This is my 35th. Laptop Price Prediction Project

Import Required Libraries

In [1]:
```
# Data handling
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Preprocessing & splitting
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline,Pipeline
from sklearn.impute import SimpleImputer

# Evaluation metrics
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Regression Models
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import (
    RandomForestRegressor,
    GradientBoostingRegressor,
    AdaBoostRegressor,
    ExtraTreesRegressor
)
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor

# XGBoost
from xgboost import XGBRegressor

# Warnings
import warnings
warnings.filterwarnings("ignore")

# Save the model
import joblib
```

Load Dataset

In [2]:
```
data = pd.read_csv('laptopData.csv')
data.head()
```

Out[2]:

	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory
0	0.0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD
1	1.0	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage
2	2.0	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD
3	3.0	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD
4	4.0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD

In [3]:
```
data.duplicated().sum()
```

Out[3]:
```
29
```

Remove Duplicates

In [4]:
```
data = data.drop_duplicates()
```

In [5]:
```
data.duplicated().sum()
```

Out[5]:
```
0
```

In [6]:
```
data.isna().sum()
```

Out[6]:
```
Unnamed: 0          1
Company             1
TypeName            1
Inches              1
ScreenResolution    1
Cpu                 1
Ram                 1
Memory              1
Gpu                 1
OpSys               1
Weight              1
Price               1
dtype: int64
```

Handle Missing values

In [7]:
```
data.dropna(inplace=True)
```

In [8]:
```
cat = data.select_dtypes(include='object')
for i in cat.columns:
    print(f"{i}")
    print(f"{cat[i].nunique()}")
```

```
Company
19
TypeName
6
Inches
25
ScreenResolution
40
Cpu
118
Ram
10
Memory
40
Gpu
110
OpSys
9
Weight
189
```

In [9]:
```
for i in cat.columns:
    print(f"{i}")
    print(f"{cat[i].unique()}")
```

```
Company
['Apple' 'HP' 'Acer' 'Asus' 'Dell' 'Lenovo' 'Chuwi' 'MSI' 'Microsoft'
 'Toshiba' 'Huawei' 'Xiaomi' 'Vero' 'Razer' 'Mediacom' 'Samsung' 'Google'
 'Fujitsu' 'LG']
TypeName
['Ultrabook' 'Notebook' 'Gaming' '2 in 1 Convertible' 'Workstation'
 'Netbook']
Inches
['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']
ScreenResolution
['IPS Panel Retina Display 2560x1600' '1440x900' 'Full HD 1920x1080'
 'IPS Panel Retina Display 2880x1800' '1366x768'
 'IPS Panel Full HD 1920x1080' 'IPS Panel Retina Display 2304x1440'
 'IPS Panel Full HD / Touchscreen 1920x1080'
 'Full HD / Touchscreen 1920x1080' 'Touchscreen / Quad HD+ 3200x1800'
 'Touchscreen 2256x1504' 'Quad HD+ / Touchscreen 3200x1800'
 'IPS Panel 1366x768' 'IPS Panel 4K Ultra HD / Touchscreen 3840x2160'
 'IPS Panel Full HD 2160x1440' '4K Ultra HD / Touchscreen 3840x2160'
 '1600x900' 'IPS Panel 4K Ultra HD 3840x2160' '4K Ultra HD 3840x2160'
 'Touchscreen 1366x768' 'Touchscreen 2560x1440'
 'IPS Panel Full HD 1366x768' 'IPS Panel 2560x1440'
 'IPS Panel Full HD 2560x1440' 'IPS Panel Retina Display 2736x1824'
 'Touchscreen 2400x1600' '2560x1440' 'IPS Panel Quad HD+ 2560x1440'
 'IPS Panel Quad HD+ 3200x1800'
 'IPS Panel Quad HD+ / Touchscreen 3200x1800'
 'IPS Panel Touchscreen 1366x768' '1920x1080'
 'IPS Panel Full HD 1920x1200'
 'IPS Panel Touchscreen / 4K Ultra HD 3840x2160'
 'IPS Panel Touchscreen 2560x1440' 'Touchscreen / Full HD 1920x1080'
 'Quad HD+ 3200x1800' 'IPS Panel Touchscreen 1920x1200'
 'Touchscreen / 4K Ultra HD 3840x2160' 'IPS Panel Touchscreen 2400x1600']
Cpu
['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 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 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' 'AMD A12-Series 9720P 2.7GHz'
 'Intel Core i5 7300U 2.6GHz' 'AMD A12-Series 9720P 3.6GHz'
 'Intel Celeron Quad Core N3450 1.1GHz'
 'Intel Celeron Dual Core N3060 1.60GHz' 'Intel Core i5 6440HQ 2.6GHz'
 'Intel Core i7 6820HQ 2.7GHz' 'AMD Ryzen 1600 3.2GHz'
 'Intel Core i7 7Y75 1.3GHz' 'Intel Core i5 7440HQ 2.8GHz'
 'Intel Core i7 7660U 2.5GHz' 'Intel Core i7 7700HQ 2.7GHz'
 'Intel Core M m3-7Y30 2.2GHz' 'Intel Core i5 7Y57 1.2GHz'
 'Intel Core i7 6700HQ 2.6GHz' 'Intel Core i3 6100U 2.3GHz'
 'Intel Atom x5-Z8350 1.44GHz' 'AMD A10-Series 9620P 2.5GHz'
 'AMD E-Series 7110 1.8GHz' 'Intel Celeron Dual Core N3350 2.0GHz'
 'AMD A9-Series A9-9420 3GHz' 'Intel Core i7 6820HK 2.7GHz'
 'Intel Core M 7Y30 1.0GHz' 'Intel Xeon E3-1535M v6 3.1GHz'
 'Intel Celeron Quad Core N3160 1.6GHz' 'Intel Core i5 6300U 2.4GHz'
 'Intel Core i3 6100U 2.1GHz' 'AMD E-Series E2-9000 2.2GHz'
 'Intel Celeron Dual Core N3050 1.6GHz' 'Intel Core M M3-6Y30 0.9GHz'
 'AMD A9-Series 9420 2.9GHz' 'Intel Core i5 6300HQ 2.3GHz'
 'AMD A6-Series 7310 2GHz' 'Intel Atom Z8350 1.92GHz'
 'Intel Xeon E3-1535M v5 2.9GHz' 'Intel Core i5 6260U 1.8GHz'
 'Intel Pentium Dual Core N4200 1.1GHz'
 'Intel Celeron Quad Core N3710 1.6GHz' 'Intel Core M 1.2GHz'
 'AMD A12-Series 9700P 2.5GHz' 'Intel Core i7 7500U 2.5GHz'
 'Intel Pentium Dual Core 4405U 2.1GHz' 'AMD A4-Series 7210 2.2GHz'
 'Intel Core i7 6560U 2.2GHz' 'Intel Core M m7-6Y75 1.2GHz'
 'AMD FX 8800P 2.1GHz' 'Intel Core M M7-6Y75 1.2GHz'
 'Intel Core i5 7200U 2.50GHz' 'Intel Core i5 7200U 2.70GHz'
 'Intel Atom X5-Z8350 1.44GHz' 'Intel Core i5 7200U 2.7GHz'
 'Intel Core M 1.1GHz' 'Intel Atom x5-Z8550 1.44GHz'
 'Intel Pentium Dual Core 4405Y 1.5GHz'
 'Intel Pentium Quad Core N3700 1.6GHz' 'Intel Core M 6Y54 1.1GHz'
 'Intel Core i7 6500U 2.50GHz' 'Intel Celeron Dual Core N3350 2GHz'
 'Samsung Cortex A72&A53 2.0GHz' 'AMD E-Series 9000 2.2GHz'
 'Intel Core M 6Y30 0.9GHz' 'AMD A9-Series 9410 2.9GHz']
Ram
['8GB' '16GB' '4GB' '2GB' '12GB' '64GB' '6GB' '32GB' '24GB' '1GB']
Memory
['128GB SSD' '128GB Flash Storage' '256GB SSD' '512GB SSD' '500GB HDD'
 '256GB Flash Storage' '1TB HDD' '128GB SSD +  1TB HDD'
 '256GB SSD +  256GB SSD' '64GB Flash Storage' '32GB Flash Storage'
 '256GB SSD +  1TB HDD' '256GB SSD +  2TB HDD' '32GB SSD' '2TB HDD'
 '64GB SSD' '1.0TB Hybrid' '512GB SSD +  1TB HDD' '1TB SSD'
 '256GB SSD +  500GB HDD' '128GB SSD +  2TB HDD' '512GB SSD +  512GB SSD'
 '16GB SSD' '16GB Flash Storage' '512GB SSD +  256GB SSD'
 '512GB SSD +  2TB HDD' '64GB Flash Storage +  1TB HDD' '180GB SSD'
 '1TB HDD +  1TB HDD' '32GB HDD' '1TB SSD +  1TB HDD' '?'
 '512GB Flash Storage' '128GB HDD' '240GB SSD' '8GB SSD' '508GB Hybrid'
 '1.0TB HDD' '512GB SSD +  1.0TB Hybrid' '256GB SSD +  1.0TB Hybrid']
Gpu
['Intel Iris Plus Graphics 640' 'Intel HD Graphics 6000'
 'Intel HD Graphics 620' 'AMD Radeon Pro 455'
 'Intel Iris Plus Graphics 650' 'AMD Radeon R5' 'Intel Iris Pro Graphics'
 'Nvidia GeForce MX150' 'Intel UHD Graphics 620' 'Intel HD Graphics 520'
 'AMD Radeon Pro 555' 'AMD Radeon R5 M430' 'Intel HD Graphics 615'
 'AMD Radeon Pro 560' 'Nvidia GeForce 940MX' 'Nvidia GeForce GTX 1050'
 'AMD Radeon R2' 'AMD Radeon 530' 'Nvidia GeForce 930MX'
 'Intel HD Graphics' 'Intel HD Graphics 500' 'Nvidia GeForce 930MX '
 'Nvidia GeForce GTX 1060' 'Nvidia GeForce 150MX'
 'Intel Iris Graphics 540' 'AMD Radeon RX 580' 'Nvidia GeForce 920MX'
```

```
 'AMD Radeon R4 Graphics' 'AMD Radeon 520' 'Nvidia GeForce GTX 1070'
 'Nvidia GeForce GTX 1050 Ti' 'Intel HD Graphics 400'
 'Nvidia GeForce MX130' 'AMD R4 Graphics' 'Nvidia GeForce GTX 940MX'
 'AMD Radeon RX 560' 'Nvidia GeForce 920M' 'AMD Radeon R7 M445'
 'AMD Radeon RX 550' 'Nvidia GeForce GTX 1050M' 'Intel HD Graphics 515'
 'AMD Radeon R5 M420' 'Intel HD Graphics 505' 'Nvidia GTX 980 SLI'
 'AMD R17M-M1-70' 'Nvidia GeForce GTX 1080' 'Nvidia Quadro M1200'
 'Nvidia GeForce 920MX ' 'Nvidia GeForce GTX 950M' 'AMD FirePro W4190M '
 'Nvidia GeForce GTX 980M' 'Intel Iris Graphics 550' 'Nvidia GeForce 930M'
 'Intel HD Graphics 630' 'AMD Radeon R5 430' 'Nvidia GeForce GTX 940M'
 'Intel HD Graphics 510' 'Intel HD Graphics 405' 'AMD Radeon RX 540'
 'Nvidia GeForce GT 940MX' 'AMD FirePro W5130M' 'Nvidia Quadro M2200M'
 'AMD Radeon R4' 'Nvidia Quadro M620' 'AMD Radeon R7 M460'
 'Intel HD Graphics 530' 'Nvidia GeForce GTX 965M'
 'Nvidia GeForce GTX1080' 'Nvidia GeForce GTX1050 Ti'
 'Nvidia GeForce GTX 960M' 'AMD Radeon R2 Graphics' 'Nvidia Quadro M620M'
 'Nvidia GeForce GTX 970M' 'Nvidia GeForce GTX 960<U+039C>'
 'Intel Graphics 620' 'Nvidia GeForce GTX 960' 'AMD Radeon R5 520'
 'AMD Radeon R7 M440' 'AMD Radeon R7' 'Nvidia Quadro M520M'
 'Nvidia Quadro M2200' 'Nvidia Quadro M2000M' 'Intel HD Graphics 540'
 'Nvidia Quadro M1000M' 'AMD Radeon 540' 'Nvidia GeForce GTX 1070M'
 'Nvidia GeForce GTX1060' 'Intel HD Graphics 5300' 'AMD Radeon R5 M420X'
 'AMD Radeon R7 Graphics' 'Nvidia GeForce 920' 'Nvidia GeForce 940M'
 'Nvidia GeForce GTX 930MX' 'AMD Radeon R7 M465' 'AMD Radeon R3'
 'Nvidia GeForce GTX 1050Ti' 'AMD Radeon R7 M365X' 'AMD Radeon R9 M385'
 'Intel HD Graphics 620 ' 'Nvidia Quadro 3000M' 'Nvidia GeForce GTX 980 '
 'AMD Radeon R5 M330' 'AMD FirePro W4190M' 'AMD FirePro W6150M'
 'AMD Radeon R5 M315' 'Nvidia Quadro M500M' 'AMD Radeon R7 M360'
 'Nvidia Quadro M3000M' 'Nvidia GeForce 960M' 'ARM Mali T860 MP4']
OpSys
['macOS' 'No OS' 'Windows 10' 'Mac OS X' 'Linux' 'Windows 10 S'
 'Chrome OS' 'Windows 7' 'Android']
Weight
['1.37kg' '1.34kg' '1.86kg' '1.83kg' '2.1kg' '2.04kg' '1.3kg' '1.6kg'
 '2.2kg' '0.92kg' '1.22kg' '2.5kg' '1.62kg' '1.91kg' '2.3kg' '1.35kg'
 '1.88kg' '1.89kg' '1.65kg' '2.71kg' '1.2kg' '1.44kg' '2.8kg' '2kg'
 '2.65kg' '2.77kg' '3.2kg' '1.49kg' '2.4kg' '2.13kg' '2.43kg' '1.7kg'
 '1.4kg' '1.8kg' '1.9kg' '3kg' '1.252kg' '2.7kg' '2.02kg' '1.63kg'
 '1.96kg' '1.21kg' '2.45kg' '1.25kg' '1.5kg' '2.62kg' '1.38kg' '1.58kg'
 '1.85kg' '1.23kg' '2.16kg' '2.36kg' '7.2kg' '2.05kg' '1.32kg' '1.75kg'
 '0.97kg' '2.56kg' '1.48kg' '1.74kg' '1.1kg' '1.56kg' '2.03kg' '1.05kg'
 '5.4kg' '4.4kg' '1.90kg' '1.29kg' '2.0kg' '1.95kg' '2.06kg' '1.12kg'
 '3.49kg' '3.35kg' '2.23kg' '?' '2.9kg' '4.42kg' '2.69kg' '2.37kg' '4.7kg'
 '3.6kg' '2.08kg' '4.3kg' '1.68kg' '1.41kg' '4.14kg' '2.18kg' '2.24kg'
 '2.67kg' '4.1kg' '2.14kg' '1.36kg' '2.25kg' '2.15kg' '2.19kg' '2.54kg'
 '3.42kg' '5.8kg' '1.28kg' '2.33kg' '1.45kg' '2.79kg' '8.23kg' '1.26kg'
 '1.84kg' '0.0002kg' '2.6kg' '2.26kg' '3.25kg' '1.59kg' '1.13kg' '1.42kg'
 '1.78kg' '1.10kg' '1.15kg' '1.27kg' '1.43kg' '2.31kg' '1.16kg' '1.64kg'
 '2.17kg' '1.47kg' '3.78kg' '1.79kg' '0.91kg' '1.99kg' '4.33kg' '1.93kg'
 '1.87kg' '2.63kg' '3.4kg' '3.14kg' '1.94kg' '1.24kg' '4.6kg' '4.5kg'
 '8.4kg' '2.73kg' '1.39kg' '2.29kg' '2.59kg' '2.94kg' '11.1kg' '1.14kg'
 '3.8kg' '6.2kg' '3.31kg' '1.09kg' '3.21kg' '1.19kg' '1.98kg' '1.17kg'
 '4.36kg' '1.71kg' '2.32kg' '4.2kg' '1.55kg' '0.81kg' '1.18kg' '2.72kg'
 '1.31kg' '0.920kg' '3.74kg' '1.76kg' '1.54kg' '2.83kg' '2.07kg' '2.38kg'
 '3.58kg' '1.08kg' '2.20kg' '0.98kg' '2.75kg' '1.70kg' '2.99kg' '1.11kg'
 '2.09kg' '4kg' '3.0kg' '0.99kg' '0.69kg' '3.52kg' '2.591kg' '2.21kg'
 '3.3kg' '2.191kg' '2.34kg' '4.0kg']
```

In [10]:
```
data['Inches'].head()
```

Out[10]:
```
0    13.3
1    13.3
2    15.6
3    15.4
4    13.3
Name: Inches, dtype: object
```

Feature Engineering – Numeric Conversion

In [11]:
```
# Replace '?' with NaN
data.replace('?', np.nan, inplace=True)

data['Inches'] = data['Inches'].astype(float)
```

In [12]:
```
data['Ram'] = data['Ram'].str.replace('GB', '').astype(int)
```

In [13]:
```
data['Weight'] = data['Weight'].str.replace('kg', '').astype(float)
```

Screen Resolution Features

In [14]:
```
data['IPS'] = data['ScreenResolution'].str.contains('IPS', case=False).astype(in
data['Retina'] = data['ScreenResolution'].str.contains('Retina', case=False).ast
data['Touchscreen'] = data['ScreenResolution'].str.contains('Touchscreen', case=
```

In [15]:
```
data[['Width','Height']] = data['ScreenResolution'] \
    .str.extract(r'(\d+)x(\d+)') \
    .astype(int)
```

In [16]:
```
def resolution_type(res):
    if '3840x2160' in res:
        return '4K'
    elif '3200x1800' in res or '2560x1440' in res:
        return 'Quad HD'
    elif '1920x1080' in res or '1920x1200' in res:
        return 'Full HD'
    elif '1366x768' in res:
        return 'HD'
    else:
        return 'Other'

data['Resolution_Type'] = data['ScreenResolution'].apply(resolution_type)
```

In [17]:
```
data['PPI'] = ((data['Width']**2 + data['Height']**2) ** 0.5) / data['Inches']
```

In [18]:
```
data['Cpu_Brand'] = data['Cpu'].apply(lambda x: x.split()[0])
data['Cpu_GHz'] = data['Cpu'].str.extract(r'(\d+\.\d+)').astype(float)
```

In [19]:
```
def cpu_family(cpu):
    cpu = cpu.lower()
    if 'i3' in cpu:
        return 'i3'
    elif 'i5' in cpu:
```

```
        return 'i5'
    elif 'i7' in cpu:
        return 'i7'
    elif 'ryzen' in cpu:
        return 'Ryzen'
    elif 'xeon' in cpu:
        return 'Xeon'
    elif 'celeron' in cpu:
        return 'Celeron'
    elif 'pentium' in cpu:
        return 'Pentium'
    elif 'atom' in cpu:
        return 'Atom'
    else:
        return 'Other'

data['Cpu_Family'] = data['Cpu'].apply(cpu_family)
```

CPU Feature Extraction

In [20]:
```
data['Cpu_Gen'] = data['Cpu'].str.extract(r'(\d{4})').astype(float)

data['Cpu_Gen'] = pd.to_numeric(data['Cpu_Gen'], errors='coerce')
for col in ['Cpu_Gen', 'Inches', 'Weight', 'PPI', 'Cpu_GHz']:
    data[col].fillna(data[col].median(), inplace=True)

data['Memory'] = data['Memory'].fillna(data['Memory'].mode()[0])
```

Memory Features

In [21]:
```
data['SSD'] = data['Memory'].str.contains('SSD', na=False).astype(int)
data['HDD'] = data['Memory'].str.contains('HDD', na=False).astype(int)
```

OS Cleaning

In [22]:
```
def os_clean(os):
    if 'Windows' in os:
        return 'Windows'
    elif 'Mac' in os:
        return 'Mac'
    elif 'Linux' in os:
        return 'Linux'
    else:
        return 'Other'

data['OpSys'] = data['OpSys'].apply(os_clean)
```

Drop Unwanted Columns

In [23]:
```
data.drop(columns=['Unnamed: 0', 'ScreenResolution', 'Cpu', 'Memory'],inplace=Tr
```

In [24]:
```
data.shape
```

Out[24]: (1273, 21)

In [25]:
```
pd.options.display.max_columns = 22

data.head()
```

Out[25]:

	Company	TypeName	Inches	Ram	Gpu	OpSys	Weight	Price	IPS	Ret
0	Apple	Ultrabook	13.3	8	Intel Iris Plus Graphics 640	Other	1.37	71378.6832	1	
1	Apple	Ultrabook	13.3	8	Intel HD Graphics 6000	Other	1.34	47895.5232	0	
2	HP	Notebook	15.6	8	Intel HD Graphics 620	Other	1.86	30636.0000	0	
3	Apple	Ultrabook	15.4	16	AMD Radeon Pro 455	Other	1.83	135195.3360	1	
4	Apple	Ultrabook	13.3	8	Intel Iris Plus Graphics 650	Other	1.37	96095.8080	1	

Encode Categorical Columns

In [26]:
```
le = LabelEncoder()
cat = data.select_dtypes(include='object').columns
for i in cat:
    data[i] = le.fit_transform(data[i])
```

In [27]:
```
data.head(2)
```

Out[27]:

	Company	TypeName	Inches	Ram	Gpu	OpSys	Weight	Price	IPS	Retina
0	1	4	13.3	8	58	2	1.37	71378.6832	1	1
1	1	4	13.3	8	51	2	1.34	47895.5232	0	0

Split X and y & Train - Test Split

In [28]:
```
X = data.drop(['Price'],axis=1)
y = data['Price']
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.20,random_state=

scaler = StandardScaler()

#scale features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled.shape,X_test_scaled.shape
```

Out[28]: ((1018, 20), (255, 20))

Preprocessing Pipeline

In [29]:
```
preprocessor = Pipeline([
    ('imputer',SimpleImputer(strategy='median')),
    ('scaler',StandardScaler())
])
```

Train Multiple Models (Baseline)

In [30]:
```
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=537),
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=100, random_stat
    "AdaBoost": AdaBoostRegressor(n_estimators=100, random_state=537),
    "Extra Trees": ExtraTreesRegressor(n_estimators=100, random_state=537),
    "K-Nearest Neighbors": KNeighborsRegressor(n_neighbors=5),
    "XGBoost": XGBRegressor(n_estimators=100, random_state=537)
}
```

In [31]:
```
results = []

for name, model in models.items():
    pipeline = make_pipeline(preprocessor, model)
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results.append({
        "Model": name,
        "RMSE": rmse,
        'MAE': mae,
        "R2 Score": r2
    })

results_df = pd.DataFrame(results).sort_values(by="R2 Score", ascending=False)
results_df
```

Out[31]:

	Model	RMSE	MAE	R2 Score
1	Random Forest	14699.390196	9049.912622	0.851055
2	Gradient Boosting	14786.848636	9422.844288	0.849277
6	XGBoost	15728.384469	9331.856765	0.829472
4	Extra Trees	15903.581529	9647.199551	0.825651
5	K-Nearest Neighbors	19429.572016	12265.460092	0.739771
0	Linear Regression	21508.940715	15539.384653	0.681091
3	AdaBoost	22997.767088	18985.762590	0.635414

GridSearchCV – Best Model Selection

In [32]:

```
models = {
    # ------------------------------
    "XGBoost": {
        "model": XGBRegressor(objective="reg:squarederror", random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [3, 5, 7],
            "learning_rate": [0.01, 0.1, 0.2],
            "subsample": [0.8, 1.0],
            "colsample_bytree": [0.8, 1.0]
        }
    },
    # ------------------------------
    "Linear Regression": {
        "model": LinearRegression(),
        "param_grid": {
            "fit_intercept": [True, False],
            "copy_X": [True, False]
        }
    },
    # ------------------------------
    "Decision Tree": {
        "model": DecisionTreeRegressor(random_state=42),
        "param_grid": {
            "max_depth": [3, 5, 10, None],
            "min_samples_split": [2, 5, 10],
            "min_samples_leaf": [1, 2, 4]
        }
    },
    # ------------------------------
    "Random Forest": {
        "model": RandomForestRegressor(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
    },
    # ------------------------------
```

```
    "Gradient Boosting": {
        "model": GradientBoostingRegressor(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "learning_rate": [0.01, 0.1, 0.2],
            "max_depth": [3, 5]
        }
    },
    # ------------------------------
    "AdaBoost": {
        "model": AdaBoostRegressor(random_state=42),
        "param_grid": {
            "n_estimators": [50, 100, 200],
            "learning_rate": [0.01, 0.1, 1.0]
        }
    },
    # ------------------------------
    "Extra Trees": {
        "model": ExtraTreesRegressor(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
    },
    # ------------------------------
    "K-Nearest Neighbors": {
        "model": KNeighborsRegressor(),
        "param_grid": {
            "n_neighbors": [3, 5, 7, 9],
            "weights": ["uniform", "distance"],
            "p": [1, 2]
        }
    },
    # ------------------------------
    "MLP Regressor": {
        "model": MLPRegressor(random_state=42, max_iter=500),
        "param_grid": {
            "hidden_layer_sizes": [(50,), (100,)],
            "activation": ["relu", "tanh"],
            "solver": ["adam"],
            "learning_rate_init": [0.001, 0.01]
        }
    }
}

print(f" Total Regression Models Loaded: {len(models)}")
```

```
Total Regression Models Loaded: 9
```

Model Evaluation

In [33]:
```
results = []
best_score = -float('inf')
best_model = None
best_name = None
```

```
for name, config in models.items():
    print(f"\n Running GridSearchCV for {name}...")

    model = config["model"]
    param_grid = config["param_grid"]

    grid = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        cv=3,
        scoring='r2',
        n_jobs=-1,
        verbose=1
    )

    grid.fit(X_train, y_train)
    y_pred = grid.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    results.append({
        'Model': name,
        'Best Params': grid.best_params_,
        'MSE': mse,
        'RMSE': rmse,
        'MAE': mae,
        'R2 Score': r2
    })

    print(f" {name} Best CV R²: {grid.best_score_:.4f}")
    print(f"   Best Params: {grid.best_params_}")

    if grid.best_score_ > best_score:
        best_score = grid.best_score_
        best_model = grid.best_estimator_
        best_name = name

# Convert results to DataFrame
results_df = pd.DataFrame(results).sort_values(by='R2 Score', ascending=False)

print("\n Summary Results:")
print(results_df)

print(f"\n Best Model: {best_name}")
print(f" Best CV Score (R²): {best_score:.4f}")
```

```
 Running GridSearchCV for XGBoost...
Fitting 3 folds for each of 72 candidates, totalling 216 fits
 XGBoost Best CV R²: 0.8327
   Best Params: {'colsample_bytree': 1.0, 'learning_rate': 0.2, 'max_depth': 5,
'n_estimators': 100, 'subsample': 1.0}

 Running GridSearchCV for Linear Regression...
Fitting 3 folds for each of 4 candidates, totalling 12 fits
 Linear Regression Best CV R²: 0.6368
   Best Params: {'copy_X': True, 'fit_intercept': True}

 Running GridSearchCV for Decision Tree...
Fitting 3 folds for each of 36 candidates, totalling 108 fits
 Decision Tree Best CV R²: 0.7369
   Best Params: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 1
0}

 Running GridSearchCV for Random Forest...
Fitting 3 folds for each of 24 candidates, totalling 72 fits
 Random Forest Best CV R²: 0.8072
   Best Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split':
2, 'n_estimators': 200}

 Running GridSearchCV for Gradient Boosting...
Fitting 3 folds for each of 12 candidates, totalling 36 fits
 Gradient Boosting Best CV R²: 0.8302
   Best Params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}

 Running GridSearchCV for AdaBoost...
Fitting 3 folds for each of 9 candidates, totalling 27 fits
 AdaBoost Best CV R²: 0.7132
   Best Params: {'learning_rate': 0.1, 'n_estimators': 100}

 Running GridSearchCV for Extra Trees...
Fitting 3 folds for each of 24 candidates, totalling 72 fits
 Extra Trees Best CV R²: 0.7936
   Best Params: {'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 2,
'n_estimators': 100}

 Running GridSearchCV for K-Nearest Neighbors...
Fitting 3 folds for each of 16 candidates, totalling 48 fits
 K-Nearest Neighbors Best CV R²: 0.7452
   Best Params: {'n_neighbors': 9, 'p': 1, 'weights': 'distance'}

 Running GridSearchCV for MLP Regressor...
Fitting 3 folds for each of 8 candidates, totalling 24 fits
 MLP Regressor Best CV R²: 0.5388
   Best Params: {'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_ra
te_init': 0.01, 'solver': 'adam'}

 Summary Results:
                Model                                        Best Params  \
4    Gradient Boosting  {'learning_rate': 0.1, 'max_depth': 3, 'n_esti...
3        Random Forest  {'max_depth': None, 'min_samples_leaf': 1, 'mi...
0              XGBoost  {'colsample_bytree': 1.0, 'learning_rate': 0.2...
6          Extra Trees  {'max_depth': 20, 'min_samples_leaf': 2, 'min_...
7  K-Nearest Neighbors  {'n_neighbors': 9, 'p': 1, 'weights': 'distance'}
2        Decision Tree  {'max_depth': None, 'min_samples_leaf': 2, 'mi...
5             AdaBoost        {'learning_rate': 0.1, 'n_estimators': 100}
1    Linear Regression            {'copy_X': True, 'fit_intercept': True}
```

```
8        MLP Regressor  {'activation': 'relu', 'hidden_layer_sizes': (...
```

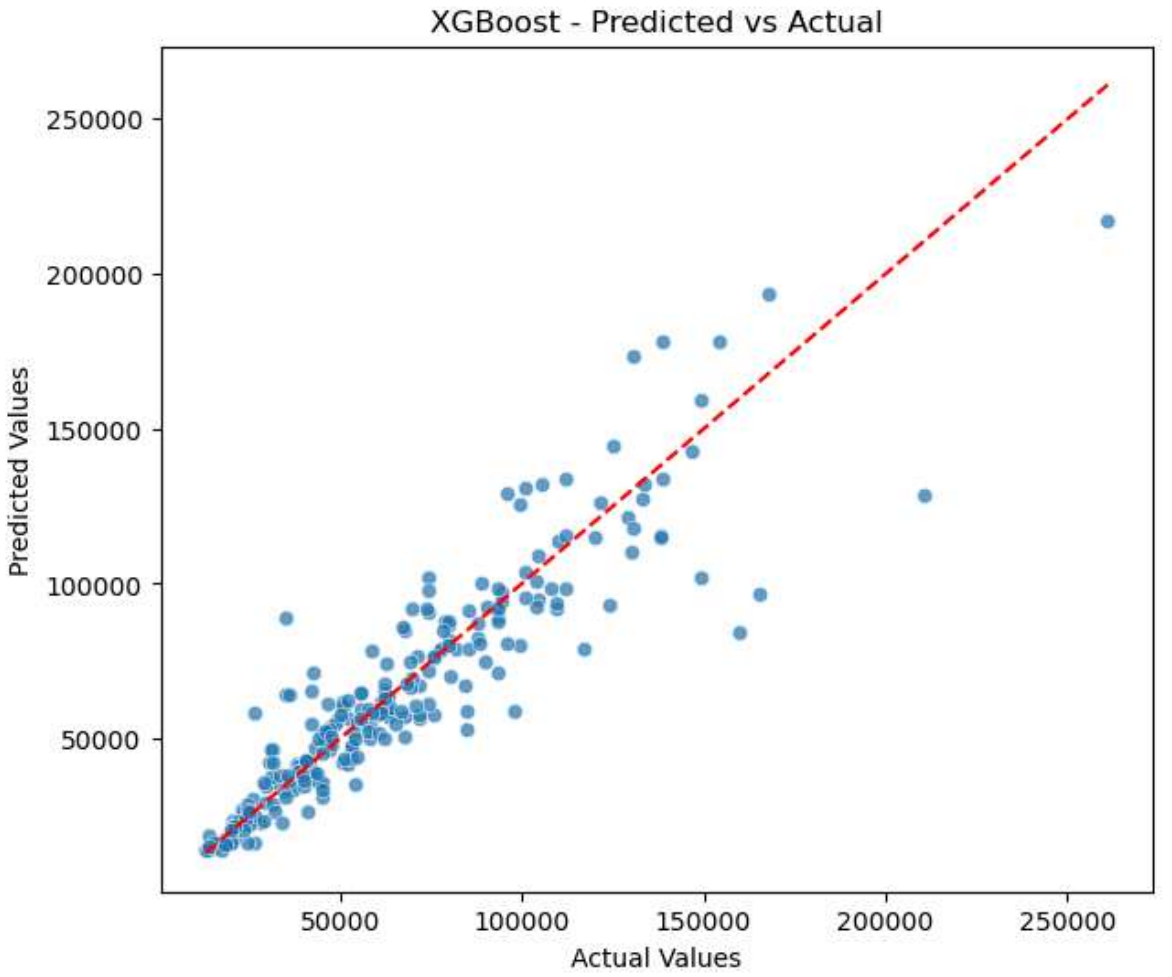
	MSE	RMSE	MAE	R2 Score
4	2.086252e+08	14443.863402	9021.628540	0.856188
3	2.150258e+08	14663.757785	8937.260238	0.851776
0	2.297939e+08	15158.954057	9279.183834	0.841596
6	2.529194e+08	15903.438846	9405.726760	0.825654
7	3.037181e+08	17427.509230	10484.484992	0.790637
2	3.880924e+08	19700.061884	11754.571514	0.732475
5	4.396024e+08	20966.697708	14524.208431	0.696968
1	4.626345e+08	21508.940715	15539.384653	0.681091
8	6.036202e+08	24568.683904	19897.909820	0.583905

```
Best Model: XGBoost
Best CV Score (R²): 0.8327
```

Visualization – Prediction vs Actual

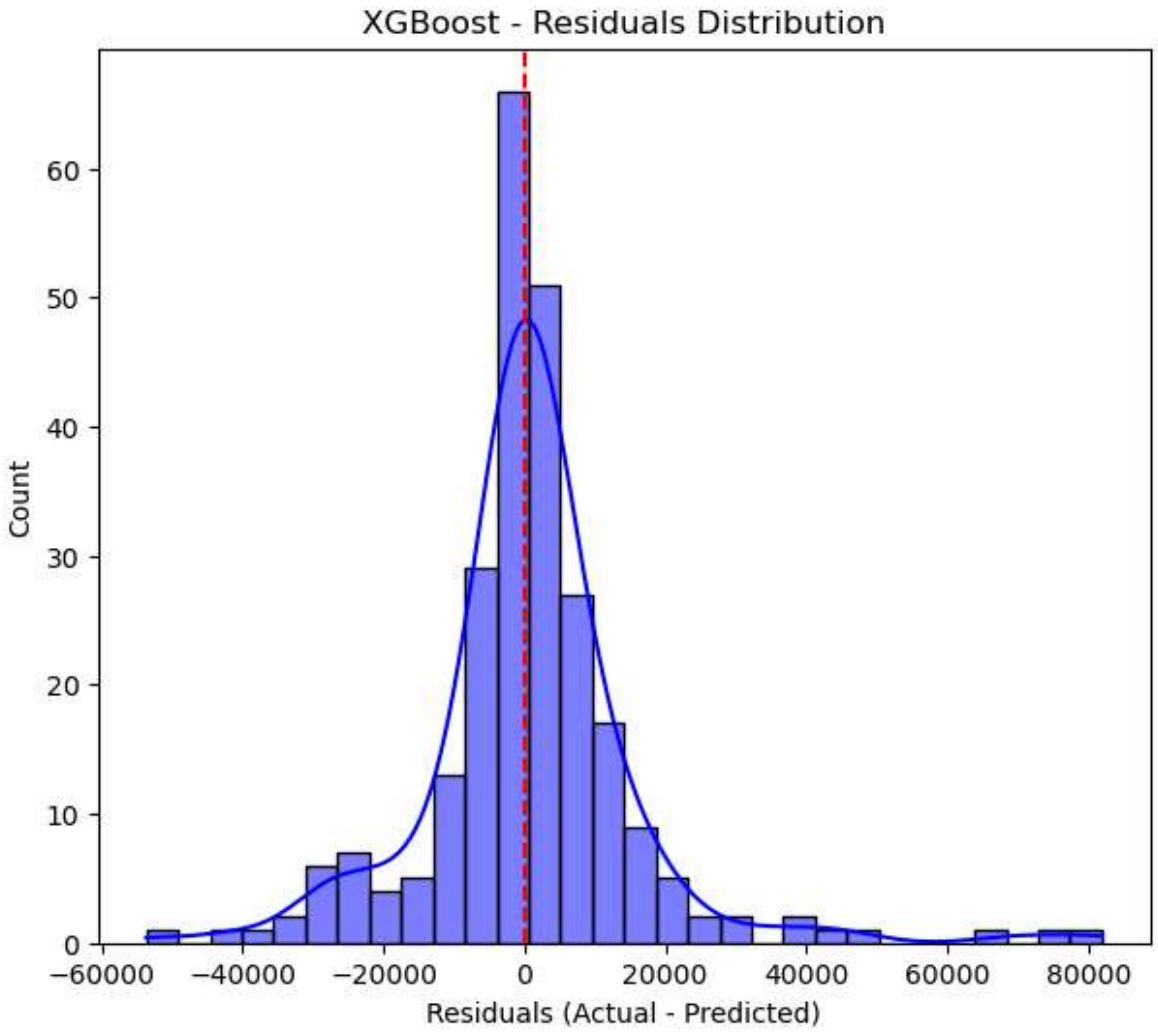
In [34]:
```
y_pred = best_model.predict(X_test)

# Scatter Plot: Predicted vs Actual
plt.figure(figsize=(7,6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')  # a
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title(f"{best_name} - Predicted vs Actual")
plt.show()
```



Residuals Distribution (Actual - Predicted)

In [35]:
```
residuals = y_test - y_pred
plt.figure(figsize=(7,6))
sns.histplot(residuals, kde=True, bins=30, color="blue")
plt.axvline(0, color='red', linestyle='--')
plt.xlabel("Residuals (Actual - Predicted)")
plt.title(f"{best_name} - Residuals Distribution")
plt.show()
```

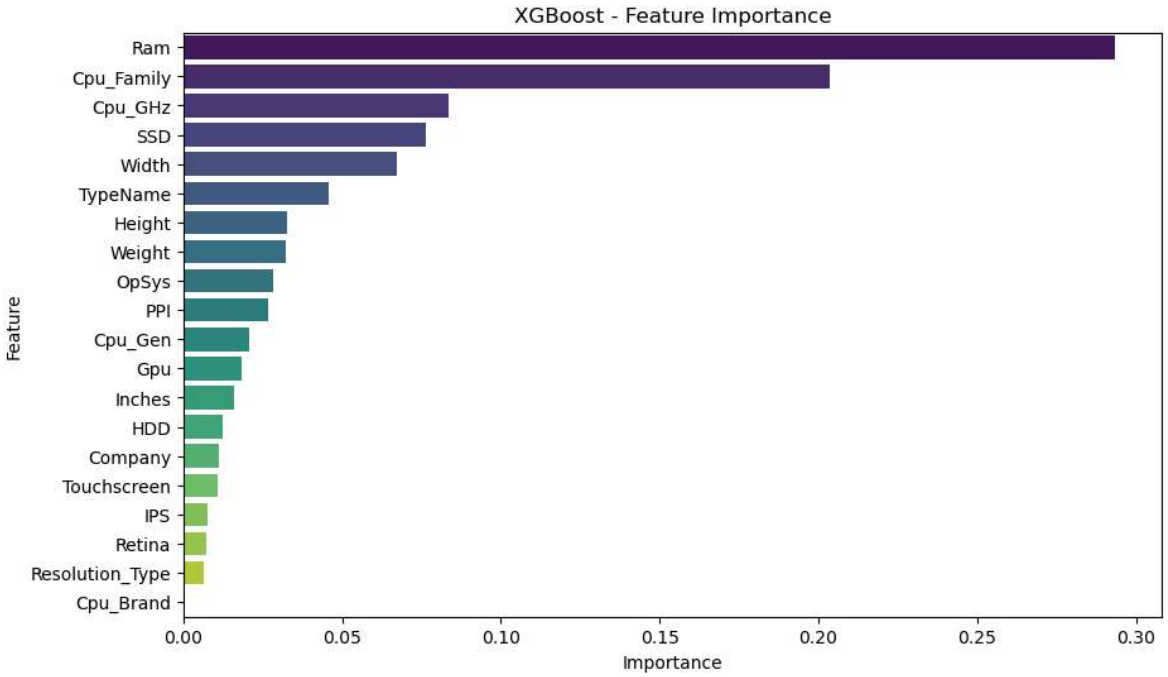


Feature Importance

In [36]:
```
if hasattr(best_model, "feature_importances_"):
    feature_importance = pd.DataFrame({
        "Feature": X_train.columns,
        "Importance": best_model.feature_importances_
    }).sort_values(by="Importance", ascending=False)

    plt.figure(figsize=(10,6))
    sns.barplot(x="Importance", y="Feature", data=feature_importance, palette="v
    plt.title(f"{best_name} - Feature Importance")
    plt.show()
```

```
else:
    print(f" Feature importance not available for {best_name}")
```



Save Model & Features

In [37]:
```
feature_columns = X.columns.to_list()
joblib.dump(feature_columns,'features 35.Laptop Price Prediction.joblib')
```

Out[37]:
```
['features 35.Laptop Price Prediction.joblib']
```

In [38]:
```
joblib.dump(best_model,'best model 35.Laptop Price Prediction.joblib')
```

Out[38]:
```
['best model 35.Laptop Price Prediction.joblib']
```

If you have any suggestions, please DM me.

Even a small message from you can make a big impact on my career

I Am Arun

In [ ]:
```

```