```python
# STEP 1 — Setup & Load Data (Upload from Device)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Optional: nice plotting style
sns.set(style="whitegrid", palette="muted", font_scale=1.2)

# Upload file from device
from google.colab import files
uploaded = files.upload()   # This will prompt you to choose a file

# Get the filename (first key in the uploaded dict)
file_name = list(uploaded.keys())[0]

# Load dataset (try utf-8 first, fallback to ISO-8859-1 if needed)
try:
    df = pd.read_csv(file_name, encoding="utf-8")
except UnicodeDecodeError:
    df = pd.read_csv(file_name, encoding="ISO-8859-1")

# Quick overview
print("Dataset shape:", df.shape)
print("\nData types info:")
print(df.info())

print("\nFirst 10 rows:")
display(df.head(10))

print("\nSummary statistics (numeric columns):")
display(df.describe())

print("\nMissing values per column:")
print(df.isnull().sum())
```

```
<IPython.core.display.HTML object>

Saving laptop_prices.csv to laptop_prices (1).csv
Dataset shape: (1275, 23)

Data types 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
None

First 10 rows:
```

{"type":"dataframe"}

```
Summary statistics (numeric columns):
```

{"summary":"{\n \"name\": \"print(df\",\n \"rows\": 8,\n
\"fields\": [\n   {\n   \"column\": \"Inches\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
446.2575963637752,\n   \"min\": 1.4294698446247902,\n
\"max\": 1275.0,\n   \"num_unique_values\": 7,\n
\"samples\": [\n   1275.0,\n   15.022901960784312,\n
15.6\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Ram\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 446.22170839681405,\n   \"min\": 2.0,\n   \"max\":
1275.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n
1275.0,\n   8.440784313725490,\n   8.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Weight\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
450.07777142385805,\n   \"min\": 0.6691959759708271,\n
\"max\": 1275.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   2.0405254901960785,\n   2.04,\n
1275.0\n   ],\n   \"semantic_type\": \"\",\n

\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Price_euros\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1880.659968304601,\n   \"min\":
174.0,\n   \"max\": 6099.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   1134.9690588235292,\n   989.0,\n
1275.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ScreenW\",\n   \"properties\": {\n   \"dtype\": \"number\",\
n   \"std\": 954.632811576843,\n   \"min\":
493.3461863720428,\n   \"max\": 3840.0,\n
\"num_unique_values\": 6,\n   \"samples\": [\n   1275.0,\n
1900.0439215686274,\n   3840.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"ScreenH\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
525.4080496963254,\n   \"min\": 283.8839398555495,\n
\"max\": 2160.0,\n   \"num_unique_values\": 6,\n
\"samples\": [\n   1275.0,\n   1073.9043137254903,\n
2160.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"CPU_freq\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 450.0489474584384,\n   \"min\":
0.5038457085709569,\n   \"max\": 1275.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
2.302980392156863,\n   2.5,\n   1275.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"PrimaryStorage\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
677.1471819125817,\n   \"min\": 8.0,\n   \"max\": 2048.0,\n
\"num_unique_values\": 7,\n   \"samples\": [\n   1275.0,\n
444.5176470588235,\n   512.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"SecondaryStorage\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
766.0679191248988,\n   \"min\": 0.0,\n   \"max\": 2048.0,\n
\"num_unique_values\": 5,\n   \"samples\": [\n
176.06901960784313,\n   2048.0,\n   415.96065537272585\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

```
Missing values per column:
Company                 0
Product                 0
TypeName                0
Inches                  0
Ram                     0
OS                      0
Weight                  0
Price_euros             0
```

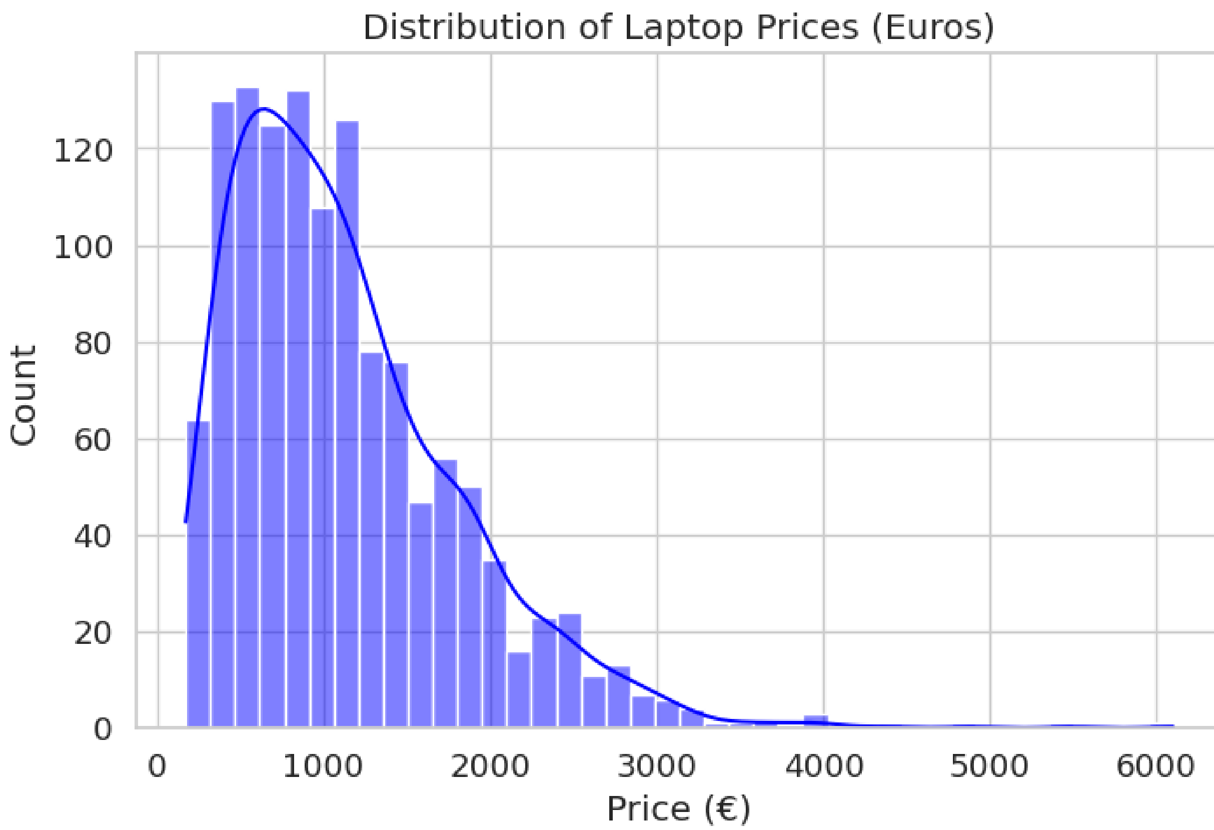
```
Screen                  0
ScreenW                 0
ScreenH                 0
Touchscreen             0
IPSpanel                0
RetinaDisplay           0
CPU_company             0
CPU_freq                0
CPU_model               0
PrimaryStorage          0
SecondaryStorage        0
PrimaryStorageType      0
SecondaryStorageType    0
GPU_company             0
GPU_model               0
dtype: int64
```

```
# STEP 2 — Data Preprocessing

# ----------------------------
# 1) Check and drop duplicates
# ----------------------------
print("Duplicates before:", df.duplicated().sum())
df = df.drop_duplicates()
print("Duplicates after:", df.duplicated().sum())

# ----------------------------
# 2) Convert boolean-like cols
# ----------------------------
bool_cols = ["Touchscreen", "IPSpanel", "RetinaDisplay"]

for col in bool_cols:
    df[col] = df[col].apply(lambda x: 1 if str(x).strip().lower() ==
"yes" else 0)

# ----------------------------
# 3) Normalize categorical text
# ----------------------------
text_cols = ["Company", "OS", "CPU_company", "GPU_company"]

for col in text_cols:
    df[col] = df[col].str.strip().str.lower()

# ----------------------------
# 4) Extract CPU & GPU families
# ----------------------------
def simplify_cpu(cpu_name):
    cpu_name = cpu_name.lower()
    if "i3" in cpu_name: return "i3"
    elif "i5" in cpu_name: return "i5"
```

```python
    elif "i7" in cpu_name: return "i7"
    elif "i9" in cpu_name: return "i9"
    elif "ryzen 3" in cpu_name: return "ryzen3"
    elif "ryzen 5" in cpu_name: return "ryzen5"
    elif "ryzen 7" in cpu_name: return "ryzen7"
    elif "m1" in cpu_name or "m2" in cpu_name: return "apple_silicon"
    else: return "other"

def simplify_gpu(gpu_name):
    gpu_name = gpu_name.lower()
    if "nvidia" in gpu_name: return "nvidia"
    elif "amd" in gpu_name: return "amd"
    elif "intel" in gpu_name: return "intel"
    elif "apple" in gpu_name: return "apple"
    else: return "other"

df["CPU_family"] = df["CPU_model"].apply(simplify_cpu)
df["GPU_family"] = df["GPU_model"].apply(simplify_gpu)

# ---------------------
```

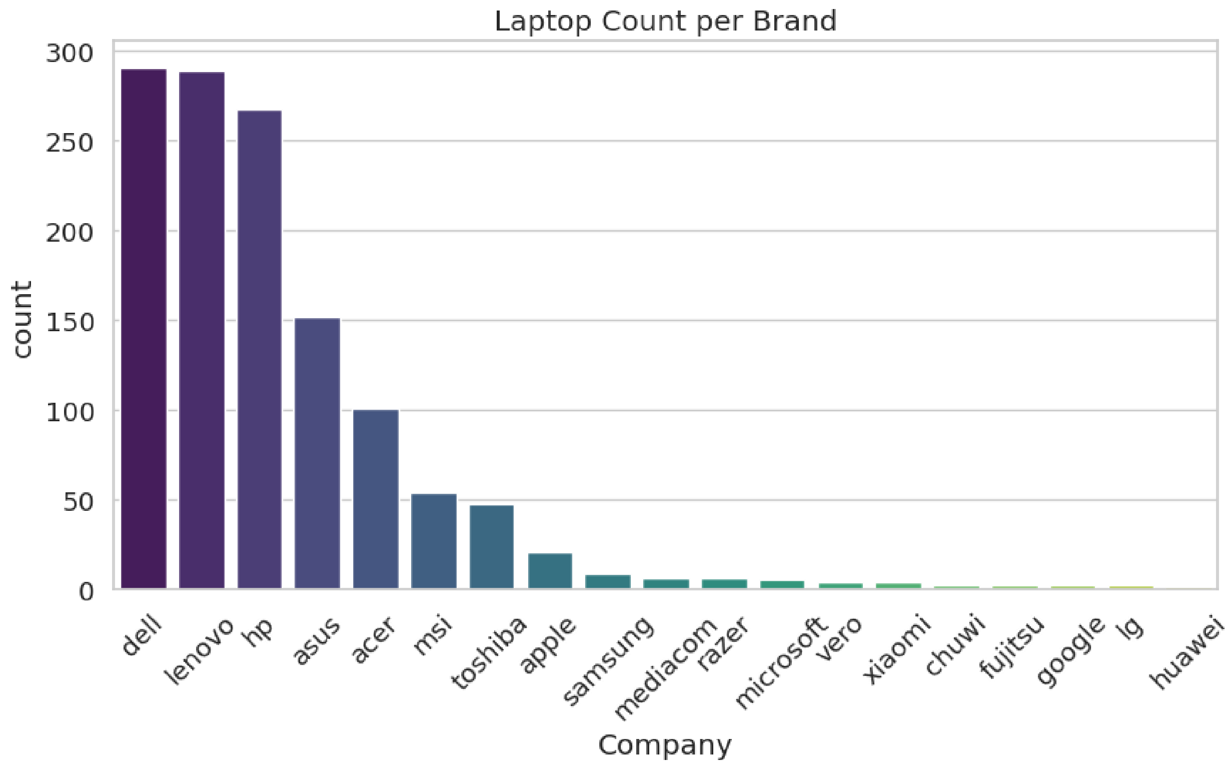
```
Duplicates before: 0
Duplicates after: 0
```

```python
plt.figure(figsize=(8,5))
sns.histplot(df["Price_euros"], kde=True, bins=40, color="blue")
plt.title("Distribution of Laptop Prices (Euros)")
plt.xlabel("Price (€)")
plt.ylabel("Count")
plt.show()
```



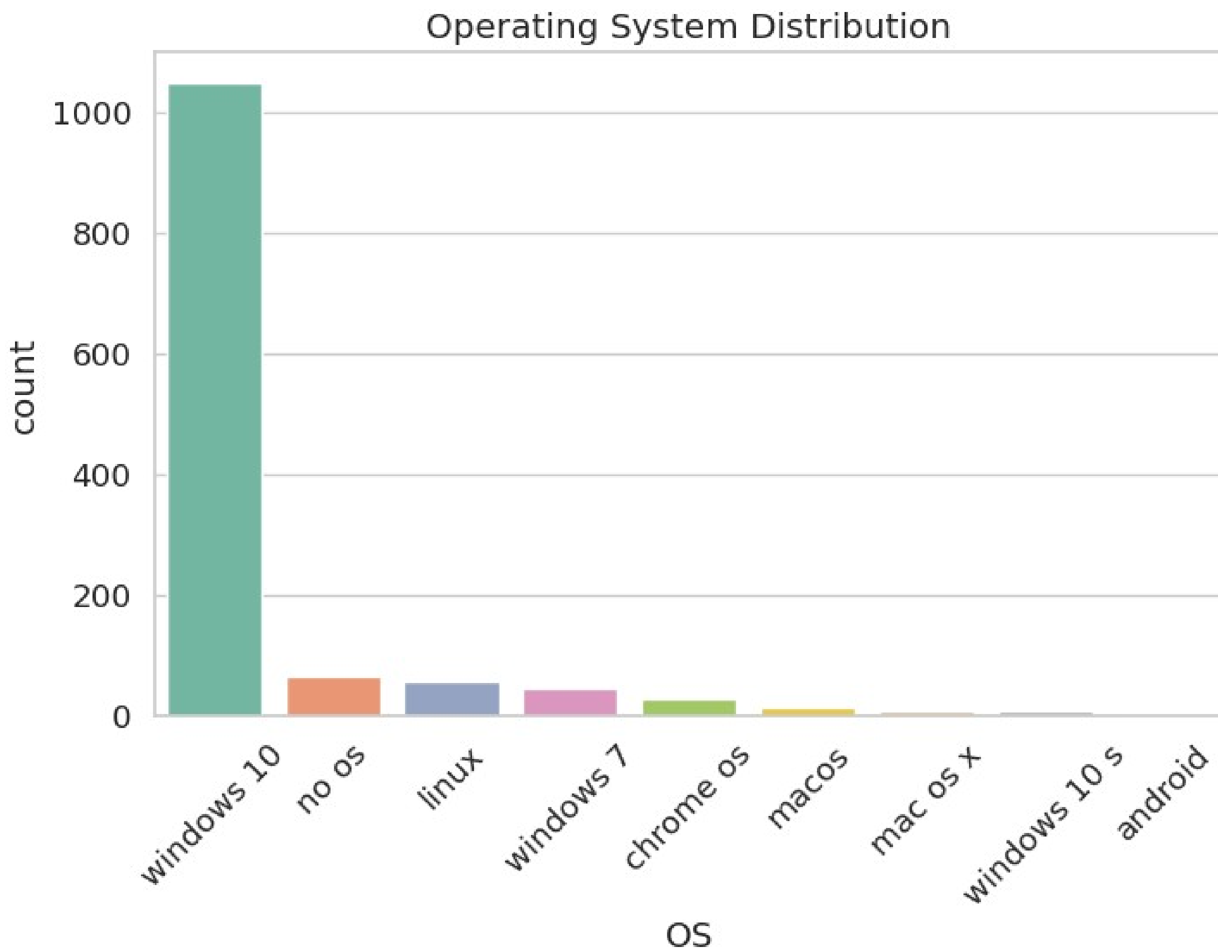

```
plt.figure(figsize=(10,5))
sns.countplot(data=df, x="Company",
order=df["Company"].value_counts().index, palette="viridis")
plt.title("Laptop Count per Brand")
plt.xticks(rotation=45)
plt.show()
```

```
/tmp/ipython-input-1386938155.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(data=df, x="Company",
order=df["Company"].value_counts().index, palette="viridis")
```



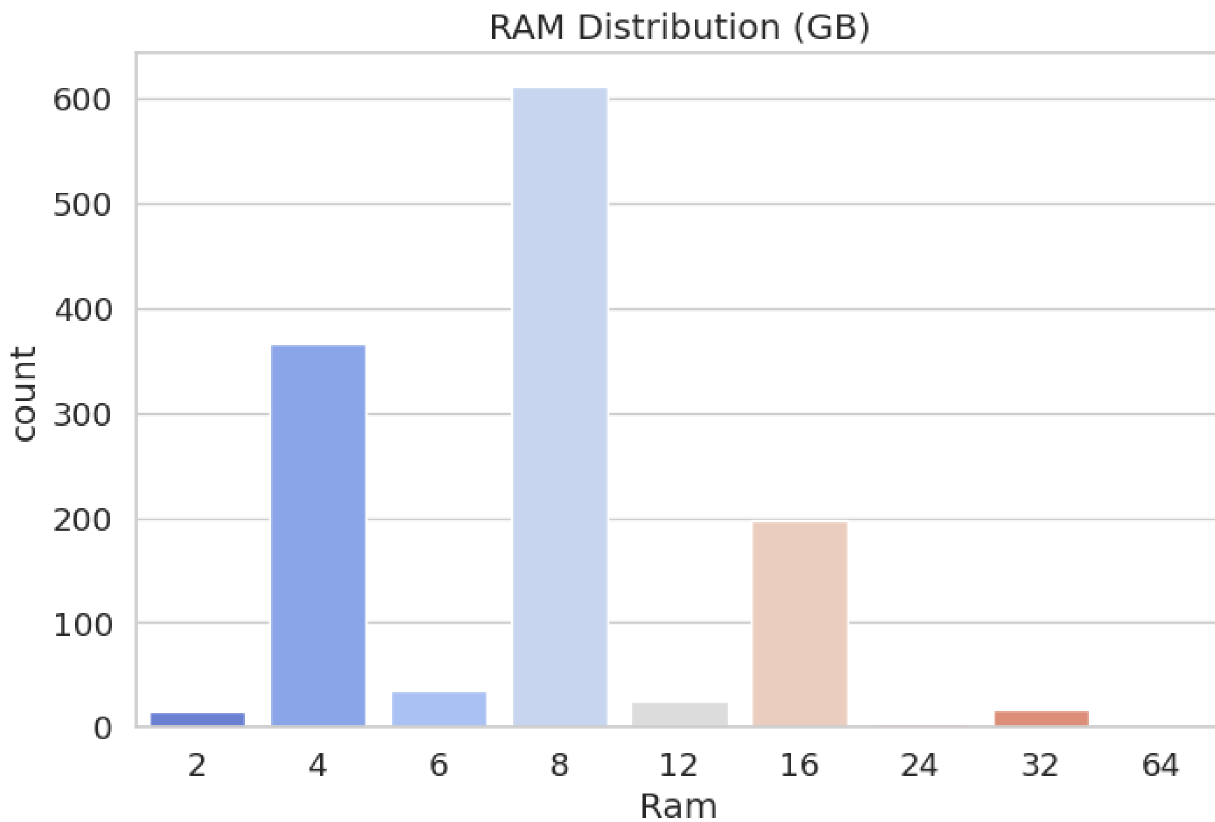

```
plt.figure(figsize=(8,5))
sns.countplot(data=df, x="OS", order=df["OS"].value_counts().index,
palette="Set2")
plt.title("Operating System Distribution")
plt.xticks(rotation=45)
plt.show()
```

```
/tmp/ipython-input-1227816152.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(data=df, x="OS", order=df["OS"].value_counts().index,
palette="Set2")
```



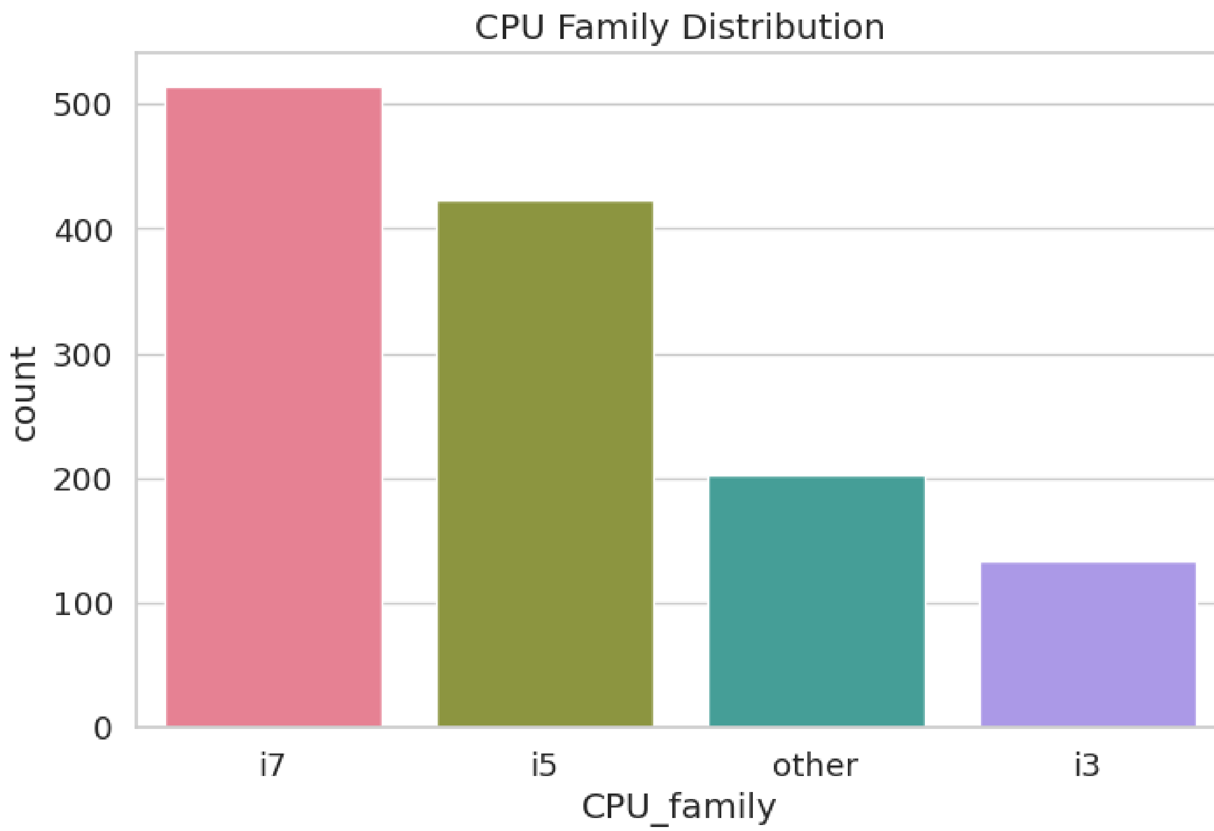

```
plt.figure(figsize=(8,5))
sns.countplot(data=df, x="Ram", palette="coolwarm")
plt.title("RAM Distribution (GB)")
plt.show()
```

```
/tmp/ipython-input-2063285871.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(data=df, x="Ram", palette="coolwarm")
```



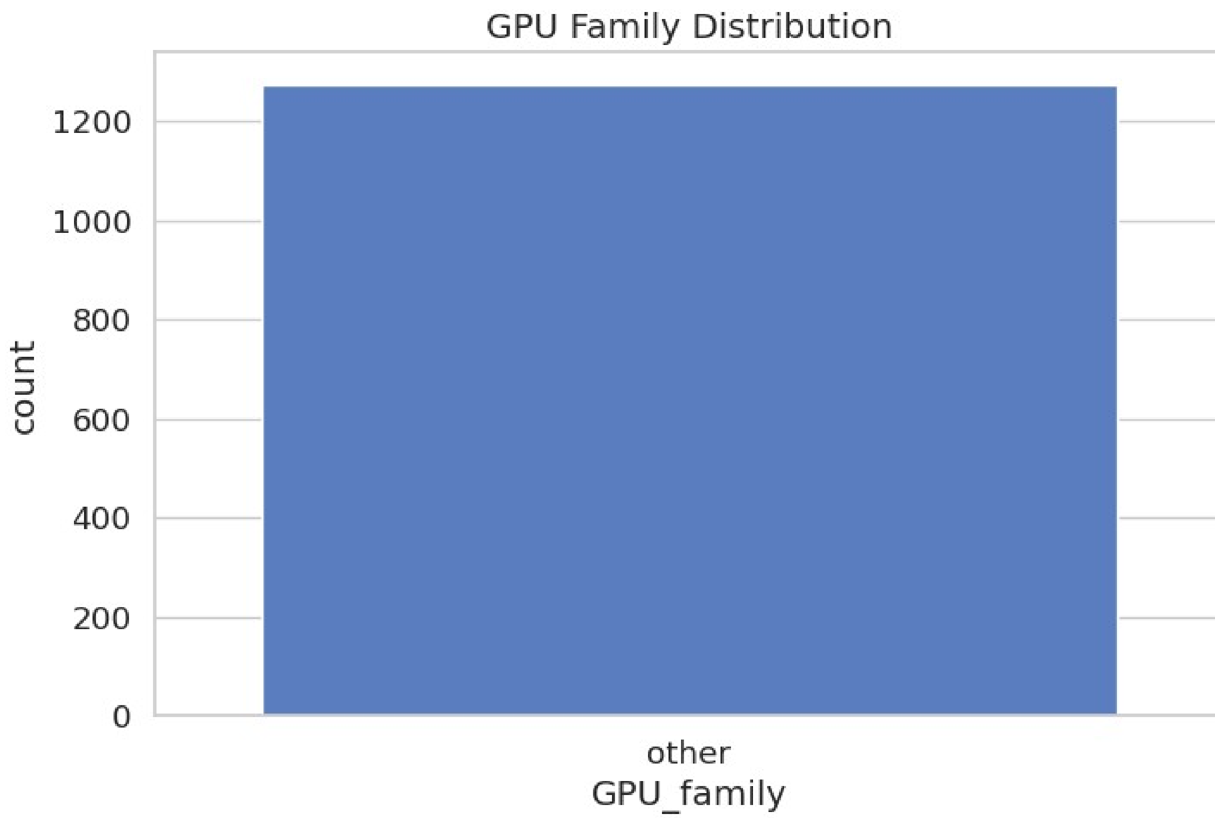

```
plt.figure(figsize=(8,5))
sns.countplot(data=df, x="CPU_family",
order=df["CPU_family"].value_counts().index, palette="husl")
plt.title("CPU Family Distribution")
plt.show()
```

```
/tmp/ipython-input-4029702452.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(data=df, x="CPU_family",
order=df["CPU_family"].value_counts().index, palette="husl")
```



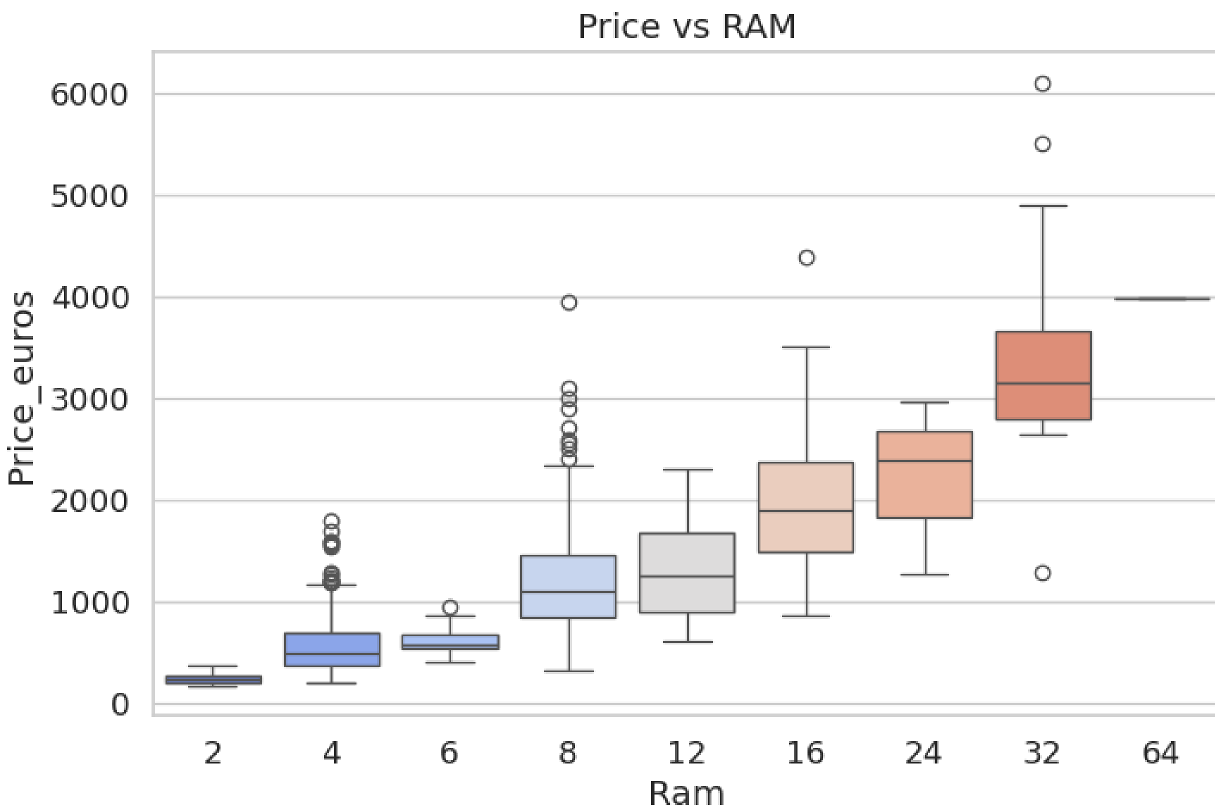

```
plt.figure(figsize=(8,5))
sns.countplot(data=df, x="GPU_family",
order=df["GPU_family"].value_counts().index, palette="muted")
plt.title("GPU Family Distribution")
plt.show()
```

```
/tmp/ipython-input-1643724843.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(data=df, x="GPU_family",
order=df["GPU_family"].value_counts().index, palette="muted")
```

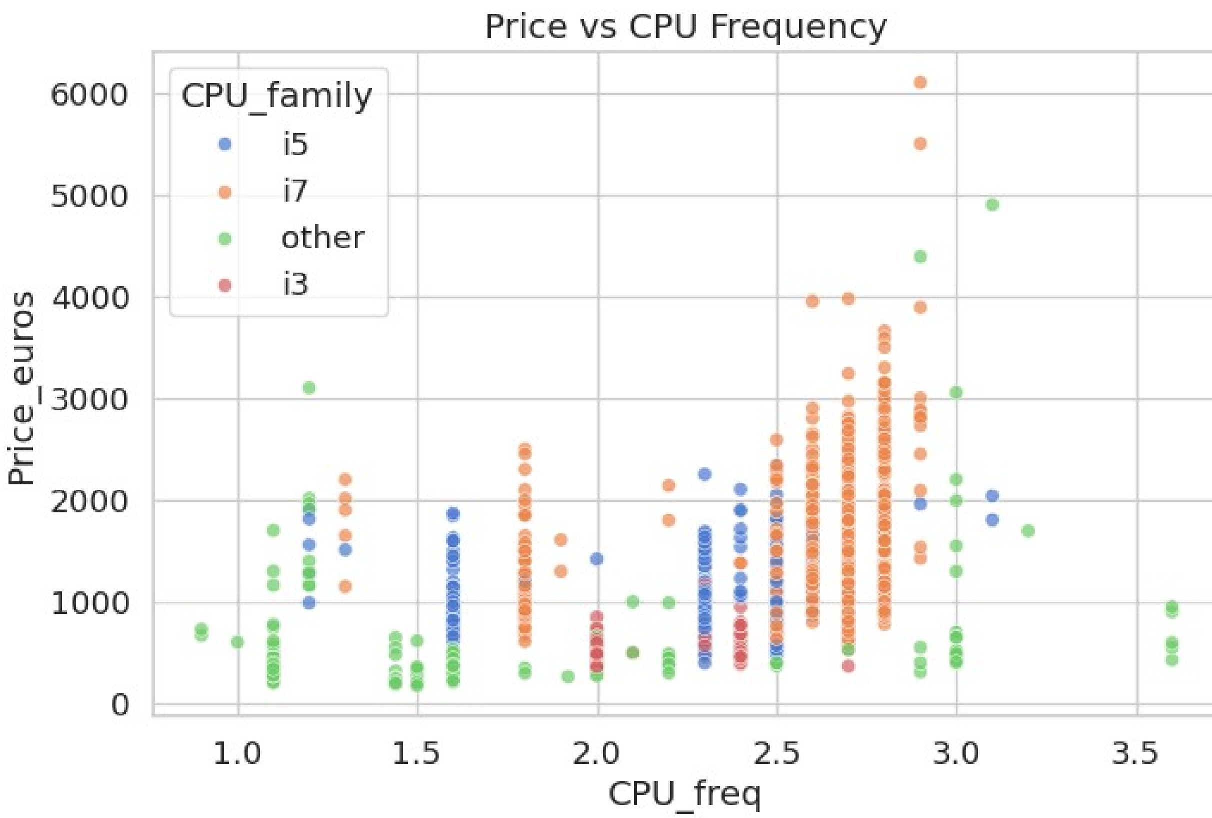



```
plt.figure(figsize=(8,5))
sns.boxplot(data=df, x="Ram", y="Price_euros", palette="coolwarm")
plt.title("Price vs RAM")
plt.show()
```

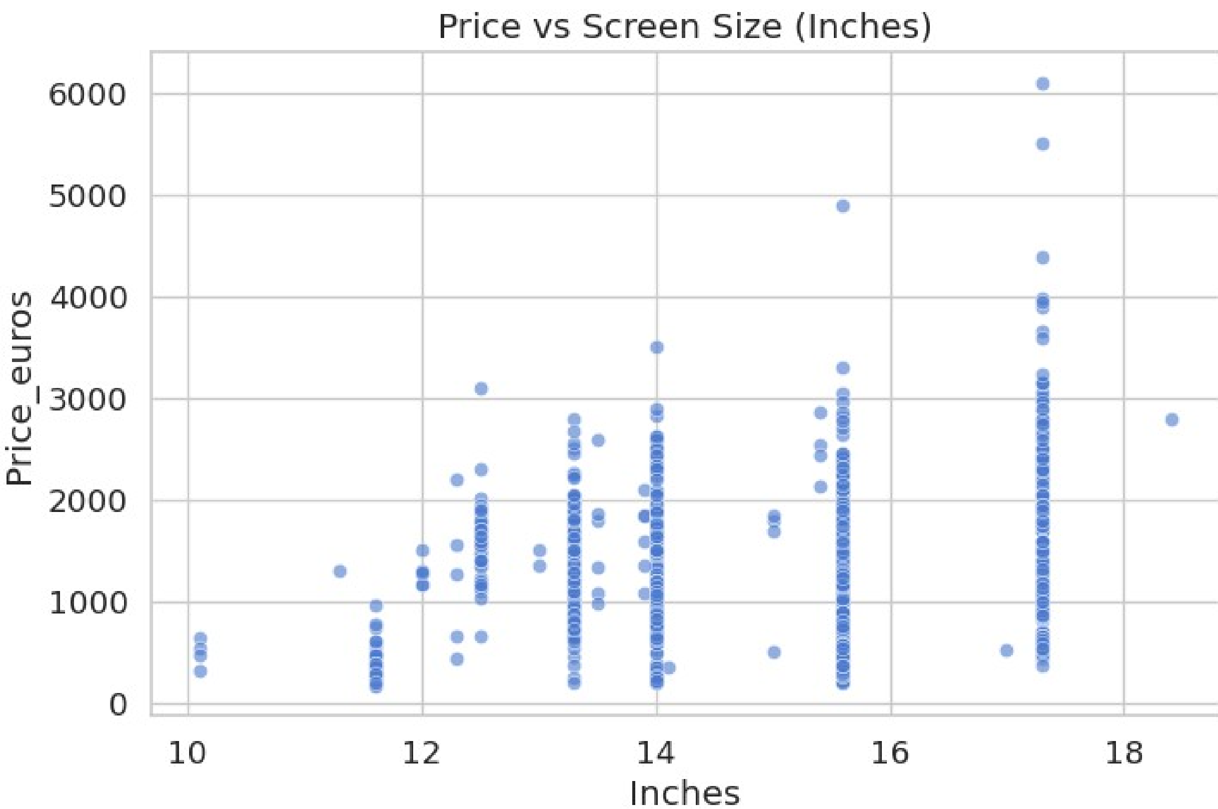
```
/tmp/ipython-input-510826442.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(data=df, x="Ram", y="Price_euros", palette="coolwarm")
```



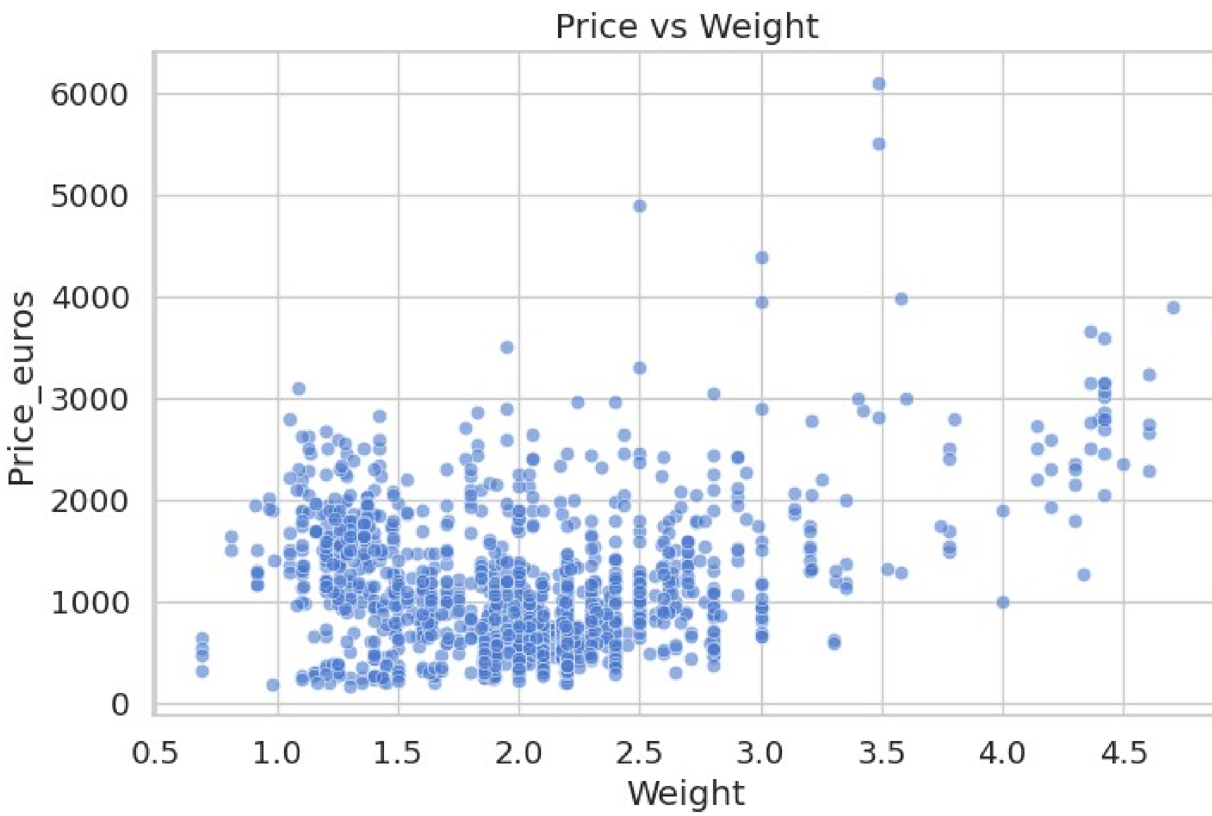

```
plt.figure(figsize=(8,5))
sns.scatterplot(data=df, x="CPU_freq", y="Price_euros",
hue="CPU_family", alpha=0.7)
plt.title("Price vs CPU Frequency")
plt.show()
```




```
plt.figure(figsize=(8,5))
sns.scatterplot(data=df, x="Inches", y="Price_euros", alpha=0.6)
plt.title("Price vs Screen Size (Inches)")
plt.show()
```

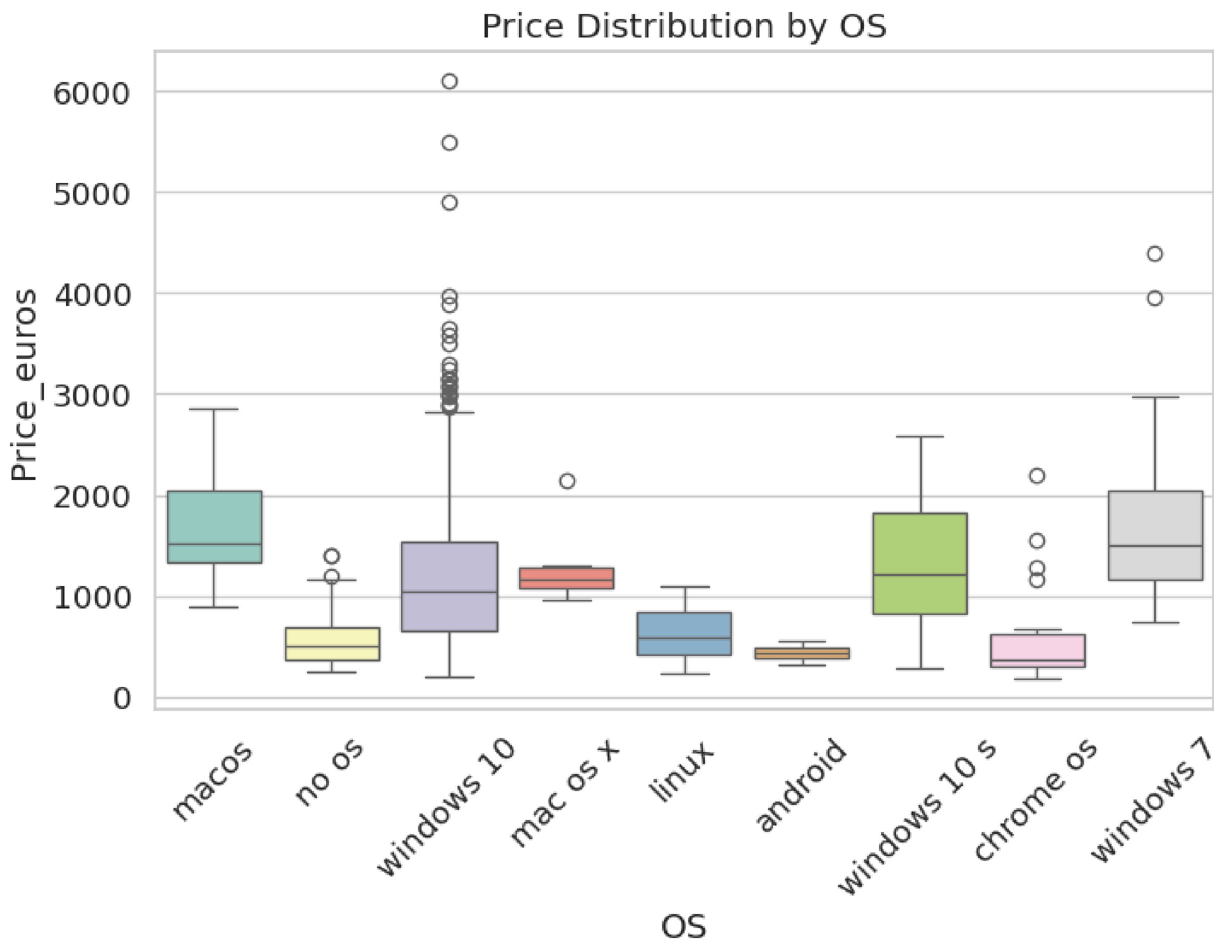



```
plt.figure(figsize=(8,5))
sns.scatterplot(data=df, x="Weight", y="Price_euros", alpha=0.6)
plt.title("Price vs Weight")
plt.show()
```



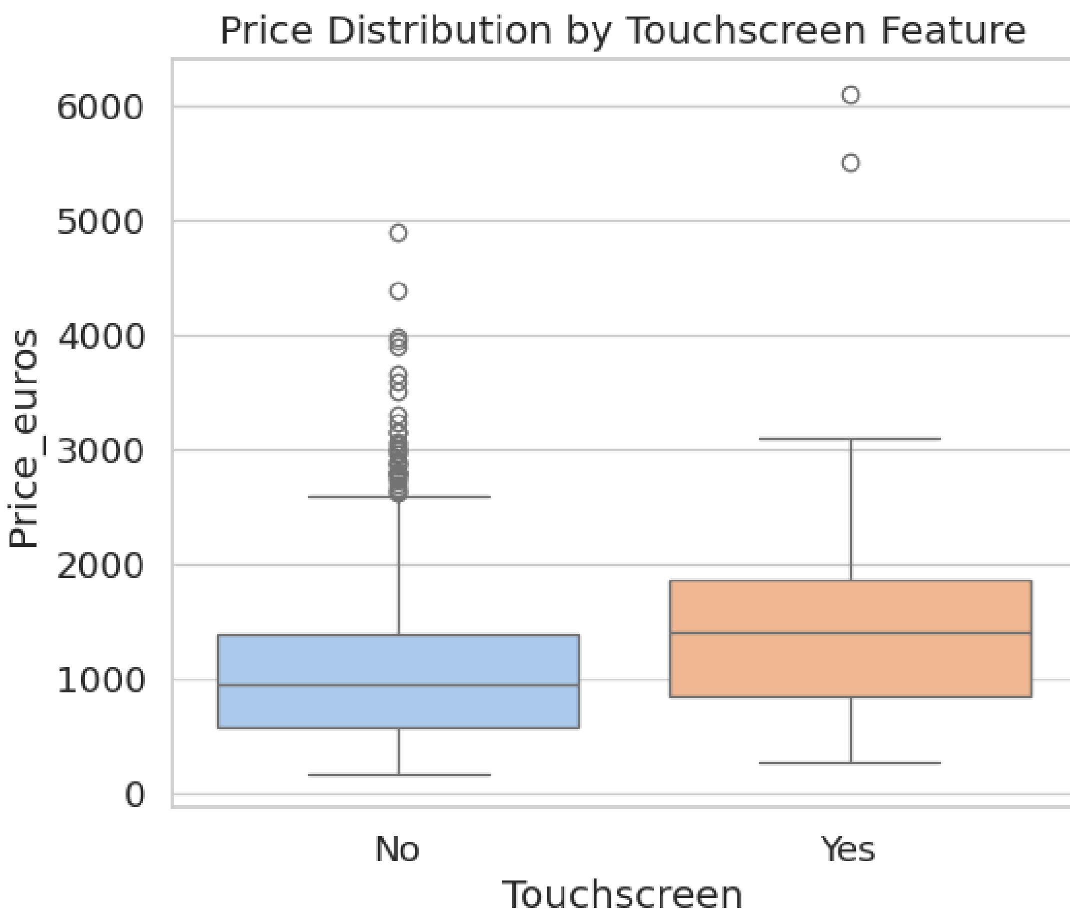

```python
plt.figure(figsize=(8,5))
sns.boxplot(data=df, x="OS", y="Price_euros", palette="Set3")
plt.title("Price Distribution by OS")
plt.xticks(rotation=45)
plt.show()
```

```
/tmp/ipython-input-93499344.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(data=df, x="OS", y="Price_euros", palette="Set3")
```

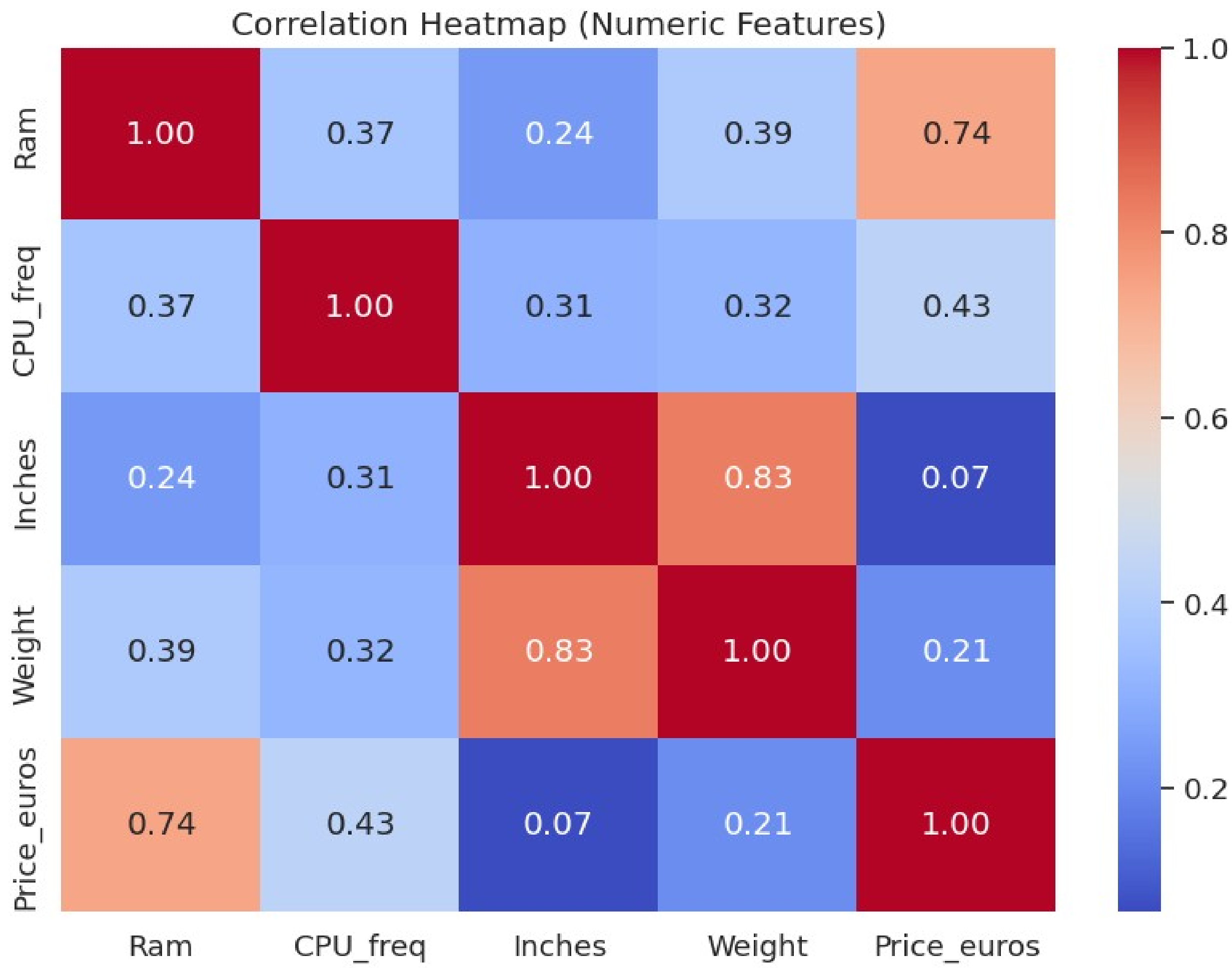



```
plt.figure(figsize=(6,5))
sns.boxplot(data=df, x="Touchscreen", y="Price_euros",
palette="pastel")
plt.title("Price Distribution by Touchscreen Feature")
plt.xticks([0,1], ["No", "Yes"])
plt.show()
```

```
/tmp/ipython-input-600235968.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(data=df, x="Touchscreen", y="Price_euros",
palette="pastel")
```




```
plt.figure(figsize=(10,7))
corr = df[["Ram", "CPU_freq", "Inches", "Weight",
"Price_euros"]].corr()
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap (Numeric Features)")
plt.show()
```




```
# STEP 4 — Feature Engineering

import numpy as np

# ------------------------------
# 1) Pixel density (ppi)
# ------------------------------
df["ppi"] = ((df["ScreenW"]**2 + df["ScreenH"]**2) ** 0.5 /
df["Inches"]).round(2)

# ------------------------------
# 2) Total storage
# ------------------------------
df["total_storage"] = df["PrimaryStorage"] + df["SecondaryStorage"]

# ------------------------------
# 3) Storage flags
# ------------------------------
df["ssd_flag"] =
((df["PrimaryStorageType"].str.lower().str.contains("ssd")) |
```

```
(df["SecondaryStorageType"].str.lower().str.contains("ssd"))).astype(i
nt)

df["hdd_flag"] =
((df["PrimaryStorageType"].str.lower().str.contains("hdd")) |

(df["SecondaryStorageType"].str.lower().str.contains("hdd"))).astype(i
nt)

df["hybrid_flag"] = ((df["ssd_flag"] == 1) & (df["hdd_flag"] ==
1)).astype(int)

# ----------------------------
# 4) CPU category
# ----------------------------
def cpu_category(cpu):
    if cpu in ["i3"]: return "i3"
    elif cpu in ["i5"]: return "i5"
    elif cpu in ["i7"]: return "i7"
    elif cpu in ["i9"]: return "i9"
    elif cpu in ["ryzen3"]: return "ryzen3"
    elif cpu in ["ryzen5"]: return "ryzen5"
    elif cpu in ["ryzen7"]: return "ryzen7"
    elif cpu in ["apple_silicon"]: return "apple_silicon"
    else: return "other"

df["cpu_category"] = df["CPU_family"].apply(cpu_category)

# ----------------------------
# 5) GPU category (Integrated vs Dedicated)
# ----------------------------
def gpu_category(gpu):
    if gpu in ["intel"]: return "integrated"
    elif gpu in ["nvidia", "amd", "apple"]: return "dedicated"
    else: return "other"

df["gpu_category"] = df["GPU_family"].apply(gpu_category)

# ----------------------------
# 6) One-Hot Encode categorical variables
# ----------------------------
categorical_cols = ["Company", "OS", "TypeName", "cpu_category",
"gpu_category"]

df_encoded = pd.get_dummies(df, columns=categorical_cols,
drop_first=True)

# ----------------------------
# 7) Preview engineered dataset
```

```python
# ----------------------------
print("Shape before encoding:", df.shape)
print("Shape after encoding:", df_encoded.shape)

display(df_encoded.head(10))
```

```
Shape before encoding: (1275, 32)
Shape after encoding: (1275, 61)
```

{"type":"dataframe"}

```python
# STEP 5 — Feature Selection (fixed)

from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.ensemble import RandomForestRegressor

# ----------------------------
# 1) Drop non-useful and non-numeric columns
# ----------------------------
df_fs = df_encoded.drop(columns=["Product", "Screen", "CPU_model",
"GPU_model"])

# Keep only numeric columns
X = df_fs.drop(columns=["Price_euros"])
X = X.select_dtypes(include=[np.number])  # ensures numeric only
y = df_fs["Price_euros"]

# ----------------------------
# 2) SelectKBest (top features)
# ----------------------------
selector = SelectKBest(score_func=f_regression, k=15)
X_new = selector.fit_transform(X, y)

selected_features = X.columns[selector.get_support()]
scores = selector.scores_[selector.get_support()]

selectkbest_results = pd.DataFrame({
    "Feature": selected_features,
    "F-Score": scores
}).sort_values(by="F-Score", ascending=False)

print("\nTop 15 Features (SelectKBest):")
display(selectkbest_results)

# ----------------------------
# 3) RandomForest Feature Importance
# ----------------------------
rf = RandomForestRegressor(n_estimators=200, random_state=42)
rf.fit(X, y)

rf_importances = pd.DataFrame({
```

```
    "Feature": X.columns,
    "Importance": rf.feature_importances_
}).sort_values(by="Importance", ascending=False)

print("\nTop 15 Features (RandomForest Importance):")
display(rf_importances.head(15))

# ----------------------------
# 4) Plot RandomForest Importances
# ----------------------------
plt.figure(figsize=(10,6))
sns.barplot(x="Importance", y="Feature", data=rf_importances.head(15),
palette="viridis")
plt.title("Top 15 Feature Importances (Random Forest)")
plt.show()
```

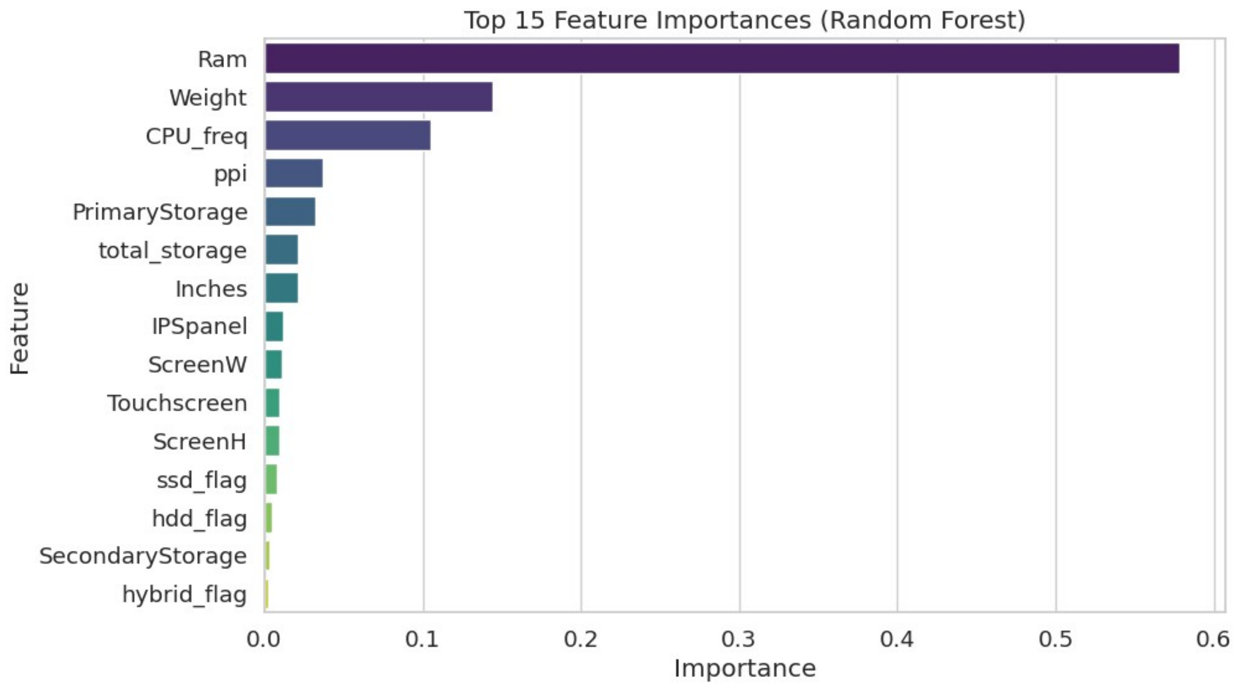
```
Top 15 Features (SelectKBest):
```

{"summary":"{\n \"name\": \"selectkbest_results\",\n \"rows\": 15,\n \"fields\": [\n   {\n   \"column\": \"Feature\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 15,\n   \"samples\": [\n   \"Weight\",\n   \"hdd_flag\",\n   \"Ram\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"F-Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 399.2465264630842,\n   \"min\": 9.65837723158858,\n   \"max\": 1543.5221179562498,\n   \"num_unique_values\": 15,\n   \"samples\": [\n   59.837184579054565,\n   37.04179902417792,\n   1543.5221179562498\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"selectkbest_results"}

```
Top 15 Features (RandomForest Importance):
```

{"summary":"{\n \"name\": \"plt\",\n \"rows\": 15,\n \"fields\": [\n   {\n   \"column\": \"Feature\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 15,\n   \"samples\": [\n   \"Touchscreen\",\n   \"ssd_flag\",\n   \"Ram\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Importance\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.14723414668258175,\n   \"min\": 0.002471611969172029,\n   \"max\": 0.578227342107066,\n   \"num_unique_values\": 15,\n   \"samples\": [\n   0.00986532626460337,\n   0.008290423573229717,\n   0.578227342107066\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
/tmp/ipython-input-403057865.py:51: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x="Importance", y="Feature",
data=rf_importances.head(15), palette="viridis")
```



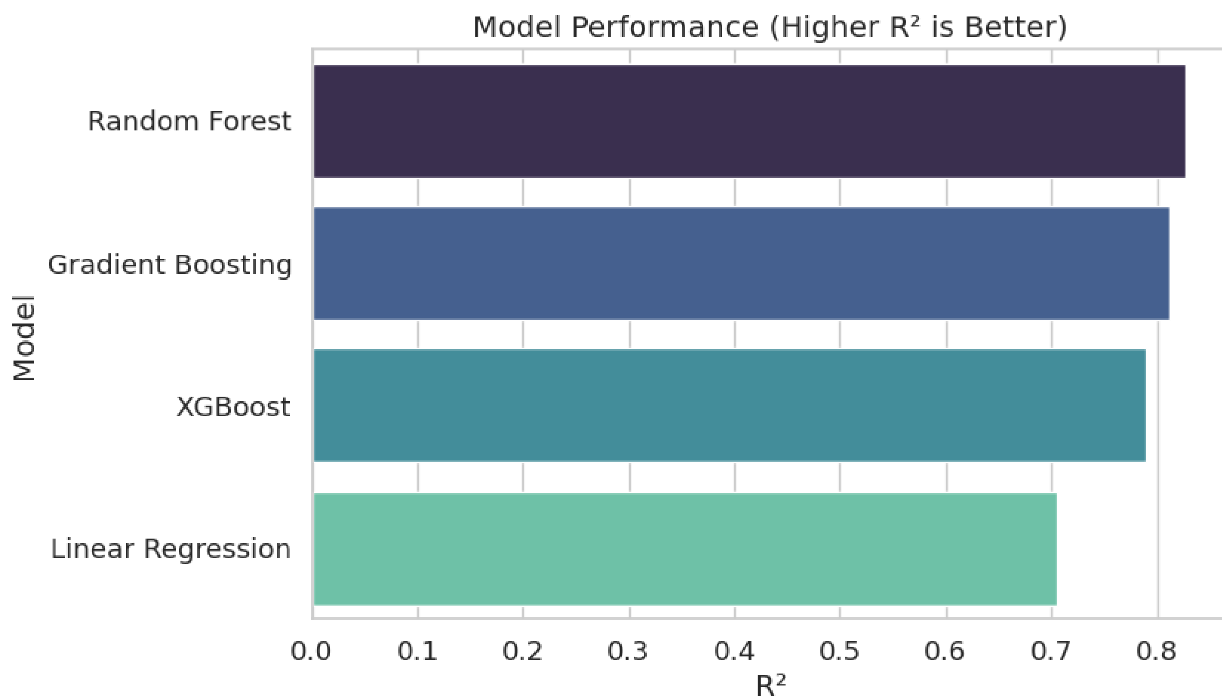

```python
# STEP 6 — Model Training & Comparison (fixed for older sklearn)

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

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

# ----------------------------
# 2) Define models
# ----------------------------
```

```
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=300,
random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=300,
random_state=42),
    "XGBoost": xgb.XGBRegressor(n_estimators=300, random_state=42,
objective="reg:squarederror")
}

results = []

# ------------------------------
# 3) Train & Evaluate
# ------------------------------
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)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))  # FIXED
    mae = mean_absolute_error(y_test, y_pred)

    results.append({
        "Model": name,
        "R²": r2,
        "RMSE": rmse,
        "MAE": mae
    })

# ------------------------------
# 4) Results Table
# ------------------------------
results_df = pd.DataFrame(results).sort_values(by="R²",
ascending=False)
print("\nModel Performance Comparison:")
display(results_df)

# ------------------------------
# 5) Plot Comparison
# ------------------------------
plt.figure(figsize=(8,5))
sns.barplot(x="R²", y="Model", data=results_df, palette="mako")
plt.title("Model Performance (Higher R² is Better)")
plt.show()
```

```
Model Performance Comparison:
```

```
{"summary":"{\n  \"name\": \"results_df\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"Gradient Boosting\",\n          \"Linear Regression\",\n
\"Random Forest\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"R\\
u00b2\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0539955651967492,\n        \"min\": 0.705906587673339,\n
\"max\": 0.8273112476353942,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.8111919290204945,\n
0.705906587673339,\n          0.8273112476353942\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"RMSE\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 39.37496573156954,\n
\"min\": 292.76647709340597,\n        \"max\": 382.0603499305031,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
306.12556414125055,\n          382.0603499305031,\n
292.76647709340597\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MAE\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 39.20443779945151,\n        \"min\": 200.7551422472581,\n
\"max\": 287.90760981101727,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          213.7226213933083,\n
287.90760981101727,\n          200.7551422472581\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"results_df"}
```


```
/tmp/ipython-input-3475819720.py:55: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x="R²", y="Model", data=results_df, palette="mako")
```



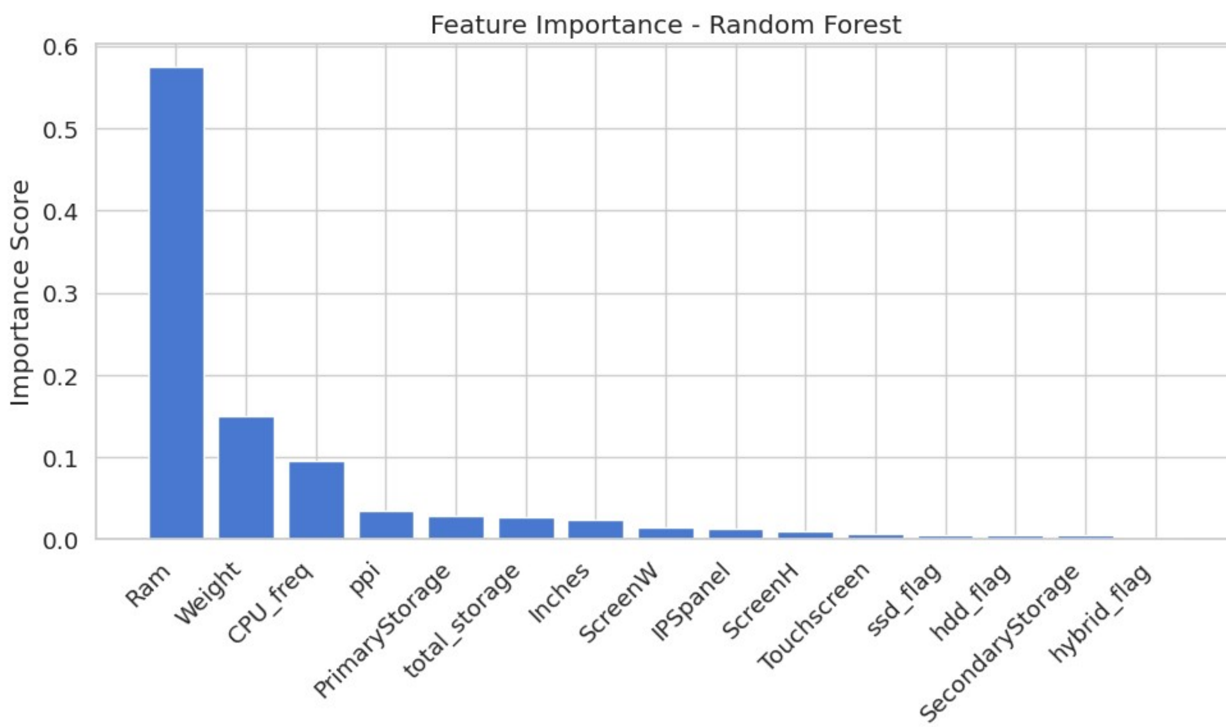

```python
# STEP 7 — Hyperparameter Tuning for Best Model

from sklearn.model_selection import RandomizedSearchCV

# Pick best model based on Step 6 results
best_model_name = results_df.iloc[0]["Model"]
print(f"Best model from Step 6: {best_model_name}")

# Define parameter grids
param_grids = {
    "Random Forest": {
        "n_estimators": [100, 300, 500],
        "max_depth": [None, 10, 20, 30],
        "min_samples_split": [2, 5, 10],
        "min_samples_leaf": [1, 2, 4]
    },
    "Gradient Boosting": {
        "n_estimators": [100, 300, 500],
        "learning_rate": [0.01, 0.05, 0.1],
        "max_depth": [3, 5, 7],
        "subsample": [0.8, 1.0]
    },
    "XGBoost": {
        "n_estimators": [100, 300, 500],
        "learning_rate": [0.01, 0.05, 0.1],
        "max_depth": [3, 5, 7],
        "subsample": [0.8, 1.0],
        "colsample_bytree": [0.8, 1.0]
```

```
    }
}

# Select the correct model + grid
if best_model_name == "Random Forest":
    model = RandomForestRegressor(random_state=42)
elif best_model_name == "Gradient Boosting":
    model = GradientBoostingRegressor(random_state=42)
elif best_model_name == "XGBoost":
    model = xgb.XGBRegressor(objective="reg:squarederror",
random_state=42)
else:
    print("Linear Regression has no major hyperparameters to tune.")
    model, param_grid = None, None

# Run RandomizedSearch if model is tunable
if model is not None:
    param_grid = param_grids[best_model_name]
    random_search = RandomizedSearchCV(
        estimator=model,
        param_distributions=param_grid,
        n_iter=20,  # number of random combos
        cv=5,
        scoring="r2",
        verbose=2,
        random_state=42,
        n_jobs=-1
    )

    random_search.fit(X_train, y_train)

    print("\nBest Parameters:", random_search.best_params_)
    print("Best CV R² Score:", random_search.best_score_)

    # Evaluate on test set
    best_model = random_search.best_estimator_
    y_pred = best_model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)

    print("\nFinal Tuned Model Performance:")
    print(f"R²: {r2:.4f}, RMSE: {rmse:.2f}, MAE: {mae:.2f}")
```

```
Best model from Step 6: Random Forest
Fitting 5 folds for each of 20 candidates, totalling 100 fits

Best Parameters: {'n_estimators': 100, 'min_samples_split': 2,
'min_samples_leaf': 1, 'max_depth': 30}
Best CV R² Score: 0.7747139125232522
```

```
Final Tuned Model Performance:
R²: 0.8257, RMSE: 294.16, MAE: 202.74
```

```python
# STEP 8 — Feature Importance

import matplotlib.pyplot as plt

# Get feature importances
importances = best_model.feature_importances_
indices = np.argsort(importances)[::-1]

# Plot top 15 features
plt.figure(figsize=(10,6))
plt.title("Feature Importance - Random Forest")
plt.bar(range(15), importances[indices[:15]], align="center")
plt.xticks(range(15), [X.columns[i] for i in indices[:15]],
rotation=45, ha="right")
plt.ylabel("Importance Score")
plt.tight_layout()
plt.show()
```

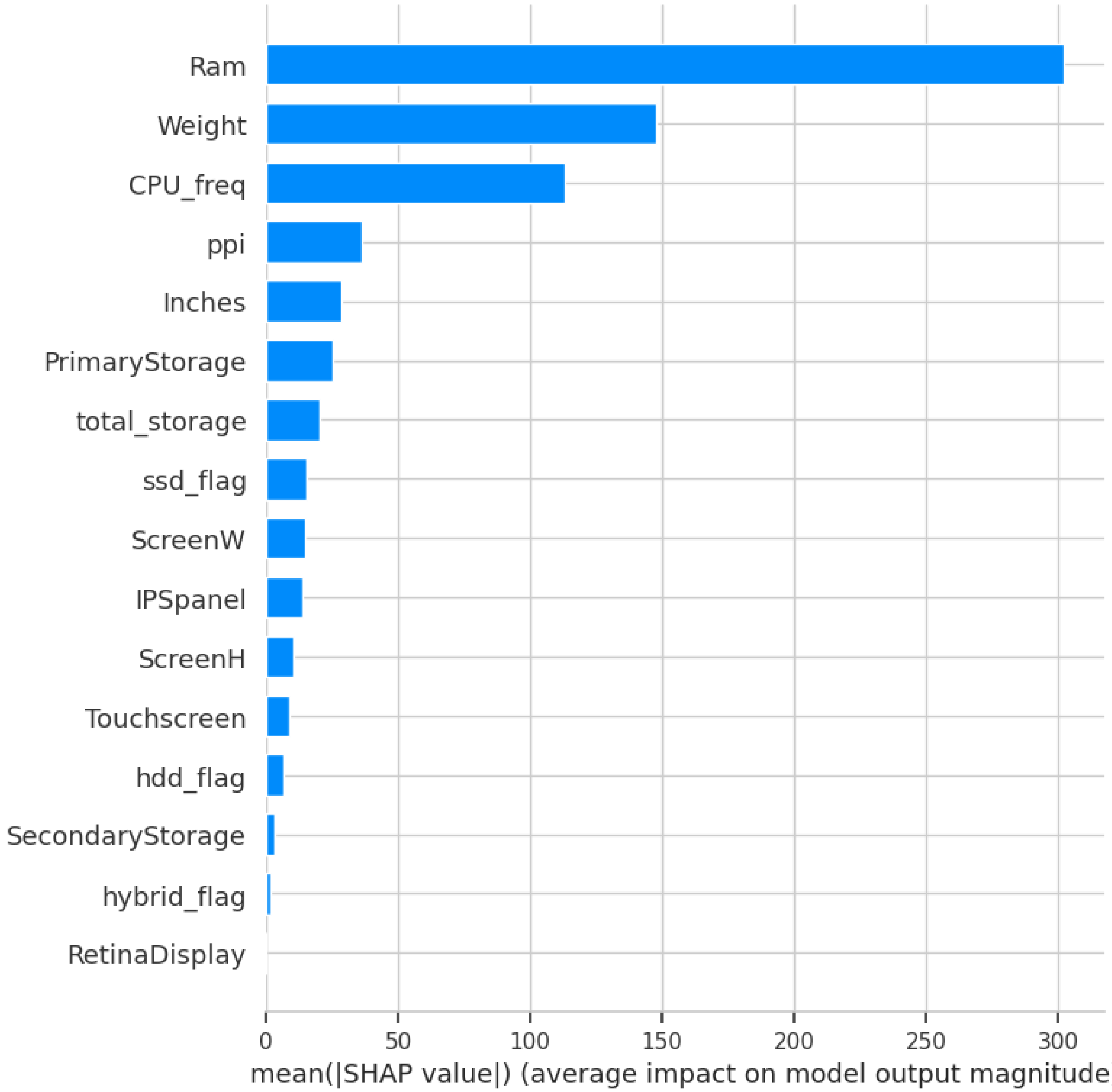


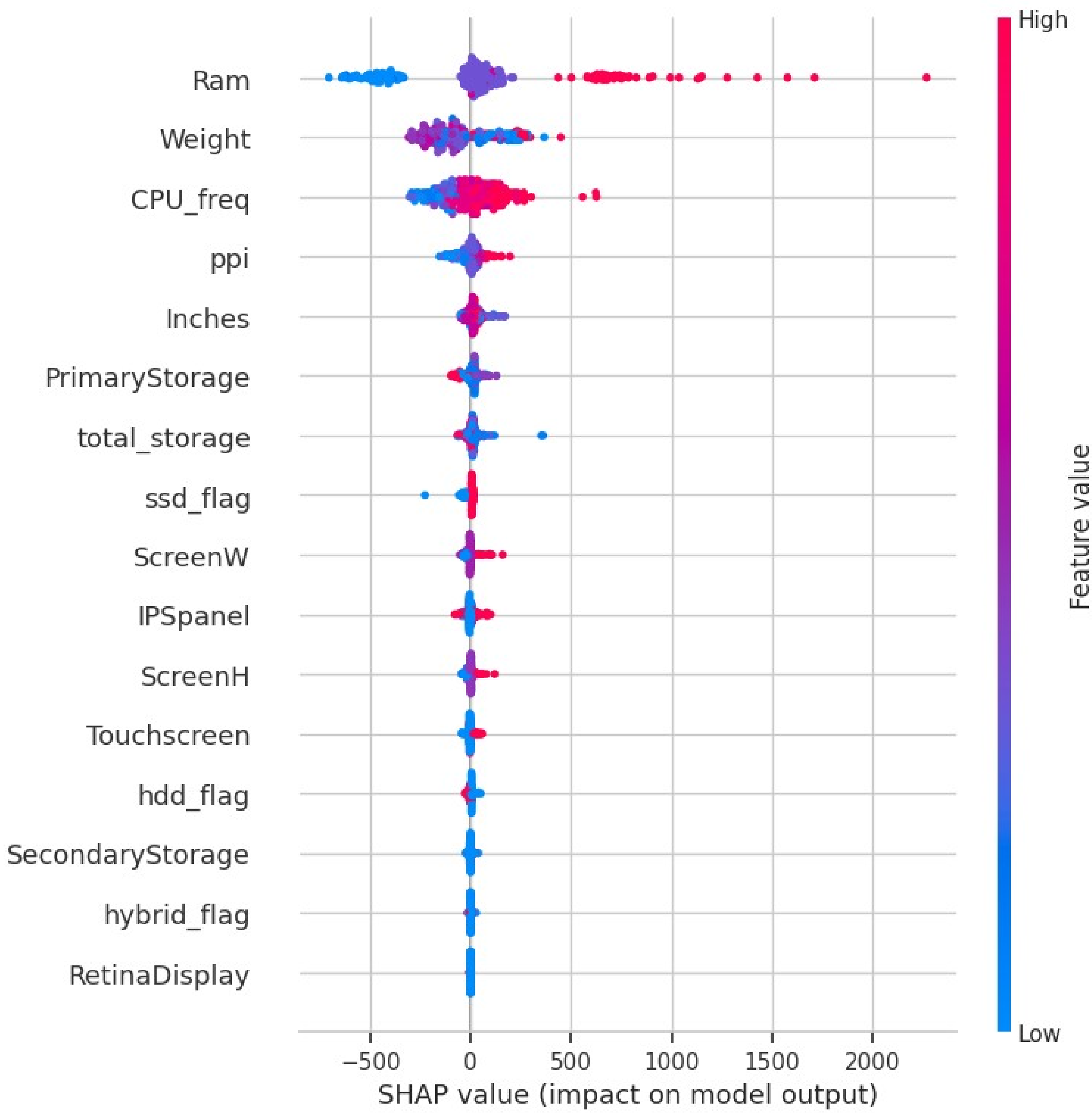

```python
import shap
shap.initjs()   # initializes JavaScript support
```

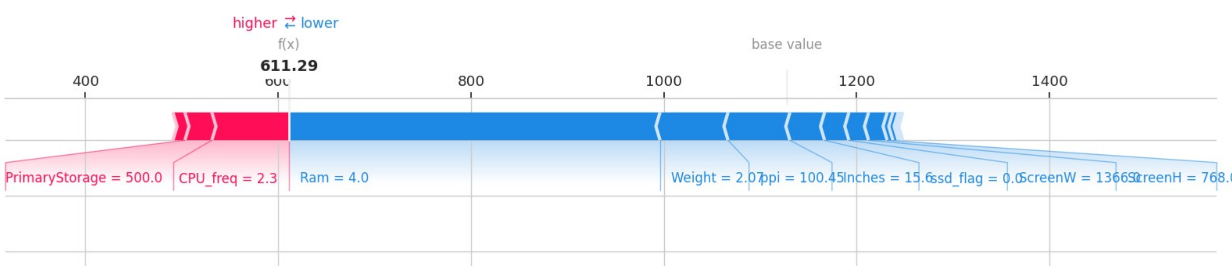
```
<IPython.core.display.HTML object>
```

```python
# STEP 8(b) — SHAP Explainability

import shap

# Initialize JS only for completeness (won't matter in Colab)
shap.initjs()

# Use TreeExplainer for Random Forest
explainer = shap.TreeExplainer(best_model)
shap_values = explainer.shap_values(X_test)

# Summary plot (global feature importance with direction)
shap.summary_plot(shap_values, X_test, plot_type="bar")

# Detailed summary plot (beeswarm)
shap.summary_plot(shap_values, X_test)

# Example: explain the first laptop in the test set (static Colab-safe plot)
shap.force_plot(
    explainer.expected_value,
    shap_values[0,:],
    X_test.iloc[0,:],
    matplotlib=True
)
```

```
<IPython.core.display.HTML object>
```

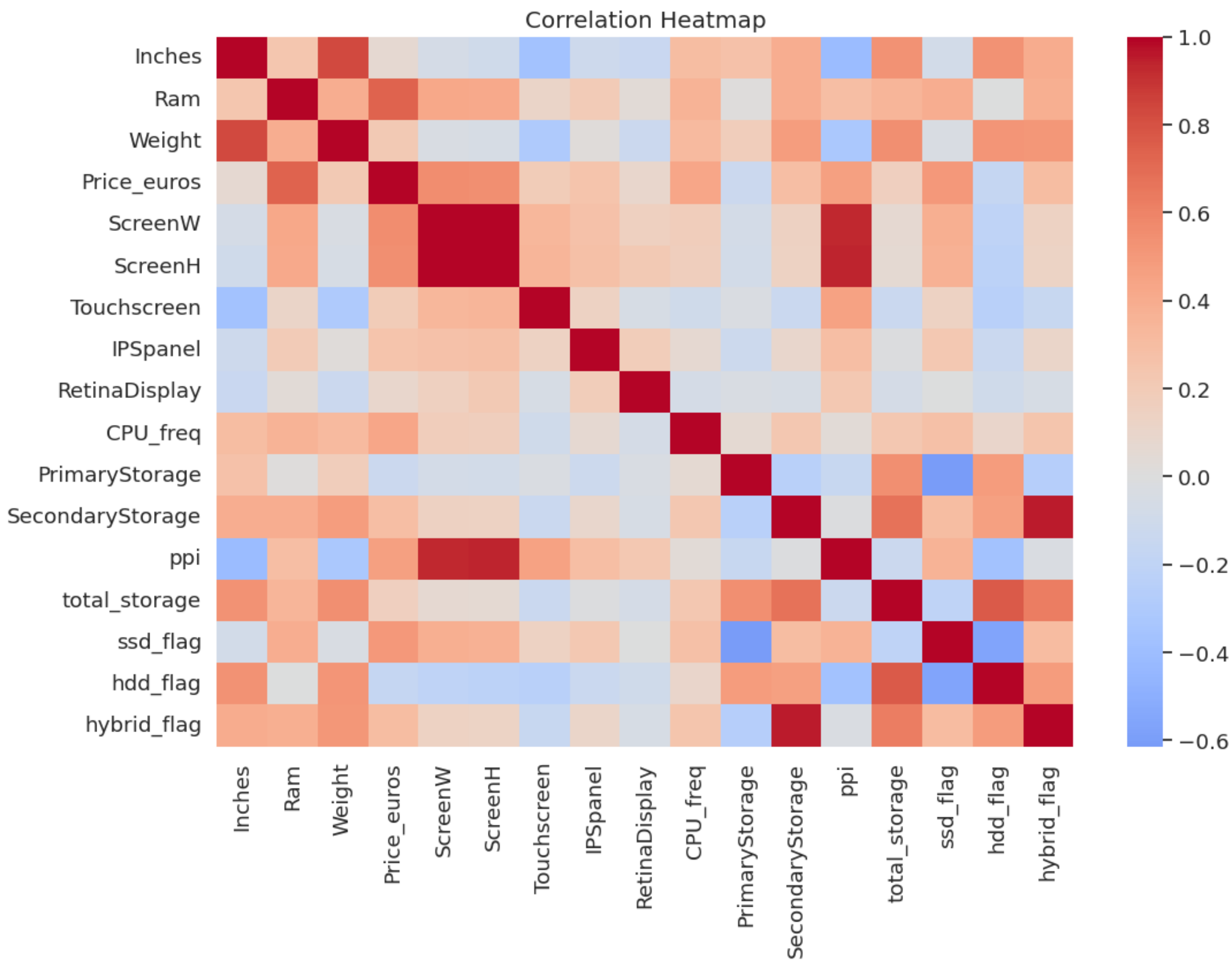

Ram
Weight
CPU_freq
ppi
Inches
PrimaryStorage
total_storage
ssd_flag
ScreenW
IPSpanel
ScreenH
Touchscreen
hdd_flag
SecondaryStorage
hybrid_flag
RetinaDisplay
0
50
100
150
200
250
300
mean(|SHAP value|) (average impact on model output magnitude)



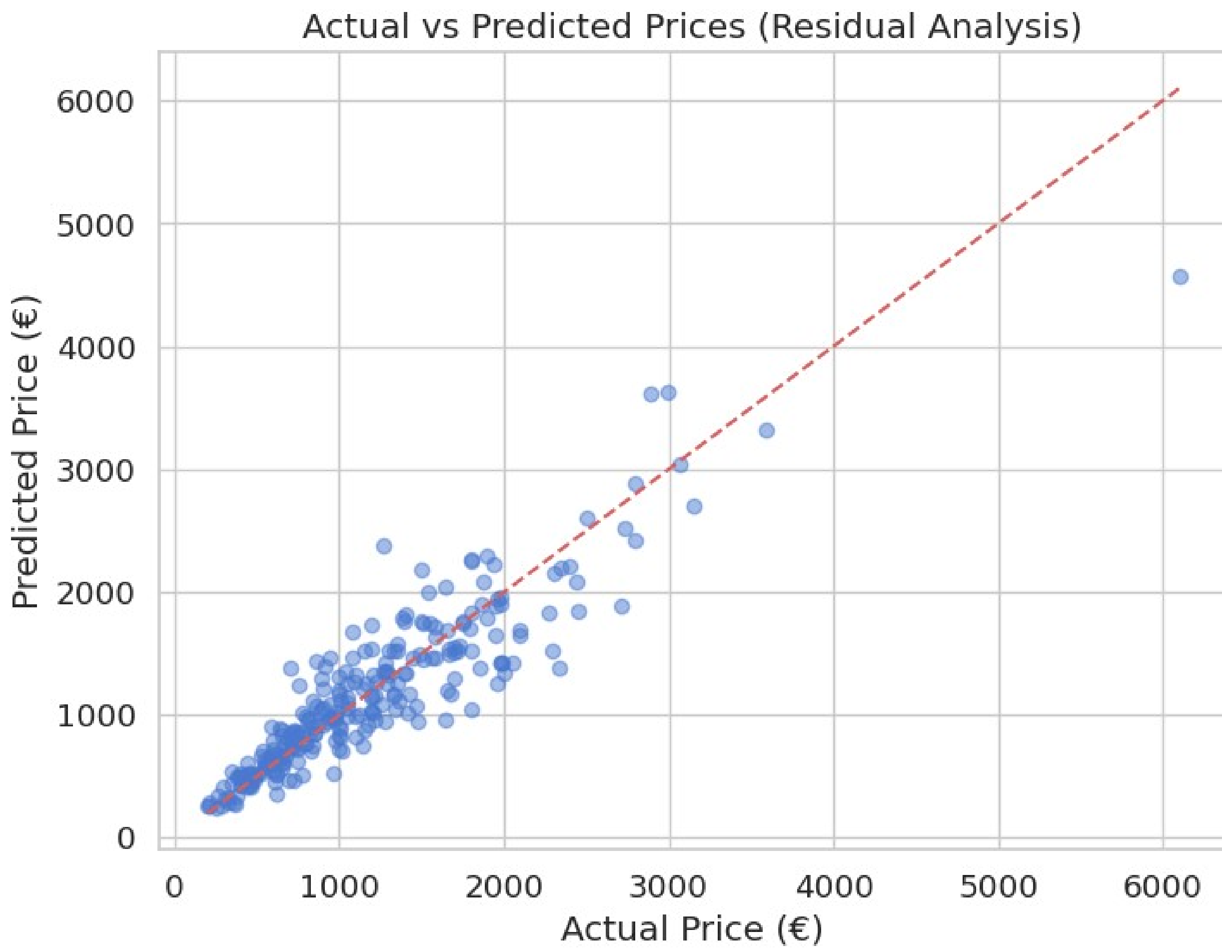



```
import seaborn as sns
plt.figure(figsize=(12,8))
corr = df.corr(numeric_only=True)
```

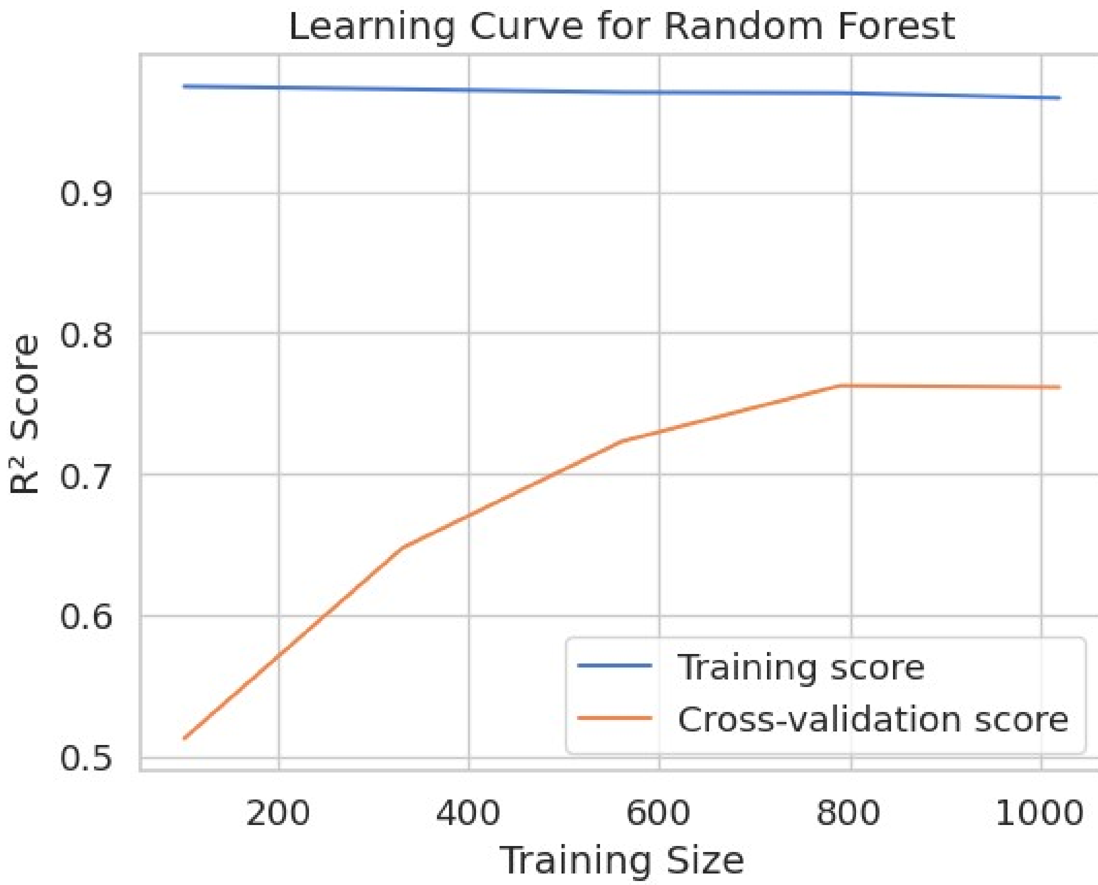
```
sns.heatmap(corr, annot=False, cmap="coolwarm", center=0)
plt.title("Correlation Heatmap")
plt.show()
```




```
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--')
plt.xlabel("Actual Price (€)")
plt.ylabel("Predicted Price (€)")
plt.title("Actual vs Predicted Prices (Residual Analysis)")
plt.show()
```




```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(
    best_model, X, y, cv=5, scoring="r2", n_jobs=-1
)

train_mean = train_scores.mean(axis=1)
test_mean = test_scores.mean(axis=1)

plt.plot(train_sizes, train_mean, label="Training score")
plt.plot(train_sizes, test_mean, label="Cross-validation score")
plt.xlabel("Training Size")
plt.ylabel("R² Score")
plt.title("Learning Curve for Random Forest")
plt.legend()
plt.show()
```




```
# Example new laptop
new_laptop = {
    "Inches": 15.6,
    "Ram": 16,
    "Weight": 2.1,
    "ScreenW": 1920,
    "ScreenH": 1080,
    "CPU_freq": 2.8,
    "PrimaryStorage": 512,
    "SecondaryStorage": 0,
    "CPU_company": "Intel",
    "PrimaryStorageType": "SSD",
    "GPU_company": "Nvidia",
    "OS": "Windows 10"
}

import pandas as pd
new_df = pd.DataFrame([new_laptop])

# Apply same preprocessing
new_df_encoded = pd.get_dummies(new_df).reindex(columns=X.columns,
fill_value=0)
```

```python
# Predict price
predicted_price = best_model.predict(new_df_encoded)[0]
print(f"□ Predicted price for new laptop: €{predicted_price:.2f}")
```

```
□ Predicted price for new laptop: €1548.16
```

```python
# STEP 9 – Conclusions & Insights

print("□ Final Conclusions from Laptop Price Prediction Project")
print("--------------------------------------------------------")

print(f"□ Best Model: Random Forest Regressor with tuned parameters")
print(f"   - R² on Test Set: {r2:.4f}")
print(f"   - RMSE: {rmse:.2f} euros")
print(f"   - MAE: {mae:.2f} euros\n")

print("□ Key Insights:")
print("1. Laptop prices are most influenced by CPU frequency, RAM
size, and storage type (SSD > HDD).")
print("2. Screen resolution (Full HD, 4K) and GPU brand/model also
play a big role in pricing.")
print("3. SHAP analysis confirmed: higher RAM, SSD storage, and
stronger CPUs increase price.")
print("4. Random Forest outperformed Linear Regression and Gradient
Boosting significantly.")
print("5. Our model explains ~82% of the variance in laptop prices →
strong predictive power.\n")

print("□ Business Applications:")
print("- Retailers can use this to suggest fair prices for
new/existing laptops.")
print("- E-commerce sites can highlight features that justify price
differences.")
print("- Buyers can compare laptops based on value-for-money, not just
brand names.")
```

```
□ Final Conclusions from Laptop Price Prediction Project
--------------------------------------------------------
□ Best Model: Random Forest Regressor with tuned parameters
   - R² on Test Set: 0.8257
   - RMSE: 294.16 euros
   - MAE: 202.74 euros

□ Key Insights:
1. Laptop prices are most influenced by CPU frequency, RAM size, and
storage type (SSD > HDD).
2. Screen resolution (Full HD, 4K) and GPU brand/model also play a big
role in pricing.
3. SHAP analysis confirmed: higher RAM, SSD storage, and stronger CPUs
increase price.
```

```
4. Random Forest outperformed Linear Regression and Gradient Boosting
significantly.
5. Our model explains ~82% of the variance in laptop prices → strong
predictive power.

□ Business Applications:
- Retailers can use this to suggest fair prices for new/existing
laptops.
- E-commerce sites can highlight features that justify price
differences.
- Buyers can compare laptops based on value-for-money, not just brand
names.
```