###**Machine Learing Project** ###

Laptop Price Prediction for SmartTech Co

1 **Project Introduction**

SmartTech Co. aims to strengthen its market position by understanding and predicting laptop prices using data-driven insights. With hundreds of models from various brands differing in RAM, CPU, storage, GPU, screen type, and operating systems, determining the right price is complex. Hence, this project develops a machine learning model that accurately predicts laptop prices and provides meaningful insights into the factors that influence pricing

2 Dataset Overview

The dataset (laptop.csv) contains detailed specifications of laptops collected from different brands. Each record represents a laptop configuration along with its price. | Column Name | Description | Type | | ---------- | ---------- | | Company | Brand or manufacturer of the laptop | Categorical | | TypeName | Category of laptop (e.g., Ultrabook, Gaming, Notebook) | Categorical | | Inches | Screen size of the laptop in inches | Numerical | | ScreenResolution | Screen resolution and display type (e.g., 1920x1080 Touchscreen) | Text | | Cpu | Processor details (brand and model) | Text | | Ram | Size of the RAM in GB | Numerical | | Memory | Storage configuration (SSD, HDD, eMMC, Flash) | Text | | Gpu | Graphics processor (brand and model) | Text | | OpSys | Operating system of the laptop | Categorical | | Weight | Weight of the laptop in kilograms | Numerical | | Price | Target variable (price of the laptop) | Numerical |

3 **Business Understanding**

Business Problem

SmartTech Co. faces challenges in:

Accurately pricing laptops with diverse configurations.

Understanding the impact of specifications and brand on pricing.

Quickly evaluating new models' prices to stay competitive.

Business Objective

Build a predictive model to estimate laptop prices based on features.

Derive data insights to understand which features most influence pricing.

Enable real-time prediction for new models.

4 Machine Learning Project Workflow | Phase | Description | | ---------- | ---------- | | *Phase 1: Data Understanding* | Explore dataset shape, data types, missing values, and

distributions. | | *Phase 2: Data Preprocessing* | Handle missing data, clean text, encode categorical variables, and normalize features. | | *Phase 3: Feature Engineering* | Extract numeric and categorical insights like resolution, storage type, and processor brand. | | *Phase 4: Model Development* | Train multiple models (Linear Regression, Random Forest, XGBoost). | | *Phase 5: Model Evaluation* | Evaluate models using RMSE, MAE, and R^2 metrics. | | *Phase 6: Hyperparameter Tuning* | Optimize the best-performing model using GridSearchCV. | | *Phase 7: Insights & Interpretability* | Analyze feature importance and business takeaways. | | *Phase 8: Deployment Ready Model* | Save the best model for real-time predictions. |

5 Success Criteria | Type | Description | Example Metric | | ---------- | ---------- | ---------- | | *Business Success* | Enable SmartTech Co. to predict prices accurately and gain competitive advantage. | Decision confidence & faster pricing | | *ML Success* | Achieve high R^2 and low RMSE on test data. | $R^2 \geq 0.75$, RMSE ≤ ₹20,000 | | *Economic Success* | Reduce overpricing/underpricing losses. | Increased profit margins & market share |

6 Data Understanding & Insights

Initial data analysis showed:

Price increases with RAM, SSD, and screen resolution.

Gaming laptops and high PPI screens are priced higher.

Brand plays a significant role — Apple and MSI laptops tend to be premium.

Operating System: Windows-based laptops dominate the dataset.

Weight: Ultrabooks are lighter but more expensive due to design factors.

```
import pandas as pd
import numpy as np

df = pd.read_csv(r"/content/laptop.csv")
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 1303,\n  \"fields\":
[\n    {\n      \"column\": \"Unnamed: 0.1\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 376,\n
\"min\": 0,\n        \"max\": 1302,\n        \"num_unique_values\":
1303,\n        \"samples\": [\n          479,\n          1022,\n
298\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Unnamed: 0\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 376.4930271562809,\n        \"min\":
0.0,\n        \"max\": 1302.0,\n        \"num_unique_values\": 1273,\n
\"samples\": [\n          44.0,\n          1188.0,\n          133.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Company\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 19,\n        \"samples\": [\n
\"Apple\",\n          \"Lenovo\",\n          \"Xiaomi\"\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TypeName\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
6,\n        \"samples\": [\n          \"Ultrabook\",\n
\"Notebook\",\n          \"Netbook\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Inches\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
25,\n        \"samples\": [\n          \"13\",\n          \"24\",\n
\"13.3\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ScreenResolution\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 40,\n
\"samples\": [\n          \"Touchscreen 1366x768\",\n
\"1600x900\",\n          \"4K Ultra HD / Touchscreen 3840x2160\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Cpu\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\":
118,\n        \"samples\": [\n          \"AMD A12-Series 9720P
2.7GHz\",\n          \"Intel Atom Z8350 1.92GHz\",\n          \"Intel
Core i5 3.1GHz\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Ram\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n
\"24GB\",\n          \"16GB\",\n          \"64GB\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Memory\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
40,\n        \"samples\": [\n          \"256GB SSD +  500GB HDD\",\n
\"1.0TB Hybrid\",\n          \"64GB SSD\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Gpu\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 110,\n
\"samples\": [\n          \"AMD Radeon R7\",\n          \"AMD Radeon
Pro 555\",\n          \"Intel Iris Plus Graphics 650\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"OpSys\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\": 9,\n
\"samples\": [\n          \"Windows 7\",\n          \"No OS\",\n
\"Windows 10 S\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Weight\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 189,\n
\"samples\": [\n          \"2.21kg\",\n          \"3.74kg\",\n
\"1.65kg\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Price\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 37332.251004569865,\n        \"min\": 9270.72,\n
\"max\": 324954.72,\n        \"num_unique_values\": 777,\n
```

\"samples\": [\n   68464.8,\n   71341.92,\n
77788.8\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"df"}

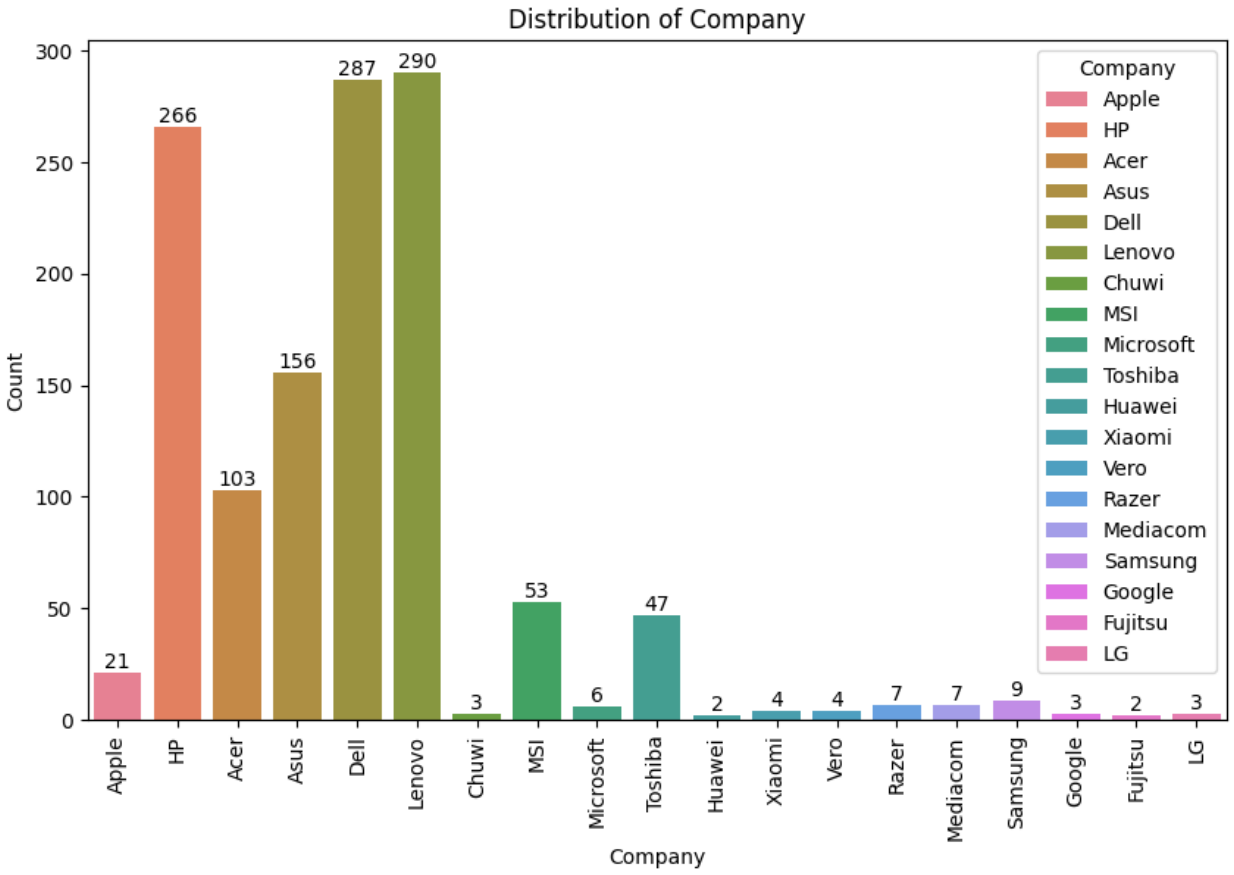
```
# drop the unwanted colums
df.drop(columns=['Unnamed: 0','Unnamed: 0.1'],axis=1, inplace=True)
df.head()
```

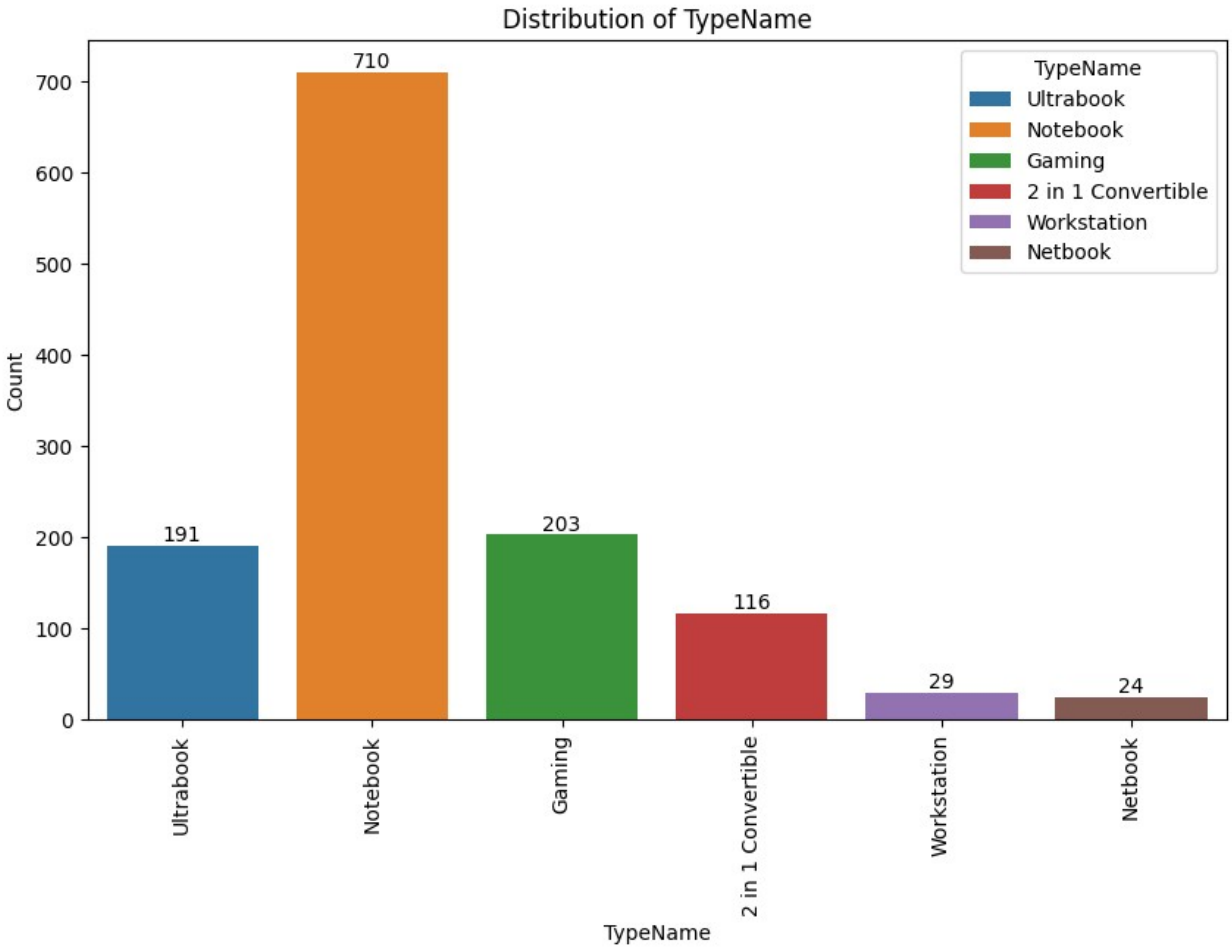
{"summary":"{\n \"name\": \"df\",\n \"rows\": 1303,\n \"fields\":
[\n   {\n   \"column\": \"Company\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 19,\n
\"samples\": [\n   \"Apple\",\n   \"Lenovo\",\n
\"Xiaomi\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"TypeName\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 6,\n   \"samples\":
[\n   \"Ultrabook\",\n   \"Notebook\",\n
\"Netbook\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Inches\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 25,\n
\"samples\": [\n   \"13\",\n   \"24\",\n
\"13.3\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ScreenResolution\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 40,\n
\"samples\": [\n   \"Touchscreen 1366x768\",\n
\"1600x900\",\n   \"4K Ultra HD / Touchscreen 3840x2160\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Cpu\",\n   \"properties\": {\
n   \"dtype\": \"category\",\n   \"num_unique_values\":
118,\n   \"samples\": [\n   \"AMD A12-Series 9720P
2.7GHz\",\n   \"Intel Atom Z8350 1.92GHz\",\n   \"Intel
Core i5 3.1GHz\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Ram\",\n   \"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n
\"24GB\",\n   \"16GB\",\n   \"64GB\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Memory\",\n   \"properties\":
{\n   \"dtype\": \"category\",\n   \"num_unique_values\":
40,\n   \"samples\": [\n   \"256GB SSD + 500GB HDD\",\n
\"1.0TB Hybrid\",\n   \"64GB SSD\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Gpu\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 110,\n
\"samples\": [\n   \"AMD Radeon R7\",\n   \"AMD Radeon
Pro 555\",\n   \"Intel Iris Plus Graphics 650\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

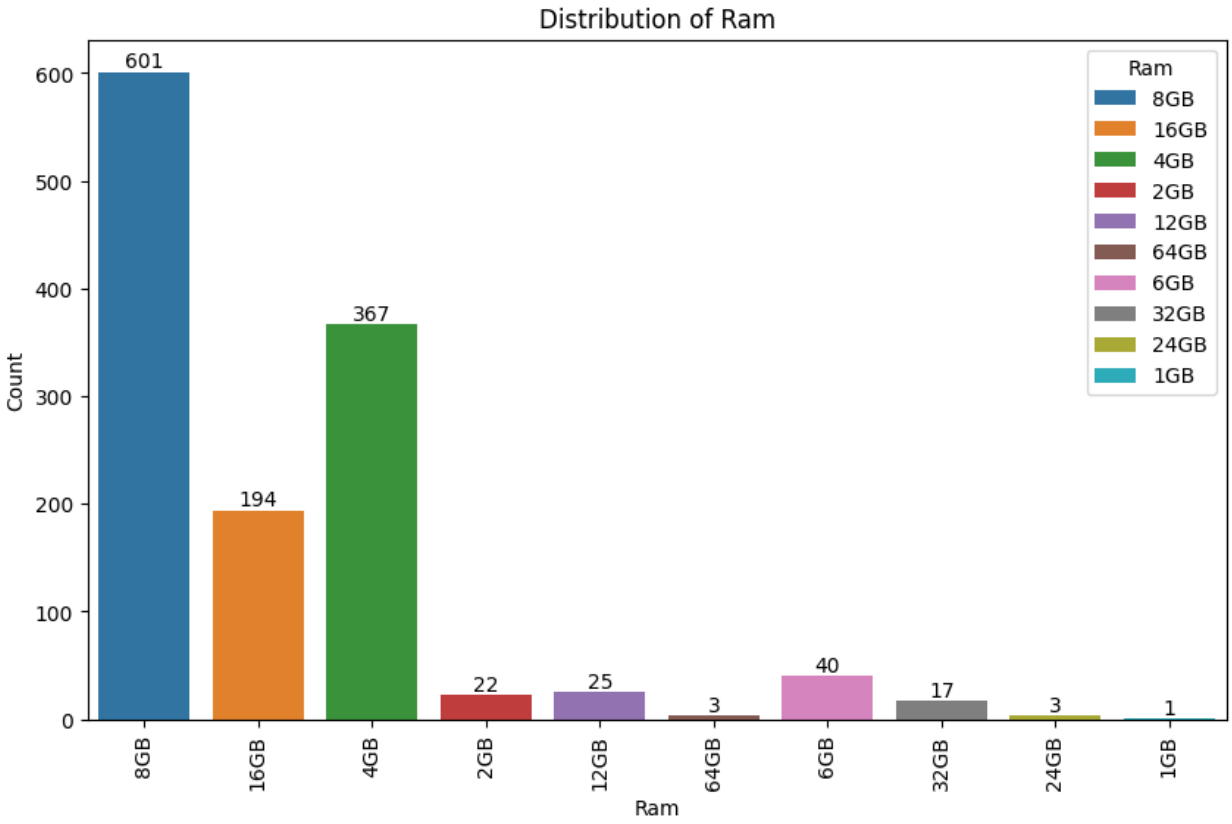
```
n    },\n    {\n      \"column\": \"OpSys\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\": 9,\n
\"samples\": [\n          \"Windows 7\",\n          \"No OS\",\n
\"Windows 10 S\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Weight\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 189,\n
\"samples\": [\n          \"2.21kg\",\n          \"3.74kg\",\n
\"1.65kg\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Price\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 37332.251004569865,\n        \"min\": 9270.72,\n
\"max\": 324954.72,\n        \"num_unique_values\": 777,\n
\"samples\": [\n          68464.8,\n          71341.92,\n
77788.8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```

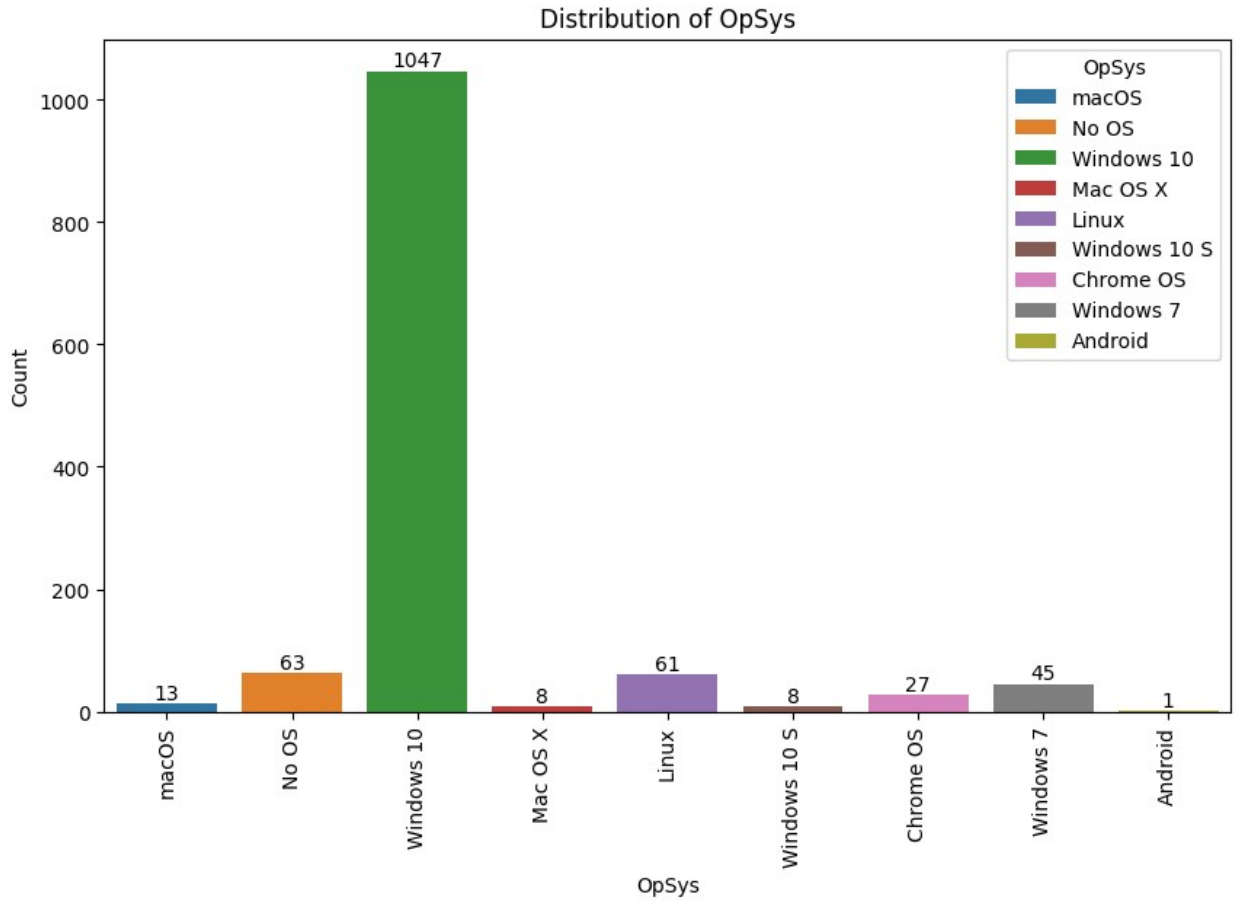

```
import numpy as np
# Replace "?" with np.nan
df = df.replace('?', np.nan)
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Distribution of categorical variables
categorical_cols = ['Company', 'TypeName','Ram','OpSys']
for col in categorical_cols:
    plt.figure(figsize=(10, 6))
    ax = sns.countplot(x=col, data=df,hue=col)
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.xticks(rotation=90)
    ## add data label
    for container in ax.containers:
        ax.bar_label(container)
    plt.show()
```


Distribution of Company
Count
Company
Apple
HP
Acer
Asus
Dell
Lenovo
Chuwi
MSI
Microsoft
Toshiba
Huawei
Xiaomi
Vero
Razer
Mediacom
Samsung
Google
Fujitsu
LG
21
266
103
156
287
290
3
53
6
47
2
4
4
7
7
9
3
2
3
0
50
100
150
200
250
300

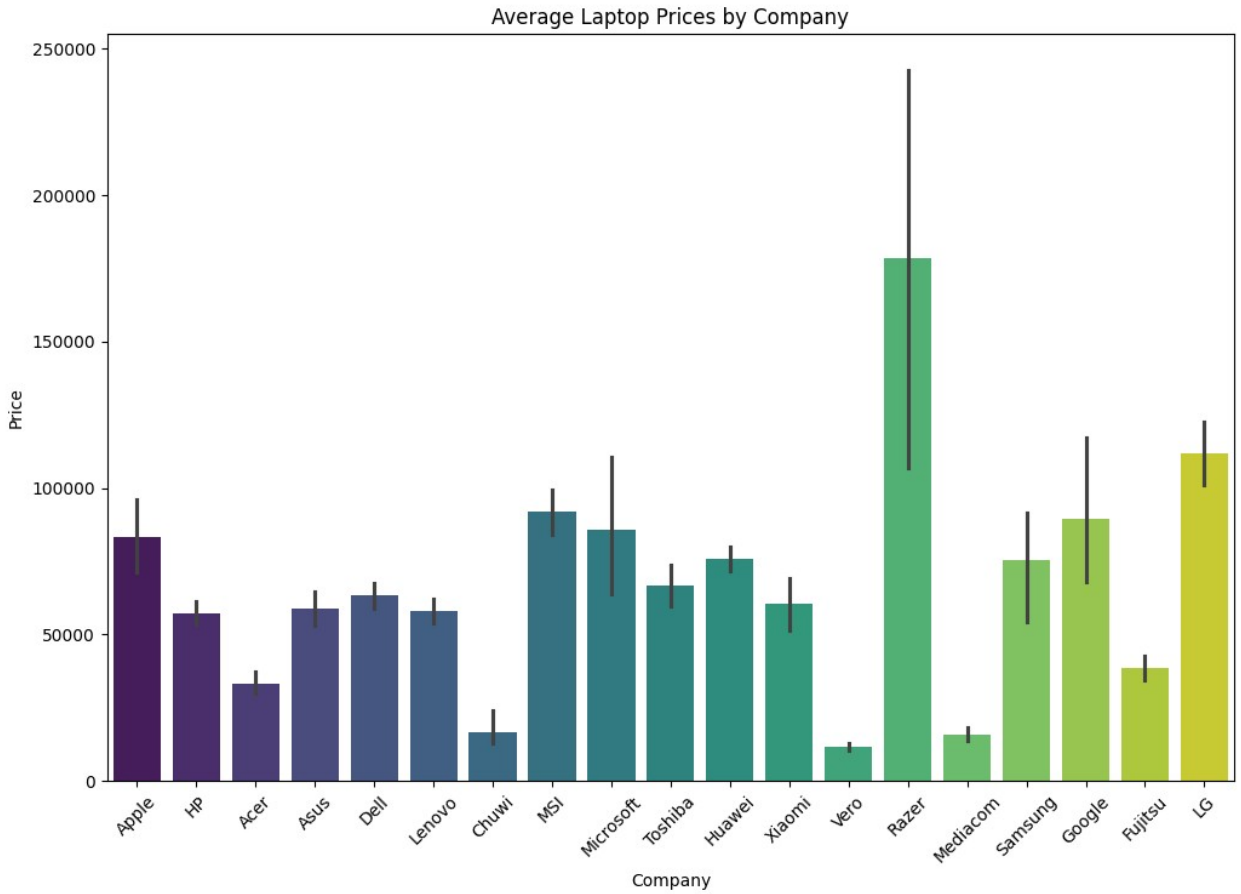

Distribution of TypeName
TypeName
Ultrabook
Notebook
Gaming
2 in 1 Convertible
Workstation
Netbook
Count
0
100
200
300
400
500
600
700
191
710
203
116
29
24
Ultrabook
Notebook
Gaming
2 in 1 Convertible
Workstation
Netbook
TypeName

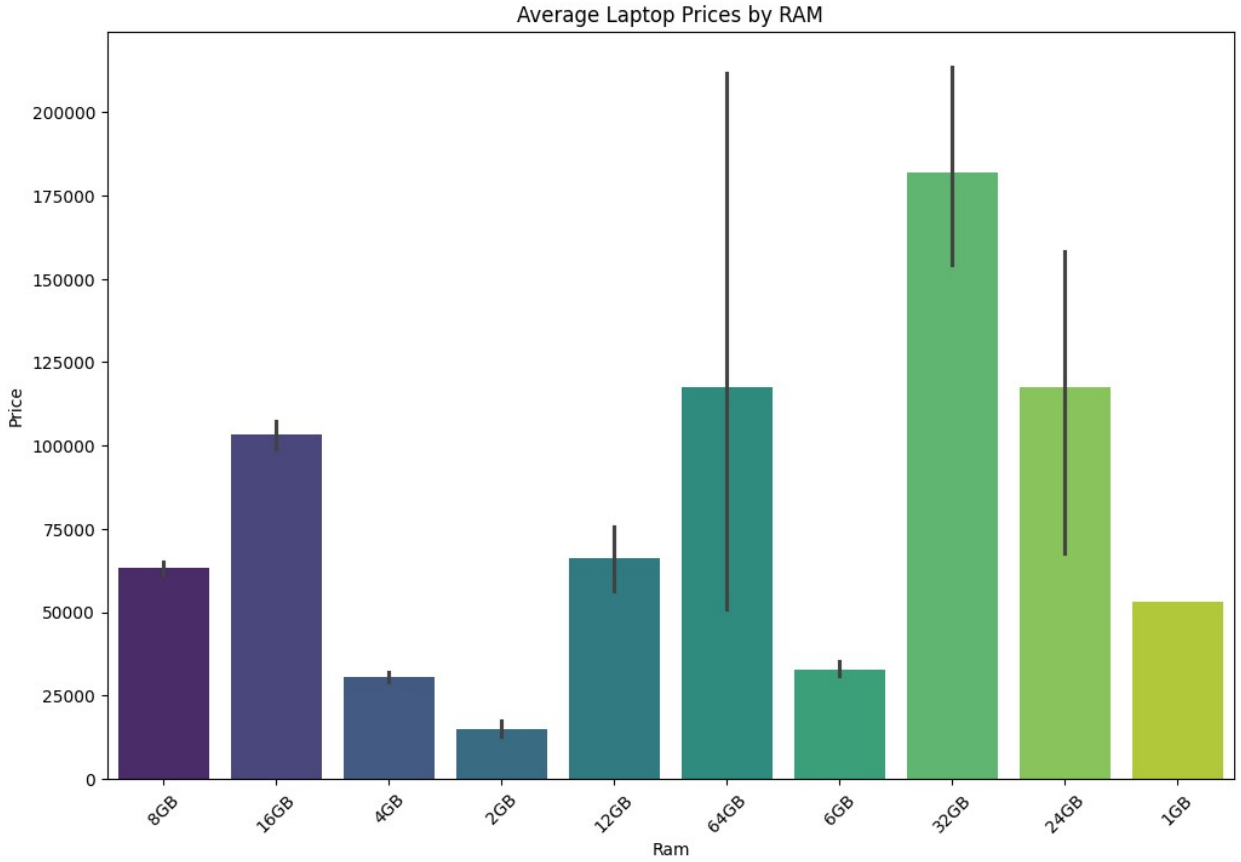

Distribution of Ram
Count
Ram
601
194
367
22
25
3
40
17
3
1
8GB
16GB
4GB
2GB
12GB
64GB
6GB
32GB
24GB
1GB
0
100
200
300
400
500
600

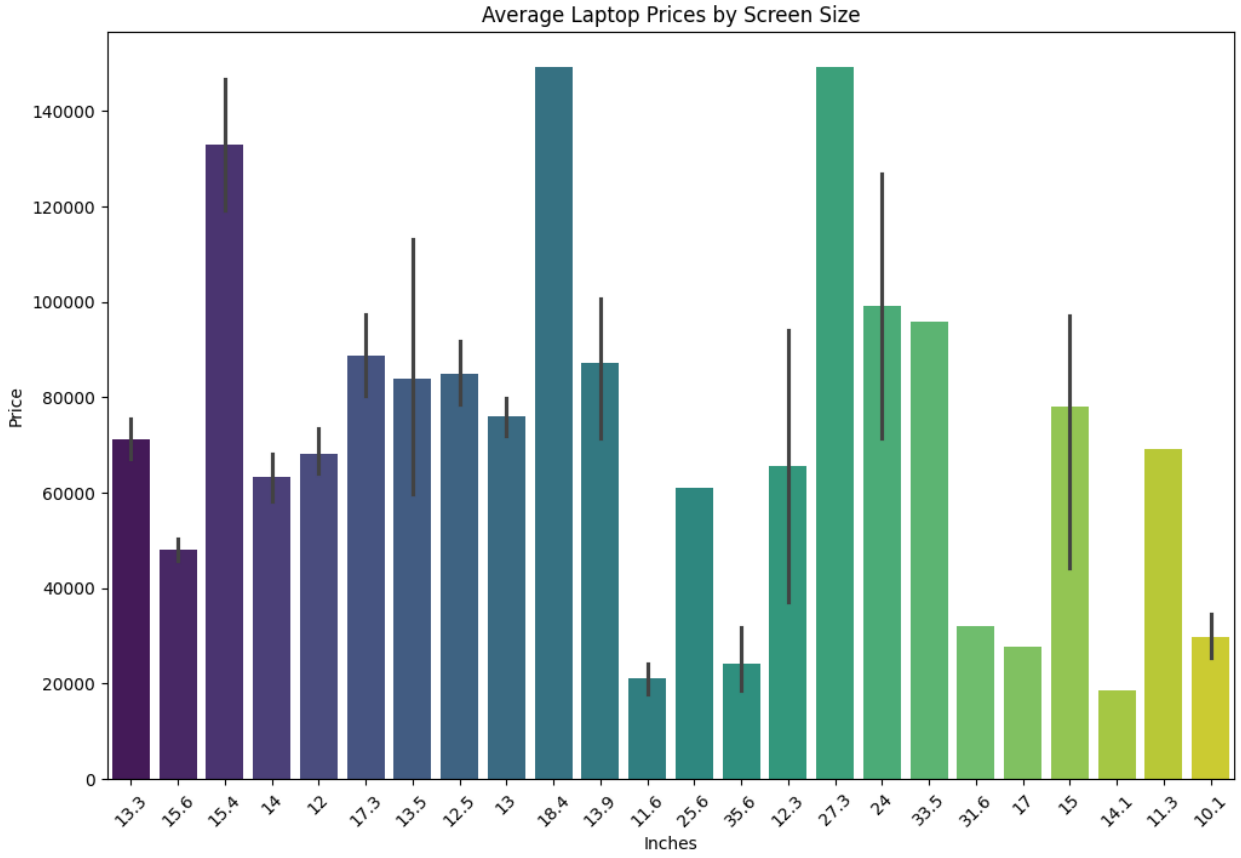


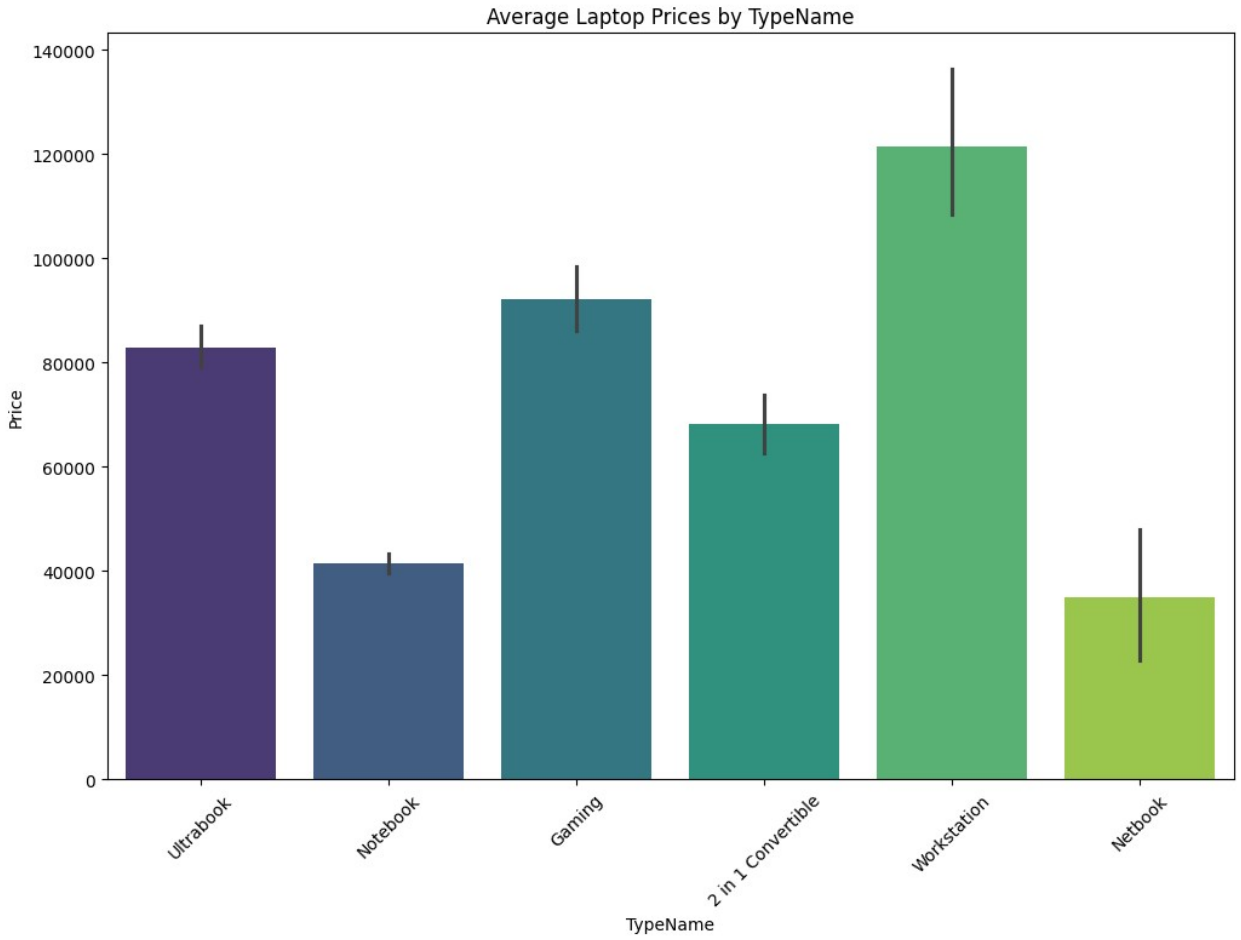

```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# Calculate the average price of laptops within each category
average_price_by_company = df.groupby('Company')
['Price'].mean().reset_index()
average_price_by_ram = df.groupby('Ram')['Price'].mean().reset_index()
average_price_by_inches = df.groupby('Inches')
['Price'].mean().reset_index()
average_price_by_inches = df.groupby('TypeName')
['Price'].mean().reset_index()
# Plot bar plots
plt.figure(figsize=(12, 8))
sns.barplot(x='Company', y='Price', data=df, estimator=lambda x:
sum(x) / len(x),palette='viridis')
plt.title('Average Laptop Prices by Company')
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(12, 8))
sns.barplot(x='Ram', y='Price', data=df, estimator=lambda x: sum(x) /
```

```
len(x),palette='viridis')
plt.title('Average Laptop Prices by RAM')
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(12, 8))
sns.barplot(x='Inches', y='Price', data=df, estimator=lambda x: sum(x)
/ len(x),palette='viridis')
plt.title('Average Laptop Prices by Screen Size')
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(12, 8))
sns.barplot(x='TypeName', y='Price', data=df, estimator=lambda x:
sum(x) / len(x),palette='viridis')
plt.title('Average Laptop Prices by TypeName')
plt.xticks(rotation=45)
plt.show()
```




Average Laptop Prices by RAM
Price
200000
175000
150000
125000
100000
75000
50000
25000
0
8GB
16GB
4GB
2GB
12GB
64GB
6GB
32GB
24GB
1GB
Ram


Average Laptop Prices by Screen Size
Price
0
20000
40000
60000
80000
100000
120000
140000
13.3
15.6
15.4
14
12
17.3
13.5
12.5
13
18.4
13.9
11.6
25.6
35.6
12.3
27.3
24
33.5
31.6
17
15
14.1
11.3
10.1
Inches



###**Checking null values**

```
df.isnull().sum()
```

```
Company             30
TypeName            30
Inches              31
ScreenResolution    30
Cpu                 30
Ram                 30
Memory              31
Gpu                 30
OpSys               30
Weight              31
Price               30
dtype: int64
```

###Dropped null values and checking after imputation

```
df.dropna(inplace=True)
print(df.isnull().sum())
```

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

###Feature Engineering

```
import pandas as pd
import numpy as np
# Define the function to extract CPU series
def extract_series(cpu):
    if pd.isna(cpu):
        return 'Unknown'
    if 'Intel Core i3' in cpu:
        return 'Intel Core i3'
    elif 'Intel Core i5' in cpu:
        return 'Intel Core i5'
    elif 'Intel Core i7' in cpu:
        return 'Intel Core i7'
    elif 'Intel Core M' in cpu:
        return 'Intel Core M'
    elif 'Intel Atom' in cpu:
        return 'Intel Atom'
    elif 'Intel Pentium' in cpu:
        return 'Intel Pentium'
    elif 'Intel Celeron' in cpu:
        return 'Intel Celeron'
    elif 'Intel Xeon' in cpu:
        return 'Intel Xeon'
    elif 'AMD E-Series' in cpu:
        return 'AMD E-Series'
    elif 'AMD A6-Series' in cpu:
        return 'AMD A6-Series'
    elif 'AMD A9-Series' in cpu:
        return 'AMD A9-Series'
    elif 'AMD A10-Series' in cpu:
```

```
        return 'AMD A10-Series'
    elif 'AMD A12-Series' in cpu:
        return 'AMD A12-Series'
    elif 'AMD FX' in cpu:
        return 'AMD FX'
    elif 'AMD Ryzen' in cpu:
        return 'AMD Ryzen'
    else:
        return 'Other'

# Apply the function to the 'Cpu' column and create a new column
'Cpu_Series'
df['Cpu_Series'] = df['Cpu'].apply(extract_series)

df['Touchscreen'] = df['ScreenResolution'].apply(lambda x:
'Touchscreen' if 'Touchscreen' in x else 'NonTouchscreen')
df['Touchscreen'].value_counts()
```

```
Touchscreen
NonTouchscreen    1085
Touchscreen        185
Name: count, dtype: int64
```

```
import pandas as pd

def categorize_os(os_name):
    if 'Windows 10' in os_name or 'Windows 7' in os_name or 'Windows
10 S' in os_name:
        return 'Windows'
    elif 'No OS' in os_name:
        return 'No OS'
    elif 'Mac OS X' in os_name or 'macOS' in os_name:
        return 'macOS'
    elif 'Linux' in os_name or 'Chrome OS' in os_name or 'Android' in
os_name:
        return 'other/Linux'
    else:
        return 'Unknown'

# Assuming 'df' is your DataFrame and 'OS' is the column containing
the OS names
df['OS'] = df['OpSys'].apply(categorize_os)

# Verify the result
print(df[['OpSys', 'OS']].head())
print(df['OS'].value_counts())
```

```
   OpSys     OS
0  macOS  macOS
1  macOS  macOS
2  No OS  No OS
```

```
3  macOS  macOS
4  macOS  macOS
OS
Windows        1097
other/Linux      89
No OS            63
macOS            21
Name: count, dtype: int64
```

```
def categorize_gpu(gpu_name):
    if 'Intel' in gpu_name:
        return 'Intel'
    elif 'AMD' in gpu_name:
        return 'AMD'
    elif 'Nvidia' in gpu_name or 'NVIDIA' in gpu_name:
        return 'Nvidia'
    elif 'ARM Mali' in gpu_name:
        return 'ARM Mali'
    else:
        return 'other'

# Apply the function to the 'Gpu' column and create a new column
'GpuCategory'
df['GpuCategory'] = df['Gpu'].apply(categorize_gpu)

# Verify the result
print(df[['Gpu', 'GpuCategory']].head())
print(df['GpuCategory'].value_counts())
```

```
                          Gpu GpuCategory
0  Intel Iris Plus Graphics 640       Intel
1        Intel HD Graphics 6000       Intel
2         Intel HD Graphics 620       Intel
3            AMD Radeon Pro 455         AMD
4  Intel Iris Plus Graphics 650       Intel
GpuCategory
Intel       702
Nvidia      392
AMD         175
ARM Mali      1
Name: count, dtype: int64
```

```
import re

# Function to extract SSD and HDD storage from memory string
def extract_storage(memory):
    ssd = 0
    hdd = 0
    flash_storage = 0
    hybrid = 0
```

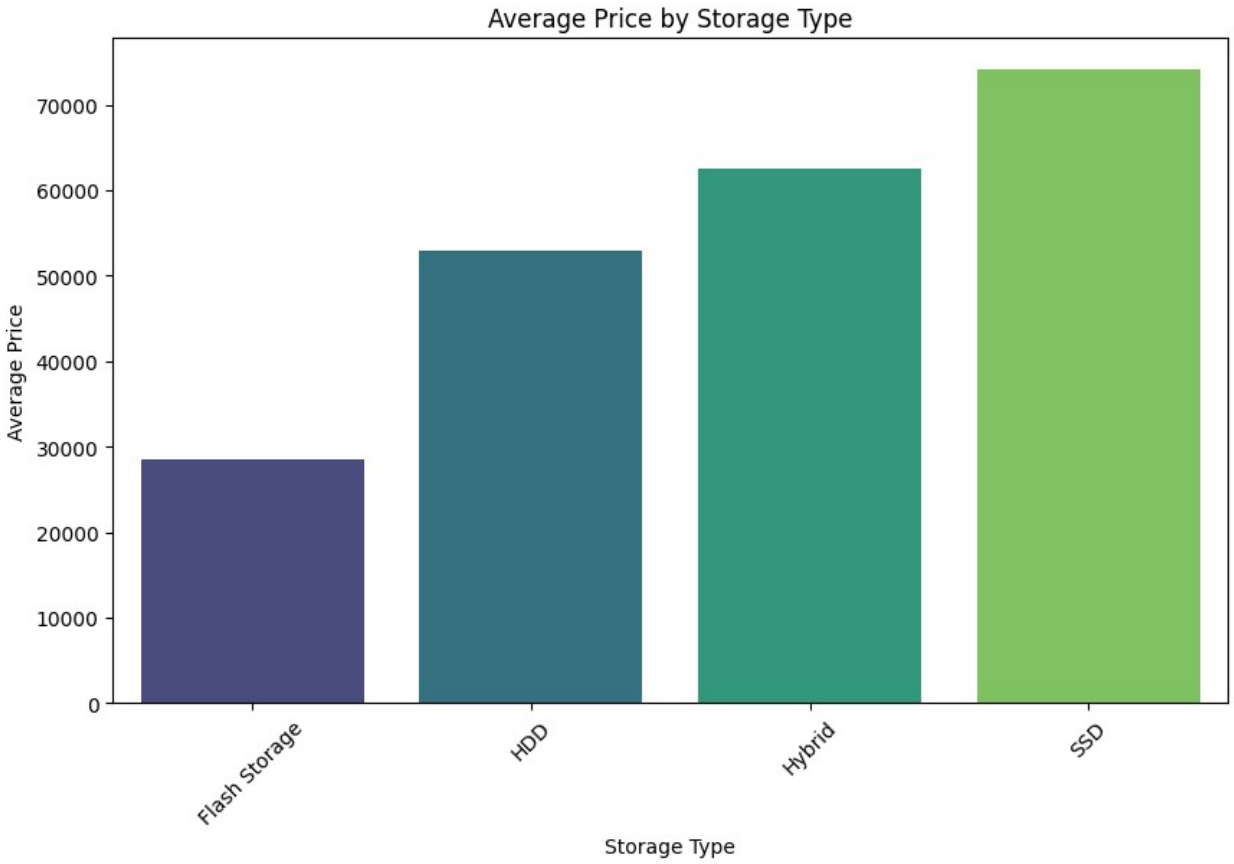
```
    ssd_match = re.findall(r'(\d+(?:\.\d+)?)(?=GB SSD|TB SSD)',
memory)
    hdd_match = re.findall(r'(\d+(?:\.\d+)?)(?=GB HDD|TB HDD)',
memory)
    flash_storage_match = re.findall(r'(\d+(?:\.\d+)?)(?=GB Flash
Storage|TB Flash Storage)', memory)
    hybrid_match = re.findall(r'(\d+(?:\.\d+)?)(?=GB Hybrid|TB
Hybrid)', memory)

    if ssd_match:
        ssd = sum(int(float(x.replace('TB', '1024').replace('GB',
''))) for x in ssd_match)

    if hdd_match:
        hdd = sum(int(float(x.replace('TB', '1024').replace('GB',
''))) for x in hdd_match)

    if flash_storage_match:
        flash_storage = sum(int(float(x.replace('TB',
'1024').replace('GB', ''))) for x in flash_storage_match)

    if hybrid_match:
        hybrid = sum(int(float(x.replace('TB', '1024').replace('GB',
''))) for x in hybrid_match)

    return pd.Series([ssd, hdd,flash_storage,hybrid])

# Apply the function to the DataFrame and create new columns
df[['SSD', 'HDD','Flash Storage','Hybrid']] =
df['Memory'].apply(extract_storage)

# Print the result
print(df[['Memory', 'SSD', 'HDD','Flash
Storage','Hybrid']].value_counts())
```

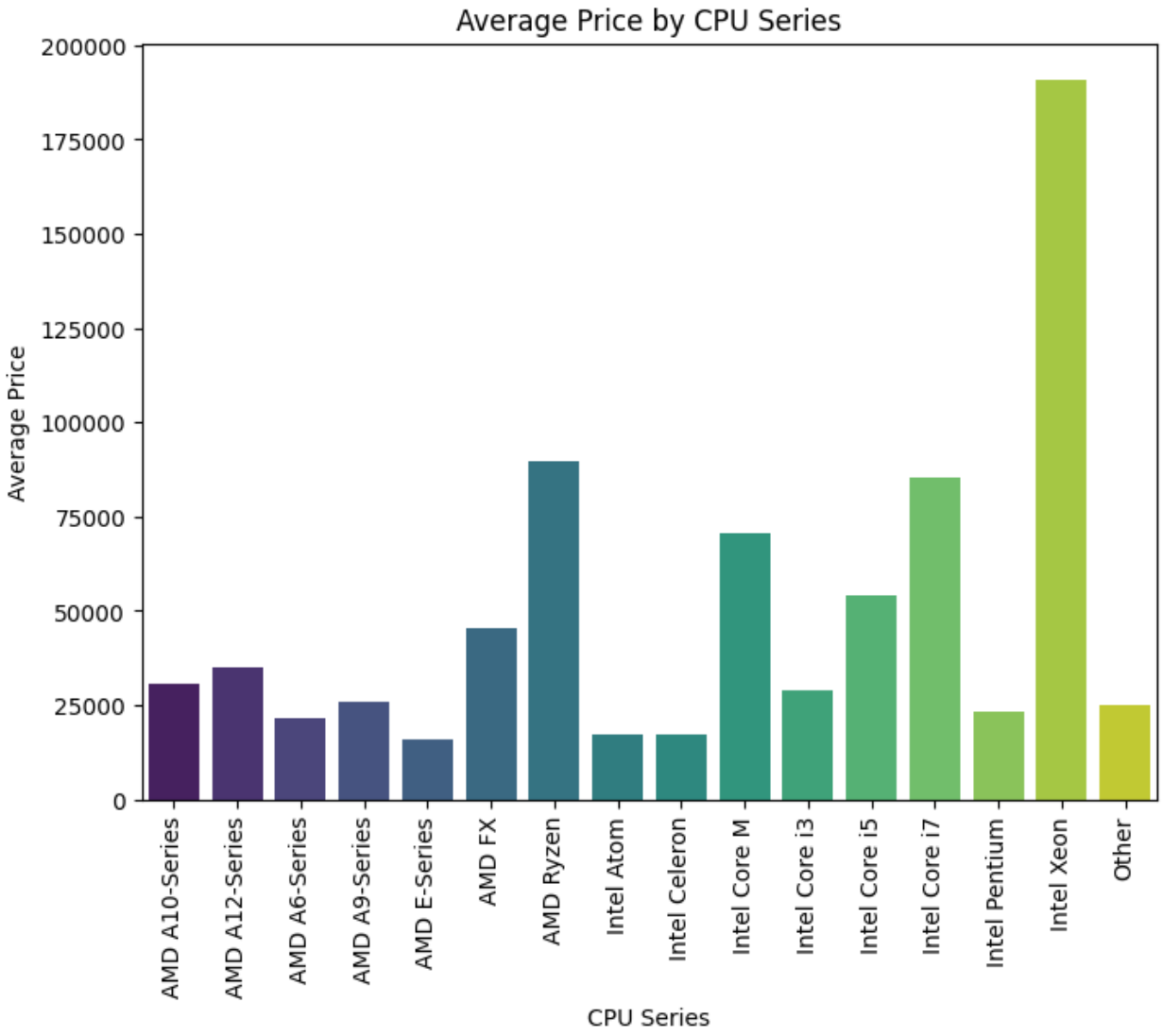
Memory	SSD	HDD	Flash Storage	Hybrid	
256GB SSD	256	0	0	0	399
1TB HDD	0	1	0	0	217
500GB HDD	0	500	0	0	130
512GB SSD	512	0	0	0	116
128GB SSD + 1TB HDD	128	1	0	0	92
128GB SSD	128	0	0	0	74
256GB SSD + 1TB HDD	256	1	0	0	71
32GB Flash Storage	0	0	32	0	37
2TB HDD	0	2	0	0	16
64GB Flash Storage	0	0	64	0	14
512GB SSD + 1TB HDD	512	1	0	0	14
1TB SSD	1	0	0	0	13
256GB SSD + 2TB HDD	256	2	0	0	10
1.0TB Hybrid	0	0	0	1	9

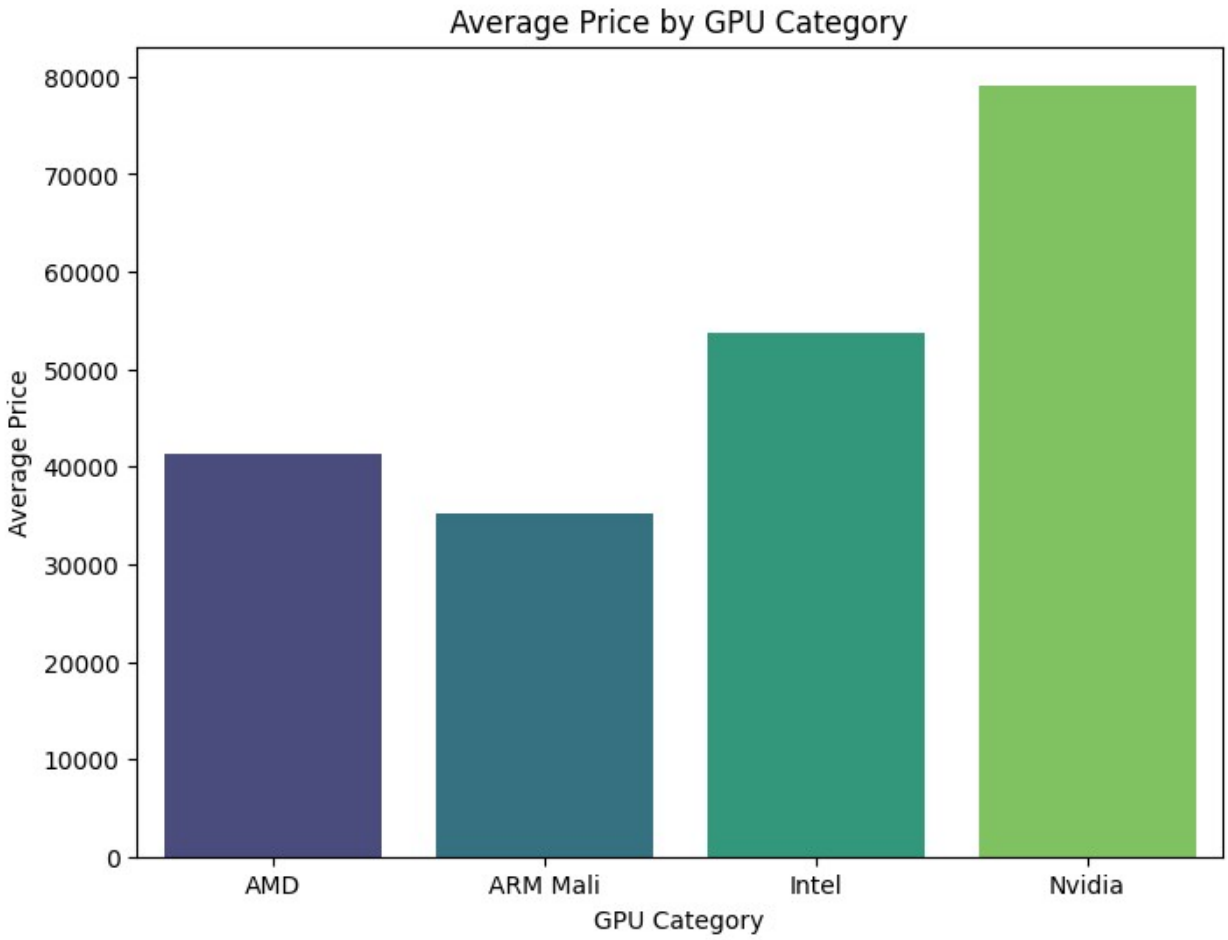
```
256GB Flash Storage             0     0    256             0            8
16GB Flash Storage              0     0    16              0            7
32GB SSD                        32    0    0               0            6
180GB SSD                       180   0    0               0            4
128GB Flash Storage             0     0    128             0            4
512GB SSD +  2TB HDD            512   2    0               0            3
16GB SSD                        16    0    0               0            3
128GB SSD +  2TB HDD            128   2    0               0            2
1TB SSD +  1TB HDD              1     1    0               0            2
512GB Flash Storage             0     0    512             0            2
256GB SSD +  500GB HDD          256   500  0               0            2
256GB SSD +  256GB SSD          512   0    0               0            2
1.0TB HDD                       0     1    0               0            1
128GB HDD                       0     128  0               0            1
256GB SSD +  1.0TB Hybrid       256   0    0               1            1
1TB HDD +  1TB HDD              0     2    0               0            1
240GB SSD                       240   0    0               0            1
512GB SSD +  1.0TB Hybrid       512   0    0               1            1
508GB Hybrid                    0     0    0               508          1
32GB HDD                        0     32   0               0            1
512GB SSD +  256GB SSD          768   0    0               0            1
512GB SSD +  512GB SSD          1024  0    0               0            1
64GB Flash Storage +  1TB HDD   0     1    64              0            1
64GB SSD                        64    0    0               0            1
8GB SSD                         8     0    0               0            1
Name: count, dtype: int64
```

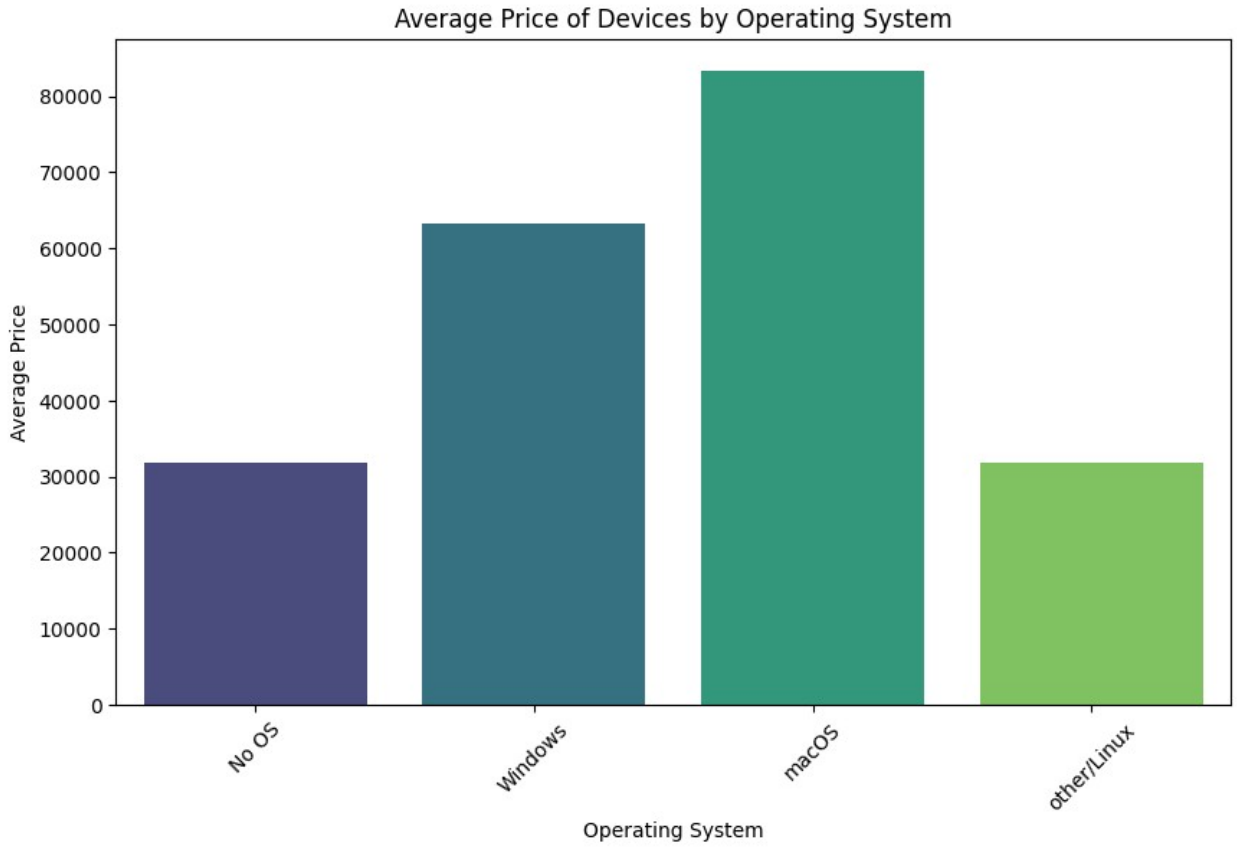
```
import matplotlib.pyplot as plt
import seaborn as sns

avg_price_cpu_series = df.groupby('Cpu_Series')
['Price'].mean().reset_index()
avg_price_gpu_category = df.groupby('GpuCategory')
['Price'].mean().reset_index()
avg_price_OS = df.groupby('OS')['Price'].mean().reset_index()
avg_price_touchscreen = df.groupby('Touchscreen')
['Price'].mean().reset_index()

# Melt the DataFrame to have 'StorageType' and 'Price' columns
melted_df = df.melt(id_vars='Price', value_vars=['SSD', 'HDD', 'Flash
Storage', 'Hybrid'], var_name='StorageType', value_name='Count')
melted_df = melted_df[melted_df['Count'] > 0]  # Filter out rows with
count 0

# Group by 'StorageType' and calculate the mean price
avg_prices_storage_types = melted_df.groupby('StorageType')
['Price'].mean().reset_index()

# Plot the average price for each storage type
plt.figure(figsize=(10, 6))
```

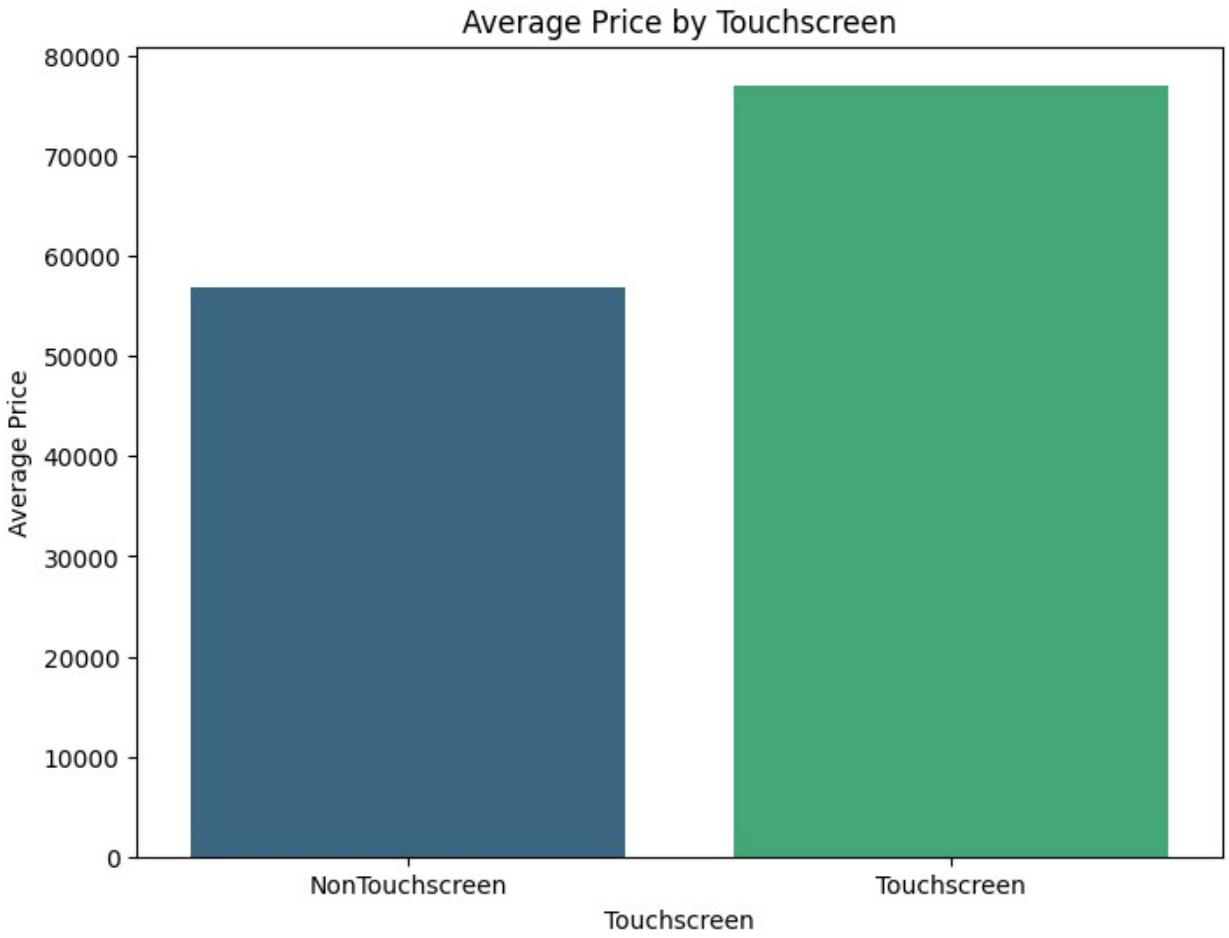
```
sns.barplot(data=avg_prices_storage_types, x='StorageType',
y='Price',palette='viridis')
plt.title('Average Price by Storage Type')
plt.xlabel('Storage Type')
plt.ylabel('Average Price')
plt.xticks(rotation=45)
plt.show()

# Plot average price by CPU Series
plt.figure(figsize=(8, 6))
sns.barplot(data=avg_price_cpu_series, x='Cpu_Series',
y='Price',palette='viridis')
plt.title('Average Price by CPU Series')
plt.xlabel('CPU Series')
plt.ylabel('Average Price')
plt.xticks(rotation=90)
plt.show()

# Plot average price by GPU Category
plt.figure(figsize=(8, 6))
sns.barplot(data=avg_price_gpu_category, x='GpuCategory',
y='Price',palette='viridis')
plt.title('Average Price by GPU Category')
plt.xlabel('GPU Category')
plt.ylabel('Average Price')
plt.show()

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(x='OS', y='Price', data=avg_price_OS,palette='viridis')
plt.title('Average Price of Devices by Operating System')
plt.xlabel('Operating System')
plt.ylabel('Average Price')
plt.xticks(rotation=45)
plt.show()


# Plot average price by Touchscreen
plt.figure(figsize=(8, 6))
sns.barplot(data=avg_price_touchscreen, x='Touchscreen',
y='Price',palette='viridis')
plt.title('Average Price by Touchscreen')
plt.xlabel('Touchscreen')
plt.ylabel('Average Price')
plt.show()
```


Average Price by Storage Type
70000
60000
50000
40000
30000
20000
10000
0
Average Price
Flash Storage
HDD
Hybrid
SSD
Storage Type


Average Price by CPU Series
Average Price
200000
175000
150000
125000
100000
75000
50000
25000
0
AMD A10-Series
AMD A12-Series
AMD A6-Series
AMD A9-Series
AMD E-Series
AMD FX
AMD Ryzen
Intel Atom
Intel Celeron
Intel Core M
Intel Core i3
Intel Core i5
Intel Core i7
Intel Pentium
Intel Xeon
Other
CPU Series


Average Price by GPU Category
80000
70000
60000
50000
40000
30000
20000
10000
0
Average Price
AMD
ARM Mali
Intel
Nvidia
GPU Category


Average Price of Devices by Operating System
Average Price
80000
70000
60000
50000
40000
30000
20000
10000
0
No OS
Windows
macOS
other/Linux
Operating System




```
# Function to extract X and Y resolutions and panel type using regular
expressions
def extract_resolution(resolution):
    match = re.match(r'(?:(?P<panel_type>.*?)(?:\sPanel)?\s?(?:\/)?\s?
(?:Touchscreen)?\s?)(?P<x_res>\d+)x(?P<y_res>\d+)', resolution)
    if match:
        panel_type = match.group('panel_type') or 'Unknown'
        if panel_type == "IPS":
            panel_type = "IPS Panel"
        x_res = int(match.group('x_res'))
        y_res = int(match.group('y_res'))
        return pd.Series([panel_type, x_res, y_res])
    else:
        return pd.Series(['Unknown', 0, 0])

# Apply the function to the 'ScreenResolution' column and create new
columns 'PanelType', 'X_res', and 'Y_res'
df[['PanelType', 'X_res', 'Y_res']] =
df['ScreenResolution'].apply(extract_resolution)

# Function to modify the PanelType column
```

```
def modify_panel_type(panel_type):
    panel_type = re.sub(r'(?i)(?:\/|\\|Touchscreen)', '', panel_type)
# Remove "/", "\", and "Touchscreen"
    if panel_type.startswith("IPS Panel"):
        panel_type = "IPS Panel"
    panel_type = re.sub(r'^\s+|\s+$', '', panel_type)  # Strip leading
and trailing whitespaces
    return panel_type

# Apply the modification to the PanelType column
df['PanelType'] = df['PanelType'].apply(modify_panel_type)

# Print the result
print(df[['ScreenResolution', 'PanelType', 'X_res', 'Y_res']])
```

```
                                    ScreenResolution  PanelType  X_res
Y_res
0              IPS Panel Retina Display 2560x1600  IPS Panel   2560
1600
1                                        1440x900    Unknown   1440
900
2                                Full HD 1920x1080    Full HD   1920
1080
3              IPS Panel Retina Display 2880x1800  IPS Panel   2880
1800
4              IPS Panel Retina Display 2560x1600  IPS Panel   2560
1600
...                                             ...        ...    ...
...
1298   IPS Panel Full HD / Touchscreen 1920x1080  IPS Panel   1920
1080
1299  IPS Panel Quad HD+ / Touchscreen 3200x1800  IPS Panel   3200
1800
1300                                     1366x768    Unknown   1366
768
1301                                     1366x768    Unknown   1366
768
1302                                     1366x768    Unknown   1366
768

[1270 rows x 4 columns]
```

###Label Encoding

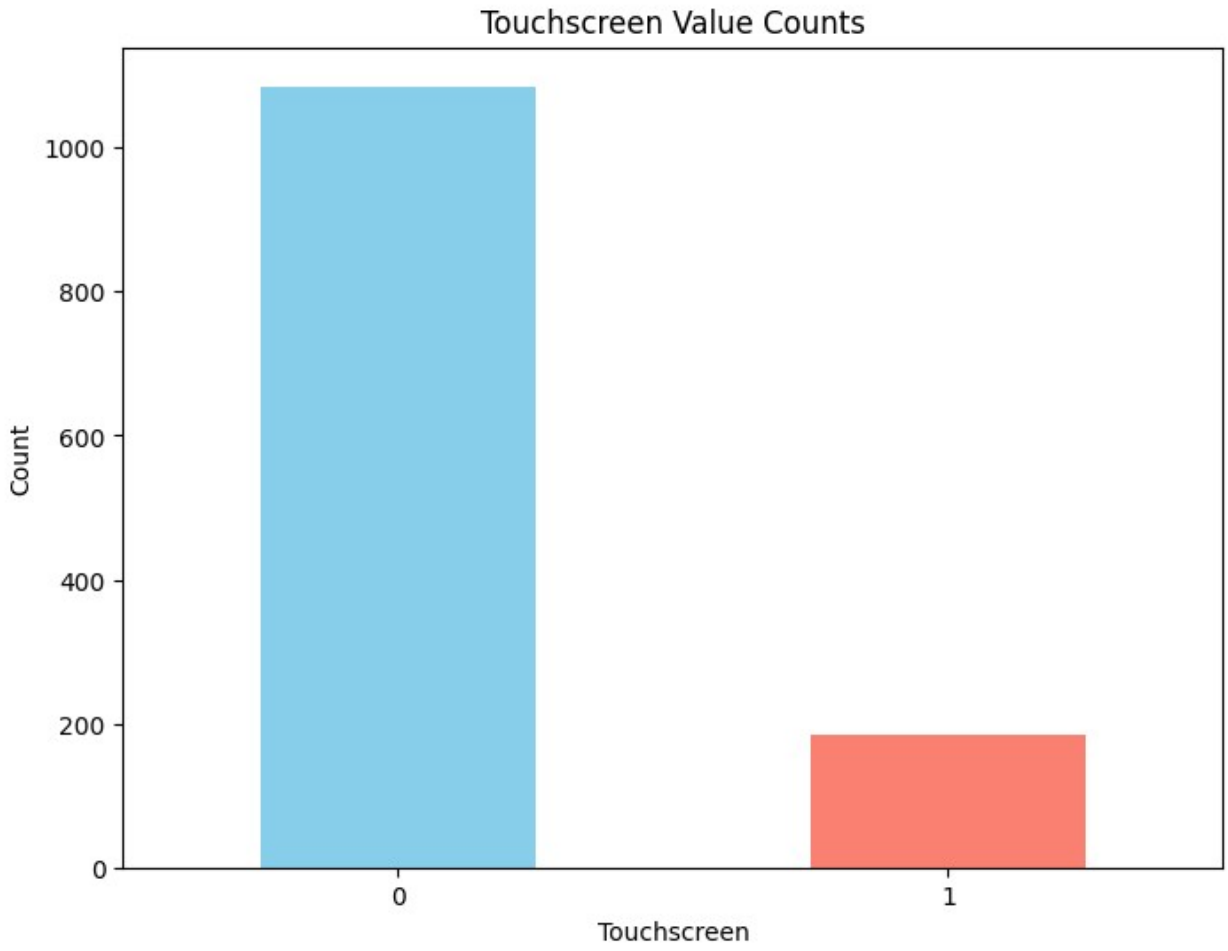
```
from sklearn.preprocessing import LabelEncoder


columns_to_encode =
['Company','TypeName','OS','Touchscreen','Cpu_Series','GpuCategory','P
anelType']
```

```python
# Dictionary to store label encoders
label_encoders = {}

# Label encode each column in the list
for col in columns_to_encode:
 le = LabelEncoder()
 df[col] = le.fit_transform(df[col])
 label_encoders[col] = le

# To print the mapping for each column
for col, le in label_encoders.items():
    print(f"Mapping for {col}:")
    for class_index, class_label in enumerate(le.classes_):
        print(f"  {class_label}: {class_index}")
```

```
Mapping for Company:
  Acer: 0
  Apple: 1
  Asus: 2
  Chuwi: 3
  Dell: 4
  Fujitsu: 5
  Google: 6
  HP: 7
  Huawei: 8
  LG: 9
  Lenovo: 10
  MSI: 11
  Mediacom: 12
  Microsoft: 13
  Razer: 14
  Samsung: 15
  Toshiba: 16
  Vero: 17
  Xiaomi: 18
Mapping for TypeName:
  2 in 1 Convertible: 0
  Gaming: 1
  Netbook: 2
  Notebook: 3
  Ultrabook: 4
  Workstation: 5
Mapping for OS:
  No OS: 0
  Windows: 1
  macOS: 2
  other/Linux: 3
Mapping for Touchscreen:
  NonTouchscreen: 0
```

```
  Touchscreen: 1
Mapping for Cpu_Series:
  AMD A10-Series: 0
  AMD A12-Series: 1
  AMD A6-Series: 2
  AMD A9-Series: 3
  AMD E-Series: 4
  AMD FX: 5
  AMD Ryzen: 6
  Intel Atom: 7
  Intel Celeron: 8
  Intel Core M: 9
  Intel Core i3: 10
  Intel Core i5: 11
  Intel Core i7: 12
  Intel Pentium: 13
  Intel Xeon: 14
  Other: 15
Mapping for GpuCategory:
  AMD: 0
  ARM Mali: 1
  Intel: 2
  Nvidia: 3
Mapping for PanelType:
  4K Ultra HD: 0
  Full HD: 1
  IPS Panel: 2
  Quad HD+: 3
  Unknown: 4
```

```python
import pandas as pd
import matplotlib.pyplot as plt


# Get the value counts
value_counts = df['Touchscreen'].value_counts()

# Plotting
plt.figure(figsize=(8, 6))
value_counts.plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Touchscreen Value Counts')
plt.xlabel('Touchscreen')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()
```




```
# Convert 'Weight' column to string type
df['Weight'] = df['Weight'].astype(str)
df['Ram'] = df['Ram'].astype(str)

# Extract only numeric values from 'Weight' column
df['Weight'] = df['Weight'].str.extract(r'([\d.]+)').astype(float)
df['Ram'] = df['Ram'].str.extract(r'(\d+)').astype(float)
df
```

```
{"type":"dataframe","variable_name":"df"}
```

```
df.columns
```

```
Index(['Company', 'TypeName', 'Inches', 'ScreenResolution', 'Cpu',
'Ram',
       'Memory', 'Gpu', 'OpSys', 'Weight', 'Price', 'Cpu_Series',
       'Touchscreen', 'OS', 'GpuCategory', 'SSD', 'HDD', 'Flash
Storage',
       'Hybrid', 'PanelType', 'X_res', 'Y_res'],
      dtype='object')
```

###Selecting features and target variable

```
x=df[['Company','TypeName','Ram','Inches','OS','Cpu_Series','Touchscreen','GpuCategory','SSD','HDD','Flash
Storage','Hybrid','PanelType','X_res','Y_res']]
x
```


```
{"summary":"{\n  \"name\": \"x\",\n  \"rows\": 1270,\n  \"fields\": [\n    {\n      \"column\": \"Company\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4,\n        \"min\": 0,\n
\"max\": 18,\n        \"num_unique_values\": 19,\n        \"samples\":
[\n          1,\n          10,\n          18\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TypeName\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 1,\n
\"min\": 0,\n        \"max\": 5,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          4,\n          3,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Ram\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5.566929534499876,\n
\"min\": 1.0,\n        \"max\": 64.0,\n        \"num_unique_values\":
10,\n        \"samples\": [\n          24.0,\n          16.0,\n
64.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Inches\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 24,\n
\"samples\": [\n          \"13\",\n          \"24\",\n
\"13.3\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"OS\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 3,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          0,\n
3,\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Cpu_Series\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 0,\n
\"max\": 15,\n        \"num_unique_values\": 16,\n        \"samples\":
[\n          11,\n          12,\n          4\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Touchscreen\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"GpuCategory\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 3,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
```

```
\"column\": \"SSD\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 164,\n        \"min\": 0,\n
\"max\": 1024,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          240,\n          768\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"HDD\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 152,\n        \"min\": 0,\n
\"max\": 500,\n        \"num_unique_values\": 6,\n        \"samples\":
[\n          0,\n          500\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Flash Storage\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 30,\n        \"min\": 0,\n
\"max\": 512,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          0,\n          128\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Hybrid\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 14,\n        \"min\": 0,\n
\"max\": 508,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"PanelType\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 4,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          4,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"X_res\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 494,\n        \"min\": 1366,\n
\"max\": 3840,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          2736,\n          2160\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Y_res\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 284,\n
\"min\": 768,\n        \"max\": 2160,\n        \"num_unique_values\":
10,\n        \"samples\": [\n          1824,\n          900\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"x"}
```


```
y = df[['Price']]
y
```


```
{"summary":"{\n  \"name\": \"y\",\n  \"rows\": 1270,\n  \"fields\": [\
n    {\n      \"column\": \"Price\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 37323.78429983366,\n
\"min\": 9270.72,\n        \"max\": 324954.72,\n
\"num_unique_values\": 775,\n        \"samples\": [\n
120831.5808,\n          115709.2416,\n          126912.96\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"y"}
```

###Splitting the data into training and testing sets

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y, test_size= 0.2,
random_state = 42)
print("train", x_train.shape)
print("test", x_test.shape)
```

```
train (1016, 15)
test (254, 15)
```

```
from sklearn.preprocessing import StandardScaler

# Feature scaling
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

x_train['Inches'] = pd.to_numeric(x_train['Inches'], errors='coerce')
x_test['Inches'] = pd.to_numeric(x_test['Inches'], errors='coerce')

from sklearn.linear_model import LinearRegression
import xgboost as xgb
from sklearn.ensemble import
RandomForestRegressor,GradientBoostingRegressor
from sklearn.metrics import r2_score,mean_squared_error

models = {
    'Linear Regression': LinearRegression(),
    'Random Forest Regression':
RandomForestRegressor(random_state=42),
    'Gradient Boosting Regressor' :
GradientBoostingRegressor(random_state=42),
    'XGBoost Regressor': xgb.XGBRegressor(random_state=42)
}

results = {}
for name, model in models.items():
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    results[name] = {'R_2 Score': r2, 'Mean Squared Error': mse ,
'Predictions': y_pred}
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/base.py:1389:
DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples,), for
example using ravel().
```

```
  return fit_method(estimator, *args, **kwargs)
/usr/local/lib/python3.12/dist-packages/sklearn/ensemble/_gb.py:672:
DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)  # TODO: Is this still required?
```

```
# Printing results
for name, metrics in results.items():
    print(f"\nModel: {name}")
    print(f"R-squared: {metrics['R_2 Score']}")
    print(f"Mean Squared Error: {metrics['Mean Squared Error']}\n")
```

```
Model: Linear Regression
R-squared: 0.6039912817054862
Mean Squared Error: 440325039.99111944


Model: Random Forest Regression
R-squared: 0.8338655456499053
Mean Squared Error: 184726135.7544251


Model: Gradient Boosting Regressor
R-squared: 0.8126058891921002
Mean Squared Error: 208364905.9317487


Model: XGBoost Regressor
R-squared: 0.7834740281105042
Mean Squared Error: 240756832.0
```

###**Hyperparameter Tuning**

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import xgboost as xgb
import pickle
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import RandomizedSearchCV

models = {
    'Linear Regression': LinearRegression(),
    'Random Forest Regression':
RandomForestRegressor(random_state=42),
```

```
    'Gradient Boosting Regressor':
GradientBoostingRegressor(random_state=42),
    'XGBoost Regressor': xgb.XGBRegressor(random_state=42)
}

# Define parameter distributions for each model
param_dists = {
    'Linear Regression': {
        'fit_intercept': [True, False],
        'copy_X': [True, False],
        'positive': [True, False]
    },
    'Random Forest Regression': {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'max_features': ['auto', 'sqrt']
    },
    'Gradient Boosting Regressor': {
        'n_estimators': [100, 200, 300, 400, 500],
        'learning_rate': [0.01, 0.05, 0.1, 0.2],
        'max_depth': [3, 4, 5, 6],
        'subsample': [0.7, 0.8, 0.9, 1.0],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    },
    'XGBoost Regressor': {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.05],
        'max_depth': [3, 4, 5],
        'subsample': [0.8, 0.9, 1.0],
        'colsample_bytree': [0.8, 0.9, 1.0]
    }
}

# Perform RandomizedSearchCV for each model
best_models = {}
for name, model in models.items():
    print(f"Tuning {name}...")
    random_search = RandomizedSearchCV(estimator=model,
param_distributions=param_dists[name], cv=5, n_iter=10, scoring='r2',
n_jobs=-1, random_state=42)
    random_search.fit(x_train, y_train)
    best_models[name] = random_search.best_estimator_
    print(f"Best parameters for {name}: {random_search.best_params_}")
    print(f"Best R-squared for {name}: {random_search.best_score_}")

# Evaluate the best models on the test set
results = {}
```

```
best_model_name = None
best_r2_score = -float('inf')

for name, model in best_models.items():
    y_pred = model.predict(x_test)
    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    results[name] = {'R_2 Score': r2, 'Mean Squared Error': mse,
'Predictions': y_pred}

    if r2 > best_r2_score:
        best_r2_score = r2
        best_model_name = name

# Save the best model
best_model = best_models[best_model_name]
with open('best_model.pkl', 'wb') as file:
    pickle.dump(best_model, file)


# Print the results
for name, metrics in results.items():
    print(f"Model: {name}")
    print(f"R-squared: {metrics['R_2 Score']}")
    print(f"Mean Squared Error: {metrics['Mean Squared Error']}\n")

print(f"Best model: {best_model_name} with R-squared:
{best_r2_score}")
```

```
Tuning Linear Regression...

/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/
_search.py:317: UserWarning: The total space of parameters 8 is
smaller than n_iter=10. Running 8 iterations. For exhaustive searches,
use GridSearchCV.
  warnings.warn(

Best parameters for Linear Regression: {'positive': True,
'fit_intercept': False, 'copy_X': True}
Best R-squared for Linear Regression: 0.3612436852658483
Tuning Random Forest Regression...

/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/
_validation.py:528: FitFailedWarning:
25 fits failed out of a total of 50.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
```

```
--------------------------------------------------------------------------
----------
25 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_vali
dation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line
1382, in wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line
436, in _validate_params
    validate_parameter_constraints(
  File
"/usr/local/lib/python3.12/dist-packages/sklearn/utils/_param_validati
on.py", line 98, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The
'max_features' parameter of RandomForestRegressor must be an int in
the range [1, inf), a float in the range (0.0, 1.0], a str among
{'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_searc
h.py:1108: UserWarning: One or more of the test scores are non-finite:
[0.74303286 0.71414601 0.7550262         nan        nan        nan
        nan        nan 0.73735684 0.7561926 ]
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/base.py:1389:
DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  return fit_method(estimator, *args, **kwargs)

Best parameters for Random Forest Regression: {'n_estimators': 200,
'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'sqrt',
'max_depth': 20}
Best R-squared for Random Forest Regression: 0.7561925973532084
Tuning Gradient Boosting Regressor...

/usr/local/lib/python3.12/dist-packages/sklearn/ensemble/_gb.py:672:
DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)  # TODO: Is this still required?

Best parameters for Gradient Boosting Regressor: {'subsample': 0.9,
'n_estimators': 500, 'min_samples_split': 5, 'min_samples_leaf': 4,
'max_depth': 5, 'learning_rate': 0.05}
```

```
Best R-squared for Gradient Boosting Regressor: 0.7729398486349339
Tuning XGBoost Regressor...
Best parameters for XGBoost Regressor: {'subsample': 0.9,
'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.05,
'colsample_bytree': 1.0}
Best R-squared for XGBoost Regressor: 0.7714925289154053
Model: Linear Regression
R-squared: 0.575990210280537
Mean Squared Error: 471459639.6233815

Model: Random Forest Regression
R-squared: 0.8509345793150533
Mean Squared Error: 165746950.23652834

Model: Gradient Boosting Regressor
R-squared: 0.8533296309538768
Mean Squared Error: 163083874.50125778

Model: XGBoost Regressor
R-squared: 0.7927759885787964
Mean Squared Error: 230413936.0

Best model: Gradient Boosting Regressor with R-squared:
0.8533296309538768



!pip install pandas xgboost scikit-learn pyqt5




Requirement already satisfied: pandas in
/usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: xgboost in
/usr/local/lib/python3.12/dist-packages (3.0.5)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: pyqt5 in
/usr/local/lib/python3.12/dist-packages (5.15.11)
Requirement already satisfied: numpy>=1.26.0 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.12/dist-packages (from xgboost) (2.27.3)
Requirement already satisfied: scipy in
```

```
/usr/local/lib/python3.12/dist-packages (from xgboost) (1.16.2)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: PyQt5-sip<13,>=12.15 in
/usr/local/lib/python3.12/dist-packages (from pyqt5) (12.17.1)
Requirement already satisfied: PyQt5-Qt5<5.16.0,>=5.15.2 in
/usr/local/lib/python3.12/dist-packages (from pyqt5) (5.15.17)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2-
>pandas) (1.17.0)
```

###Real-time Predictions:

```
import sys
import pandas as pd
import pickle
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit,
QComboBox, QPushButton, QVBoxLayout, QHBoxLayout, QMessageBox

# Load the trained model
with open('best_model.pkl', 'rb') as file:
    best_model = pickle.load(file)


# Function to convert user input features to numerical values based on
label encoding mappings
def convert_features_to_numerical(features):
    # Label encoding mappings
    company_mapping = {'Acer': 0, 'Apple': 1, 'Asus': 2, 'Chuwi': 3,
'Dell': 4, 'Fujitsu': 5,
                       'Google': 6, 'HP': 7, 'Huawei': 8, 'LG': 9,
'Lenovo': 10, 'MSI': 11,
                       'Mediacom': 12, 'Microsoft': 13, 'Razer': 14,
'Samsung': 15, 'Toshiba': 16,
                       'Vero': 17, 'Xiaomi': 18}
    typename_mapping = {'2 in 1 Convertible': 0, 'Gaming': 1,
'Netbook': 2, 'Notebook': 3,
                        'Ultrabook': 4, 'Workstation': 5}
    ops_mapping = {'No OS': 0, 'Windows': 1, 'macOS': 2,
'other/Linux': 3}
    touchscreen_mapping = {'NonTouchscreen': 0, 'Touchscreen': 1}
    cpu_series_mapping = {'AMD A10-Series': 0, 'AMD A12-Series': 1,
'AMD A6-Series': 2, 'AMD A9-Series': 3,
                          'AMD E-Series': 4, 'AMD FX': 5, 'AMD Ryzen':
6, 'Intel Atom': 7, 'Intel Celeron': 8,
                          'Intel Core M': 9, 'Intel Core i3': 10,
```

```
'Intel Core i5': 11, 'Intel Core i7': 12,
                          'Intel Pentium': 13, 'Intel Xeon': 14,
'Other': 15}
    gpu_category_mapping = {'AMD': 0, 'ARM Mali': 1, 'Intel': 2,
'Nvidia': 3}
    panel_type_mapping = {'4K Ultra HD': 0, 'Full HD': 1, 'IPS Panel':
2, 'Quad HD+': 3, 'Unknown': 4}

    # Convert "Inches" to numeric
    features['Inches'] = pd.to_numeric(features['Inches'],
errors='coerce')

    # Convert features to numerical values based on mappings
    features['Company'] = company_mapping.get(features['Company'], -1)
    features['TypeName'] = typename_mapping.get(features['TypeName'],
-1)
    features['OS'] = ops_mapping.get(features['OS'], -1)
    features['Touchscreen'] =
touchscreen_mapping.get(features['Touchscreen'], -1)
    features['Cpu_Series'] =
cpu_series_mapping.get(features['Cpu_Series'], -1)
    features['GpuCategory'] =
gpu_category_mapping.get(features['GpuCategory'], -1)
    features['PanelType'] =
panel_type_mapping.get(features['PanelType'], -1)

    return features

# Function to make predictions
def predict_price(model, data):
    # Make predictions
    prediction = model.predict(data)
    return prediction

# PyQt5 application
class PricePredictorApp(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('Laptop Price Predictor')

        layout = QVBoxLayout()

        self.company_label = QLabel('Company')
        self.company = QComboBox()
        self.company.addItems(['Acer', 'Apple', 'Asus', 'Chuwi',
'Dell', 'Fujitsu', 'Google', 'HP', 'Huawei', 'LG', 'Lenovo', 'MSI',
'Mediacom', 'Microsoft', 'Razer', 'Samsung', 'Toshiba', 'Vero',
```

```
'Xiaomi'])
        layout.addWidget(self.company_label)
        layout.addWidget(self.company)

        self.type_name_label = QLabel('Type Name')
        self.type_name = QComboBox()
        self.type_name.addItems(['2 in 1 Convertible', 'Gaming',
'Netbook', 'Notebook', 'Ultrabook', 'Workstation'])
        layout.addWidget(self.type_name_label)
        layout.addWidget(self.type_name)

        self.ram_label = QLabel('RAM (GB)')
        self.ram = QLineEdit()
        layout.addWidget(self.ram_label)
        layout.addWidget(self.ram)

        self.inches_label = QLabel('Screen Size (Inches)')
        self.inches = QLineEdit()
        layout.addWidget(self.inches_label)
        layout.addWidget(self.inches)

        self.os_label = QLabel('Operating System')
        self.os = QComboBox()
        self.os.addItems(['other/Linux', 'No OS', 'Windows','macOS'])
        layout.addWidget(self.os_label)
        layout.addWidget(self.os)

        self.cpu_series_label = QLabel('CPU Series')
        self.cpu_series = QComboBox()
        self.cpu_series.addItems(['AMD A10-Series', 'AMD A12-Series',
'AMD A6-Series', 'AMD A9-Series', 'AMD E-Series', 'AMD FX', 'AMD
Ryzen', 'Intel Atom', 'Intel Celeron', 'Intel Core M', 'Intel Core
i3', 'Intel Core i5', 'Intel Core i7', 'Intel Pentium', 'Intel Xeon',
'Other'])
        layout.addWidget(self.cpu_series_label)
        layout.addWidget(self.cpu_series)

        self.touchscreen_label = QLabel('Touchscreen')
        self.touchscreen = QComboBox()
        self.touchscreen.addItems(['NonTouchscreen', 'Touchscreen'])
        layout.addWidget(self.touchscreen_label)
        layout.addWidget(self.touchscreen)

        self.gpu_category_label = QLabel('GPU Category')
        self.gpu_category = QComboBox()
        self.gpu_category.addItems(['AMD', 'ARM Mali', 'Intel',
'Nvidia'])
        layout.addWidget(self.gpu_category_label)
        layout.addWidget(self.gpu_category)
```

```
        self.ssd_label = QLabel('SSD (GB)')
        self.ssd = QLineEdit()
        layout.addWidget(self.ssd_label)
        layout.addWidget(self.ssd)

        self.hdd_label = QLabel('HDD (GB)')
        self.hdd = QLineEdit()
        layout.addWidget(self.hdd_label)
        layout.addWidget(self.hdd)
        self.flash_storage_label = QLabel('Flash Storage (GB)')
        self.flash_storage = QLineEdit()
        layout.addWidget(self.flash_storage_label)
        layout.addWidget(self.flash_storage)

        self.hybrid_label = QLabel('Hybrid (GB)')
        self.hybrid = QLineEdit()
        layout.addWidget(self.hybrid_label)
        layout.addWidget(self.hybrid)

        self.panel_type_label = QLabel('Panel Type')
        self.panel_type = QComboBox()
        self.panel_type.addItems(['4K Ultra HD', 'Full HD', 'IPS
Panel', 'Quad HD+', 'Unknown'])
        layout.addWidget(self.panel_type_label)
        layout.addWidget(self.panel_type)

        self.x_res_label = QLabel('X Resolution')
        self.x_res = QLineEdit()
        layout.addWidget(self.x_res_label)
        layout.addWidget(self.x_res)

        self.y_res_label = QLabel('Y Resolution')
        self.y_res = QLineEdit()
        layout.addWidget(self.y_res_label)
        layout.addWidget(self.y_res)

        self.predict_button = QPushButton('Predict')
        self.predict_button.clicked.connect(self.on_predict)
        layout.addWidget(self.predict_button)

        self.setLayout(layout)

    def on_predict(self):
        try:
            input_data = {
                'Company': self.company.currentText(),
                'TypeName': self.type_name.currentText(),
                'Ram': float(self.ram.text()),
                'Inches': float(self.inches.text()),
                'OS': self.os.currentText(),
```

```
                'Cpu_Series': self.cpu_series.currentText(),
                'Touchscreen': self.touchscreen.currentText(),
                'GpuCategory': self.gpu_category.currentText(),
                'SSD': int(self.ssd.text()),
                'HDD': int(self.hdd.text()),
                'Flash Storage': int(self.flash_storage.text()),
                'Hybrid' : int(self.hybrid.text()),
                'PanelType': self.panel_type.currentText(),
                'X_res': int(self.x_res.text()),
                'Y_res': int(self.y_res.text())
            }

            input_df =
pd.DataFrame([convert_features_to_numerical(input_data)])
            prediction = predict_price(best_model, input_df)
            QMessageBox.information(self, "Predicted Price", f'$
{prediction[0]:,.2f}')
        except Exception as e:
            QMessageBox.critical(self, "Error", str(e))

def main():
    app = QApplication(sys.argv)
    ex = PricePredictorApp()
    ex.setGeometry(100, 100, 800, 600)  # Set window geometry (x_pos,
y_pos, width, height)
    ex.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()
```