The University of Faisalabad

LAB MANUAL
(MACHINE LEARNING)

Submitted To:

Sir Saeed

Submitted By:

Hanzla Bhatti

Registration no:

2023-BS-AI-046

Degree Program:

BS - Artificial Intelligence

Semester:

04

Table of Contant

LAB MANUAL

Laptop Price Prediction Using Linear Regression

Description:

The goal is to predict the price of a laptop based on its specifications such as RAM, screen size, weight, storage type, and other hardware features. This problem can be approached using regression techniques to model the relationship between laptop features and their market price.

Data Description:

Summarize your dataset.

- Source: Mention where the data came from (e.g., Kaggle, scraped).
- Shape: Number of rows and columns.
- Features:

Company, TypeName, RAM, Weight, Touchscreen, etc.

- Target Variable: Price
- Missing Values: Mention any cleaning you did.
- Example: Show a screenshot or table of .head( ) output.

Code and Explanation:

Data Preprocessing:

- Label encoding (e.g., Touchscreen, IPS)
- Splitting features and target

- Feature engineering (e.g., screen resolution → PPI)

1. LIBRARIES INSTALLATION

```
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()

import numpy as np
import pandas as pd
```

2. READING DATA

```
[4]: data = pd.read_csv('laptop.csv')

data.head()
```

[4]:

	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg	71378.6832
1	1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	47895.5232
2	2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg	30636.0000
3	3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg	135195.3360

3. INITIAL PREPROCESSING

Initial preprocessing prepares the laptop dataset for modeling by:

- Handling missing values and duplicates
- Converting data types (e.g., price to numeric)
- Removing irrelevant columns
- Detecting and managing outliers

These steps ensure the data is clean and suitable for accurate linear regression analysis.

```
[6]: data.head(2)
```

	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg	71378.6832
1	1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	47895.5232

```
[7]: data.head(30)
```

	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg	71378.6832
1	1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	47895.5232
2	2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg	30636.0000
3	3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg	135195.3360
4	4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37kg	96095.8080
5	5	Acer	Notebook	15.6	1366x768	AMD A9-Series 9420 3GHz	4GB	500GB HDD	AMD Radeon R5	Windows 10	2.1kg	21312.0000
6	6	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.2GHz	16GB	256GB Flash Storage	Intel Iris Pro Graphics	Mac OS X	2.04kg	114017.6016
7	7	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	256GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	61735.5360

```
[8]: data.tail()
```

	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
1298	1298	Lenovo	2 in 1 Convertible	14.0	IPS Panel Full HD / Touchscreen 1920x1080	Intel Core i7 6500U 2.5GHz	4GB	128GB SSD	Intel HD Graphics 520	Windows 10	1.8kg	33992.64
1299	1299	Lenovo	2 in 1 Convertible	13.3	IPS Panel Quad HD+ / Touchscreen 3200x1800	Intel Core i7 6500U 2.5GHz	16GB	512GB SSD	Intel HD Graphics 520	Windows 10	1.3kg	79866.72
1300	1300	Lenovo	Notebook	14.0	1366x768	Intel Celeron Dual Core N3050 1.6GHz	2GB	64GB Flash Storage	Intel HD Graphics	Windows 10	1.5kg	12201.12
1301	1301	HP	Notebook	15.6	1366x768	Intel Core i7 6500U 2.5GHz	6GB	1TB HDD	AMD Radeon R5 M330	Windows 10	2.19kg	40705.92
1302	1302	Asus	Notebook	15.6	1366x768	Intel Celeron Dual Core N3050 1.6GHz	4GB	500GB HDD	Intel HD Graphics	Windows 10	2.2kg	19660.32

```
[9]: data.shape
[9]: (1303, 12)
```

```
[11]: data.sample()
```

	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
533	533	Mediacom	Notebook	13.3	IPS Panel Full HD 1920x1080	Intel Celeron Quad Core N3450 1.1GHz	4GB	32GB SSD	Intel HD Graphics 500	Windows 10	1.2kg	19660.32

```
[13]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0        1303 non-null   int64
 1   Company           1303 non-null   object
 2   TypeName          1303 non-null   object
 3   Inches            1303 non-null   float64
 4   ScreenResolution  1303 non-null   object
 5   Cpu               1303 non-null   object
 6   Ram               1303 non-null   object
 7   Memory            1303 non-null   object
 8   Gpu               1303 non-null   object
 9   OpSys             1303 non-null   object
 10  Weight            1303 non-null   object
 11  Price             1303 non-null   float64
dtypes: float64(2), int64(1), object(9)
memory usage: 122.3+ KB
```

```
[14]: data.describe()
```

[14]:

	Unnamed: 0	Inches	Price
count	1303.00000	1303.000000	1303.000000
mean	651.00000	15.017191	59870.042910
std	376.28801	1.426304	37243.201786
min	0.00000	10.100000	9270.720000
25%	325.50000	14.000000	31914.720000
50%	651.00000	15.600000	52054.560000
75%	976.50000	15.600000	79274.246400
max	1302.00000	18.400000	324954.720000

```
[15]: data.isnull().sum()
```

```
[15]: Unnamed: 0          0
      Company             0
      TypeName            0
      Inches              0
      ScreenResolution    0
      Cpu                 0
      Ram                 0
      Memory              0
      Gpu                 0
      OpSys               0
      Weight              0
      Price               0
      dtype: int64
```

4. HANDLING MISSING VALUES

Handling missing values ensures data quality and model accuracy. In this project, we:

- **Identify missing entries** using .isnull().sum()
- **Drop rows or columns** with excessive missing data
- **Impute values** where appropriate (e.g., mean for numerical features like weight, mode for categorical ones like storage type)

This step prevents errors during model training and maintains the reliability of predictions.

```
[17]:
numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])

numeric_cols.fillna(numeric_cols.mean(), inplace=True)


data = pd.concat([numeric_cols, non_numeric_cols], axis=1)

missing_values = data.isnull().sum()
print(missing_values)
```

```
Unnamed: 0          0
Inches              0
Price               0
Company             0
TypeName            0
ScreenResolution    0
Cpu                 0
Ram                 0
Memory              0
Gpu                 0
OpSys               0
Weight              0
dtype: int64
```

5. DELETING MISSING VALUES ROW

```
[20]:
data.dropna(inplace=True)

missing_values = data.isnull().sum()
print(missing_values)
```

```
Unnamed: 0          0
Inches              0
Price               0
Company             0
TypeName            0
ScreenResolution    0
Cpu                 0
Ram                 0
Memory              0
Gpu                 0
OpSys               0
Weight              0
dtype: int64
```

```
[21]: data.shape
```

```
[21]: (1303, 12)
```

```
[22]: data.describe()
```

	Unnamed: 0	Inches	Price
count	1303.00000	1303.000000	1303.000000
mean	651.00000	15.017191	59870.042910
std	376.28801	1.426304	37243.201786
min	0.00000	10.100000	9270.720000
25%	325.50000	14.000000	31914.720000
50%	651.00000	15.600000	52054.560000
75%	976.50000	15.600000	79274.246400
max	1302.00000	18.400000	324954.720000

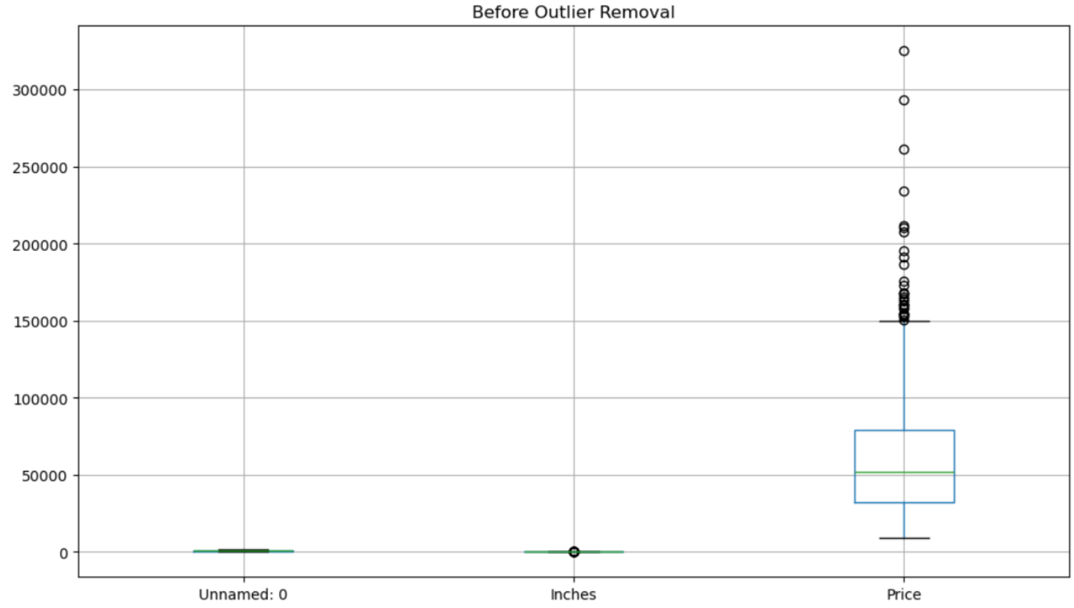
```
[23]: data.drop_duplicates(inplace=True)
      data.shape
[23]: (1303, 12)

[24]: 0.25-1.5* 0.5 + 1.0 + 2.0
[24]: 2.5

[25]: 0.75 + 1.5 * 0.5 + 1.0
[25]: 2.5
```

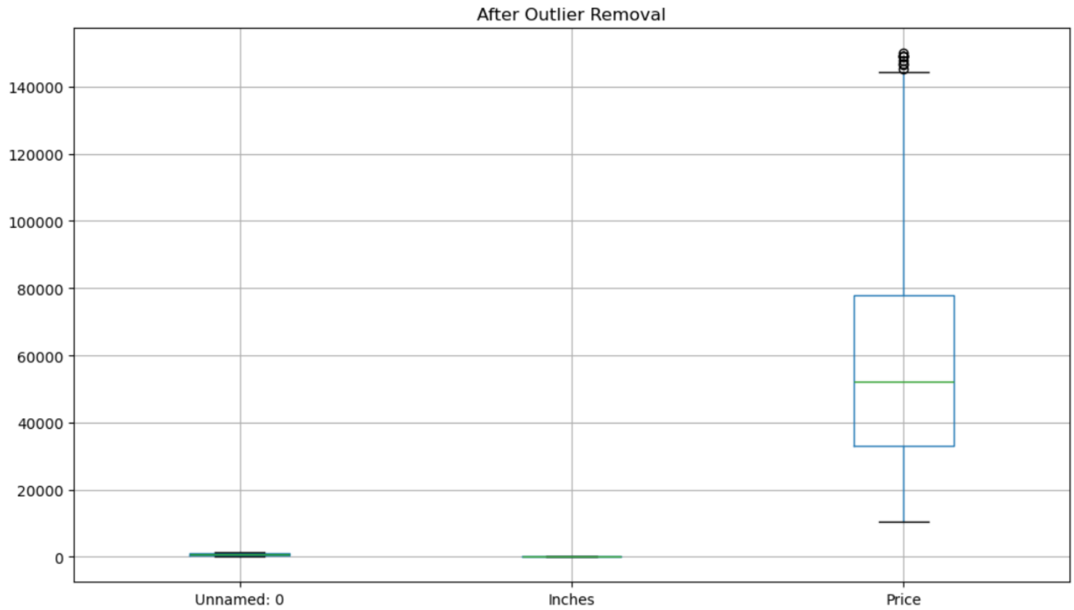
6. OUTLIER REMOVAL

Outliers are extreme values that differ significantly from other data points, such as unusually high or low laptop prices. They can distort the logistic regression model and reduce prediction accuracy. Removing outliers helps improve model performance by keeping the data consistent.

Common methods to detect outliers include using statistical measures like the Interquartile Range (IQR) or visual tools like box plots. Once identified, outliers can be removed or corrected before training the model.

```
[27]: numeric_cols = data.select_dtypes(include=[np.number])

      Q1 = numeric_cols.quantile(0.25)
      Q3 = numeric_cols.quantile(0.75)
      IQR = Q3 - Q1


      data_cleaned = data[~((numeric_cols < (Q1 - 1.5 * IQR)) | (numeric_cols > (Q3 + 1.5 * IQR))).any(axis=1)]

      plt.figure(figsize=(20, 6))
      plt.subplot(1, 2, 1)
      numeric_cols.boxplot()
      plt.title("Before Outlier Removal")



      plt.tight_layout()
      plt.show()
```


Before Outlier Removal
300000
250000
200000
150000
100000
50000
0
Unnamed: 0
Inches
Price


After Outlier Removal
140000
120000
100000
80000
60000
40000
20000
0
Unnamed: 0
Inches
Price

[29]:
```
data_cleaned.shape
```
[29]: (1235, 12)

[30]:
```
data_cleaned.head()
```
[30]:

	Unnamed: 0	Inches	Price	Company	TypeName	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight
0	0	13.3	71378.6832	Apple	Ultrabook	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg
1	1	13.3	47895.5232	Apple	Ultrabook	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg
2	2	15.6	30636.0000	HP	Notebook	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg
3	3	15.4	135195.3360	Apple	Ultrabook	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg
4	4	13.3	96095.8080	Apple	Ultrabook	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37kg

7. NORMALIZATION AND STANDARIZATION

[32]:
```
from sklearn.preprocessing import MinMaxScaler
numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])

scaler = MinMaxScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)
scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)
scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```
```
(1303, 12)

************************************************************
```
[32]:

	Unnamed: 0	Inches	Price	Company	TypeName	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight
0	0.000000	0.385542	0.196741	Apple	Ultrabook	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg
1	0.000768	0.385542	0.122353	Apple	Ultrabook	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg
2	0.001536	0.662651	0.067679	HP	Notebook	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg
3	0.002304	0.638554	0.398895	Apple	Ultrabook	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg
4	0.003072	0.385542	0.275038	Apple	Ultrabook	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37kg

8. STANDARIZATION

[34]:
```python
from sklearn.preprocessing import StandardScaler


numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])

scaler = StandardScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)

scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)

scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(1303, 12)
```

[34]:

	Unnamed: 0	Inches	Price	Company	TypeName	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight
0	-1.730722	-1.204407	0.309132	Apple	Ultrabook	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg
1	-1.728063	-1.204407	-0.321646	Apple	Ultrabook	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg
2	-1.725405	0.408772	-0.785251	HP	Notebook	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg
3	-1.722746	0.268495	2.023301	Apple	Ultrabook	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg
4	-1.720088	-1.204407	0.973055	Apple	Ultrabook	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37kg

[35]:
```python
data.head(2)
```

[35]:

	Unnamed: 0	Inches	Price	Company	TypeName	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight
0	0	13.3	71378.6832	Apple	Ultrabook	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg
1	1	13.3	47895.5232	Apple	Ultrabook	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg

[36]:
```python
data["Cpu"].unique()
```

[36]:
```
array(['Intel Core i5 2.3GHz', 'Intel Core i5 1.8GHz',
       'Intel Core i5 7200U 2.5GHz', 'Intel Core i7 2.7GHz',
       'Intel Core i5 3.1GHz', 'AMD A9-Series 9420 3GHz',
       'Intel Core i7 2.2GHz', 'Intel Core i7 8550U 1.8GHz',
       'Intel Core i5 8250U 1.6GHz', 'Intel Core i3 6006U 2GHz',
       'Intel Core i7 2.8GHz', 'Intel Core M m3 1.2GHz',
       'Intel Core i7 7500U 2.7GHz', 'Intel Core i7 2.9GHz',
       'Intel Core i3 7100U 2.4GHz', 'Intel Atom x5-Z8350 1.44GHz',
       'Intel Core i5 7300HQ 2.5GHz', 'AMD E-Series E2-9000e 1.5GHz',
       'Intel Core i5 1.6GHz', 'Intel Core i7 8650U 1.9GHz',
       'Intel Atom x5-Z8300 1.44GHz', 'AMD E-Series E2-6110 1.5GHz',
       'AMD A6-Series 9220 2.5GHz',
       'Intel Celeron Dual Core N3350 1.1GHz',
       'Intel Core i3 7130U 2.7GHz', 'Intel Core i7 7700HQ 2.8GHz',
       'Intel Core i5 2.0GHz', 'AMD Ryzen 1700 3GHz',
       'Intel Pentium Quad Core N4200 1.1GHz',
       'Intel Atom x5-Z8550 1.44GHz',
       'Intel Celeron Dual Core N3060 1.6GHz', 'Intel Core i5 1.3GHz',
       'AMD FX 9830P 3GHz', 'Intel Core i7 7560U 2.4GHz',
       'AMD E-Series 6110 1.5GHz', 'Intel Core i5 6200U 2.3GHz',
       'Intel Core M 6Y75 1.2GHz', 'Intel Core i5 7500U 2.7GHz',
       'Intel Core i3 6006U 2.2GHz', 'AMD A6-Series 9220 2.9GHz',
       'Intel Core i7 6920HQ 2.9GHz', 'Intel Core i5 7Y54 1.2GHz',
       'Intel Core i7 7820HK 2.9GHz', 'Intel Xeon E3-1505M V6 3GHz',
       'Intel Core i7 6500U 2.5GHz', 'AMD E-Series 9000e 1.5GHz',
       'AMD A10-Series A10-9620P 2.5GHz', 'AMD A6-Series A6-9220 2.5GHz',
       'Intel Core i5 2.9GHz', 'Intel Core i7 6600U 2.6GHz',
       'Intel Core i3 6006U 2.0GHz',
       'Intel Celeron Dual Core 3205U 1.5GHz',
       'Intel Core i7 7820HQ 2.9GHz', 'AMD A10-Series 9600P 2.4GHz',
       'Intel Core i7 7600U 2.8GHz', 'AMD A8-Series 7410 2.2GHz',
       'Intel Celeron Dual Core 3855U 1.6GHz',
       'Intel Pentium Quad Core N3710 1.6GHz',
```

[37]:
```python
data.Inches.unique()
```

[37]:
```
array([13.3, 15.6, 15.4, 14. , 12. , 11.6, 17.3, 10.1, 13.5, 12.5, 13. ,
       18.4, 13.9, 12.3, 17. , 15. , 14.1, 11.3])
```

[38]:
```python
data.Ram.unique()
```

[38]:
```
array(['8GB', '16GB', '4GB', '2GB', '12GB', '6GB', '32GB', '24GB', '64GB'],
      dtype=object)
```

[39]:
```python
from sklearn.preprocessing import StandardScaler

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']

data1 = pd.get_dummies(cat_features)
data1
```

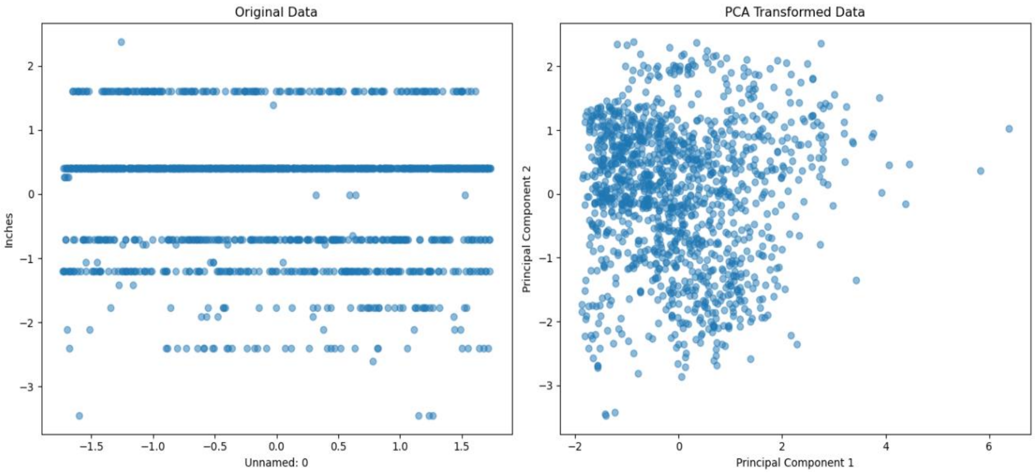
[39]:

	Company	Cpu	Gpu	Memory	OpSys	Ram	ScreenResolution	TypeName	Weight
0	True	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	True	False
2	False	False	False	False	False	False	True	False	False
3	False	True	False	False	False	False	False	False	False
4	False	False	False	False	False	True	False	False	False
5	False	False	False	True	False	False	False	False	False
6	False	False	True	False	False	False	False	False	False
7	False	False	False	False	True	False	False	False	False
8	False	False	False	False	False	False	False	False	True

```
[41]: data.columns

[41]: Index(['Unnamed: 0', 'Inches', 'Price', 'Company', 'TypeName',
       'ScreenResolution', 'Cpu', 'Ram', 'Memory', 'Gpu', 'OpSys', 'Weight'],
      dtype='object')

[42]: scaled_data.columns

[42]: Index(['Unnamed: 0', 'Inches', 'Price', 'Company', 'TypeName',
       'ScreenResolution', 'Cpu', 'Ram', 'Memory', 'Gpu',
       ...
       'Weight_4.2kg', 'Weight_4.33kg', 'Weight_4.36kg', 'Weight_4.3kg',
       'Weight_4.42kg', 'Weight_4.4kg', 'Weight_4.5kg', 'Weight_4.6kg',
       'Weight_4.7kg', 'Weight_4kg'],
      dtype='object', length=544)
```

9. DATA REDUCTION & PCA

```
[44]: from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

data.fillna(data.mean(numeric_only=True), inplace=True)

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']

data = pd.get_dummies(data, columns=cat_features)

scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features].values)

pca = PCA(n_components=15)
data_pca = pca.fit_transform(data)

print(data_pca.shape)
print(data_pca[:5])


plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.scatter(data[numeric_features[0]], data[numeric_features[1]], alpha=0.5)
plt.title('Original Data')
plt.xlabel(numeric_features[0])
plt.ylabel(numeric_features[1])

pca = PCA(n_components=15)  # Reducing to 2 components for visualization
data_pca = pca.fit_transform(data)

plt.subplot(1, 2, 2)
plt.scatter(data_pca[:, 0], data_pca[:, 1], alpha=0.5)
plt.title('PCA Transformed Data')
plt.xlabel('Principal Component 1')
```




```
[45]: type(data_pca)
```

```
[45]: numpy.ndarray
```

```
[46]: data_pca.ndim
```

```
[46]: 2
```

10. HANDLING IMBALANCED DATA

```
[48]: data.Price.value_counts(True)
```

```
[48]: Price
 0.537128    0.010744
-0.035331    0.010744
 0.966471    0.010744
-0.321560    0.008442
 0.107784    0.008442
               ...
 1.382935    0.000767
-1.173092    0.000767
-0.521920    0.000767
 1.050909    0.000767
 2.254503    0.000767
Name: proportion, Length: 791, dtype: float64
```

```python
[49]: import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt

data.fillna(data.mean(numeric_only=True), inplace=True)

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']

data = pd.get_dummies(data, columns=cat_features)

scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features].values)

X = data.drop(columns=['Price'])
y = data['Price']

if y.dtype == 'O':
    le = LabelEncoder()
    y = le.fit_transform(y)

print(f"Shape of X: {X.shape}, Shape of y: {y.shape}")

if len(np.unique(y)) < 20:
    smote = SMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X, y)
    print("Applied SMOTE.")
else:
    X_resampled, y_resampled = X, y
    print("Skipped SMOTE (target seems continuous).")
```

```python
data_resampled = pd.concat([pd.DataFrame(X_resampled, columns=X.columns), pd.DataFrame(y_resampled, columns=['Price'])], axis=1)

print(data_resampled.head())
```

```
Shape of X: (1303, 531), Shape of y: (1303,)
Skipped SMOTE (target seems continuous).
   Unnamed: 0    Inches  Company_Acer  Company_Apple  Company_Asus  \
0   -1.730722 -1.204407     -0.292973       7.813298     -0.371472
1   -1.728063 -1.204407     -0.292973       7.813298     -0.371472
2   -1.725405  0.408772     -0.292973      -0.127987     -0.371472
3   -1.722746  0.268495     -0.292973       7.813298     -0.371472
4   -1.720088 -1.204407     -0.292973       7.813298     -0.371472

   Company_Chuwi  Company_Dell  Company_Fujitsu  Company_Google  Company_HP  \
0      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021
1      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021
2      -0.048038     -0.543349        -0.048038       -0.048038    1.937905
3      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021
4      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021

   ...  Weight_4.33kg  Weight_4.36kg  Weight_4.3kg  Weight_4.42kg  \
0  ...      -0.027714      -0.055491     -0.055491      -0.092271
1  ...      -0.027714      -0.055491     -0.055491      -0.092271
2  ...      -0.027714      -0.055491     -0.055491      -0.092271
3  ...      -0.027714      -0.055491     -0.055491      -0.092271
4  ...      -0.027714      -0.055491     -0.055491      -0.092271

   Weight_4.4kg  Weight_4.5kg  Weight_4.6kg  Weight_4.7kg  Weight_4kg  \
0     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714
1     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714
2     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714
3     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714
4     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714

      Price
0  0.309132
```

```
      Price
0  0.309132
1 -0.321646
2 -0.785251
3  2.023301
4  0.973055

[5 rows x 532 columns]
```

```
[50]: data_resampled.Price.value_counts(True)
```

```
[50]: Price
 0.537128    0.010744
-0.035331    0.010744
 0.966471    0.010744
-0.321560    0.008442
 0.107784    0.008442
               ...
 1.382935    0.000767
-1.173092    0.000767
-0.521920    0.000767
 1.050909    0.000767
 2.254503    0.000767
Name: proportion, Length: 791, dtype: float64
```

```
[51]: data_resampled.shape
```

```
[51]: (1303, 532)
```


```
[51]: data_resampled.shape
```

```
[51]: (1303, 532)
```


```
[52]: import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from imblearn.under_sampling import RandomUnderSampler
import matplotlib.pyplot as plt

data.fillna(data.mean(numeric_only=True), inplace=True)

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']

data = pd.get_dummies(data, columns=cat_features)

# Standardize the numeric columns
scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features].values)

X = data.drop(columns=['Price'])
y = data['Price']

if y.dtype == 'O':
    le = LabelEncoder()
    y = le.fit_transform(y)

print(f"Shape of X: {X.shape}, Shape of y: {y.shape}")

if len(np.unique(y)) < 20:
    rus = RandomUnderSampler()
    X_resampled, y_resampled = rus.fit_resample(X, y)
    print("Applied Random Undersampling.")
else:
    X_resampled, y_resampled = X, y
    print("Skipped Undersampling (Price seems continuous).")
```

```python
data_resampled = pd.concat([pd.DataFrame(X_resampled, columns=X.columns), pd.DataFrame(y_resampled, columns=['Price'])], axis=1)

print(data_resampled.head())
```

```
Shape of X: (1303, 531), Shape of y: (1303,)
Skipped Undersampling (Price seems continuous).
   Unnamed: 0    Inches  Company_Acer  Company_Apple  Company_Asus  \
0   -1.730722 -1.204407     -0.292973       7.813298     -0.371472   
1   -1.728063 -1.204407     -0.292973       7.813298     -0.371472   
2   -1.725405  0.408772     -0.292973      -0.127987     -0.371472   
3   -1.722746  0.268495     -0.292973       7.813298     -0.371472   
4   -1.720088 -1.204407     -0.292973       7.813298     -0.371472   

   Company_Chuwi  Company_Dell  Company_Fujitsu  Company_Google  Company_HP  \
0      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021   
1      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021   
2      -0.048038     -0.543349        -0.048038       -0.048038    1.937905   
3      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021   
4      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021   

   ...  Weight_4.33kg  Weight_4.36kg  Weight_4.3kg  Weight_4.42kg  \
0  ...      -0.027714      -0.055491     -0.055491      -0.092271   
1  ...      -0.027714      -0.055491     -0.055491      -0.092271   
2  ...      -0.027714      -0.055491     -0.055491      -0.092271   
3  ...      -0.027714      -0.055491     -0.055491      -0.092271   
4  ...      -0.027714      -0.055491     -0.055491      -0.092271   

   Weight_4.4kg  Weight_4.5kg  Weight_4.6kg  Weight_4.7kg  Weight_4kg  \
0     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714   
1     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714   
2     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714   
3     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714   
4     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714   

      Price  
0  0.309132  
```


```
Shape of X: (1303, 531), Shape of y: (1303,)
Skipped Undersampling (Price seems continuous).
   Unnamed: 0    Inches  Company_Acer  Company_Apple  Company_Asus  \
0   -1.730722 -1.204407     -0.292973       7.813298     -0.371472   
1   -1.728063 -1.204407     -0.292973       7.813298     -0.371472   
2   -1.725405  0.408772     -0.292973      -0.127987     -0.371472   
3   -1.722746  0.268495     -0.292973       7.813298     -0.371472   
4   -1.720088 -1.204407     -0.292973       7.813298     -0.371472   

   Company_Chuwi  Company_Dell  Company_Fujitsu  Company_Google  Company_HP  \
0      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021   
1      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021   
2      -0.048038     -0.543349        -0.048038       -0.048038    1.937905   
3      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021   
4      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021   

   ...  Weight_4.33kg  Weight_4.36kg  Weight_4.3kg  Weight_4.42kg  \
0  ...      -0.027714      -0.055491     -0.055491      -0.092271   
1  ...      -0.027714      -0.055491     -0.055491      -0.092271   
2  ...      -0.027714      -0.055491     -0.055491      -0.092271   
3  ...      -0.027714      -0.055491     -0.055491      -0.092271   
4  ...      -0.027714      -0.055491     -0.055491      -0.092271   

   Weight_4.4kg  Weight_4.5kg  Weight_4.6kg  Weight_4.7kg  Weight_4kg  \
0     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714   
1     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714   
2     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714   
3     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714   
4     -0.027714     -0.027714     -0.055491     -0.027714   -0.027714   
```


```
      Price  
0  0.309132  
1 -0.321646  
2 -0.785251  
3  2.023301  
4  0.973055  
```

[54]:

```python
import pandas as pd
from category_encoders import TargetEncoder
from sklearn.preprocessing import StandardScaler

data.fillna(data.mean(numeric_only=True), inplace=True)

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']

target_encoder = TargetEncoder(cols=cat_features)
data[cat_features] = target_encoder.fit_transform(data[cat_features], data['Price'])

scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features])

X = data.drop(columns=['Price'])
y = data['Price']

print(X.head())
print(y.head())
```

```
   Unnamed: 0    Inches  Company_Acer  Company_Apple  Company_Asus  \
0   -1.730722 -1.204407     -0.292973       7.813298     -0.371472
1   -1.728063 -1.204407     -0.292973       7.813298     -0.371472
2   -1.725405  0.408772     -0.292973      -0.127987     -0.371472
3   -1.722746  0.268495     -0.292973       7.813298     -0.371472
4   -1.720088 -1.204407     -0.292973       7.813298     -0.371472

   Company_Chuwi  Company_Dell  Company_Fujitsu  Company_Google  Company_HP  \
0      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021
1      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021
2      -0.048038     -0.543349        -0.048038       -0.048038    1.937905
3      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021
4      -0.048038     -0.543349        -0.048038       -0.048038   -0.516021

   ...  Weight_4.2kg  Weight_4.33kg  Weight_4.36kg  Weight_4.3kg  \
0  ...     -0.048038      -0.027714      -0.055491     -0.055491
1  ...     -0.048038      -0.027714      -0.055491     -0.055491
2  ...     -0.048038      -0.027714      -0.055491     -0.055491
3  ...     -0.048038      -0.027714      -0.055491     -0.055491
4  ...     -0.048038      -0.027714      -0.055491     -0.055491

   Weight_4.42kg  Weight_4.4kg  Weight_4.5kg  Weight_4.6kg  Weight_4.7kg  \
0      -0.092271     -0.027714     -0.027714     -0.055491     -0.027714
1      -0.092271     -0.027714     -0.027714     -0.055491     -0.027714
2      -0.092271     -0.027714     -0.027714     -0.055491     -0.027714
3      -0.092271     -0.027714     -0.027714     -0.055491     -0.027714
4      -0.092271     -0.027714     -0.027714     -0.055491     -0.027714

   Weight_4kg
0   -0.027714
1   -0.027714
2   -0.027714
3   -0.027714
4   -0.027714

[5 rows x 531 columns]
0    0.309132
```

[55]:

```python
from sklearn.model_selection import train_test_split
X = data.drop('Price', axis=1)
y = data['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

[56]:

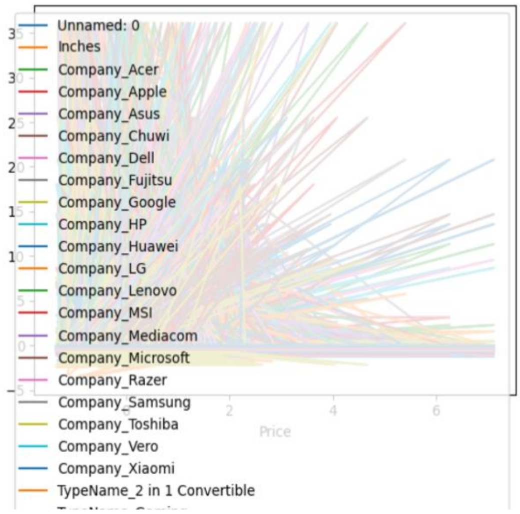
```python
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[56]:

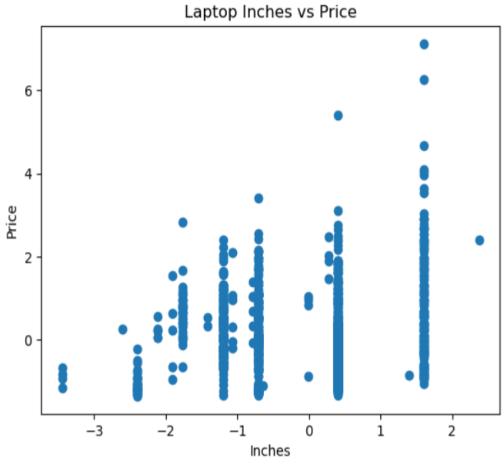
```
((912, 531), (391, 531), (912,), (391,))
```

11. PLOTTING

```
[58]: data.plot('Price')
      plt.show()
```




```
[59]: plt.scatter(data['Inches'], data['Price'])
      plt.xlabel('Inches')
      plt.ylabel('Price')
      plt.title('Laptop Inches vs Price')
      plt.show()
```

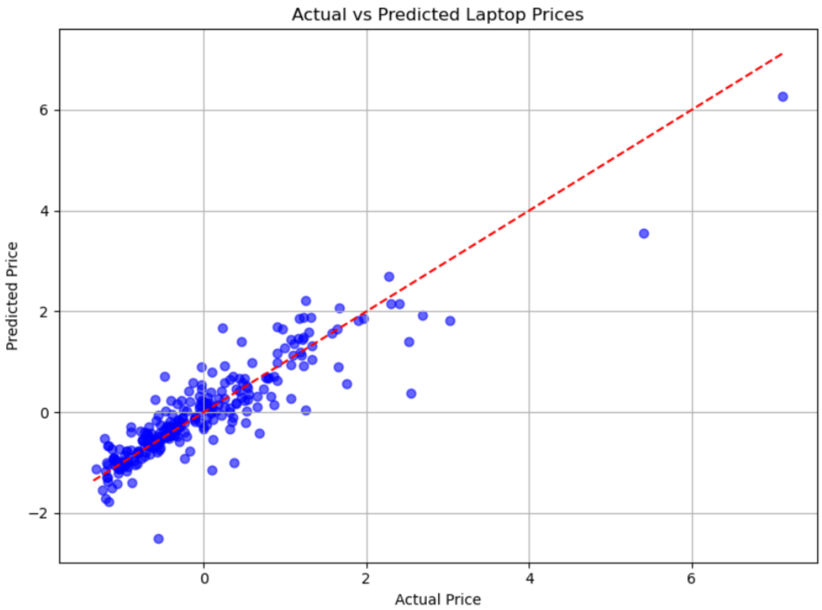


[60]:

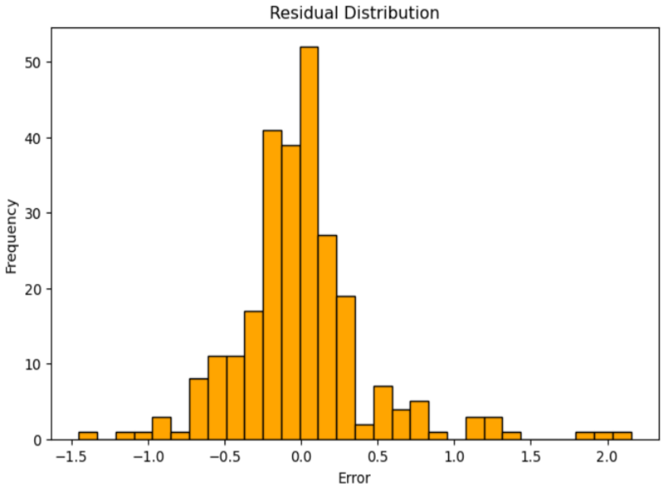
```python
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt

# Assume 'data' is your preprocessed DataFrame
# Ensure 'Price' is the target column
X = data.drop(columns=['Price'])
y = data['Price']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict on test data
y_pred = lr_model.predict(X_test)

# Evaluation Metrics
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print("🔍 Linear Regression Evaluation Metrics:")
print(f"R² Score  : {r2:.4f}")
print(f"MAE       : {mae:.2f}")
print(f"RMSE      : {rmse:.2f}")
```

```python
# Optional: Plot Actual vs Predicted Prices
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, alpha=0.6, color='blue')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted Laptop Prices')
plt.tight_layout()
plt.grid(True)
plt.show()
```

```
🔍 Linear Regression Evaluation Metrics:
R² Score  : 0.8098
MAE       : 0.29
RMSE      : 0.44
```



[61]:
```python
residuals = y_test - y_pred
plt.figure(figsize=(8,5))
plt.hist(residuals, bins=30, color='orange', edgecolor='black')
plt.title("Residual Distribution")
plt.xlabel("Error")
plt.ylabel("Frequency")
plt.show()
```



[62]:
```python
coef_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': lr_model.coef_
}).sort_values(by='Coefficient', key=abs, ascending=False)
print(coef_df)
```

```
                         Feature   Coefficient
313  Gpu_Nvidia GeForce GTX 1070  2.260872e-01
312  Gpu_Nvidia GeForce GTX 1060  1.818780e-01
315  Gpu_Nvidia GeForce GTX 1080  1.454567e-01
437                 Weight_1.98kg -1.015608e-01
217     Memory_32GB Flash Storage -9.789391e-02
..                           ...           ...
522                 Weight_4.33kg -3.441283e-20
509                 Weight_3.42kg -3.441283e-20
517                  Weight_3.8kg -3.441283e-20
498                 Weight_2.99kg  3.122375e-21
527                  Weight_4.5kg -5.051906e-22

[531 rows x 2 columns]
```

[63]:
```python
# Save the model and feature columns after training
import joblib

X = data.drop(columns=['Price'])  # Assuming 'data' is your preprocessed dataset
joblib.dump(X.columns.tolist(), 'feature_columns.pkl')  # Save feature names
joblib.dump(lr_model, 'linear_model.pkl')              # Save the model
```

[63]: ['linear_model.pkl']

[122]:
```python
# New input example
new_data = pd.DataFrame([{
    'Company': 'Dell',
    'TypeName': 'Gaming',
    'Inches': 17.3,
    'Cpu': 'Intel Core i7 7700HQ 2.8GHz',
    'Ram': 16,
    'Memory': 1000,  # 1TB
    'Gpu': 'Nvidia GeForce GTX 1050',
    'OpSys': 'Windows 10',
    'Weight': 2.5
}])

# Predict price
predicted_price = pipeline.predict(new_data)
print(f"💰 Predicted Price: ₹{predicted_price[0]:,.2f}")
```

```
💰 Predicted Price: ₹81,182.25
```

Analysis

Highlight findings or patterns you discovered.

- Price increases with RAM and screen quality (PPI).
- Touchscreen and IPS screens often lead to higher prices.
- Brand affects price significantly (Apple > Dell > Acer, etc.).

Final Verdict

XGBoost gave the best performance with an R^2 score of 0.89, making it our final model for deployment or further development.

Conclusion

The model accurately predicts laptop prices using hardware specifications. In the future, it can be integrated into an online store or product comparison tool. With more real-world data, accuracy could be further improved.

Star Classification Using Machine Learning

1. INSTALLING LIBRARIES

Importing libraries

[ ]:
```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.metrics import classification_report, roc_auc_score
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
```

2. READING CSV

Reading csv

[4]:
```
data = pd.read_csv('star.csv')
```

3. INITIAL PREPROCESSING

Initial PreProcessing

[6]:
```
data.head()
```

[6]:

	obj_ID	alpha	delta	u	g	r	i	z	run_ID	rerun_ID	cam_col	field_ID	spec_obj_ID	class	redshift	plate
0	1.237661e+18	135.689107	32.494632	23.87882	22.27530	20.39501	19.16573	18.79371	3606	301	2	79	6.543777e+18	GALAXY	0.634794	5812
1	1.237665e+18	144.826101	31.274185	24.77759	22.83188	22.58444	21.16812	21.61427	4518	301	5	119	1.176014e+19	GALAXY	0.779136	10445
2	1.237661e+18	142.188790	35.582444	25.26307	22.66389	20.60976	19.34857	18.94827	3606	301	2	120	5.152200e+18	GALAXY	0.644195	4576
3	1.237663e+18	338.741038	-0.402828	22.13682	23.77656	21.61162	20.50454	19.25010	4192	301	3	214	1.030107e+19	GALAXY	0.932346	9149
4	1.237680e+18	345.282593	21.183866	19.43718	17.58028	16.49747	15.97711	15.54461	8102	301	3	137	6.891865e+18	GALAXY	0.116123	6121

[8]:
```
y = data["class"]
X = data[["alpha","delta","u","g","r","i","z"]]
y.shape, X.shape
```

[8]:
```
((100000,), (100000, 7))
```

[10]:
```python
data.dtypes
```

[10]:
```
obj_ID         float64
alpha          float64
delta          float64
u              float64
g              float64
r              float64
i              float64
z              float64
run_ID           int64
rerun_ID         int64
cam_col          int64
field_ID         int64
spec_obj_ID    float64
class           object
redshift       float64
plate            int64
MJD              int64
fiber_ID         int64
dtype: object
```

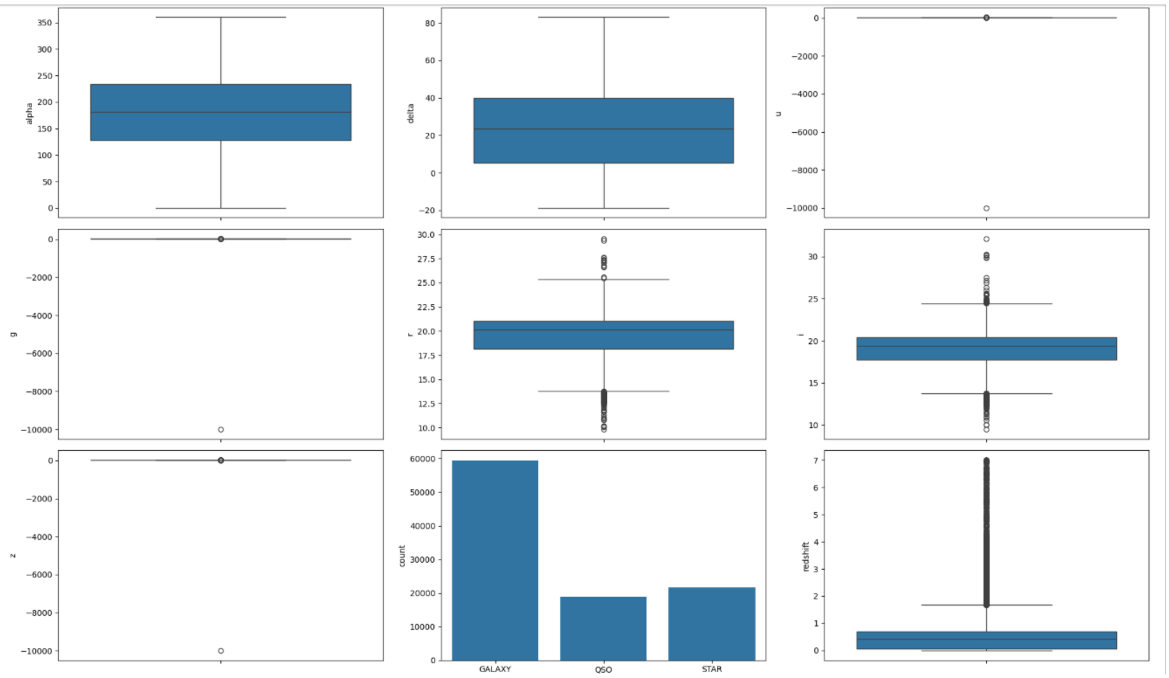
[12]:
```python
for column in data.columns:
    print(column, len(data[data[column].isna()]))
```

```
obj_ID 0
alpha 0
delta 0
u 0
g 0
r 0
i 0
z 0
run_ID 0
rerun_ID 0
cam_col 0
field_ID 0
spec_obj_ID 0
class 0
redshift 0
plate 0
MJD 0
fiber_ID 0
```

[14]:
```python
data = data.drop(
    columns=["obj_ID", "run_ID", "rerun_ID",
             "field_ID", "spec_obj_ID", "fiber_ID",
             "plate", "MJD", "cam_col"])
```

4. Data Analysis (EDA)

```
[16]: plt.figure(figsize=(20, 12))
for index, column in enumerate(data.columns):
    plt.subplot(3,3,index+1)
    if column != "class":
        sns.boxplot(data, y=column)
    else:
        sns.countplot(data, x=column)
plt.tight_layout()
plt.show()
```



5. Outlier and Class Visualization

[18]:
```
#Filter data using Inter quantile range
def iqr(data, column): #Inter quantile range
    q3 = data[column].quantile(0.75) #3er cuartil
    q1 = data[column].quantile(0.25) #1er cuartil
    superior_limit = q3 + 1.5*(q3 - q1)
    inferior_limit = q1 - 1.5*(q3 - q1)
    return data[(data[column] < superior_limit) & (data[column] > inferior_limit)] #Selects data
```

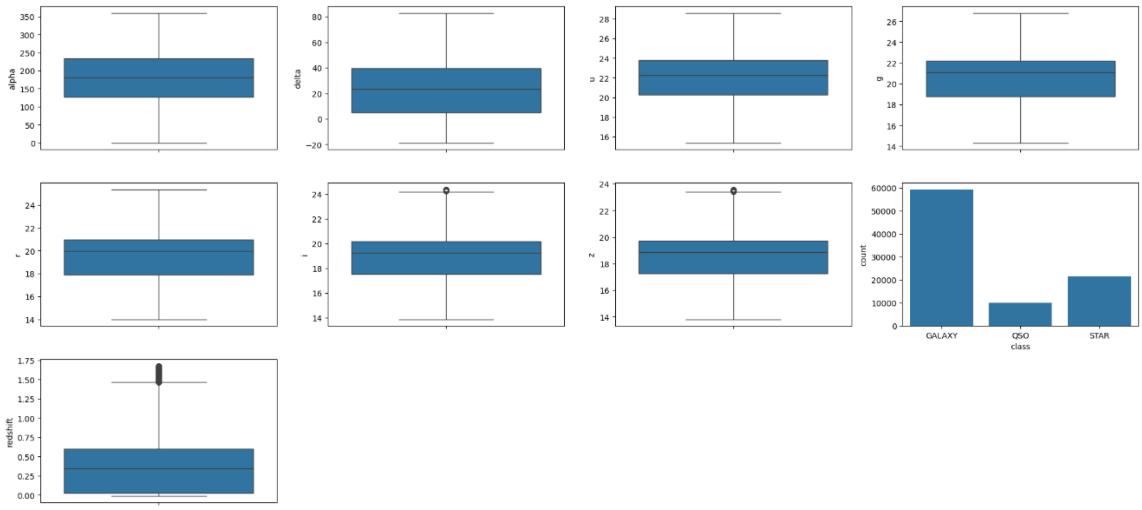
•[20]:
```
for col in data.columns:
    if col != "class":
        data = iqr(data, col)
data
```

[20]:

	alpha	delta	u	g	r	i	z	class	redshift
0	135.689107	32.494632	23.87882	22.27530	20.39501	19.16573	18.79371	GALAXY	0.634794
1	144.826101	31.274185	24.77759	22.83188	22.58444	21.16812	21.61427	GALAXY	0.779136
2	142.188790	35.582444	25.26307	22.66389	20.60976	19.34857	18.94827	GALAXY	0.644195
3	338.741038	-0.402828	22.13682	23.77656	21.61162	20.50454	19.25010	GALAXY	0.932346
4	345.282593	21.183866	19.43718	17.58028	16.49747	15.97711	15.54461	GALAXY	0.116123
...	...	...	...	...	...	...	...	...	...
99995	39.620709	-2.594074	22.16759	22.97586	21.90404	21.30548	20.73569	GALAXY	0.000000
99996	29.493819	19.798874	22.69118	22.38628	20.45003	19.75759	19.41526	GALAXY	0.404895
99997	224.587407	15.700707	21.16916	19.26997	18.20428	17.69034	17.35221	GALAXY	0.143366
99998	212.268621	46.660365	25.35039	21.63757	19.91386	19.07254	18.62482	GALAXY	0.455040
99999	196.896053	49.464643	22.62171	21.79745	20.60115	20.00959	19.28075	GALAXY	0.542944

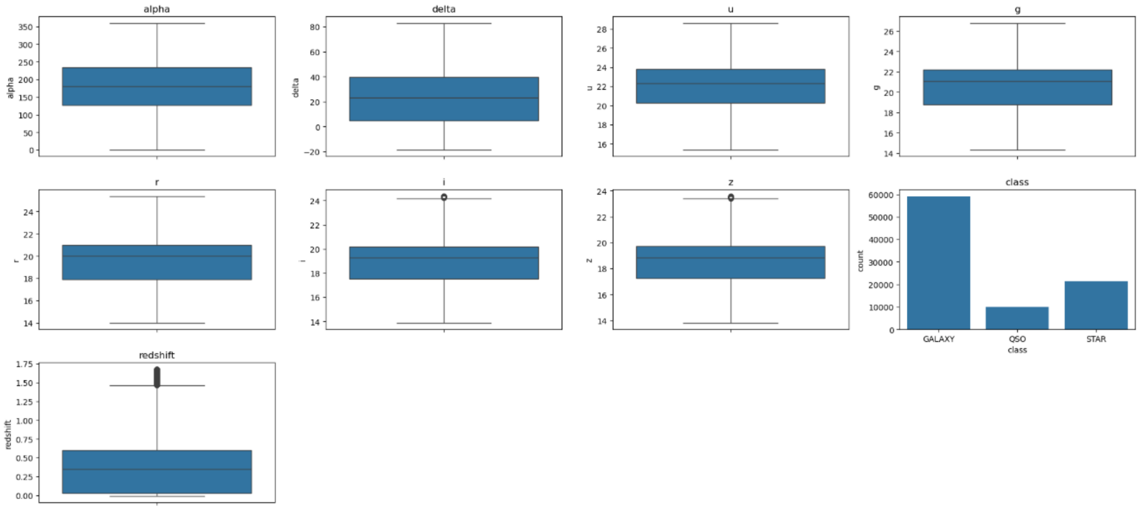
90600 rows × 9 columns

[22]:
```
plt.figure(figsize=(20, 12))
for index, column in enumerate(data.columns):
    plt.subplot(3,3,index+1)
    if column != "class":
        sns.boxplot(data, y=column)
    else:
        sns.countplot(data, x=column)
plt.tight_layout()
plt.show()
```


alpha
delta
u
g
r
i
z
count
GALAXY
QSO
STAR
class
redshift

6. Box Plots by Class

[203]:
```
plt.figure(figsize=(20, 12))

# Loop over columns with manual index
plot_index = 1
for column in data.columns:
    plt.subplot(4, 4, plot_index)
    if column != "class":
        sns.boxplot(data=data, y=column)
    else:
        sns.countplot(data=data, x=column)
    plt.title(column)
    plot_index += 1

plt.tight_layout()
plt.show()
```



[26]:
```
data.columns
```

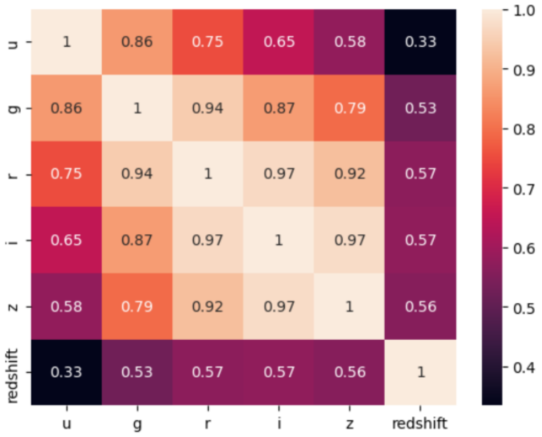
[26]: Index(['alpha', 'delta', 'u', 'g', 'r', 'i', 'z', 'class', 'redshift'], dtype='object')

[28]:
```
X = data.drop(columns=["alpha", "delta", "class"])
y = data["class"]
```

7. Feature Correlation Heatmap

[30]:
```
sns.heatmap(X.corr(), annot=True)
```

[30]: <Axes: >



8. Feature Scaling with StandardScaler & PCA

[34]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

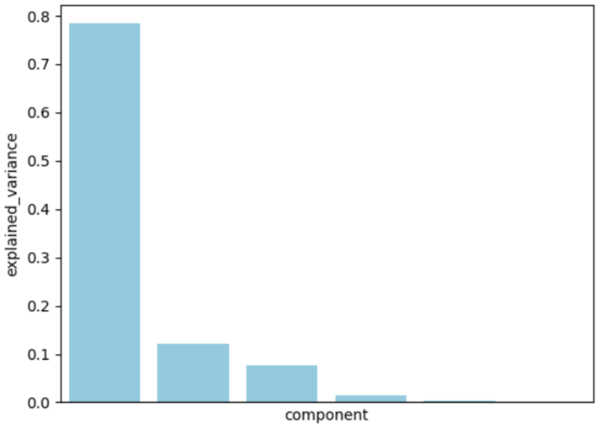
[34]: ((67950, 6), (22650, 6), (67950,), (22650,))

[36]:
```
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)
```

[38]:
```
pca = PCA()
pca.fit(X_train)
pca.explained_variance_ratio_
```

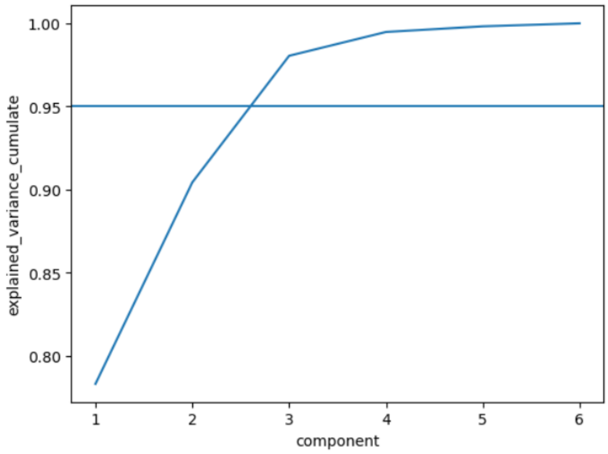
[38]:
```
array([0.78320804, 0.12115768, 0.07611468, 0.01432049, 0.0033905 ,
       0.00180861])
```

•[40]:
```
exp_variance = pd.DataFrame(pca.explained_variance_ratio_)
exp_variance["component"] = exp_variance.index + 1
exp_variance = exp_variance.rename(columns={0: "explained_variance"})
sns.barplot(exp_variance, x="component", y = "explained_variance", color="skyblue")
plt.xticks([])
plt.show()
```



9. PCA Explained Variance Plot

```
•[42]: exp_variance["explained_variance_cumulate"] = exp_variance["explained_variance"].cumsum() #Cummulative sum
sns.lineplot(exp_variance, x = "component", y="explained_variance_cumulate")
plt.axhline(0.95)
plt.show()
```




```
[44]: pca = PCA(4)
X_train_trans = pca.fit_transform(X_train)
X_test_trans = pca.transform(X_test)
```

```
[46]: X_train
```

```
[46]: array([[ 0.51829521,  2.14741077,  1.1903411 ,  0.92327428,  0.63616709,
          0.96135466],
        [-0.10028117, -0.46743038, -0.72389907, -0.76336732, -0.81561899,
         -0.49516694],
        [ 1.3364528 ,  1.06667144,  0.94628193,  0.76310088,  0.57521406,
          0.52650634],
        ...,
        [-0.33754508,  0.30351404,  0.8244193 ,  1.23728634,  0.94353157,
          1.09553655],
        [-1.07847234, -1.39909211, -1.47313316, -1.48857262, -1.48153449,
         -0.8257778 ],
        [-0.00241445,  0.15893284, -0.22352822, -0.2816943 , -0.31799461,
         -0.06934532]])
```

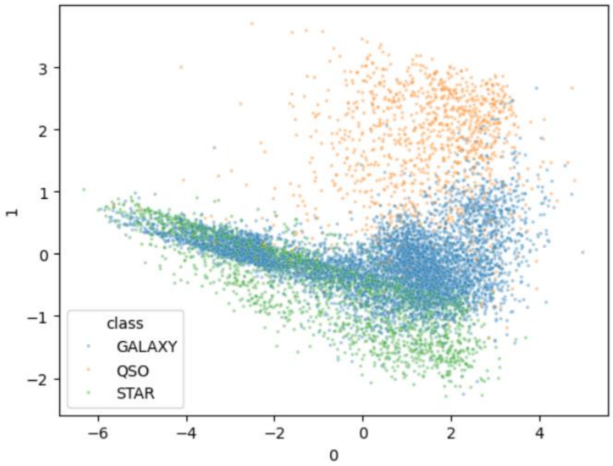
```
[48]: sns.scatterplot(pd.DataFrame(X_train_trans, index=y_train.index).sample(10000), x=0, y=1, hue=y_train , alpha=0.4, s = 5)
```

```
[48]: <Axes: xlabel='0', ylabel='1'>
```

10. PCA Scatter Plot (2D Visualization)

```
[48]: sns.scatterplot(pd.DataFrame(X_train_trans, index=y_train.index).sample(10000), x=0, y=1, hue=y_train , alpha=0.4, s = 5)
```

```
[48]: <Axes: xlabel='0', ylabel='1'>
```



11. Logistic Regression Model Training and Evaluation

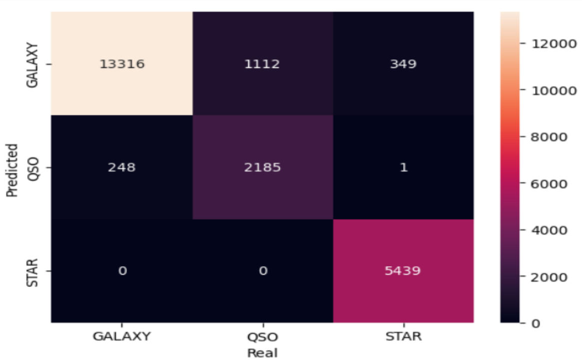
```
[52]: lr = LogisticRegression(class_weight="balanced", max_iter=1000)
cv_score = cross_validate(lr, X_train, y_train, scoring="roc_auc_ovr")["test_score"]
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
y_prob = lr.predict_proba(X_test)
cv_score.mean(), cv_score.std(), roc_auc_score(y_test, y_prob, multi_class="ovr")
```

```
[52]: (0.9825675841499691, 0.0013354704058210329, 0.9838202611960023)
```

```
[54]: lr = LogisticRegression(class_weight="balanced")
cv_score = cross_validate(lr, X_train_trans, y_train, scoring="roc_auc_ovr")["test_score"]
lr.fit(X_train_trans, y_train)
y_pred = lr.predict(X_test_trans)
y_prob = lr.predict_proba(X_test_trans)
print(classification_report(y_test, y_pred))
```

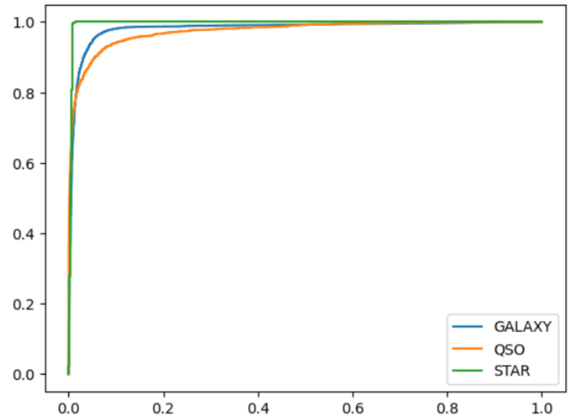
```
              precision    recall  f1-score   support

      GALAXY       0.98      0.90      0.94     14777
         QSO       0.66      0.90      0.76      2434
        STAR       0.94      1.00      0.97      5439

    accuracy                           0.92     22650
   macro avg       0.86      0.93      0.89     22650
weighted avg       0.94      0.92      0.93     22650
```

```
[56]: y_pred = lr.predict(X_test_trans)
cm = pd.DataFrame(confusion_matrix(y_test, y_pred), columns=lr.classes_, index=lr.classes_)
sns.heatmap(cm, annot=True, fmt=".0f")
plt.xlabel("Real")
plt.ylabel("Predicted")
plt.show()
```



12. Confusion Matrix Visualization

[58]:
```
from sklearn.metrics import roc_curve
fpr, tpr, threshold = roc_curve(y_test == "GALAXY", y_prob[:,0])
plt.plot(fpr, tpr, label="GALAXY")

fpr, tpr, threshold = roc_curve(y_test == "QSO", y_prob[:,1])
plt.plot(fpr, tpr, label="QSO")

fpr, tpr, threshold = roc_curve(y_test == "STAR", y_prob[:,2])
plt.plot(fpr, tpr, label="STAR")
plt.legend()
plt.show()
```



[60]:
```
coefs = pd.DataFrame(lr.coef_)
coefs
```

[60]:

	0	1	2	3
0	4.315506	10.749458	8.143111	-0.189914
1	4.485682	13.586592	7.297307	-1.582222
2	-8.801189	-24.336051	-15.440417	1.772136

13. Model Coefficients Table

[62]:
```
coefs = coefs.T.rename(columns={0: "Galaxy", 1: "Star", 2: "QSR"}).reset_index()
coefs
```

[62]:

	index	Galaxy	Star	QSR
0	0	4.315506	4.485682	-8.801189
1	1	10.749458	13.586592	-24.336051
2	2	8.143111	7.297307	-15.440417
3	3	-0.189914	-1.582222	1.772136

14. Reshaping Coefficients for Visualization

[64]:
```python
melted = coefs.melt(id_vars="index")
melted
```

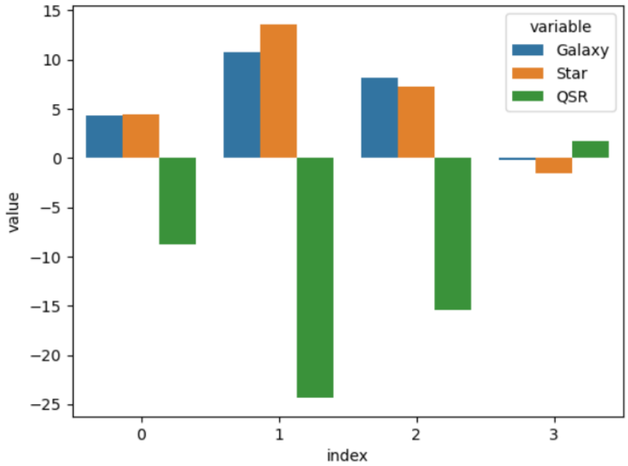
[64]:

	index	variable	value
0	0	Galaxy	4.315506
1	1	Galaxy	10.749458
2	2	Galaxy	8.143111
3	3	Galaxy	-0.189914
4	0	Star	4.485682
5	1	Star	13.586592
6	2	Star	7.297307
7	3	Star	-1.582222
8	0	QSR	-8.801189
9	1	QSR	-24.336051
10	2	QSR	-15.440417
11	3	QSR	1.772136

15. Feature Coefficients Bar Plot

```
[66]: sns.barplot(melted, y="value",x="index", hue="variable")
```

```
[66]: <Axes: xlabel='index', ylabel='value'>
```



16. Combining Predictions with Features

```
[68]: sub_data = pd.DataFrame(X_test_trans)
sub_data["y_pred"] = y_pred
sub_data["y_true"] = y_test.reset_index(drop=True)
sub_data
```

[68]:

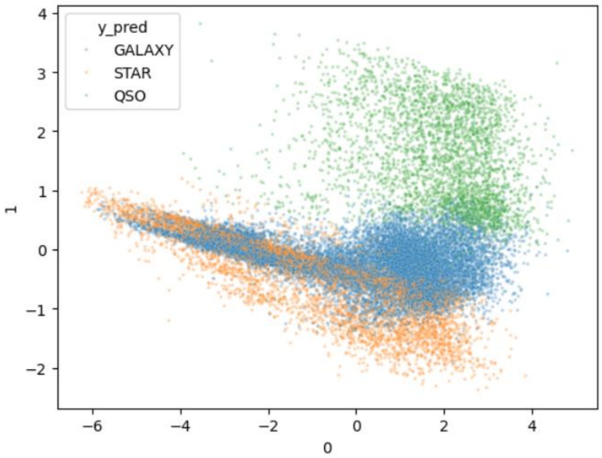
	0	1	2	3	y_pred	y_true
0	1.492390	-0.513964	0.984640	-0.274023	GALAXY	GALAXY
1	0.342604	-0.522680	-1.335681	-0.104420	STAR	STAR
2	1.694761	-0.903614	0.987576	-0.346962	GALAXY	GALAXY
3	0.646426	-0.217004	0.337338	0.206549	GALAXY	GALAXY
4	-1.730782	0.031789	-0.862782	-0.008830	STAR	GALAXY
...	...	...	...	...	...	...
22645	1.408758	-0.860379	0.872890	-0.496899	GALAXY	GALAXY
22646	3.339496	1.287065	-0.252630	0.158705	QSO	GALAXY
22647	1.094340	-0.356428	0.315394	0.104988	GALAXY	GALAXY
22648	0.963203	-0.712679	-1.426644	-0.283686	STAR	STAR
22649	2.045158	0.084790	-0.179530	0.357387	GALAXY	GALAXY

22650 rows × 6 columns

17. Scatter Plot of Predicted Classes

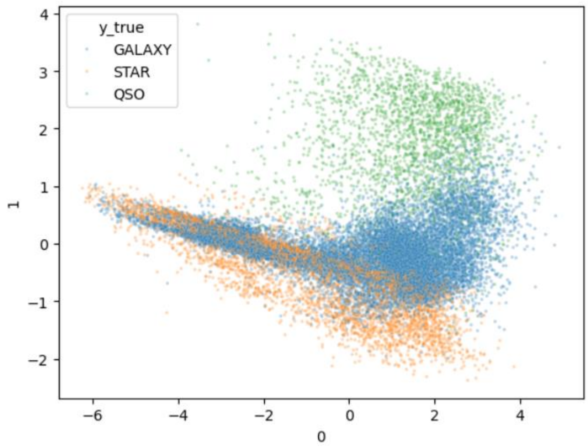
[70]:
```
sns.scatterplot(sub_data, x=0, y=1, hue="y_pred", alpha=0.3, s=5)
```

[70]: <Axes: xlabel='0', ylabel='1'>



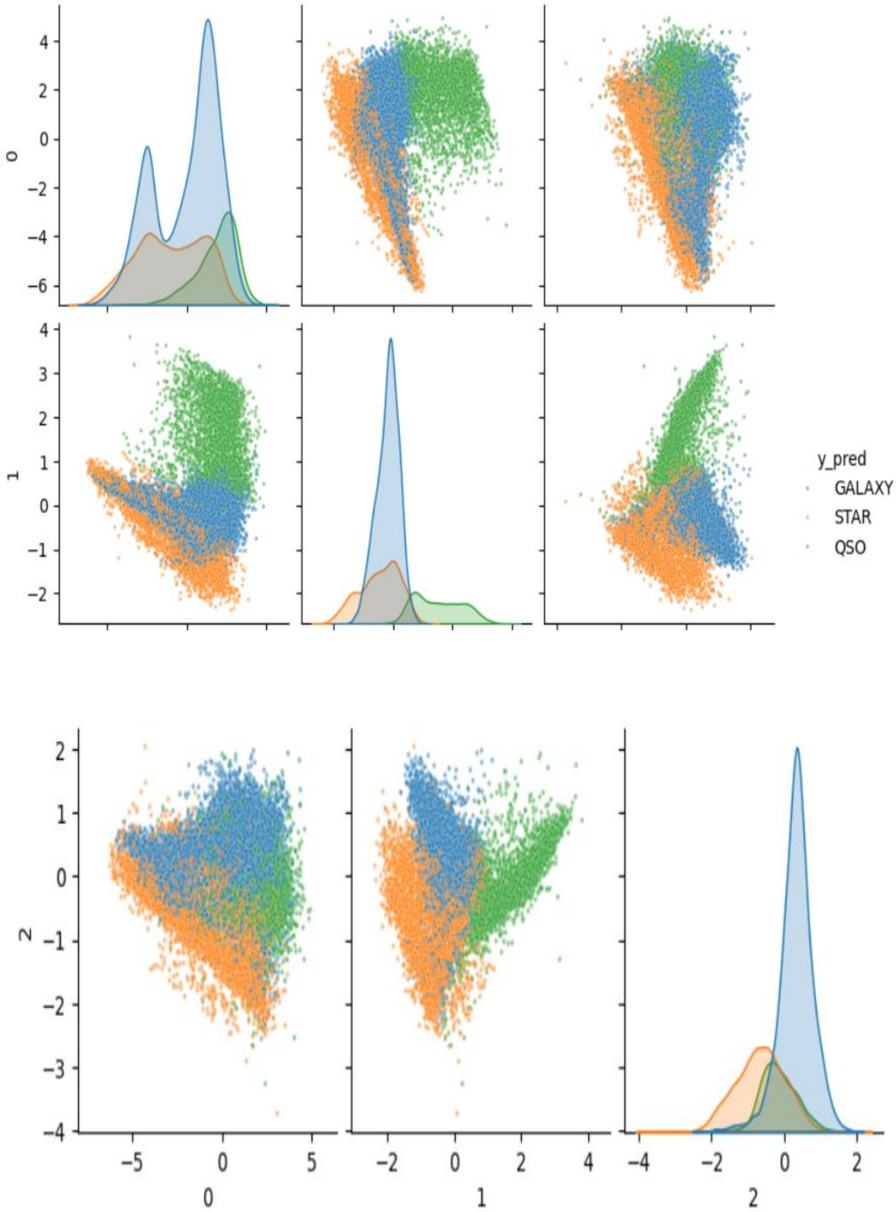
[72]:
```
sns.scatterplot(sub_data, x=0, y=1, hue="y_true", alpha=0.3, s=5)
```

[72]: <Axes: xlabel='0', ylabel='1'>



```
[74]: sns.pairplot(sub_data[[0,1,2,"y_pred"]], hue="y_pred", plot_kws={"alpha":0.5, "s": 5, "hue_order": ["QSO", "GALAXY", "STAR"]})
```

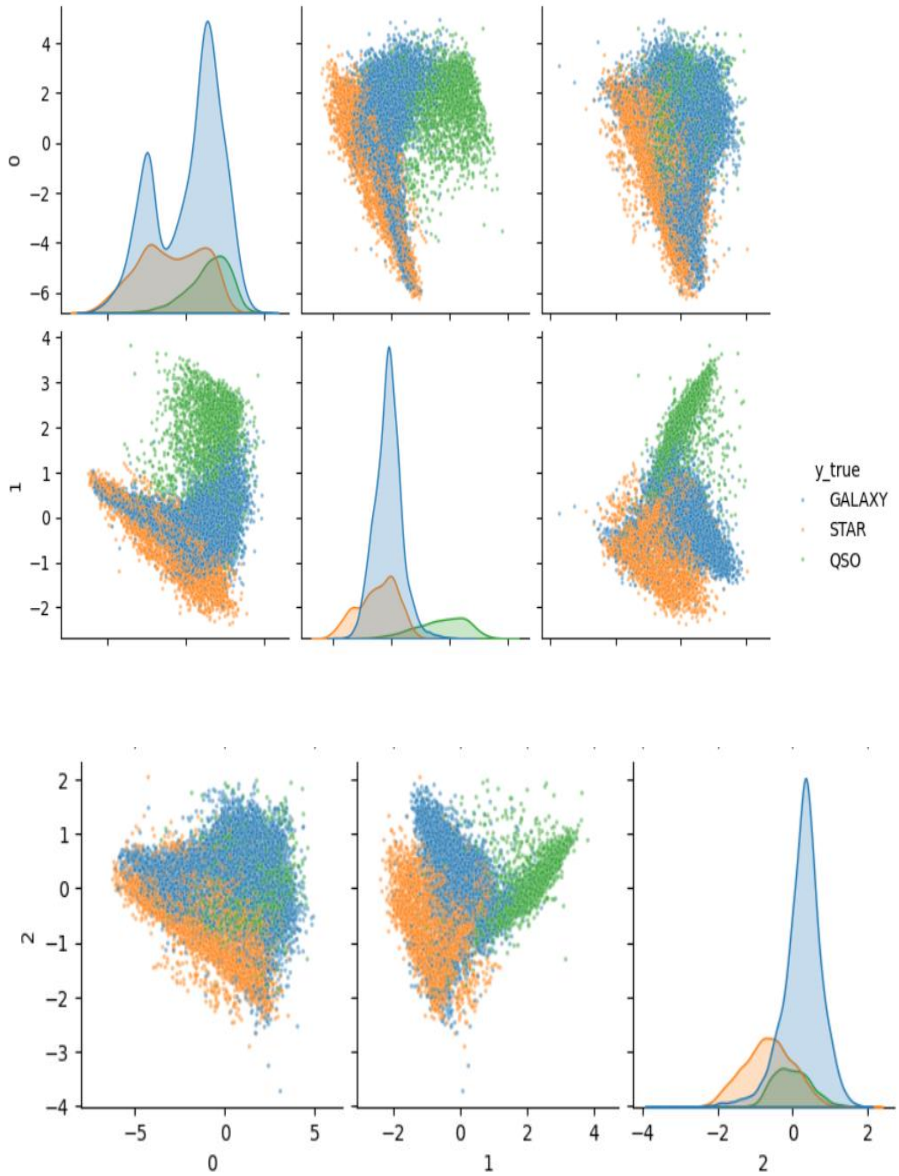
```
[74]: <seaborn.axisgrid.PairGrid at 0x20281ea54c0>
```



[96]:

```
sns.pairplot(sub_data[[0,1,2,"y_true"]], hue="y_true", plot_kws={"alpha":0.5, "s": 5, "hue_order": ["QSO", "GALAXY", "STAR"]})
```

[96]: <seaborn.axisgrid.PairGrid at 0x2028a4b95b0>



[156]:

```
print(data.columns.tolist())
```

```
['alpha', 'delta', 'u', 'g', 'r', 'i', 'z', 'class', 'redshift']
```

[157]:

```
# Load data
df = pd.read_csv('star.csv')
df.columns = df.columns.str.strip()

# Encode target
class_encoder = LabelEncoder()
df['class'] = class_encoder.fit_transform(df['class'])

# Features and target
X = df[['u', 'g', 'r', 'i', 'z', 'redshift']]
y = df['class']

# Scale
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[158]:

```
custom_input = pd.DataFrame([{
    'u': 18.5,
    'g': 17.8,
    'r': 17.2,
    'i': 16.9,
    'z': 16.7,
    'redshift': 0.01
}])


custom_input_scaled = scaler.transform(custom_input)


prediction = model.predict(custom_input_scaled)
predicted_class = class_encoder.inverse_transform(prediction)[0]

print("Predicted Class:", predicted_class)
```

```
Predicted Class: GALAXY
```