 0.818874

display\_size resolution\_width resolution\_height OS warranty

1. 3.552714e-15 0.0 0.0 0.0 0.0
2. 3.552714e-15 0.0 0.0 0.0 0.0
3. 3.552714e-15 0.0 0.0 0.0 0.0
4. 3.552714e-15 0.0 0.0 0.0 0.0
5. 3.552714e-15 0.0 0.0 0.0 0.0

**12. Implement the linear Regression**

from sklearn.linear\_model import LinearRegression from

sklearn.model\_selection import train\_test\_split from sklearn.metrics import mean\_squared\_error, r2\_score import pandas as pd

X = df[['Unnamed: 0.1', 'Unnamed: 0', 'brand', 'name', 'spec\_rating', 'processor',

'CPU', 'Ram', 'Ram\_type', 'ROM', 'ROM\_type', 'GPU', 'display\_size', 'resolution\_width', 'resolution\_height', 'OS', 'warranty']] # List of columns y = df['price'] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) model = LinearRegression() model.fit(X\_train, y\_train) y\_pred = model.predict(X\_test) mse = mean\_squared\_error(y\_test, y\_pred) # Mean Squared Error r2 = r2\_score(y\_test, y\_pred) # R-squared value

print("Mean Squared Error:", mse) print("R-squared

Value:", r2)

plt.scatter(y\_test, y\_pred, color="blue", alpha=0.5) plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color="red", linestyle="--",

linewidth=2)

plt.title("Actual vs Predicted Prices")

plt.xlabel("Actual Prices") plt.ylabel("Predicted Prices") plt.show()

**Code Explanation:**

1. **Data Preparation:** You're selecting specific columns (X) as the features and price (Y) as the target variable from the DataFrame **.**
2. **Splitting the Dataset**: The train\_test\_split function divides the data into training (80%) and testing (20%) sets.
3. **Model Initialization**: The LinearRegression class is used to create a linear regression model.
4. **Training the Model**: The fit method trains the model using the training data (X\_train,Y\_train ).
5. **Making Predictions**: The predict method generates predictions for the test data (X\_test).
6. **Evaluating Performance**: Metrics such as Mean Squared Error (mse) and R-squared (r2) are calculated to assess the model’s accuracy.
7. Finally, it prints out the error (how far predictions are from actual values) and the Rsquared value.

**Output:**

Mean Squared Error: 311155786.8100262

R-squared Value: 0.4335467427935126

