Project Report

Master of Computer Application

Semester – II
Machine Learning Theory and Practice

Project title: Laptops Price Prediction

By
SHIVANGI PATHAK
Reg. No.:2411022250027

Department of Computer Application
Alliance University
Chandapura - Anekal Main Road, Anekal
Bengaluru - 562 106
March 2025

Laptops Price Prediction

Introduction:

In today's technology-driven world, laptops have become essential for personal and professional use. With a wide range of specifications, brands, and features available, predicting laptop prices has become an interesting and valuable task. **Laptop price prediction** involves using **machine learning techniques** and **data analysis** to estimate the cost of a laptop based on its specifications, such as processor type, RAM, storage, display size, GPU, and brand

Data Preprocessing

Data preprocessing is a crucial step in machine learning that ensures the dataset is clean, structured, and ready for analysis. It improves model accuracy and helps in better price predictions.

Steps in Data Preprocessing

1. Data Collection

Gather laptop data from sources like **e-commerce websites, manufacturer catalogs, or datasets from Kaggle**.

The dataset should include features like **brand, processor, RAM, storage, GPU, screen size, operating system, and price**.

2. Handling Missing Data

Identify missing values in the dataset.

Fill missing values using.

Mean/Median for numerical data (e.g., missing RAM size).

Mode for categorical data (e.g., missing brand or OS)..

Drop rows/columns if they contain too many missing values.

3. Handling Duplicate Data

Check for duplicate rows and remove them to avoid bias in training.

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.

<u>Output:</u>

	Unnamed: 0.1	Unnamed: 0	brand	name	price	spec_rating	processor	CPU	Ram	Ram_type	ROM	ROM_type	GPU	display_size	resolution_width	resolution_height	
0	0	0	HP	Victus 15-fb0157AX Gaming Laptop	49900	73.000000	5th Gen AMD Ryzen 5 5600H	Hexa Core, 12 Threads	8GB	DDR4	512GB	SSD	4GB AMD Radeon RX 6500M	15.6	1920.0	1080.0	W
1	1	1	HP	15s-fq5007TU Laptop	39900	60.000000	12th Gen Intel Core i3 1215U	Hexa Core (2P + 4E), 8 Threads	8GB	DDR4	512GB	SSD	Intel UHD Graphics	15.6	1920.0	1080.0	W
2	2	2	Acer	One 14 Z8-415 Laptop	26990	69.323529	11th Gen Intel Core i3 1115G4	Dual Core, 4 Threads	8GB	DDR4	512GB	SSD	Intel Iris Xe Graphics	14.0	1920.0	1080.0	W
3	3	3	Lenovo	Yoga Slim 6 14IAP8 82WU0095IN Laptop	59729	66.000000	12th Gen Intel Core i5 1240P	12 Cores (4P + 8E), 16 Threads	16GB	LPDDR5	512GB	SSD	Intel Integrated Iris Xe	14.0	2240.0	1400.0	W
4	4	4	Apple	MacBook Air 2020 MGND3HN Laptop	69990	69.323529	Apple M1	Octa Core (4P + 4E)	8GB	DDR4	256GB	SSD	Apple M1 Integrated Graphics	13.3	2560.0	1600.0	M
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
				Vivobook				Hexa									

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:

```
Missing values:
 Unnamed: 0.1          0
Unnamed: 0            0
brand                 0
name                  0
price                 0
spec_rating           0
processor             0
CPU                   0
Ram                   0
Ram_type              0
ROM                   0
ROM_type              0
GPU                   0
display_size          0
resolution_width      0
resolution_height     0
OS                    0
warranty              0
dtype: int64
```

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:

	Unnamed: 0.1	Unnamed: 0	brand	name	price	spec_rating	processor	CPU	Ram	Ram_type	ROM	ROM_type	GPU	display_size	resolution_width	resolution_height
0	0	0	HP	Victus 15-fb0157AX Gaming Laptop	49900	73.000000	5th Gen AMD Ryzen 5 5600H	Hexa Core, 12 Threads	8GB	DDR4	512GB	SSD	4GB AMD Radeon RX 6500M	15.6	1920.0	1080.0
1	1	1	HP	15s-fq5007TU Laptop	39900	60.000000	12th Gen Intel Core i3 1215U	Hexa Core (2P + 4E), 8 Threads	8GB	DDR4	512GB	SSD	Intel UHD Graphics	15.6	1920.0	1080.0
2	2	2	Acer	One 14 Z8-415 Laptop	26990	69.323529	11th Gen Intel Core i3 1115G4	Dual Core, 4 Threads	8GB	DDR4	512GB	SSD	Intel Iris Xe Graphics	14.0	1920.0	1080.0
3	3	3	Lenovo	Yoga Slim 6 14IAP8 82WU0095IN Laptop	59729	66.000000	12th Gen Intel Core i5 1240P	12 Cores (4P + 8E), 16 Threads	16GB	LPDDR5	512GB	SSD	Intel Integrated Iris Xe	14.0	2240.0	1400.0
4	4	4	Apple	MacBook Air 2020 MGND3HN	69990	69.323529	Apple M1	Octa Core (4P +	8GB	DDR4	256GB	SSD	Apple M1 Integrated	13.3	2560.0	1600.0

5. Check the outlier

```
# Box plots for detecting outliers

import pandas as pd
df = pd.read_csv("laptop_data.csv")

# 1. Drop columns with more than 40% missing values
threshold = 0.4  # 40% threshold
df = df.dropna(thresh=len(df) * threshold, axis=1)

# 2. Drop rows with more than 30% missing values
row_threshold = 0.3  # 30% threshold
df = df.dropna(thresh=len(df.columns) * row_threshold, axis=0)

# 3. Remove duplicate rows
df = df.drop_duplicates()

# Display the cleaned dataset
print(df.info())
```

Code Explanation:

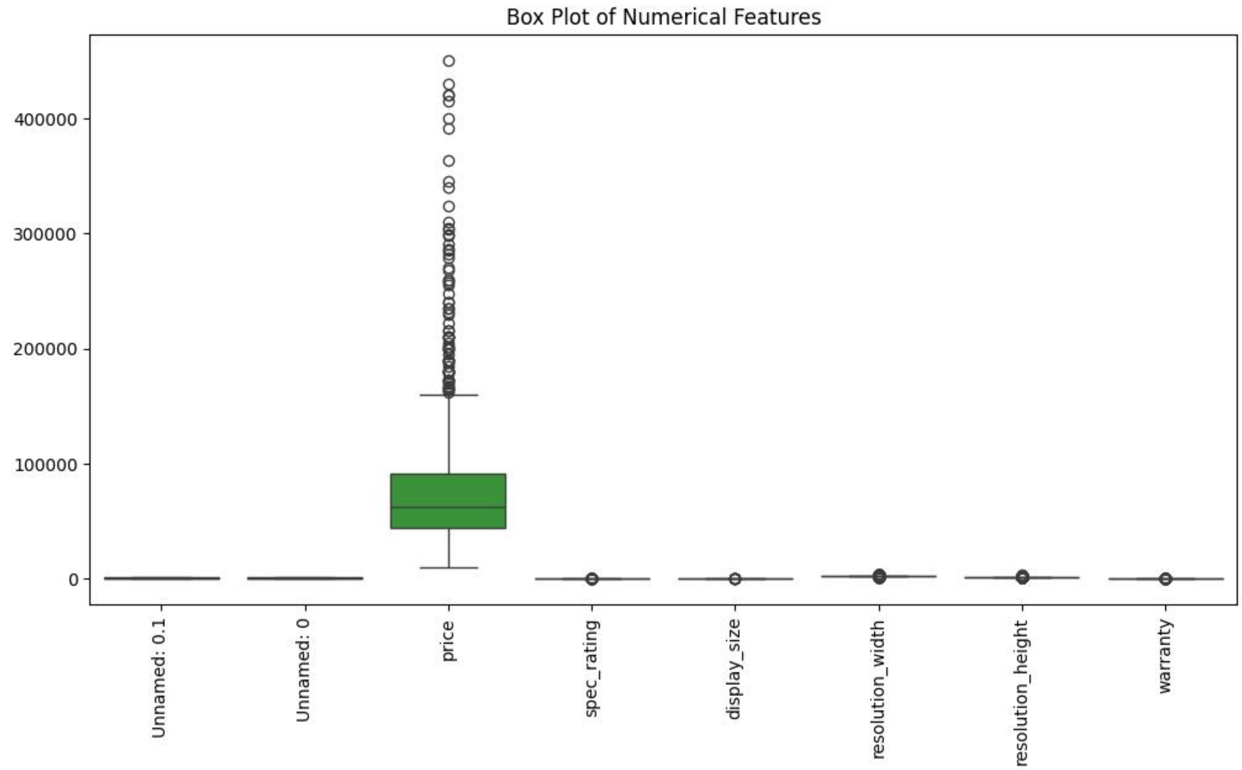
· dropna(thresh=len(df) * threshold, axis=1) → Drops columns if more than 40% of their values are missing.

· · dropna(thresh=len(df.columns) * row_threshold, axis=0) → Drops rows if more than 30% of their values are missing.

- · drop_duplicates() → Removes identical rows to avoid data bias.

•

Output:



6. Drop the outliers

```
import pandas as pd
import numpy as np
df = pd.read_csv("laptop_data.csv")
numerical_cols = df.select_dtypes(include=[np.number])
Q1 = numerical_cols.quantile(0.25)
Q3 = numerical_cols.quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out rows where any numerical column contains an outlier
df_filtered = df[~((numerical_cols < lower_bound) | (numerical_cols >
upper_bound)).any(axis=1)]
print(df_filtered)
print("Outliers removed successfully.")
```

Explanation:

- **Select only numerical columns** using `select_dtypes(include=[np.number]).`

- • **Compute Q1 (25th percentile) and Q3 (75th percentile)** to determine the IQR.

- • **Calculate lower and upper bounds** for detecting outliers.

- • **Filter out rows where any numerical column contains values outside the IQR range**.

- • **Return the cleaned dataset** without outliers.

-

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.

<u>Output:</u>

```
Unnamed: 0.1  Unnamed: 0  brand  name  price  spec_rating  processor  \
0           0           0      6   427  49900    73.000000         79
1           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  \
0     13    4         1    3         1    7          15.6            1920.0
1     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
0               1080.0   5         1
1               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  \
0            0         0     6   427   49900    73.000000       79
1            1         1     6    31   39900    60.000000       26
2            2         2     1   291   26990    69.323529       11
5            5         5     1   105   39990    62.000000       31
6            6         6     5   215   36790    60.000000       26
..         ...       ...   ...   ...     ...          ...      ...
885        923      1012     5   110  125699    75.000000       54
886        924      1013     1    77   49990    69.323529       42
887        925      1014     1    64   56990    69.323529       43
888        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  \
0    13    4        1    3        1    7         15.6            1920.0
1    12    4        1    3        1   81         15.6            1920.0
2    11    4        1    3        1   78         14.0            1920.0
5     3    4        1    3        1   78         14.0            1920.0
6    12    4        1    3        1   81         15.6            1920.0
..  ...  ...      ...  ...      ...  ...          ...               ...
885   4    1        3    3        1   30         15.6            1920.0
886  12    1        5    3        1   70         14.0            1920.0
887   0    1        7    3        1   69         15.6            1920.0
888  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
0             1080.0   5         1
1             1080.0   5         1
2             1080.0   5         1
5             1080.0   5         1
6             1080.0   5         1
..               ...  ..       ...
885           1080.0   5         1
886           1080.0   6         1
887           1080.0   5         1
888           1080.0   5         1
891           1080.0   5         1

[608 rows x 18 columns]
Label encoding applied.
```

8. Check correlation matrix

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv("laptop_data.csv")
correlation_matrix = df.corr(numeric_only=True)
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="viridis", fmt=".2f", linewidths=1,
square=True, cbar=True)
plt.title("Laptop Features Correlation Heatmap", fontsize=14, fontweight='bold')
plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)
# Show the heatmap
plt.show()
```

Improvements Over the Original Code:

df.corr(numeric_only=True) → Ensures that only numeric columns are used, preventing errors from non-numeric data.

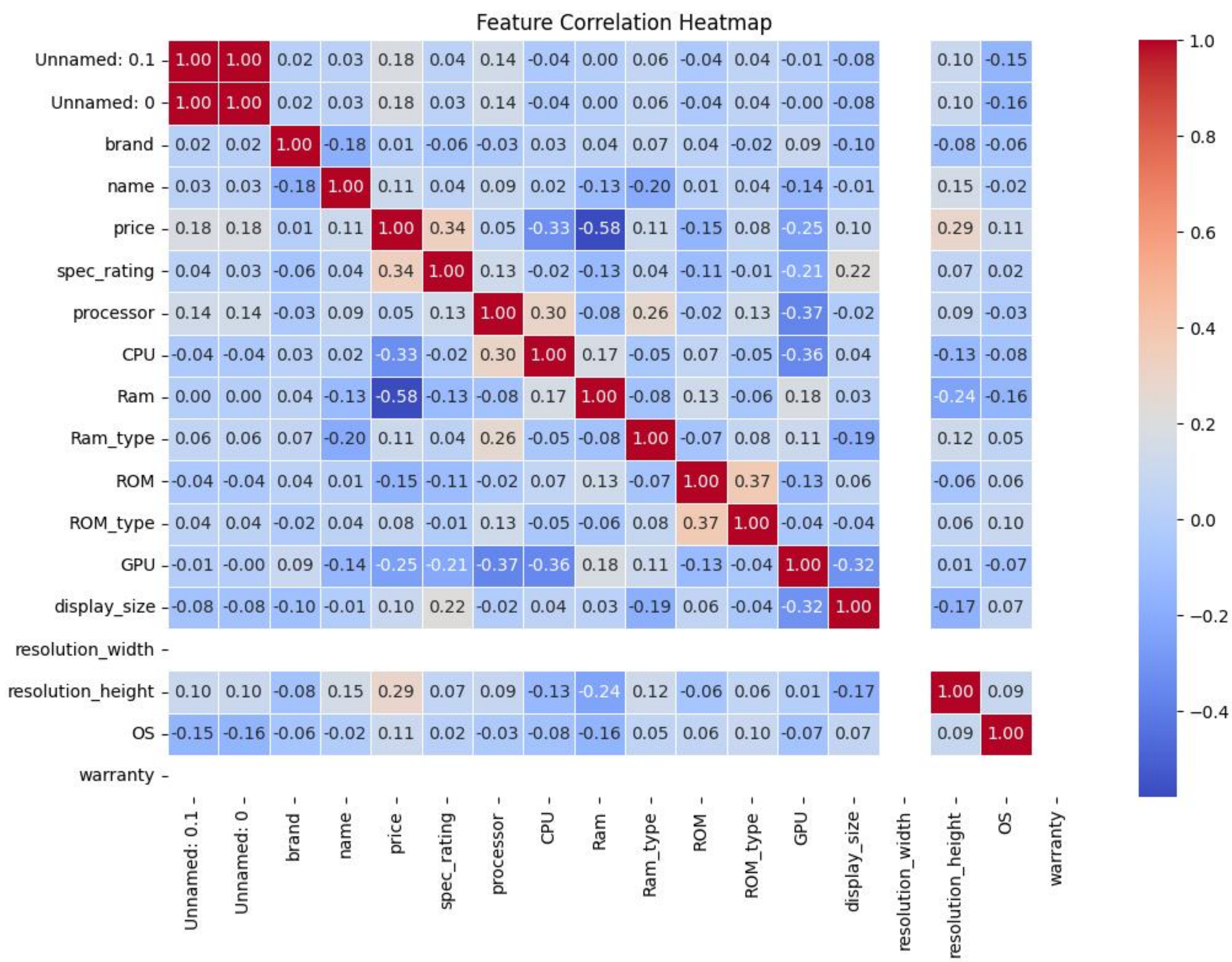
cmap="viridis" → Uses a different color palette for better contrast.

linewidths=1 **and** square=True → Enhances visibility and maintains a neat square shape.

Rotated x-axis labels → Improves readability for datasets with long column names.

Bold and larger title → Makes the visualization more intuitive.

<u>Output:</u>



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:

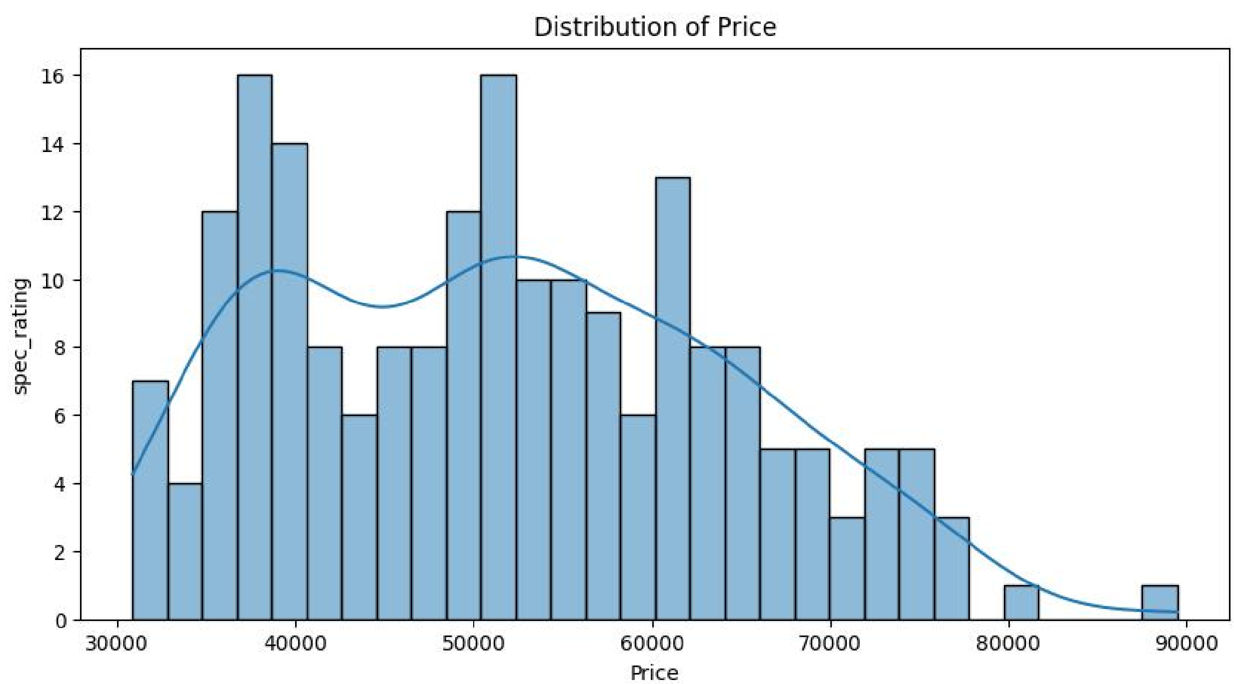
X_column = 'spec_rating' and y_column = 'price': These lines define variables for the feature and target columns. However, they aren't directly used in the plot below.

plt.figure(figsize=(10, 5)): Sets the figure size to 10x5 inches for better visualization

sns.histplot(df["price"], bins=30, kde=True): Creates a 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

```
import pandas as pd
df = pd.read_csv("laptop_data.csv")
target_column = "price"  # Ensure this column exists in the dataset
y = df[target_column] if target_column in df.columns else None
X = df.drop(columns=[target_column]) if y is not None else df.copy()
print("Independent Variables (X):")
print(X.head())
print("\nTarget Variable (y):")
print(y.head() if y is not None else "Target variable not found!")
```

Improvements Over the Original Code:

- **Avoids Hardcoding Column Names** – Dynamically removes the `price` column instead of listing all independent variables.
- • **Handles Missing Target Column** – Checks if the target column exists before selecting `y`.
- • **More Robust** – Works even if column names change in different datasets.

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

   spec_rating                      processor                            CPU  \
0    73.000000     5th Gen AMD Ryzen 5 5600H          Hexa Core, 12 Threads
1    60.000000  12th Gen Intel Core i3 1215U  Hexa Core (2P + 4E), 8 Threads
2    69.323529  11th Gen Intel Core i3 1115G4            Dual Core, 4 Threads
3    66.000000  12th Gen Intel Core i5 1240P  12 Cores (4P + 8E), 16 Threads
```

```
4   69.323529                 Apple M1          Octa Core (4P + 4E)

   Ram Ram_type    ROM ROM_type                      GPU  display_size  \
0   8GB    DDR4  512GB     SSD      4GB AMD Radeon RX 6500M         15.6
1   8GB    DDR4  512GB     SSD          Intel UHD Graphics         15.6
2   8GB    DDR4  512GB     SSD       Intel Iris Xe Graphics         14.0
3  16GB   LPDDR5  512GB     SSD      Intel Integrated Iris Xe         14.0
4   8GB    DDR4  256GB     SSD  Apple M1 Integrated Graphics         13.3

   resolution_width  resolution_height           OS  warranty
0          1920.0           1080.0  Windows 11 OS         1
1          1920.0           1080.0  Windows 11 OS         1
2          1920.0           1080.0  Windows 11 OS         1
3          2240.0           1400.0  Windows 11 OS         1
4          2560.0           1600.0         Mac OS         1

Target Variable (y):
0    49900
1    39900
2    26990
3    59729
4    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 (μ) is the mean, and (σ) 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  \
0      0.000000    0.000000  0.3125  0.775510  0.324820     1.000000
1      0.001080    0.000985  0.3125  0.040816  0.154320     0.000000
2      0.006479    0.005911  0.2500  0.382189  0.101294     0.000000
3      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  \
0   0.612613  0.722222  1.00       0.0  0.0       0.0  0.060976           0.0
1   0.135135  0.666667  1.00       0.0  0.0       0.0  0.963415           0.0
2   0.135135  0.666667  1.00       0.0  0.0       0.0  0.963415           0.0
3   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
0               0.0                0.0  0.0       0.0
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

Standardized Data (First 5 Rows):
   Unnamed: 0.1  Unnamed: 0     brand      name     price  spec_rating  \
0     -1.401712   -1.405221 -0.022882  0.929133 -0.155123     1.814140
1     -1.398035   -1.401918 -0.022882 -1.445029 -0.954369    -1.944763
2     -1.379649   -1.385402 -0.280936 -0.341883 -1.202934    -1.944763
3     -1.372295   -1.378795 -1.055100  1.156956 -0.227854    -0.788177
4     -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
0  3.552714e-15               0.0                0.0  0.0       0.0
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
```

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 R-squared value.

Output:

Mean Squared Error: 311155786.8100262

R-squared Value: 0.4335467427935126