Laptop Prices Prediction

Problem Statement: Predicting Laptop Price Based on Features

Objective:

To build a machine learning model that predicts the price of laptops based on their various features, which include specifications like RAM, screen size, processor speed, and more.

Key Points:

Data Description:

The dataset includes various features of laptops, such as:

Brand/Company (e.g., Apple, HP, Acer) Product Type (e.g., Ultrabook, Notebook) Screen Size (in inches) RAM Size (in GB) Operating System (e.g., macOS, Windows) Weight (in kg) CPU Information (company, frequency, model) Storage Type and Size (Primary and Secondary) GPU Information (company, model) Price (in euros, the target variable)

Goal:

The main goal is to predict the price of laptops based on these features.

Target Variable:

The target variable (the value we are predicting) is Price in euros. Features:

Various attributes like: Company (e.g., Apple, HP) RAM Screen Size CPU Frequency Primary Storage Type GPU Model These features are used to predict the target variable.

Machine Learning Approach:

we aim to use supervised learning (regression), where the model is trained on labeled data (features and corresponding prices).

The model will learn the relationship between the features and the price.

Model Evaluation:

The performance of the model will be evaluated using Mean Squared Error (MSE) and R-squared.We will also consider cross-validation to validate the model's generalizability.

Challenges:

The data includes both numerical and categorical features, so proper encoding and scaling will be required.Handling potential missing values and outliers in the data.

Expected Outcome:

A trained machine learning model that can accurately predict laptop prices based on the given features.

Data Collection and Preparation

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

In [2]:
```
df = pd.read_csv(r"C:\Users\chira\Downloads\laptop_prices.csv")
data = df
```

In [3]:
```
df.head()
```

Out[3]:

	Company	Product	TypeName	Inches	Ram	OS	Weight	Price_euros	Screen	ScreenW
0	Apple	MacBook Pro	Ultrabook	13.3	8	macOS	1.37	1339.69	Standard	2560
1	Apple	Macbook Air	Ultrabook	13.3	8	macOS	1.34	898.94	Standard	1440
2	HP	250 G6	Notebook	15.6	8	No OS	1.86	575.00	Full HD	1920
3	Apple	MacBook Pro	Ultrabook	15.4	16	macOS	1.83	2537.45	Standard	2880
4	Apple	MacBook Pro	Ultrabook	13.3	8	macOS	1.37	1803.60	Standard	2560

5 rows × 23 columns

In [4]:
```
df.tail()
```

Out[4]:

	Company	Product	TypeName	Inches	Ram	OS	Weight	Price_euros
1270	Lenovo	Yoga 500-14ISK	2 in 1 Convertible	14.0	4	Windows 10	1.80	638.0
1271	Lenovo	Yoga 900-13ISK	2 in 1 Convertible	13.3	16	Windows 10	1.30	1499.0
1272	Lenovo	IdeaPad 100S-14IBR	Notebook	14.0	2	Windows 10	1.50	229.0
1273	HP	15-AC110nv (i7-6500U/6GB/1TB/Radeon	Notebook	15.6	6	Windows 10	2.19	764.0
1274	Asus	X553SA-XX031T (N3050/4GB/500GB/W10)	Notebook	15.6	4	Windows 10	2.20	369.0

5 rows × 23 columns

In [5]:
```
df.shape
```

Out[5]:
```
(1275, 23)
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Company                 0
Product                 0
TypeName                0
Inches                  0
Ram                     0
OS                      0
Weight                  0
Price_euros             0
Screen                  0
ScreenW                 0
ScreenH                 0
Touchscreen             0
IPSpanel                0
RetinaDisplay           0
CPU_company             0
CPU_freq                0
CPU_model               0
PrimaryStorage          0
SecondaryStorage        0
PrimaryStorageType      0
SecondaryStorageType    0
GPU_company             0
GPU_model               0
dtype: int64
```

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

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Company                 0
Product                 0
TypeName                0
Inches                  0
Ram                     0
OS                      0
Weight                  0
Price_euros             0
Screen                  0
ScreenW                 0
ScreenH                 0
Touchscreen             0
IPSpanel                0
RetinaDisplay           0
CPU_company             0
CPU_freq                0
CPU_model               0
PrimaryStorage          0
SecondaryStorage        0
PrimaryStorageType      0
SecondaryStorageType    0
GPU_company             0
GPU_model               0
dtype: int64
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1275 entries, 0 to 1274
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Company               1275 non-null   object 
 1   Product               1275 non-null   object 
 2   TypeName              1275 non-null   object 
 3   Inches                1275 non-null   float64
 4   Ram                   1275 non-null   int64  
 5   OS                    1275 non-null   object 
 6   Weight                1275 non-null   float64
 7   Price_euros           1275 non-null   float64
 8   Screen                1275 non-null   object 
 9   ScreenW               1275 non-null   int64  
 10  ScreenH               1275 non-null   int64  
 11  Touchscreen           1275 non-null   object 
 12  IPSpanel              1275 non-null   object 
 13  RetinaDisplay         1275 non-null   object 
 14  CPU_company           1275 non-null   object 
 15  CPU_freq              1275 non-null   float64
 16  CPU_model             1275 non-null   object 
 17  PrimaryStorage        1275 non-null   int64  
 18  SecondaryStorage      1275 non-null   int64  
 19  PrimaryStorageType    1275 non-null   object 
 20  SecondaryStorageType  1275 non-null   object 
 21  GPU_company           1275 non-null   object 
 22  GPU_model             1275 non-null   object 
dtypes: float64(4), int64(5), object(14)
memory usage: 229.2+ KB
```

In [10]:
```
df.describe()
```

Out[10]:

	Inches	Ram	Weight	Price_euros	ScreenW	ScreenH	CPU_freq
count	1275.000000	1275.000000	1275.000000	1275.000000	1275.000000	1275.000000	1275.000000
mean	15.022902	8.440784	2.040525	1134.969059	1900.043922	1073.904314	2.302980
std	1.429470	5.097809	0.669196	700.752504	493.346186	283.883940	0.503846
min	10.100000	2.000000	0.690000	174.000000	1366.000000	768.000000	0.900000
25%	14.000000	4.000000	1.500000	609.000000	1920.000000	1080.000000	2.000000
50%	15.600000	8.000000	2.040000	989.000000	1920.000000	1080.000000	2.500000
75%	15.600000	8.000000	2.310000	1496.500000	1920.000000	1080.000000	2.700000
max	18.400000	64.000000	4.700000	6099.000000	3840.000000	2160.000000	3.600000

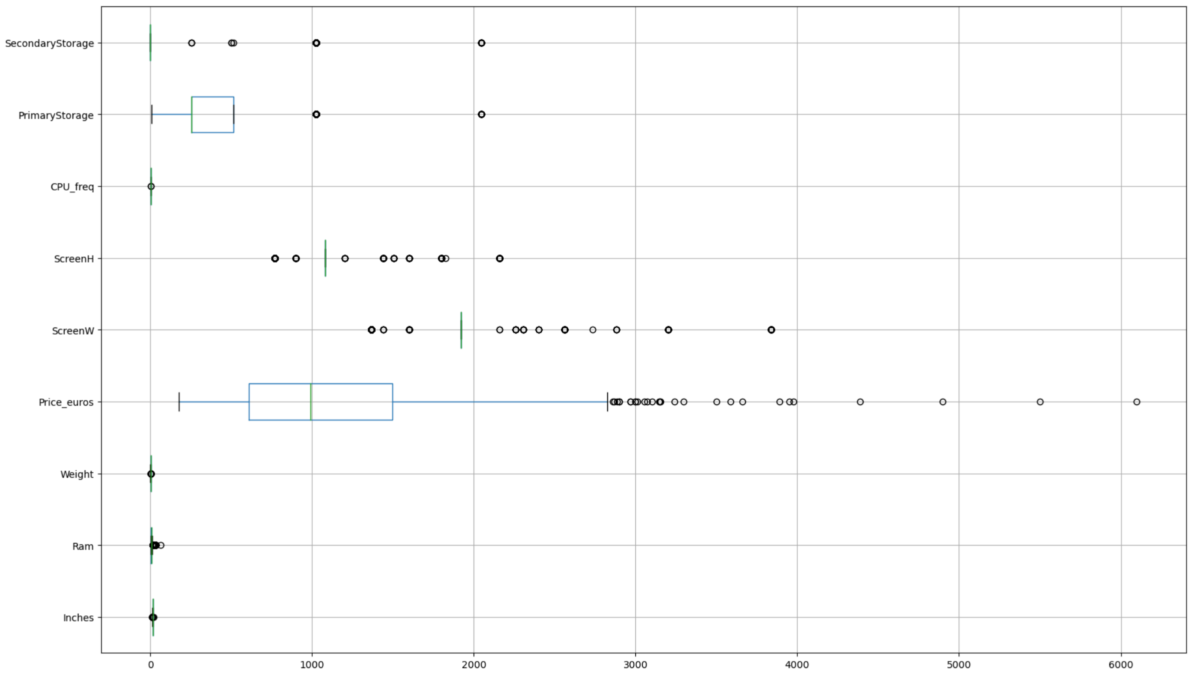
In [11]:

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

Exploratory Data Analysis

In [12]:

```
# Create a box plot
plt.figure(figsize=(20,12))
df.boxplot(vert=0)
plt.show()
```



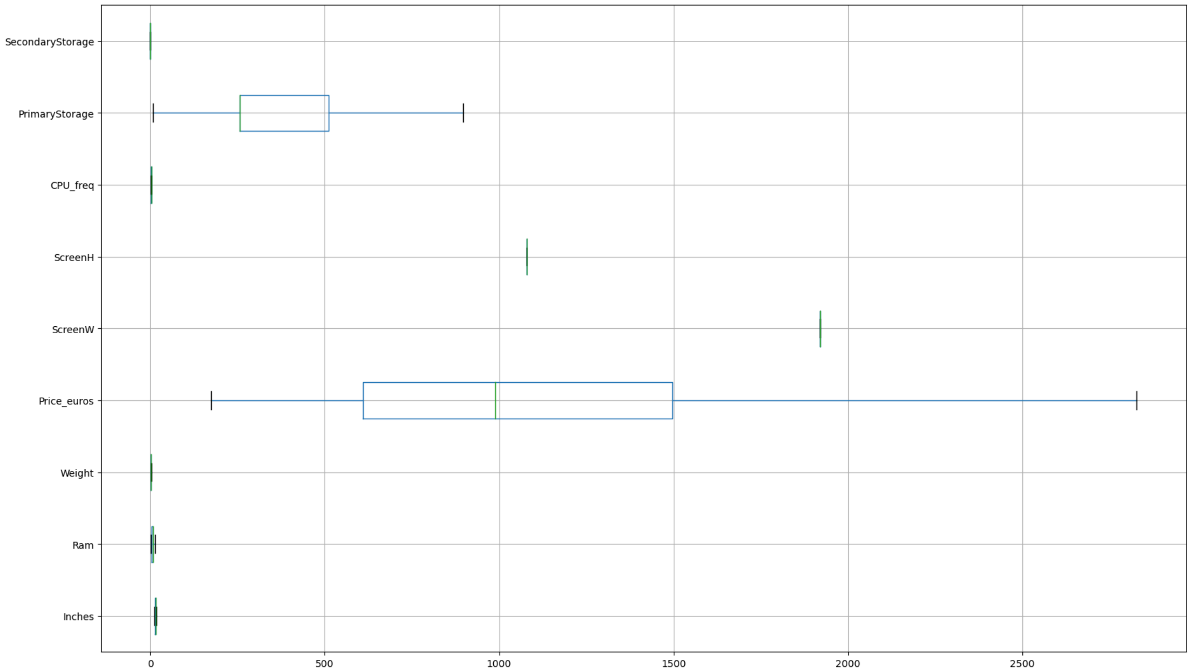
In [13]:

```
def remove_outlier(col):
  # Convert column to numeric before sorting (handling potential errors)
  col = pd.to_numeric(col, errors='coerce')
  sorted_col = sorted(col)
  Q1, Q3 = np.percentile(sorted_col, [25, 75])
  IQR = Q3 - Q1
  lower_range = Q1 - (1.5 * IQR)
  upper_range = Q3 + (1.5 * IQR)
  return lower_range, upper_range

# Assuming 'df' is your DataFrame
```

```
for column in df.columns:
  lower, upper = remove_outlier(df[column])
  df[column] = np.where(df[column] > upper, upper, df[column])
  df[column] = np.where(df[column] < lower, lower, df[column])

# Now 'df' has outliers removed (assuming numerical columns)
```

In [14]:
```
# Identification of Outliers using boxplot

plt.figure(figsize=(20,12))
df.boxplot(vert = 0)
```

Out[14]: <Axes: >



1. Univariate Graphs

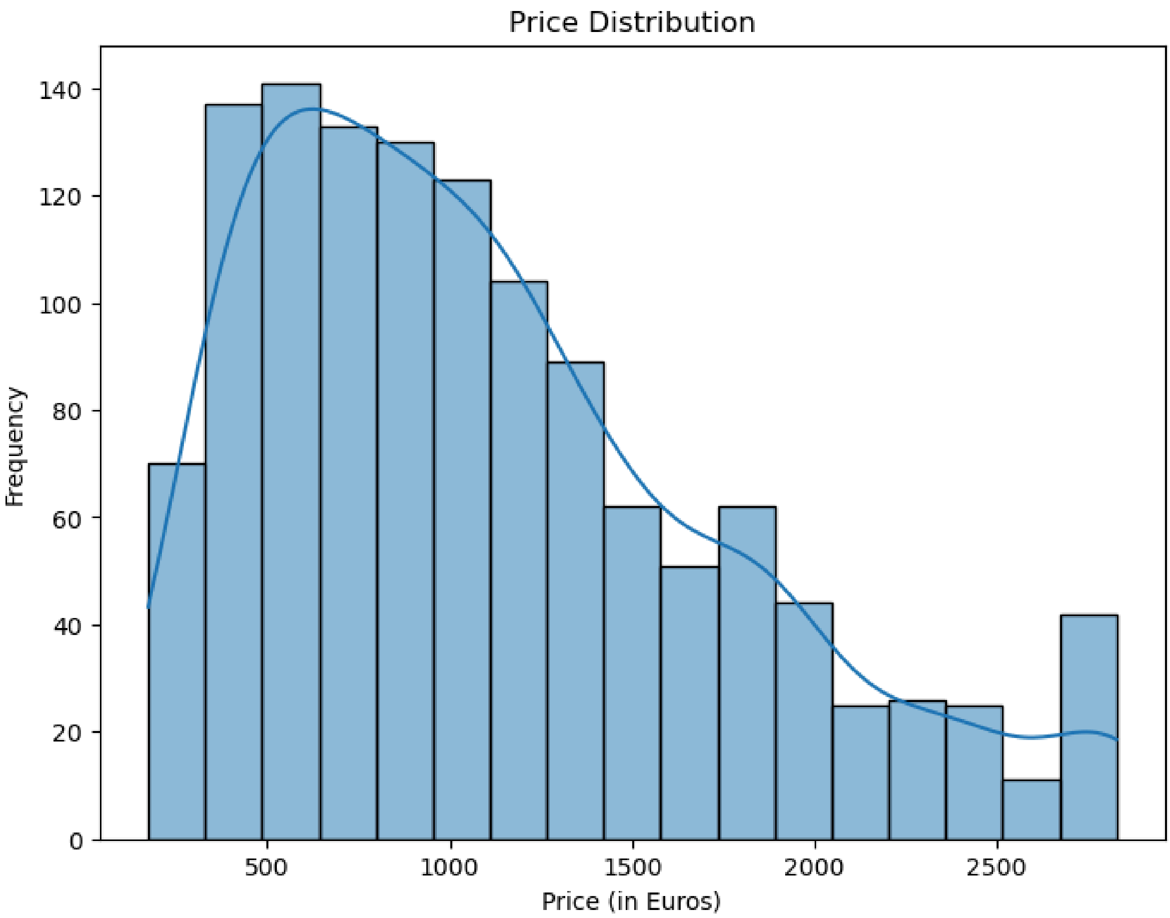
These are used to understand the distribution of individual features, particularly the target variable Price.

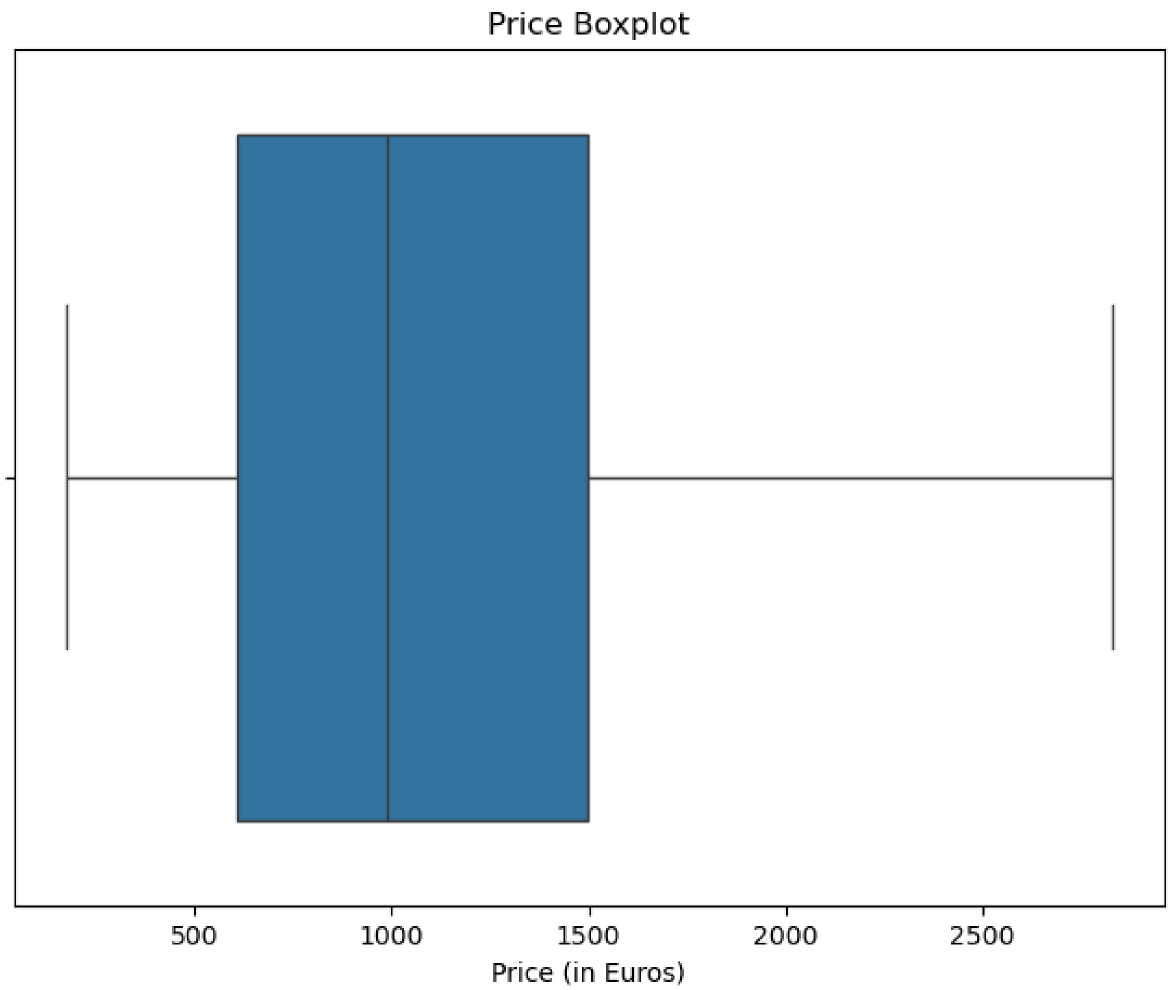
Histogram for Price: To see the distribution of laptop prices.

Boxplot for Price: To visualize the spread and outliers in the price distribution.

In [15]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Univariate Graphs
# Histogram of Price
plt.figure(figsize=(8, 6))
sns.histplot(df['Price_euros'], kde=True)
plt.title('Price Distribution')
plt.xlabel('Price (in Euros)')
plt.ylabel('Frequency')
plt.show()

# Boxplot of Price
plt.figure(figsize=(8, 6))
```

```
sns.boxplot(x=df['Price_euros'])
plt.title('Price Boxplot')
plt.xlabel('Price (in Euros)')
plt.show()
```





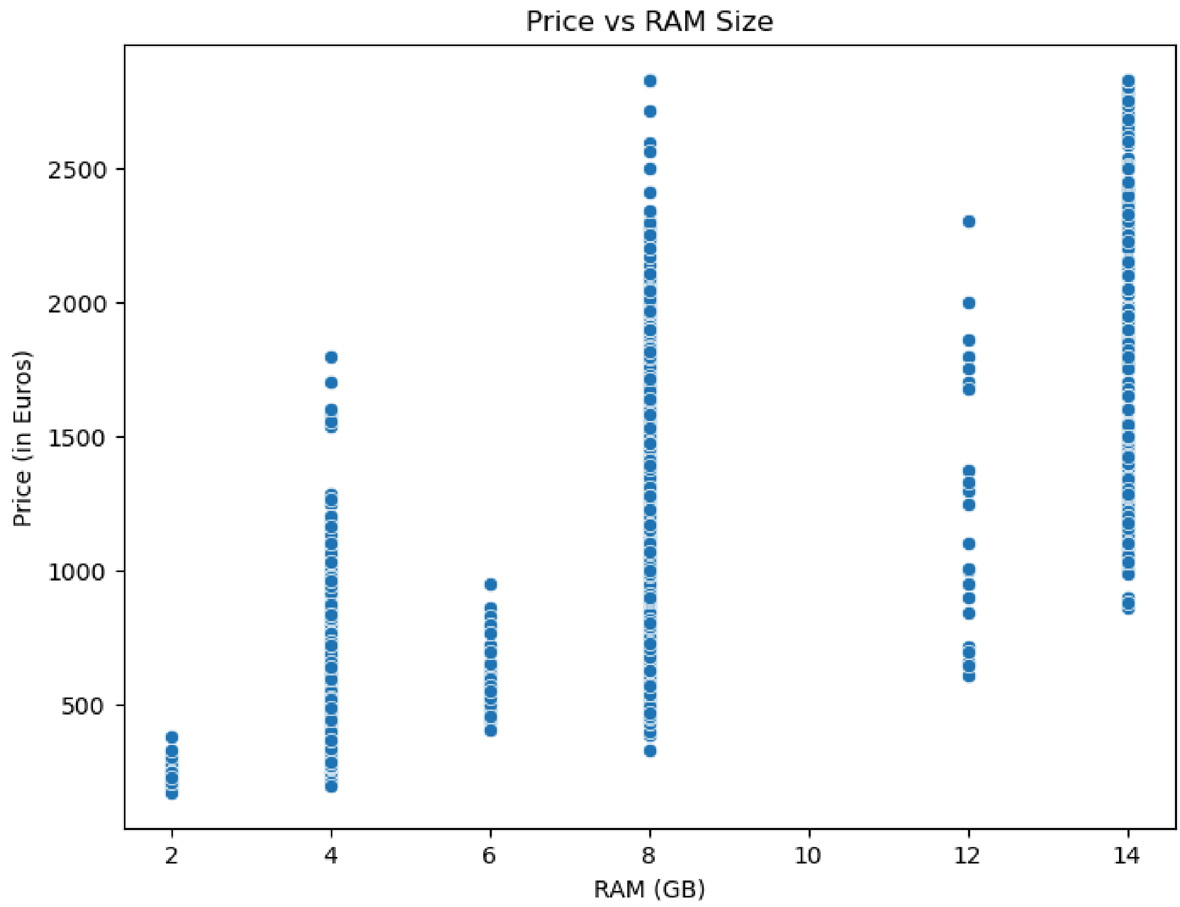
2. Bivariate Graphs

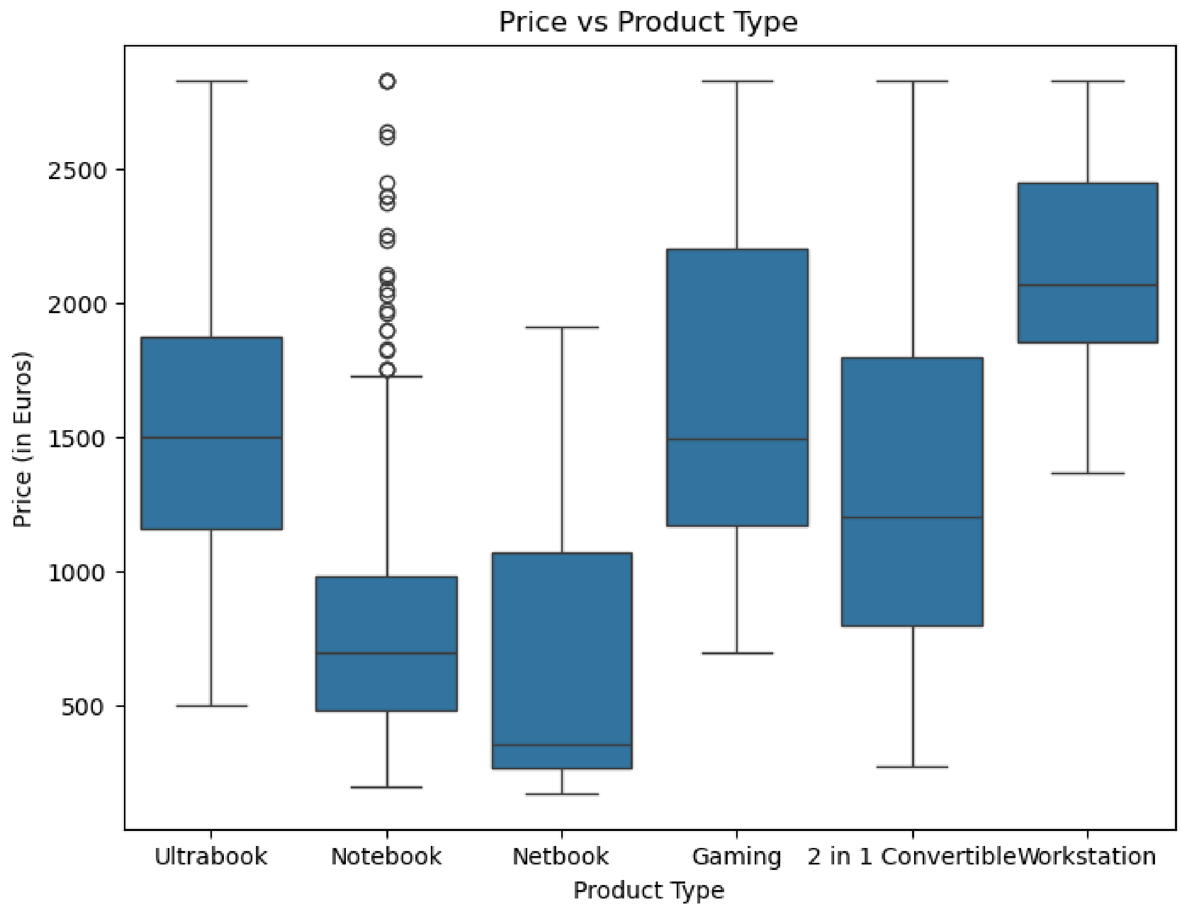
These graphs are useful to explore the relationships between two variables (features and the target variable).

Scatter plot for Price vs RAM: To see how RAM size affects laptop price.

Boxplot for Price vs Product Type: To compare prices across different laptop types.

In [16]:
```
# Bivariate Graphs
# Scatter plot for Price vs RAM
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['Ram'], y=df['Price_euros'])
plt.title('Price vs RAM Size')
plt.xlabel('RAM (GB)')
plt.ylabel('Price (in Euros)')
plt.show()

# Boxplot for Price vs Product Type
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['TypeName'], y=df['Price_euros'])
plt.title('Price vs Product Type')
plt.xlabel('Product Type')
plt.ylabel('Price (in Euros)')
plt.show()
```





3. Multivariate Graphs

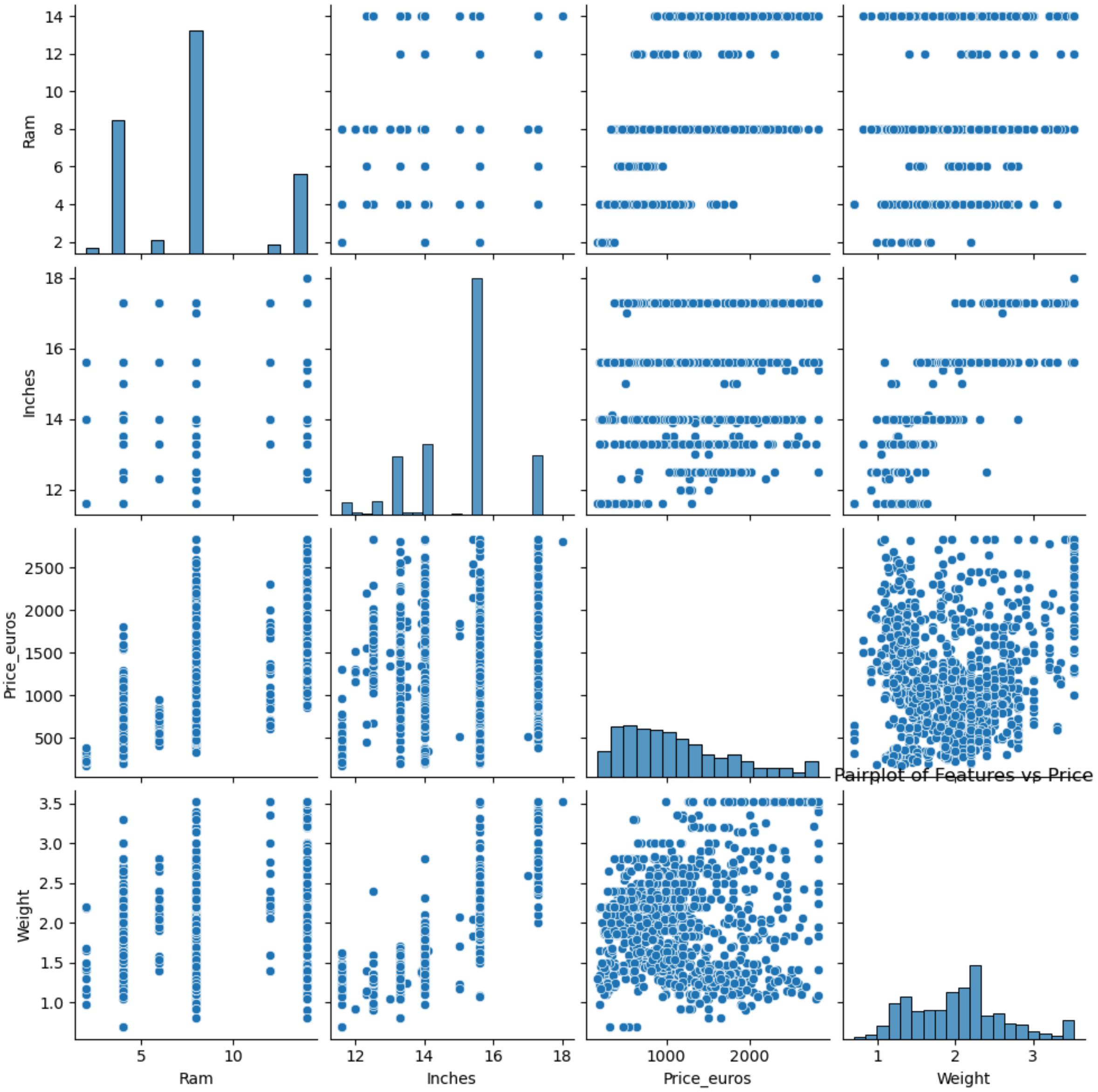
These graphs help visualize the relationships between more than two variables.

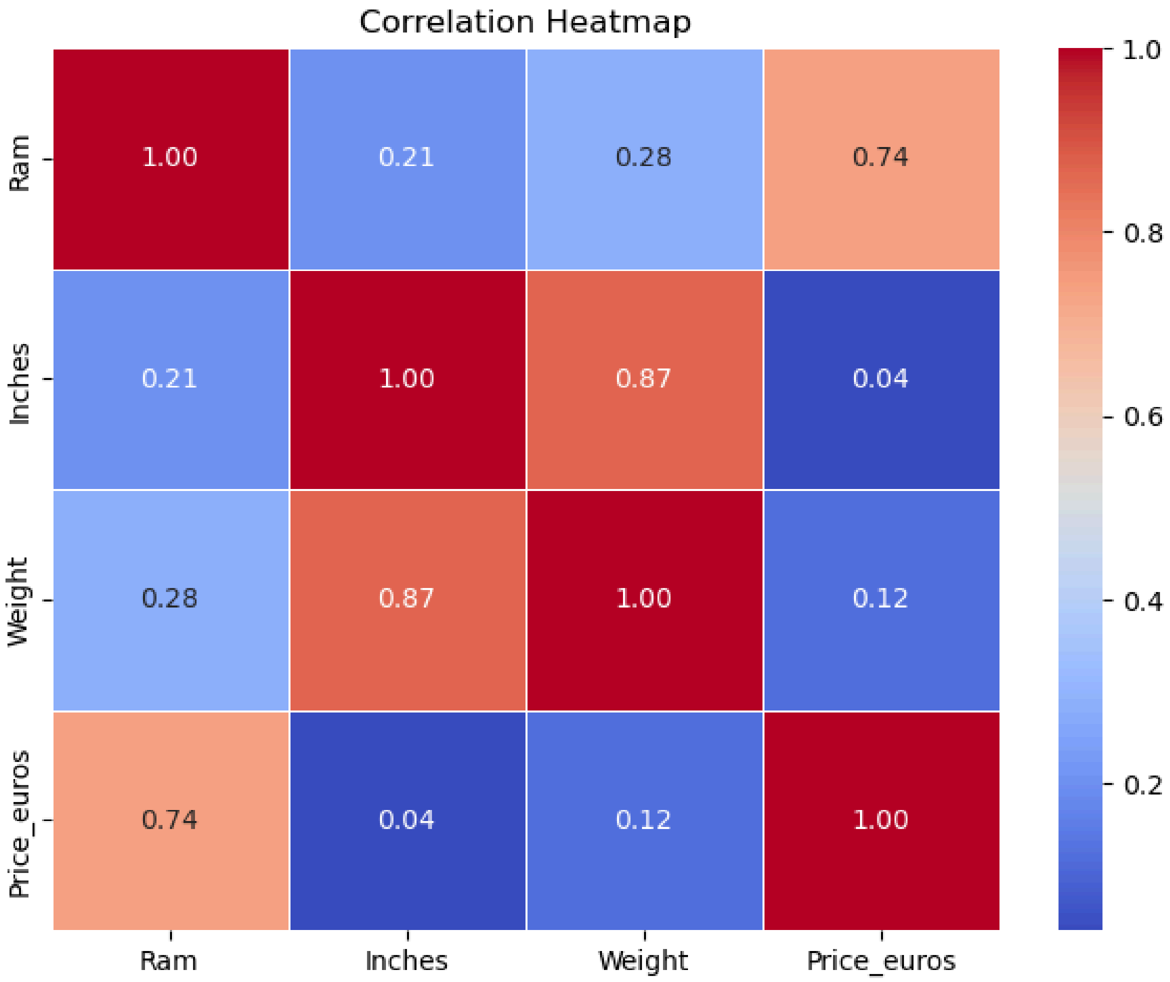
Pairplot: To visualize the relationship between several numerical features.

Heatmap of Correlation Matrix: To see the correlation between numerical features, especially with the target variable Price.

In [17]:
```
# Multivariate Graphs
# Pairplot for multiple features
sns.pairplot(df[['Ram', 'Inches', 'Price_euros', 'Weight']])
plt.title('Pairplot of Features vs Price')
plt.show()

# Correlation Heatmap for numerical features
corr_matrix = df[['Ram', 'Inches', 'Weight', 'Price_euros']].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

```
C:\Users\chira\anaconda3\Lib\site-packages\seaborn\axisgrid.py:123: UserWarning: T
he figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```





4. Categorical vs Numerical Data

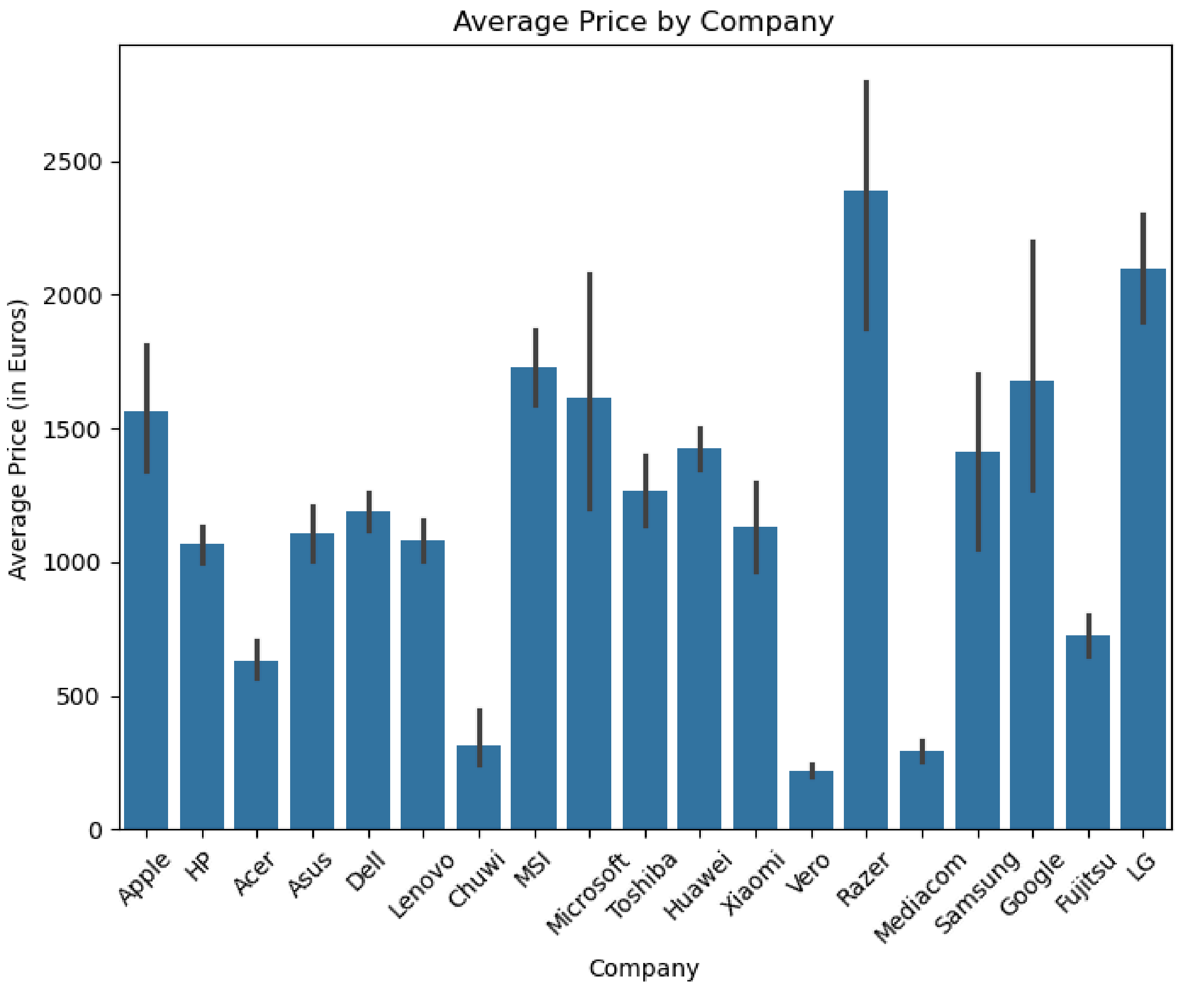
For categorical features like Company and Operating System, we can visualize how they influence the target variable (Price).

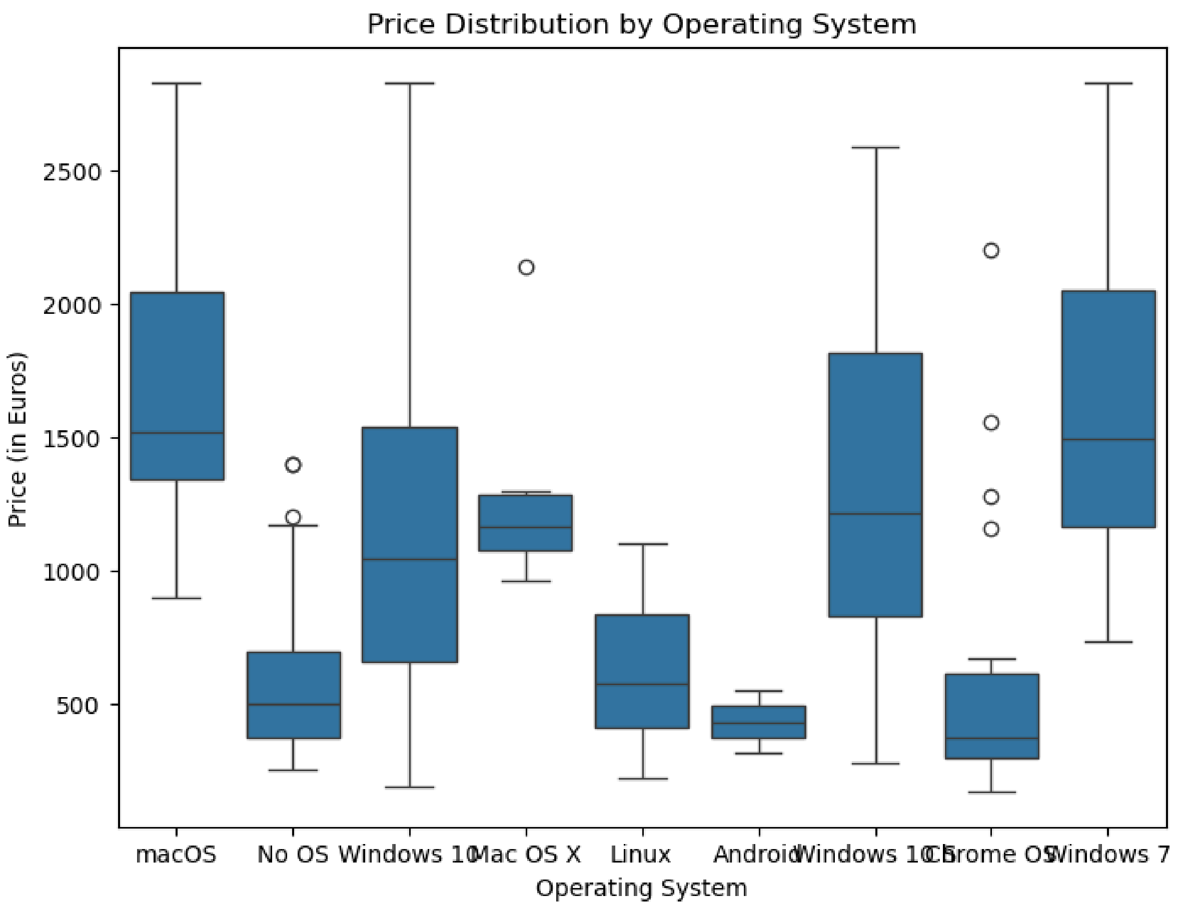
Bar plot for Price vs Company: To see how the price varies across different companies.

Boxplot for Price vs OS: To compare the price distribution across different operating systems.

In [18]:
```
# Categorical vs Numerical Graphs
# Bar plot for Price vs Company
plt.figure(figsize=(8, 6))
sns.barplot(x=df['Company'], y=df['Price_euros'])
plt.title('Average Price by Company')
plt.xlabel('Company')
plt.ylabel('Average Price (in Euros)')
plt.xticks(rotation=45)
plt.show()

# Boxplot for Price vs OS
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['OS'], y=df['Price_euros'])
plt.title('Price Distribution by Operating System')
plt.xlabel('Operating System')
```

```
plt.ylabel('Price (in Euros)')
plt.show()
```

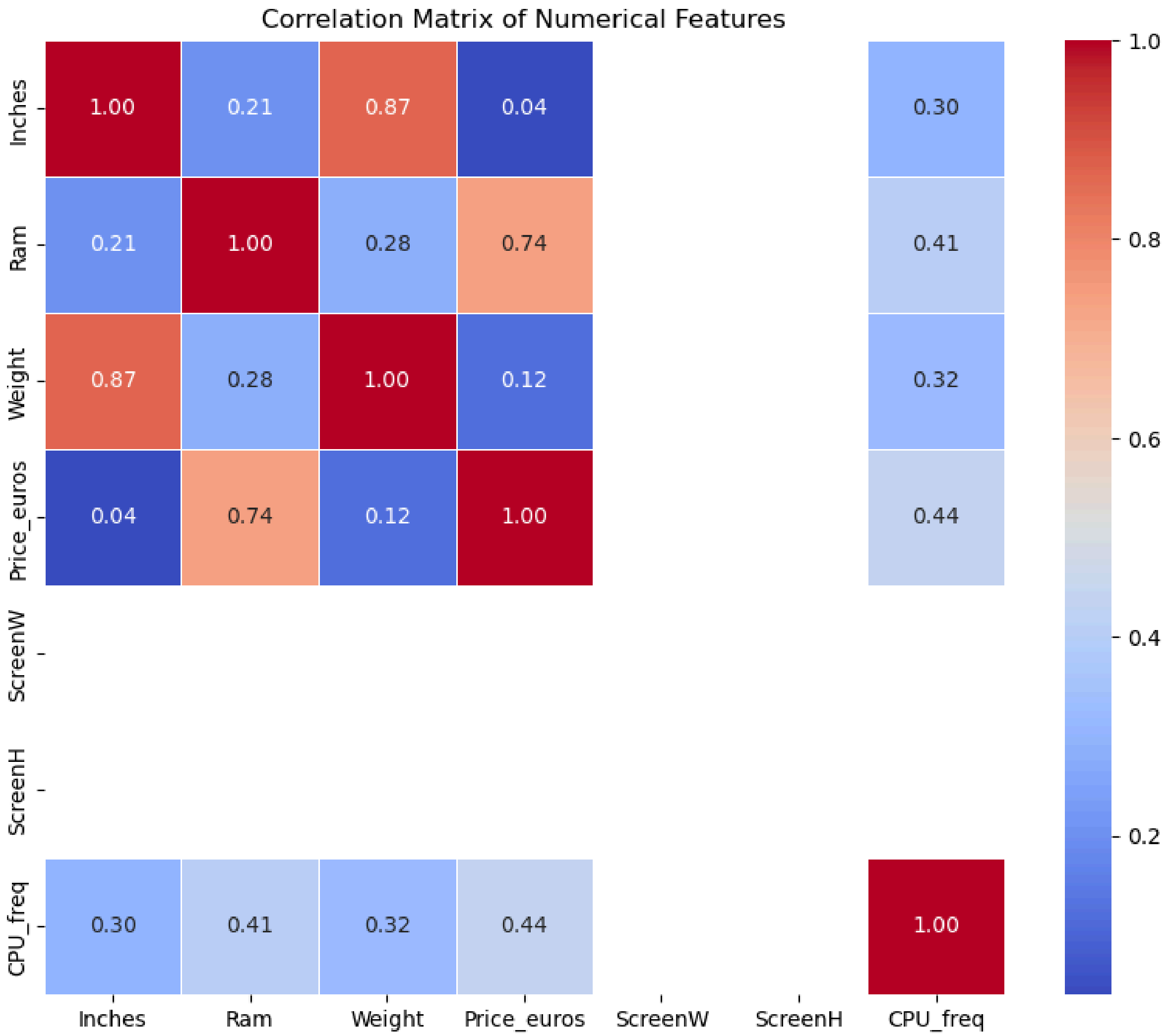




5. Correlation Between Features

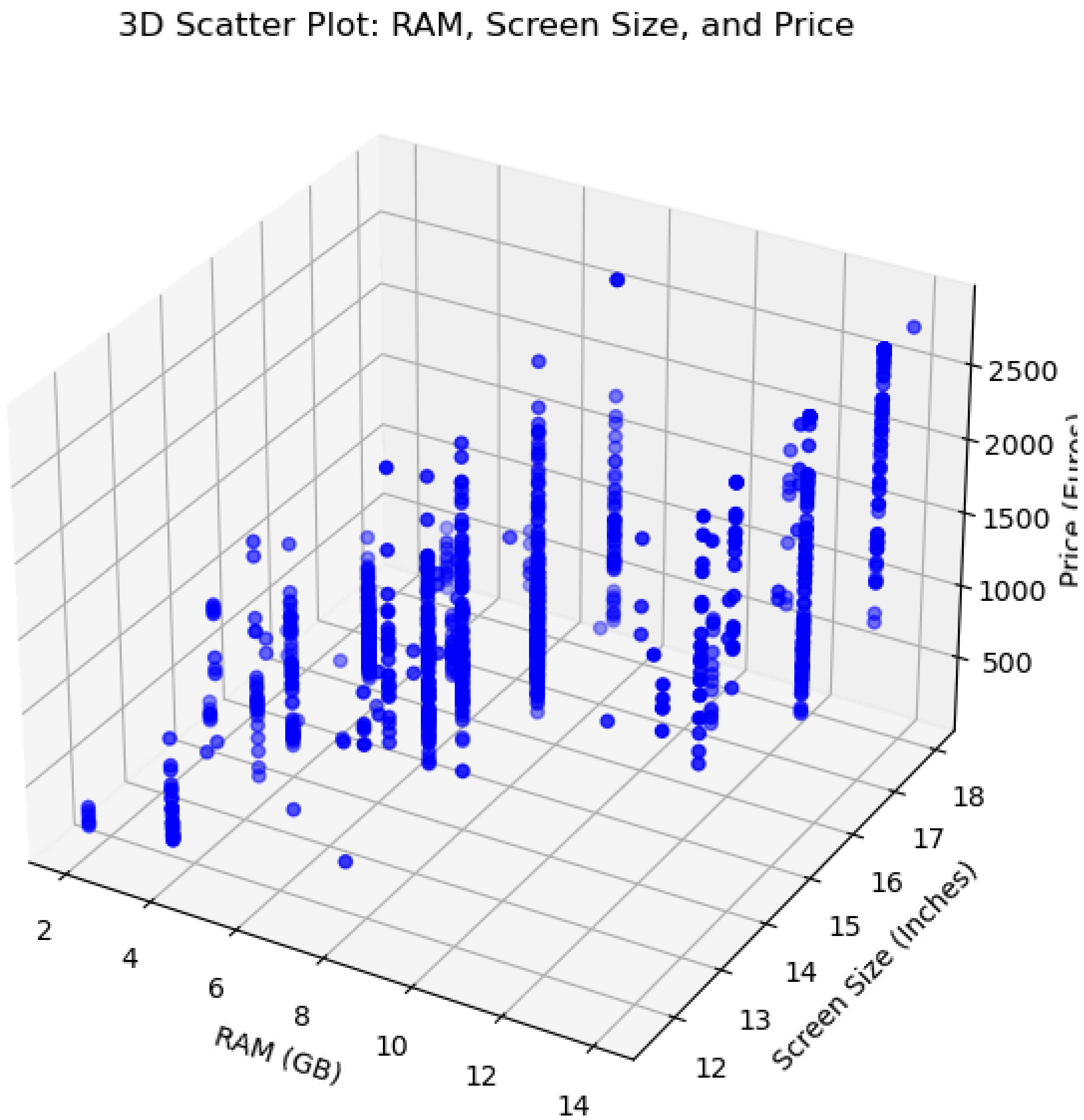
You can also visualize how all the numerical features correlate with each other and the target variable Price.

In [19]:
```
# Correlation heatmap for all numerical features
numerical_features = df[['Inches', 'Ram', 'Weight', 'Price_euros', 'ScreenW', 'Scre
corr_matrix = numerical_features.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



In [20]:
```
# Sample data (replace df with your actual DataFrame)
X = df[['Ram', 'Inches', 'Price_euros']]  # Features: RAM, Screen Size, and Price

# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
ax.scatter(X['Ram'], X['Inches'], X['Price_euros'], c='blue', marker='o')

# Set labels
ax.set_xlabel('RAM (GB)')
ax.set_ylabel('Screen Size (Inches)')
ax.set_zlabel('Price (Euros)')
```

```python
# Title of the plot
ax.set_title('3D Scatter Plot: RAM, Screen Size, and Price')

# Show plot
plt.show()
```



Features Engineering and selection

In [21]:
```python
from sklearn.preprocessing import LabelEncoder
```

In [22]:
```python
# Step 1: Label Encoding for 'Company', 'OS', and 'TypeName'
label_encoder = LabelEncoder()
df['Company'] = label_encoder.fit_transform(df['Company'])
df['OS'] = label_encoder.fit_transform(df['OS'])
df['Product'] = label_encoder.fit_transform(df['Product'])
df['TypeName'] = label_encoder.fit_transform(df['TypeName'])
df['Touchscreen'] = label_encoder.fit_transform(df['Touchscreen'])
df['IPSpanel'] = label_encoder.fit_transform(df['IPSpanel'])
df['RetinaDisplay'] = label_encoder.fit_transform(df['RetinaDisplay'])
df['CPU_company'] = label_encoder.fit_transform(df['CPU_company'])
df['PrimaryStorageType'] = label_encoder.fit_transform(df['PrimaryStorageType'])
df['SecondaryStorageType'] = label_encoder.fit_transform(df['SecondaryStorageType']
```

```
df['GPU_company'] = label_encoder.fit_transform(df['GPU_company'])
df['GPU_model'] = label_encoder.fit_transform(df['GPU_model'])
```

In [23]:
```
# Perform One-Hot Encoding using pandas get_dummies()
df_encoded = pd.get_dummies(df, columns=[
    'Company', 'Product', 'TypeName', 'OS', 'Screen', 'Touchscreen',
    'IPSpanel', 'RetinaDisplay', 'CPU_company', 'PrimaryStorageType',
    'SecondaryStorageType', 'GPU_company', 'GPU_model'])
```

In [24]:
```
from sklearn.preprocessing import StandardScaler

# Step 1: One-Hot Encode the 'Screen' column (and other categorical columns)
df_encoded = pd.get_dummies(df, drop_first=True)

# Step 2: Select only the numerical columns for scaling
numerical_cols = df_encoded.select_dtypes(include=['float64', 'int64']).columns

# Step 3: Apply scaling only to numerical columns
scaler = StandardScaler()
df_scaled = df_encoded[numerical_cols]
df_scaled = scaler.fit_transform(df_scaled)
```

In [ ]:
```

```

In [25]:
```
from sklearn.model_selection import train_test_split

# Assuming 'df_encoded' is the DataFrame after One-Hot Encoding and 'Price_euros' i
X = df_encoded.drop('Price_euros', axis=1)  # Features
y = df_encoded['Price_euros']  # Target variable

# Split the data into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [26]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the test data
X_test_scaled = scaler.transform(X_test)
```

In [27]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the test data
X_test_scaled = scaler.transform(X_test)
```

In [28]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
```

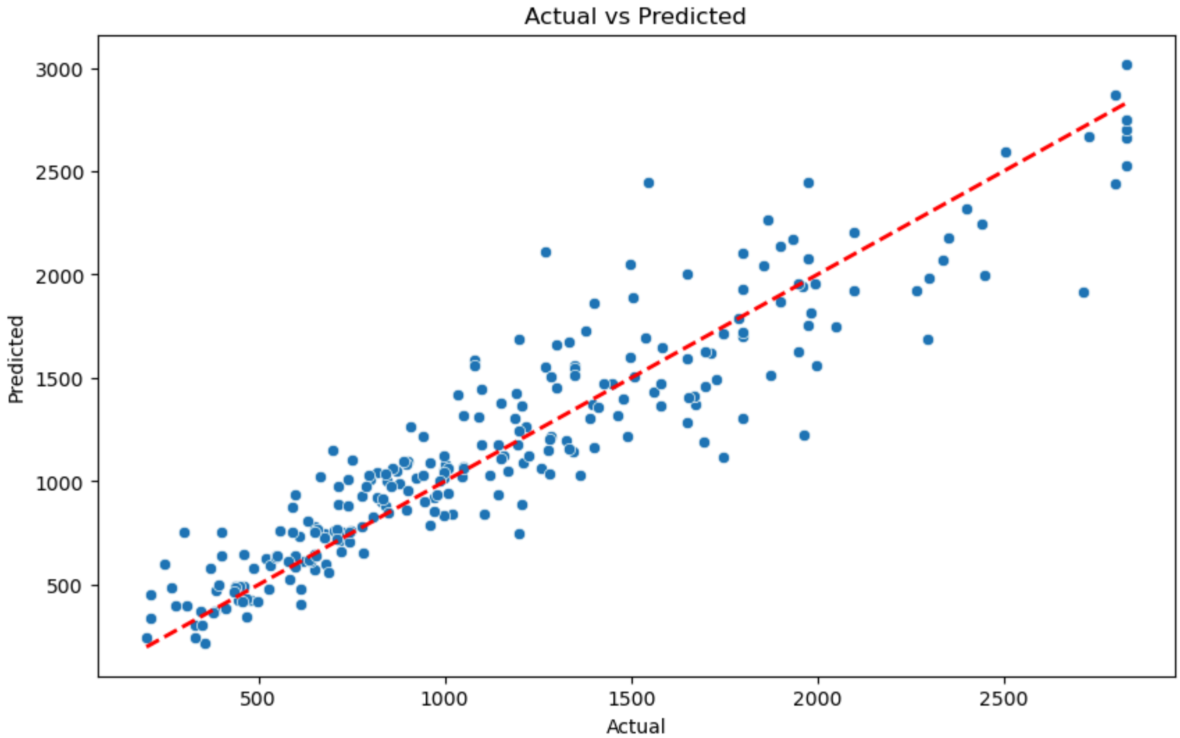
```python
from sklearn.tree import DecisionTreeRegressor
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [29]:
```python
# 1. **Linear Regression**
model_lr = LinearRegression()
model_lr.fit(X_train_scaled, y_train)
y_pred_lr = model_lr.predict(X_test_scaled)
mse_lr = mean_squared_error(y_test, y_pred_lr)
print(f'Linear Regression - MSE: {mse_lr}')

# 2. **Random Forest Regressor**
model_rf = RandomForestRegressor(n_estimators=100, random_state=42)
model_rf.fit(X_train_scaled, y_train)
y_pred_rf = model_rf.predict(X_test_scaled)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
print(f'Random Forest - MAE: {mae_rf}')

# 3. **Gradient Boosting Regressor**
model_gbr = GradientBoostingRegressor(n_estimators=100, random_state=42)
model_gbr.fit(X_train_scaled, y_train)
y_pred_gbr = model_gbr.predict(X_test_scaled)
r2_gbr = r2_score(y_test, y_pred_gbr)
print(f'Gradient Boosting Regressor - R²: {r2_gbr}')

# 4. **XGBoost**
model_xgb = xgb.XGBRegressor(n_estimators=100, random_state=42)
model_xgb.fit(X_train_scaled, y_train)
y_pred_xgb = model_xgb.predict(X_test_scaled)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
print(f'XGBoost - MSE: {mse_xgb}')

# 5. **Support Vector Regressor (SVR)**
model_svr = SVR(kernel='rbf')
model_svr.fit(X_train_scaled, y_train)
y_pred_svr = model_svr.predict(X_test_scaled)
mae_svr = mean_absolute_error(y_test, y_pred_svr)
print(f'Support Vector Regressor - MAE: {mae_svr}')

# 6. **K-Nearest Neighbors Regressor (KNN)**
model_knn = KNeighborsRegressor(n_neighbors=5)
model_knn.fit(X_train_scaled, y_train)
y_pred_knn = model_knn.predict(X_test_scaled)
mae_knn = mean_absolute_error(y_test, y_pred_knn)
print(f'K-Nearest Neighbors - MAE: {mae_knn}')

# 7. **Decision Tree Regressor**
model_dtr = DecisionTreeRegressor(random_state=42)
model_dtr.fit(X_train_scaled, y_train)
y_pred_dtr = model_dtr.predict(X_test_scaled)
mae_dtr = mean_absolute_error(y_test, y_pred_dtr)
print(f'Decision Tree Regressor - MAE: {mae_dtr}')
```

```
Linear Regression - MSE: 1.4640046271770353e+27
Random Forest - MAE: 153.287477723934
Gradient Boosting Regressor - R²: 0.864336006402988
XGBoost - MSE: 43632.79866991747
Support Vector Regressor - MAE: 471.9802322738659
K-Nearest Neighbors - MAE: 215.95213333333336
Decision Tree Regressor - MAE: 213.4928039215686
```

In [35]:
```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Assuming `final_model` is the model you selected after hyperparameter tuning
# Final Model Training with the best model (e.g., RandomForestRegressor after tunir
model_gbr.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_final = model_gbr.predict(X_test_scaled)

# Evaluate the final model performance
mse_final = mean_squared_error(y_test, y_pred_final)
mae_final = mean_absolute_error(y_test, y_pred_final)
r2_final = r2_score(y_test, y_pred_final)

# Plot predicted vs actual values
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred_final)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--r', lw=2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted')
plt.show()

# Calculate and print evaluation metrics
print(f'Mean Squared Error (MSE): {mse_final}')
print(f'Mean Absolute Error (MAE): {mae_final}')
print(f'R²: {r2_final}')
```




```
Mean Squared Error (MSE): 52698.106376150514
Mean Absolute Error (MAE): 168.4955087505294
R²: 0.864336006402988
```

In [ ]:

In [31]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: