project5-1

June 27, 2025

Data Loading:

[ ]:
```
import pandas as pd
df = pd.read_csv('laptopPrice.csv')
display(df.head())
```

```
    brand processor_brand processor_name processor_gnrtn ram_gb ram_type  \
0    ASUS           Intel        Core i3            10th   4 GB     DDR4
1  Lenovo           Intel        Core i3            10th   4 GB     DDR4
2  Lenovo           Intel        Core i3            10th   4 GB     DDR4
3    ASUS           Intel        Core i5            10th   8 GB     DDR4
4    ASUS           Intel   Celeron Dual   Not Available   4 GB     DDR4

      ssd      hdd       os  os_bit graphic_card_gb  weight     warranty  \
0    0 GB  1024 GB  Windows  64-bit            0 GB  Casual  No warranty
1    0 GB  1024 GB  Windows  64-bit            0 GB  Casual  No warranty
2    0 GB  1024 GB  Windows  64-bit            0 GB  Casual  No warranty
3  512 GB     0 GB  Windows  32-bit            2 GB  Casual  No warranty
4    0 GB   512 GB  Windows  64-bit            0 GB  Casual  No warranty

  Touchscreen msoffice  Price   rating  Number of Ratings  Number of Reviews
0          No       No  34649  2 stars                  3                  0
1          No       No  38999  3 stars                 65                  5
2          No       No  39999  3 stars                  8                  1
3          No       No  69990  3 stars                  0                  0
4          No       No  26990  3 stars                  0                  0
```

Data Exploration:

[ ]:
```
display(df.describe())
display(df.info())

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.histplot(df['Price'], kde=True)
plt.title('Distribution of Price')
plt.xlabel('Price')
```

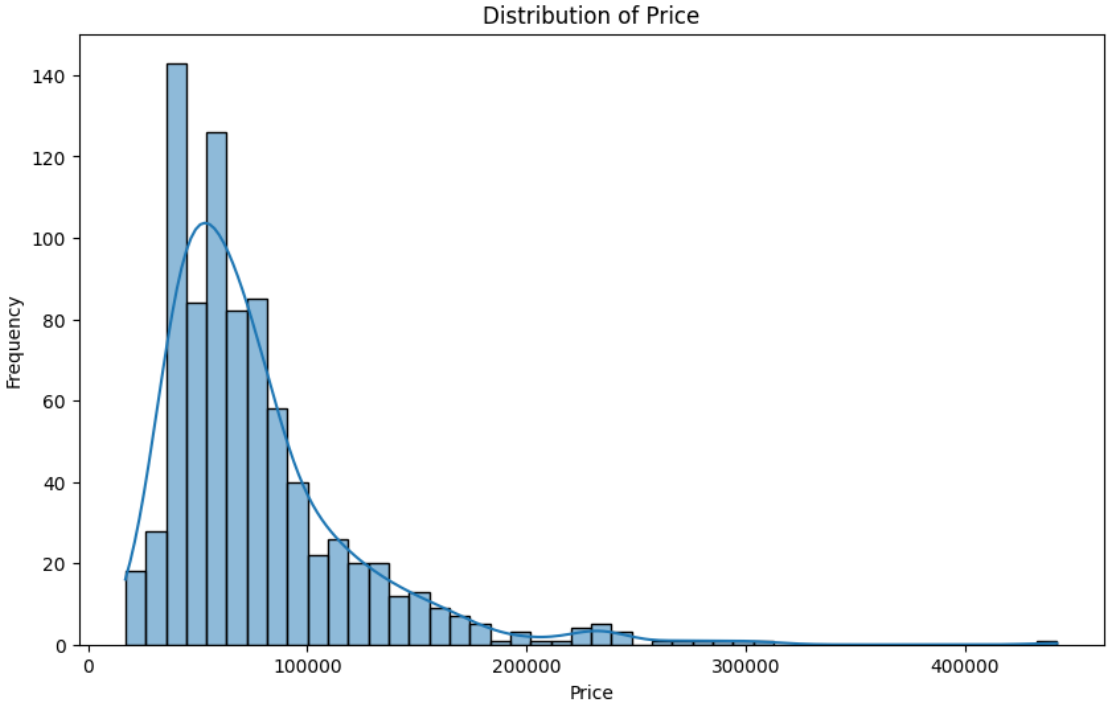
```
plt.ylabel('Frequency')
plt.show()

numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
numerical_cols.remove('Price')

for col in numerical_cols:
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=col, y='Price', data=df)
    plt.title(f'Price vs {col}')
    plt.xlabel(col)
    plt.ylabel('Price')
    plt.show()

categorical_cols = df.select_dtypes(include=['object']).columns.tolist()

for col in categorical_cols:
    plt.figure(figsize=(12, 6))
    sns.countplot(y=col, data=df, order = df[col].value_counts().index)
    plt.title(f'Distribution of {col}')
    plt.xlabel('Count')
    plt.ylabel(col)
    plt.show()

    plt.figure(figsize=(12, 6))
    sns.boxplot(x='Price', y=col, data=df)
    plt.title(f'Price by {col}')
    plt.xlabel('Price')
    plt.ylabel(col)
    plt.show()

display(df.isnull().sum())

for col in numerical_cols + ['Price']:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[col])
    plt.title(f'Box plot of {col}')
    plt.xlabel(col)
    plt.show()
```

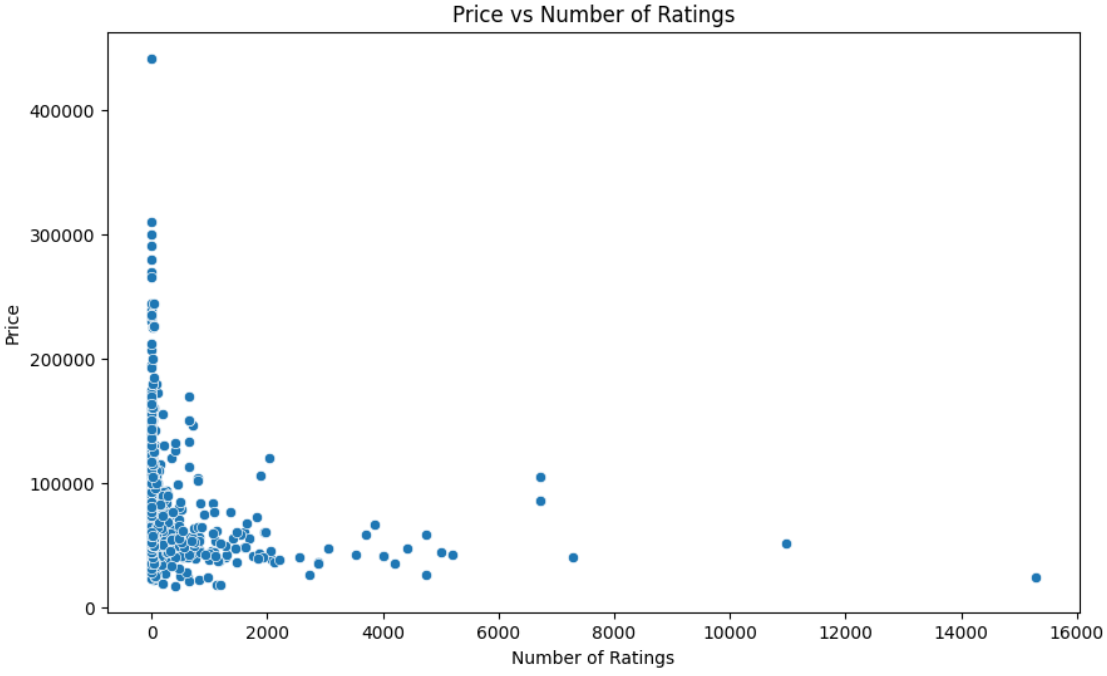
	Price	Number of Ratings	Number of Reviews
count	823.000000	823.000000	823.000000
mean	76745.177400	315.301337	37.609964
std	45101.790525	1047.382654	121.728017
min	16990.000000	0.000000	0.000000
25%	46095.000000	0.000000	0.000000
50%	64990.000000	17.000000	2.000000

```
75%     89636.000000         139.500000          18.000000
max    441990.000000       15279.000000        1947.000000
```

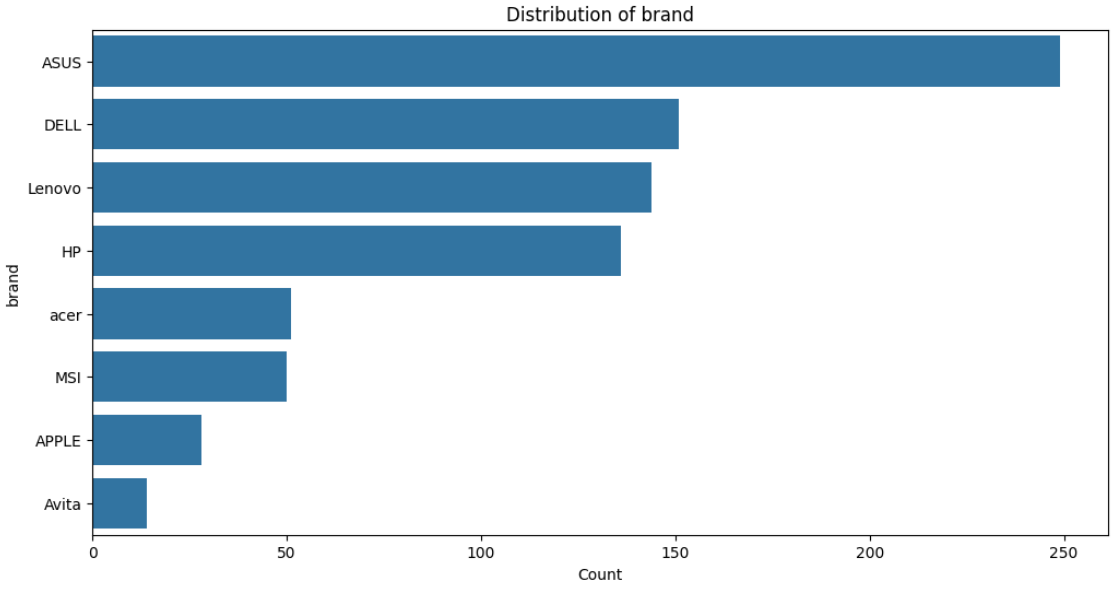
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 823 entries, 0 to 822
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   brand              823 non-null    object
 1   processor_brand    823 non-null    object
 2   processor_name     823 non-null    object
 3   processor_gnrtn    823 non-null    object
 4   ram_gb             823 non-null    object
 5   ram_type           823 non-null    object
 6   ssd                823 non-null    object
 7   hdd                823 non-null    object
 8   os                 823 non-null    object
 9   os_bit             823 non-null    object
 10  graphic_card_gb    823 non-null    object
 11  weight             823 non-null    object
 12  warranty           823 non-null    object
 13  Touchscreen        823 non-null    object
 14  msoffice           823 non-null    object
 15  Price              823 non-null    int64
 16  rating             823 non-null    object
 17  Number of Ratings  823 non-null    int64
 18  Number of Reviews  823 non-null    int64
dtypes: int64(3), object(16)
memory usage: 122.3+ KB
```

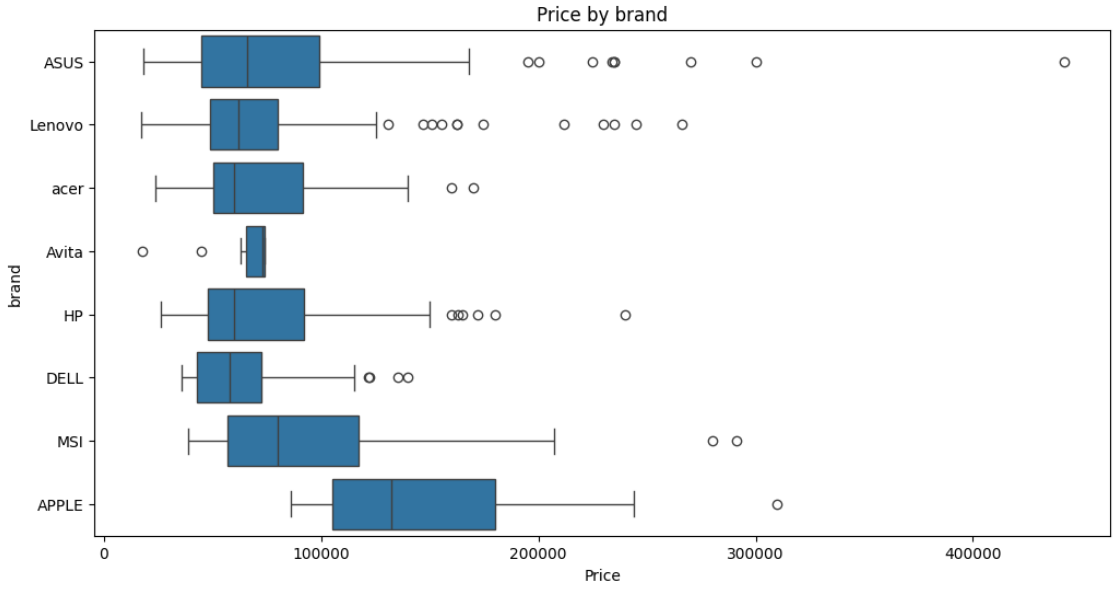
```
None
```

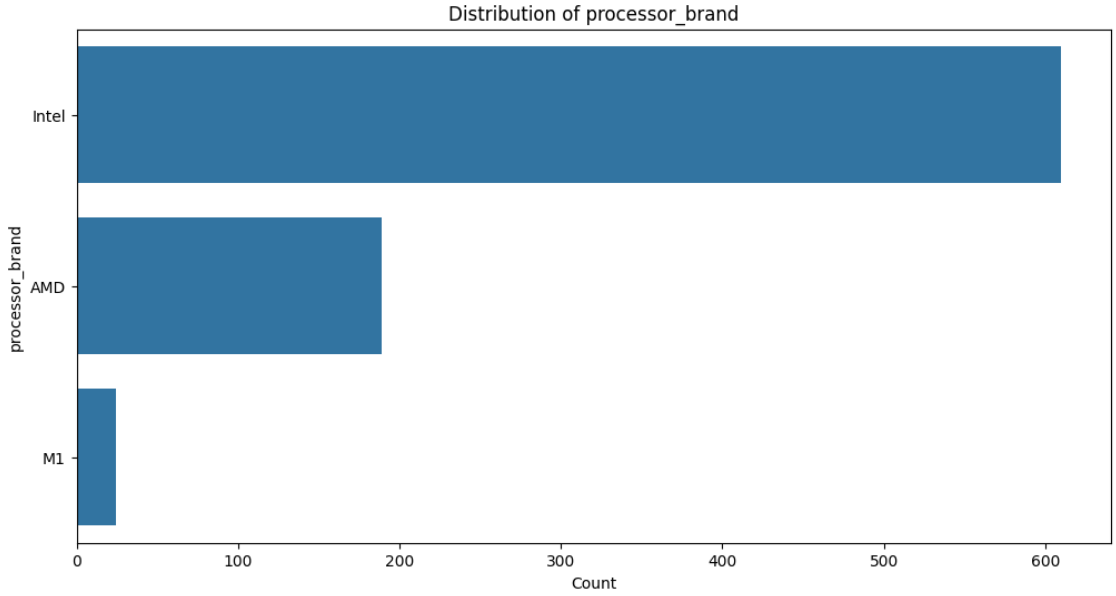

Distribution of Price
Frequency
Price
0
20
40
60
80
100
120
140
0
100000
200000
300000
400000

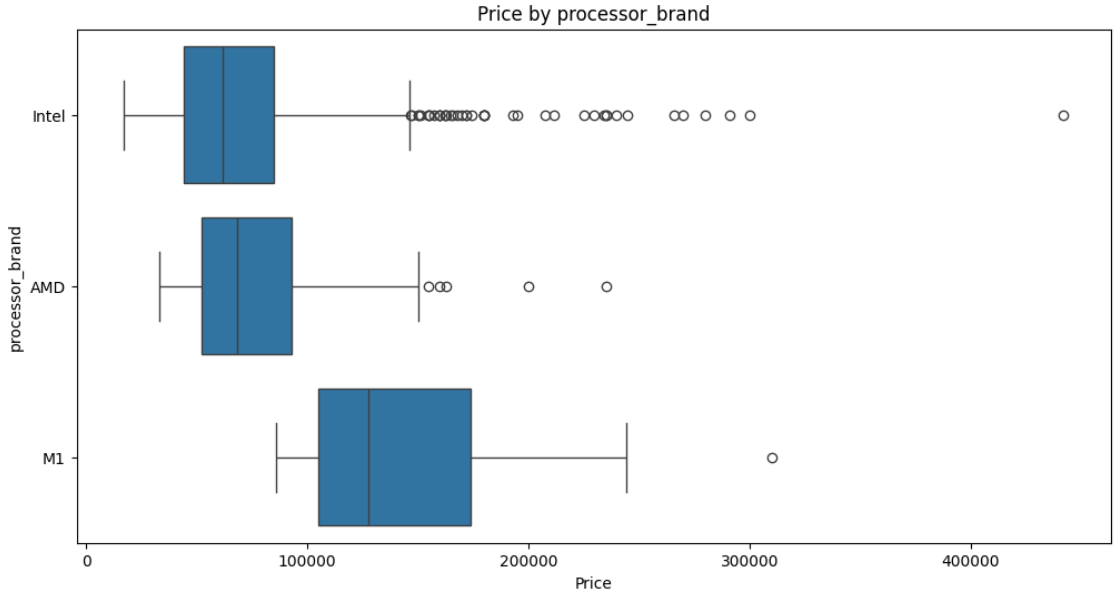

Price vs Number of Ratings
Price
Number of Ratings
0
100000
200000
300000
400000
0
2000
4000
6000
8000
10000
12000
14000
16000

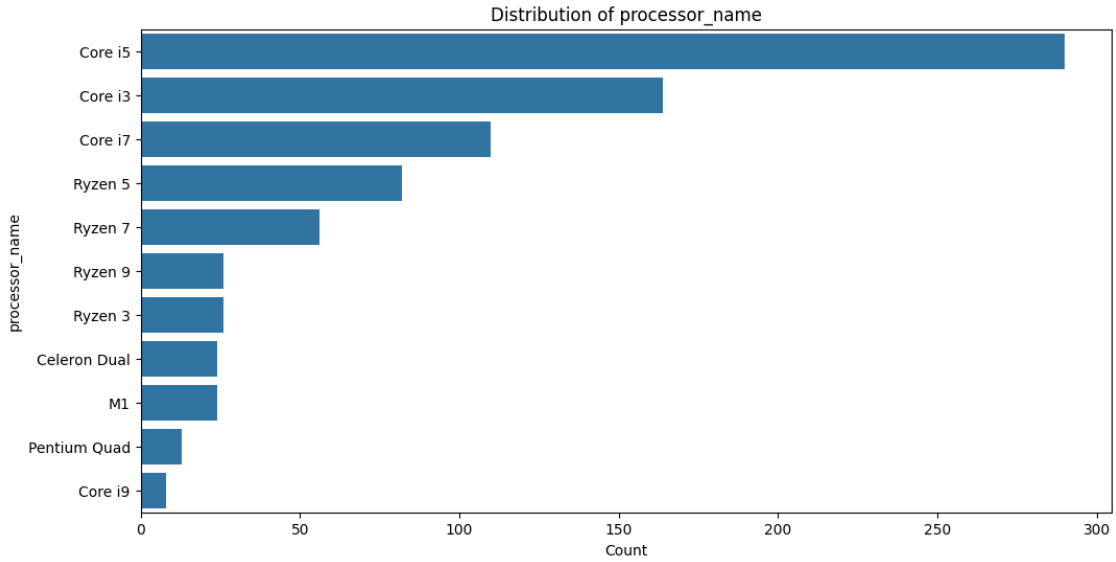

Price vs Number of Reviews
Price
Number of Reviews
0
100000
200000
300000
400000
0
250
500
750
1000
1250
1500
1750
2000

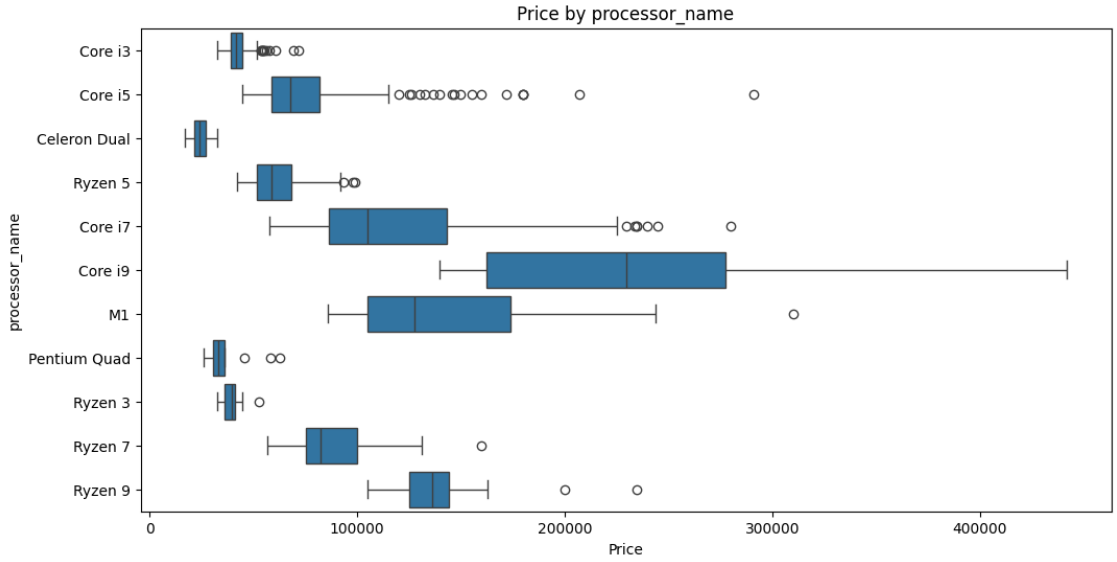

Distribution of brand
brand
Count
ASUS
DELL
Lenovo
HP
acer
MSI
APPLE
Avita
0
50
100
150
200
250

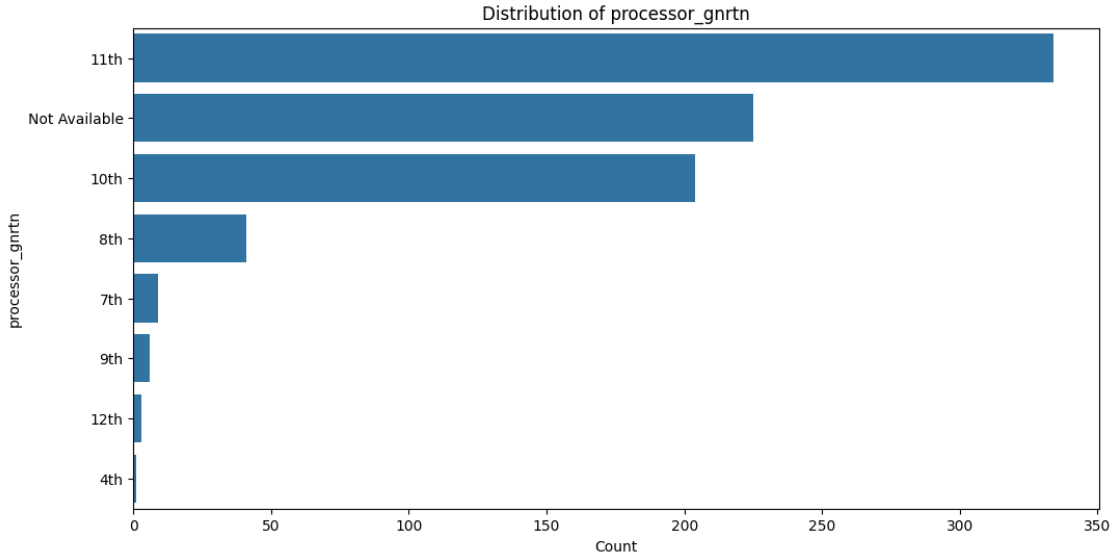

Price by brand
ASUS
Lenovo
acer
Avita
HP
DELL
MSI
APPLE
brand
0
100000
200000
300000
400000
Price

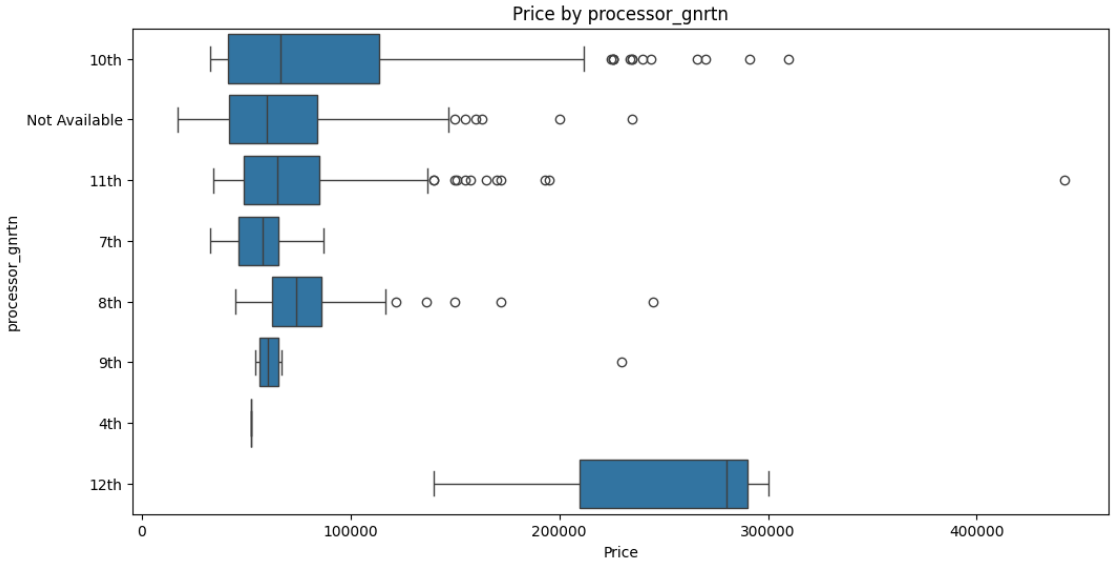

Distribution of processor_brand
Intel
AMD
M1
processor_brand
0
100
200
300
400
500
600
Count

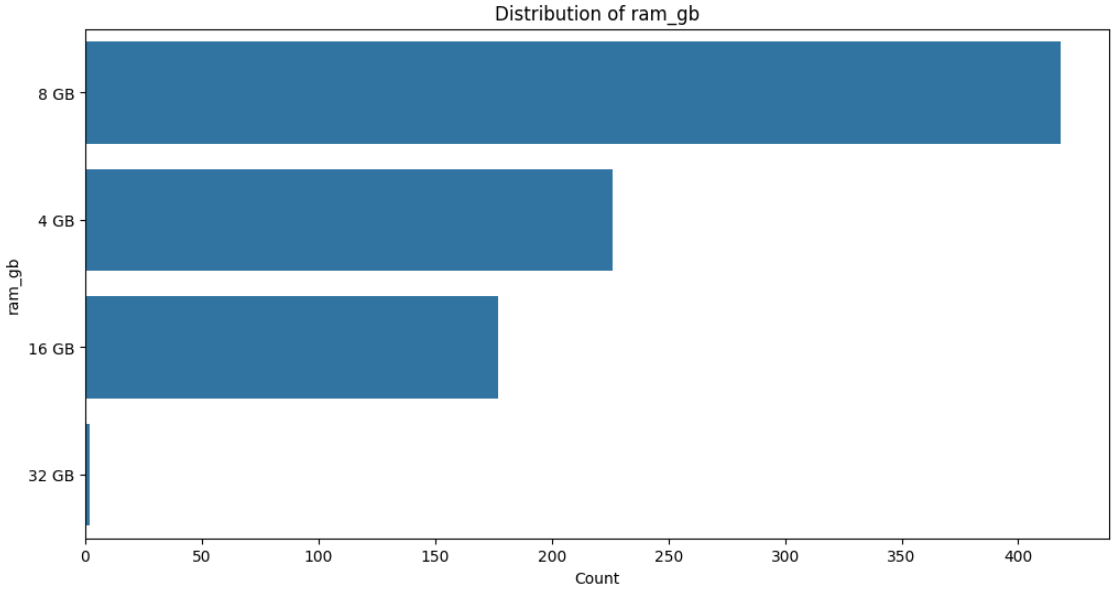

Price by processor_brand
Intel
AMD
M1
processor_brand
0
100000
200000
300000
400000
Price

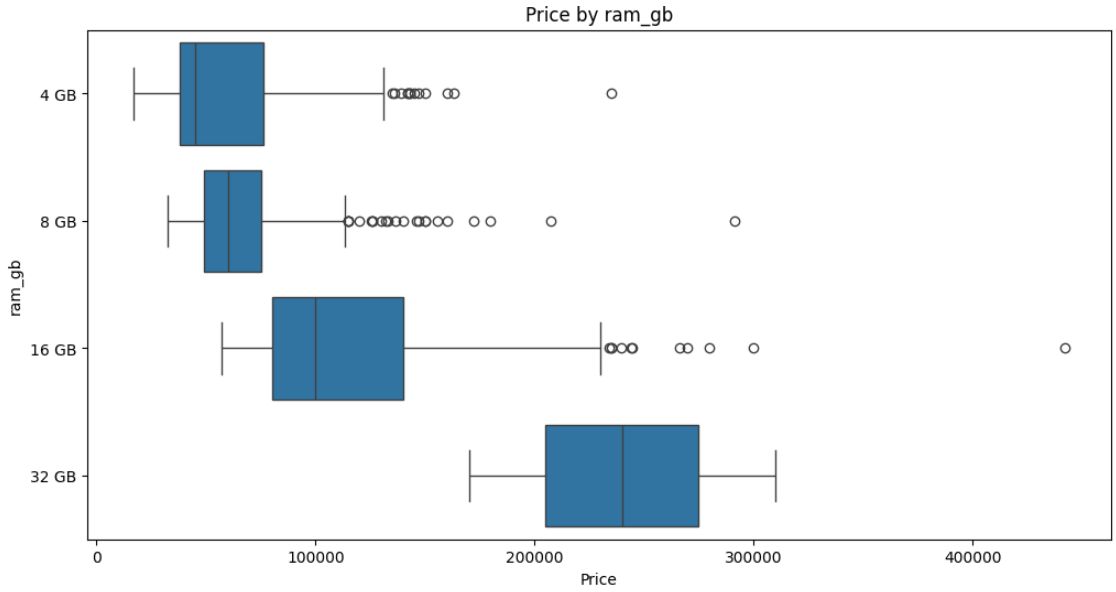

Distribution of processor_name
Core i5
Core i3
Core i7
Ryzen 5
Ryzen 7
Ryzen 9
Ryzen 3
Celeron Dual
M1
Pentium Quad
Core i9
processor_name
0
50
100
150
200
250
300
Count

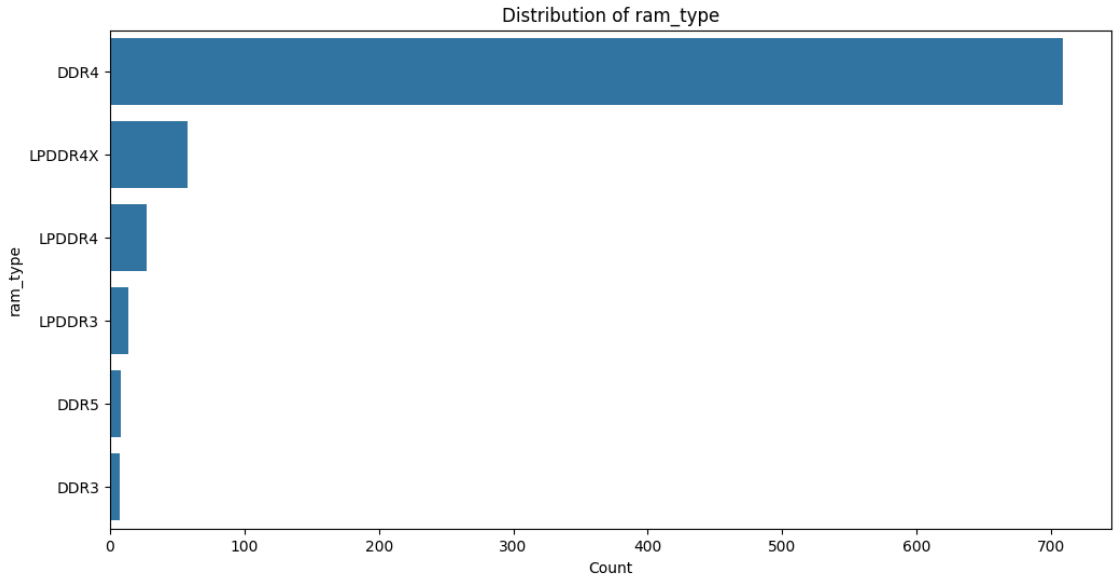

Price by processor_name
Core i3
Core i5
Celeron Dual
Ryzen 5
Core i7
Core i9
M1
Pentium Quad
Ryzen 3
Ryzen 7
Ryzen 9
processor_name
0
100000
200000
300000
400000
Price

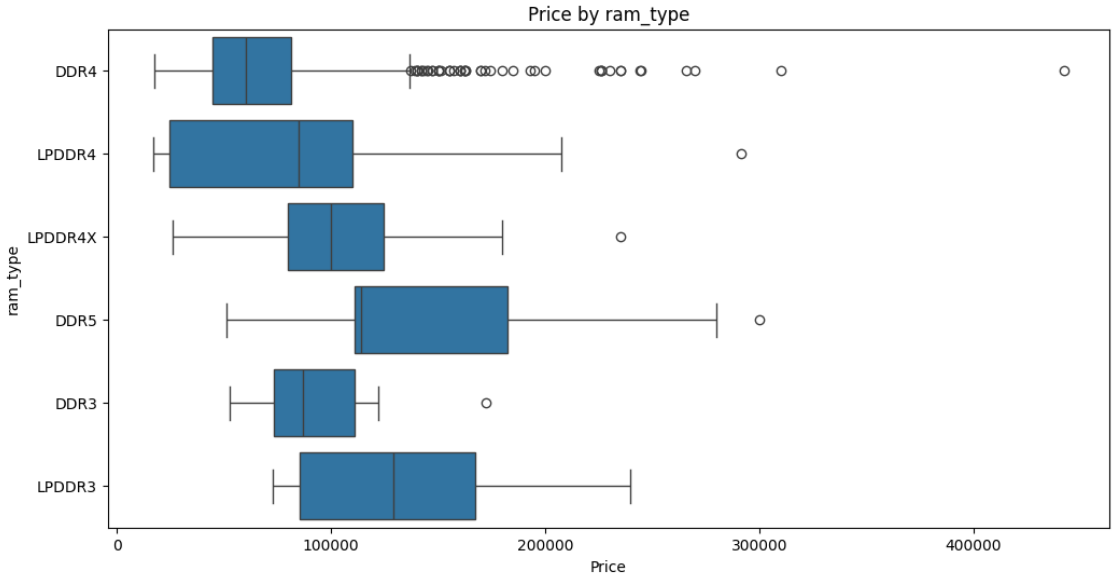

Distribution of processor_gnrtn
11th
Not Available
10th
8th
7th
9th
12th
4th
processor_gnrtn
0
50
100
150
200
250
300
350
Count

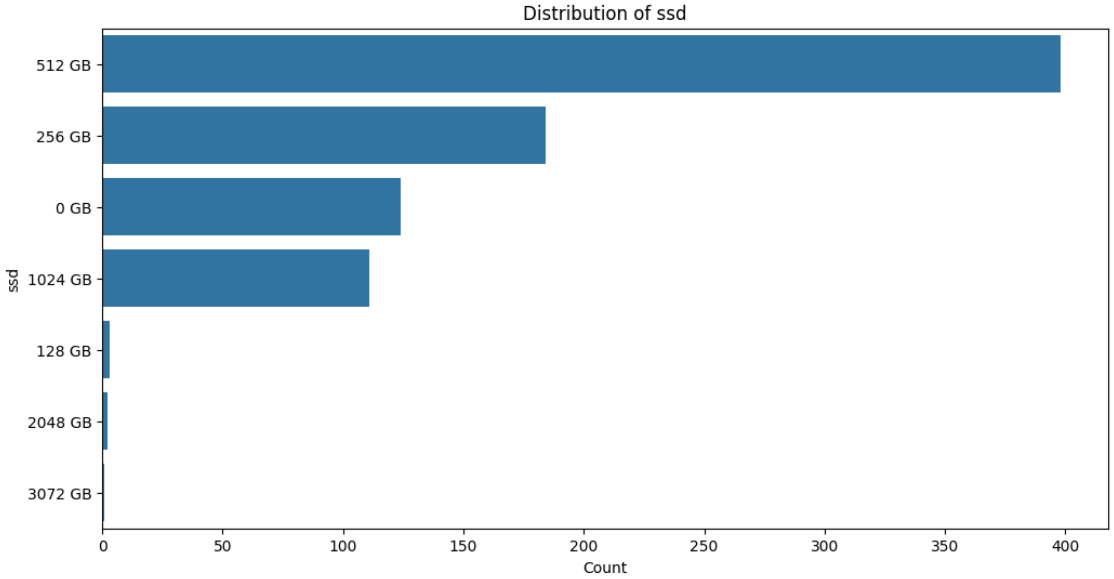

Price by processor_gnrtn
10th
Not Available
11th
7th
8th
9th
4th
12th
processor_gnrtn
0
100000
200000
300000
400000
Price

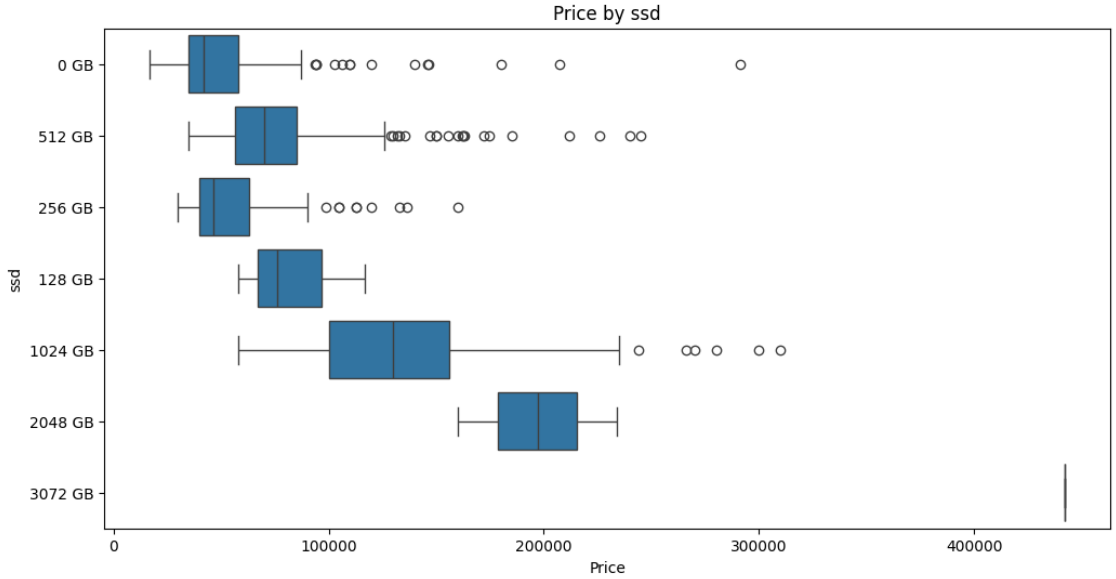

Distribution of ram_gb
8 GB
4 GB
16 GB
32 GB
ram_gb
0
50
100
150
200
250
300
350
400
Count

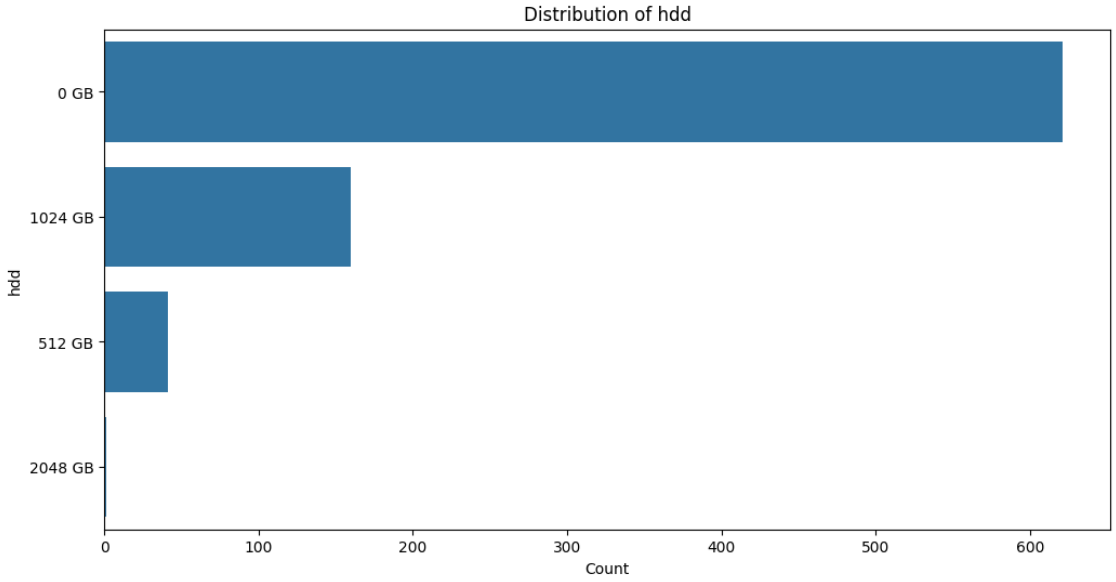

Price by ram_gb
4 GB
8 GB
16 GB
32 GB
ram_gb
0
100000
200000
300000
400000
Price

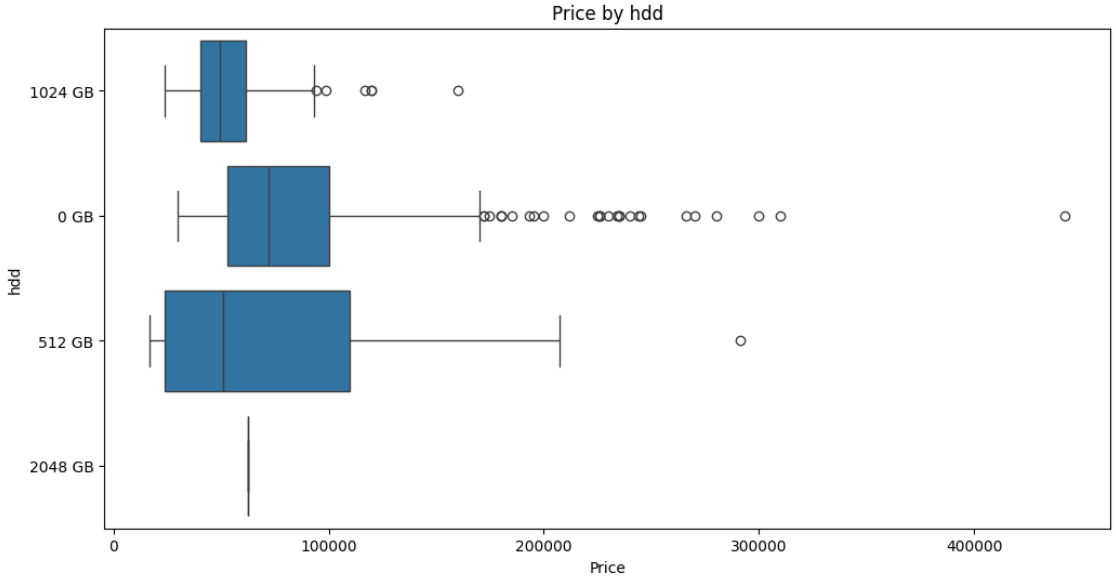

Distribution of ram_type
DDR4
LPDDR4X
LPDDR4
LPDDR3
DDR5
DDR3
ram_type
0
100
200
300
400
500
600
700
Count

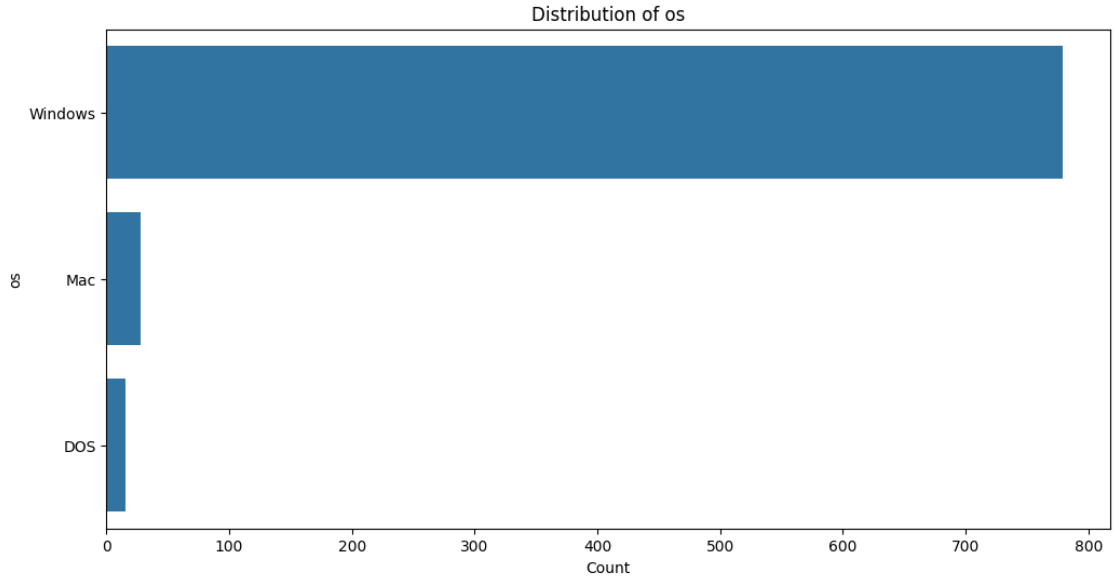

Price by ram_type
DDR4
LPDDR4
LPDDR4X
DDR5
DDR3
LPDDR3
ram_type
0
100000
200000
300000
400000
Price

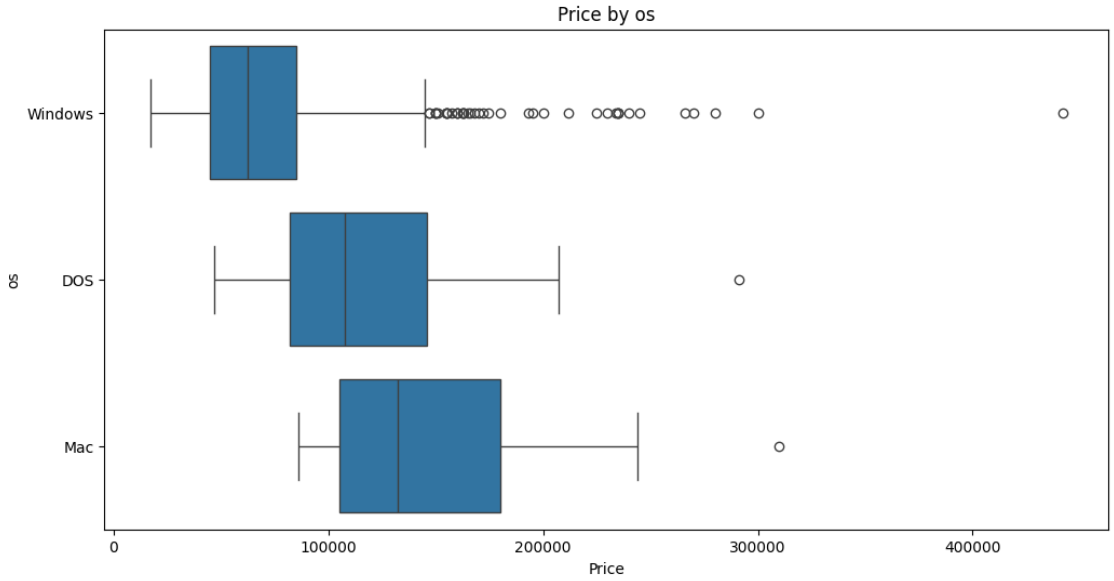

Distribution of ssd
512 GB
256 GB
0 GB
1024 GB
128 GB
2048 GB
3072 GB
ssd
0
50
100
150
200
250
300
350
400
Count

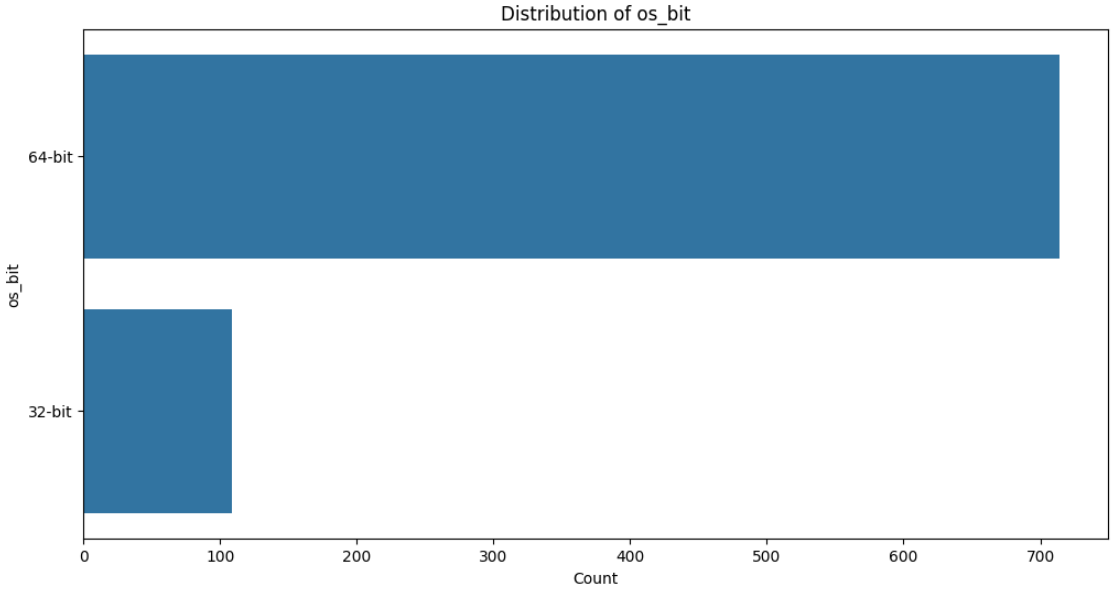

Price by ssd
0 GB
512 GB
256 GB
128 GB
1024 GB
2048 GB
3072 GB
ssd
0
100000
200000
300000
400000
Price

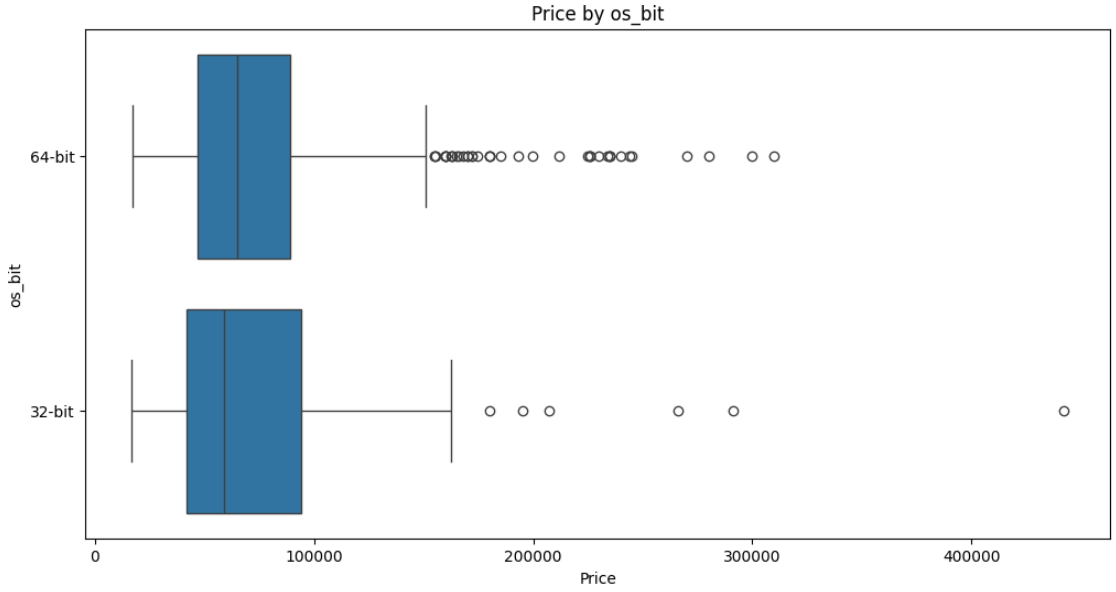

Distribution of hdd
0 GB
1024 GB
512 GB
2048 GB
hdd
0
100
200
300
400
500
600
Count

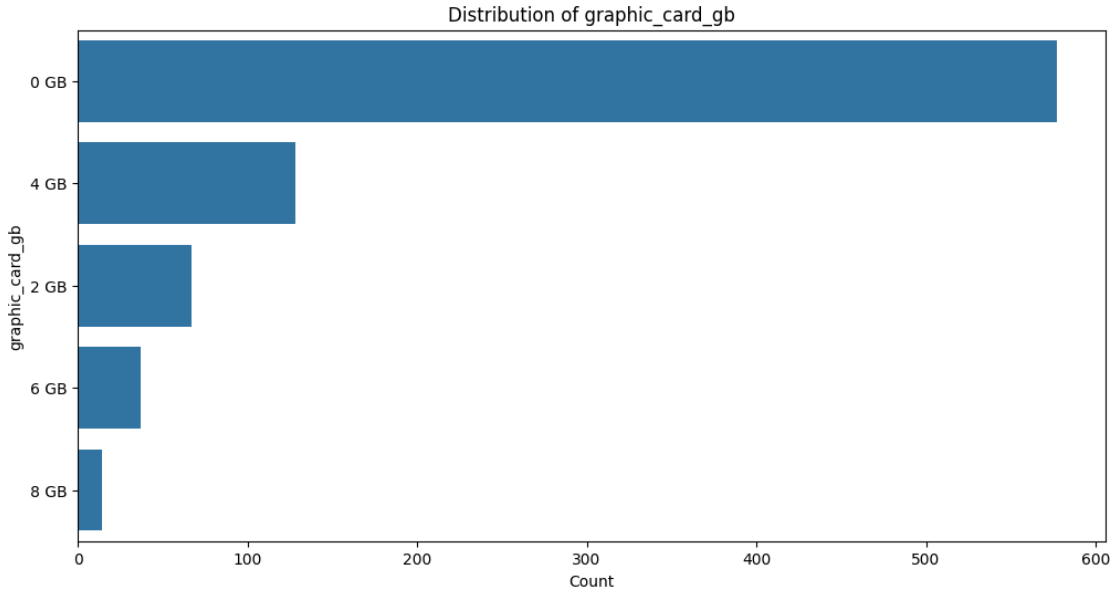

Price by hdd
1024 GB
0 GB
512 GB
2048 GB
hdd
0
100000
200000
300000
400000
Price

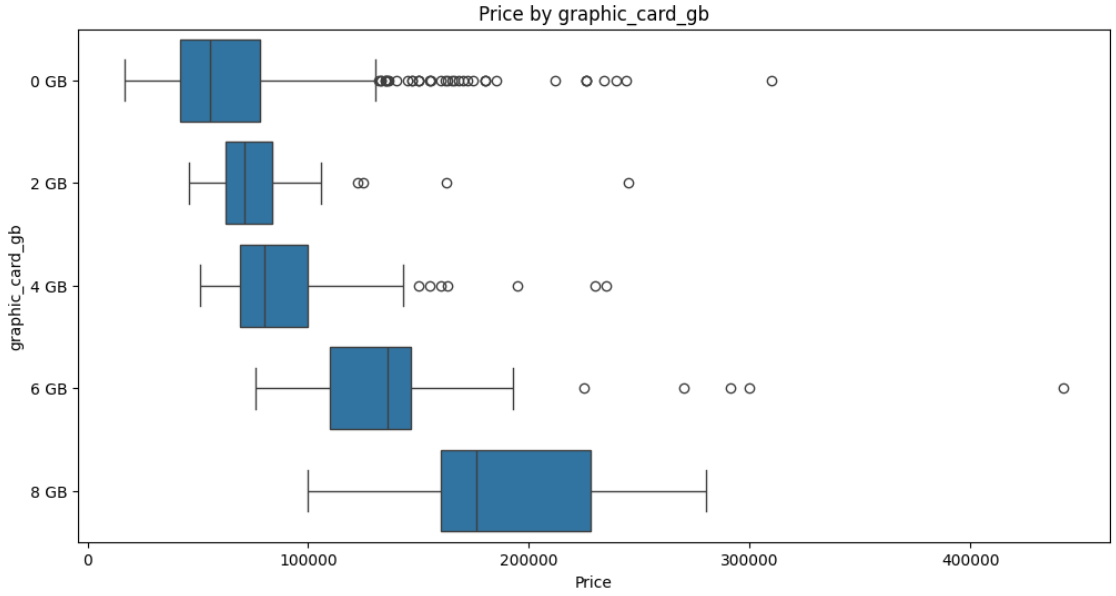

Distribution of os
Windows
Mac
DOS
os
0
100
200
300
400
500
600
700
800
Count

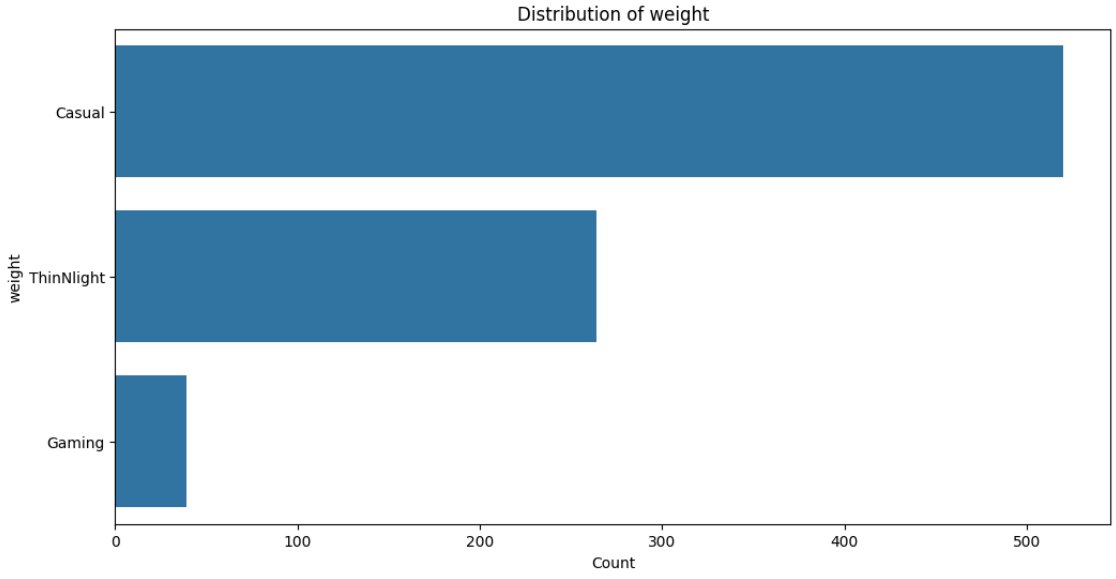

Price by os
Windows
DOS
Mac
os
0
100000
200000
300000
400000
Price

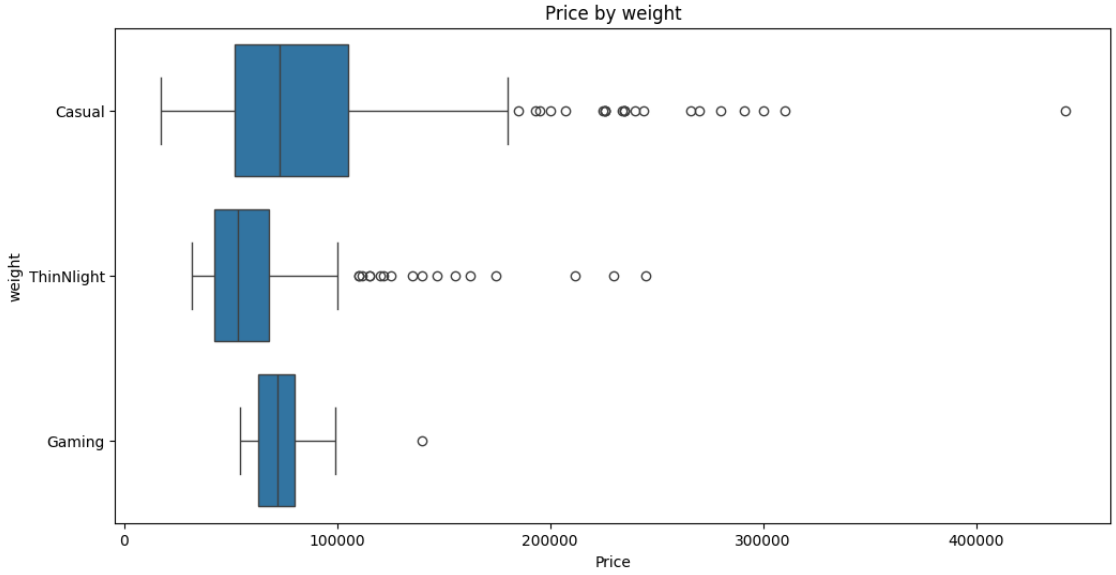

Distribution of os_bit
64-bit
32-bit
os_bit
0
100
200
300
400
500
600
700
Count

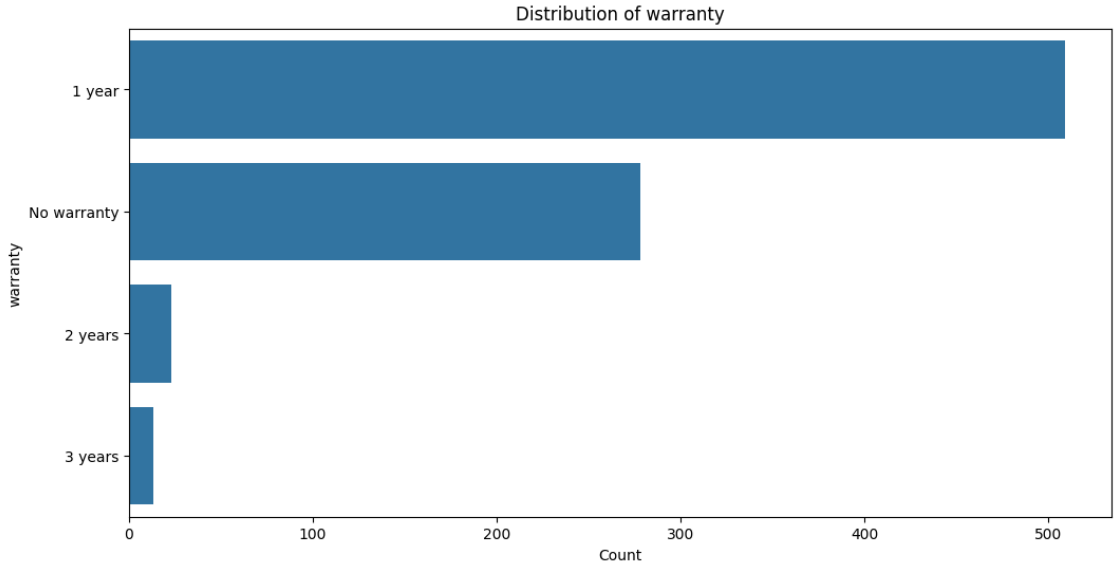

Price by os_bit
64-bit
32-bit
os_bit
0
100000
200000
300000
400000
Price

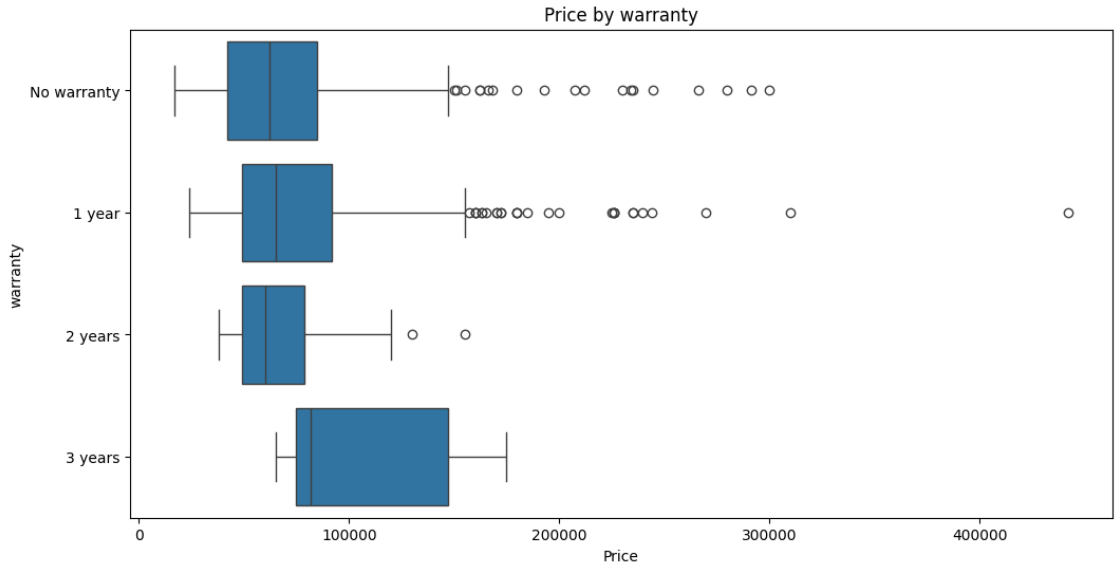

Distribution of graphic_card_gb
0 GB
4 GB
2 GB
6 GB
8 GB
graphic_card_gb
0
100
200
300
400
500
600
Count

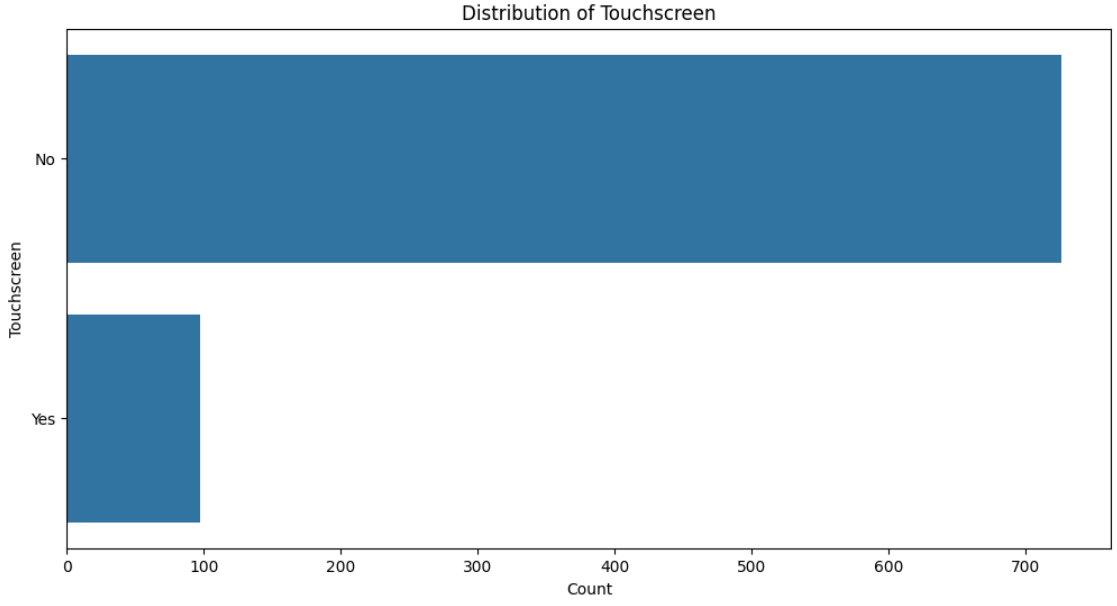

Price by graphic_card_gb
graphic_card_gb
0 GB
2 GB
4 GB
6 GB
8 GB
0
100000
200000
300000
400000
Price

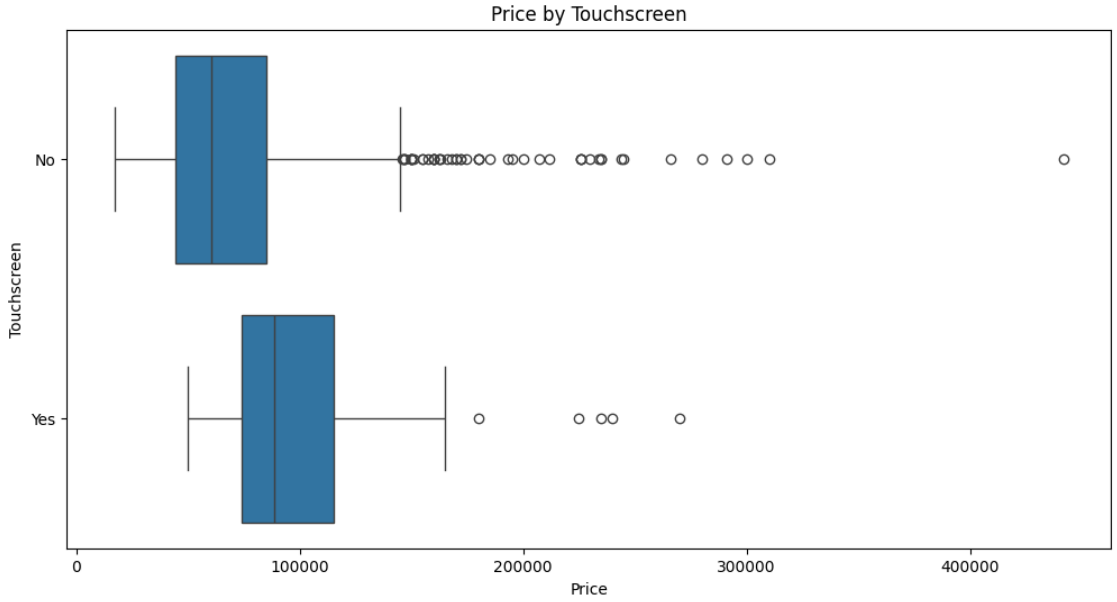

Distribution of weight
weight
Casual
ThinNlight
Gaming
0
100
200
300
400
500
Count

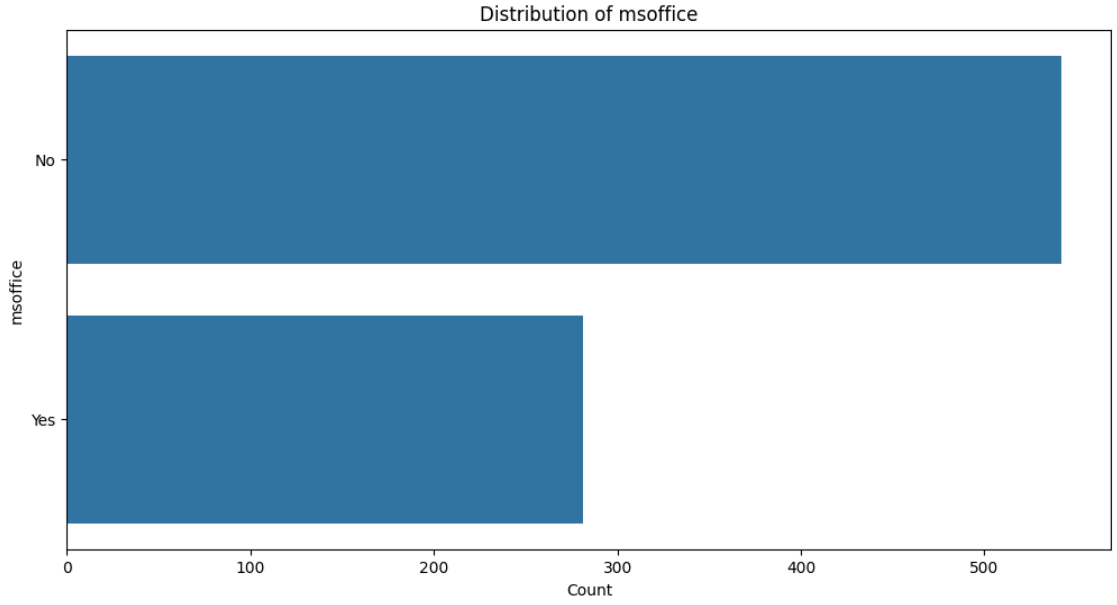

Price by weight
Casual
ThinNlight
Gaming
weight
0
100000
200000
300000
400000
Price

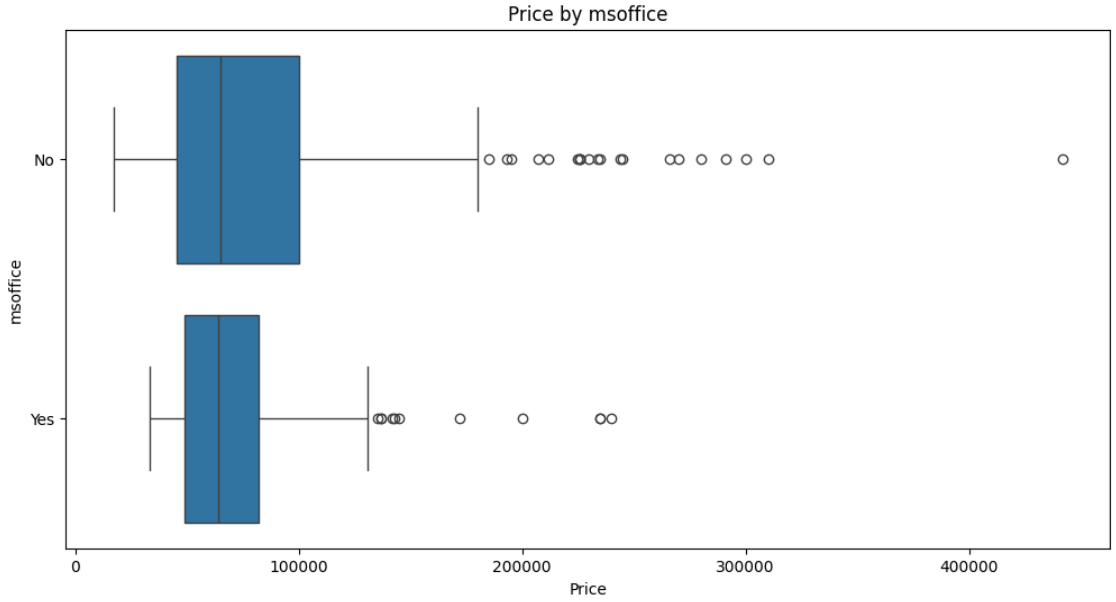

Distribution of warranty
1 year
No warranty
2 years
3 years
warranty
0
100
200
300
400
500
Count

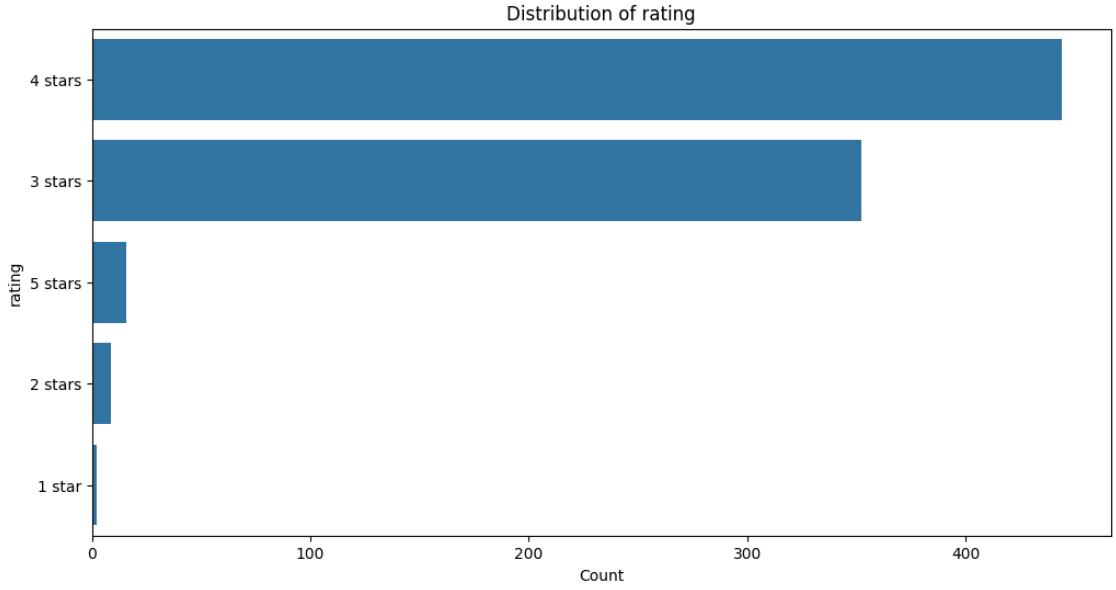

Price by warranty
No warranty
1 year
2 years
3 years
warranty
0
100000
200000
300000
400000
Price

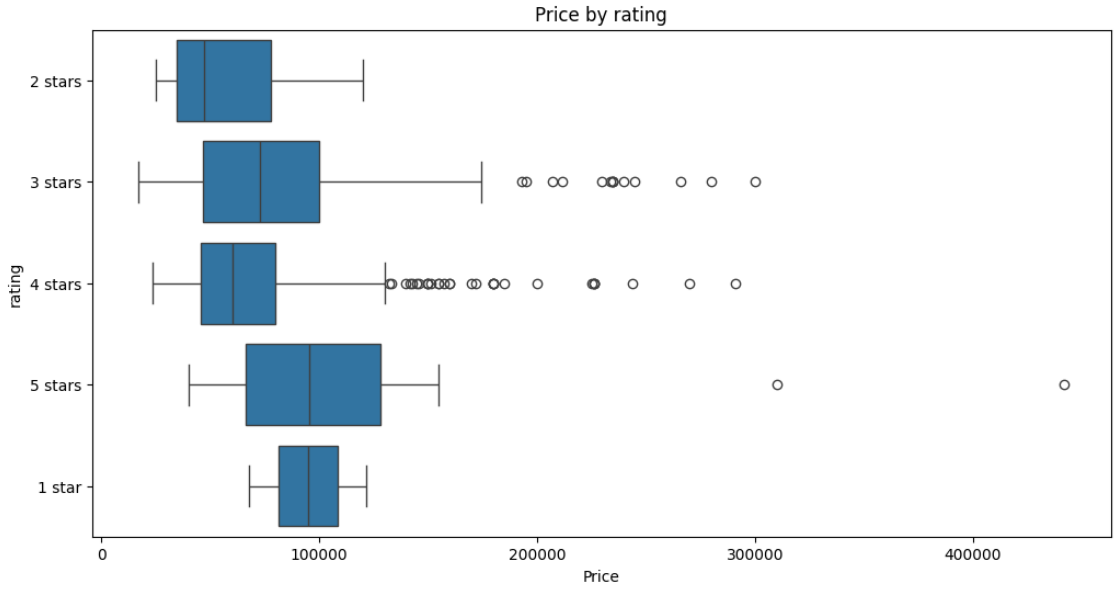

Distribution of Touchscreen
No
Yes
Touchscreen
0
100
200
300
400
500
600
700
Count


Price by Touchscreen
No
Yes
Touchscreen
0
100000
200000
300000
400000
Price

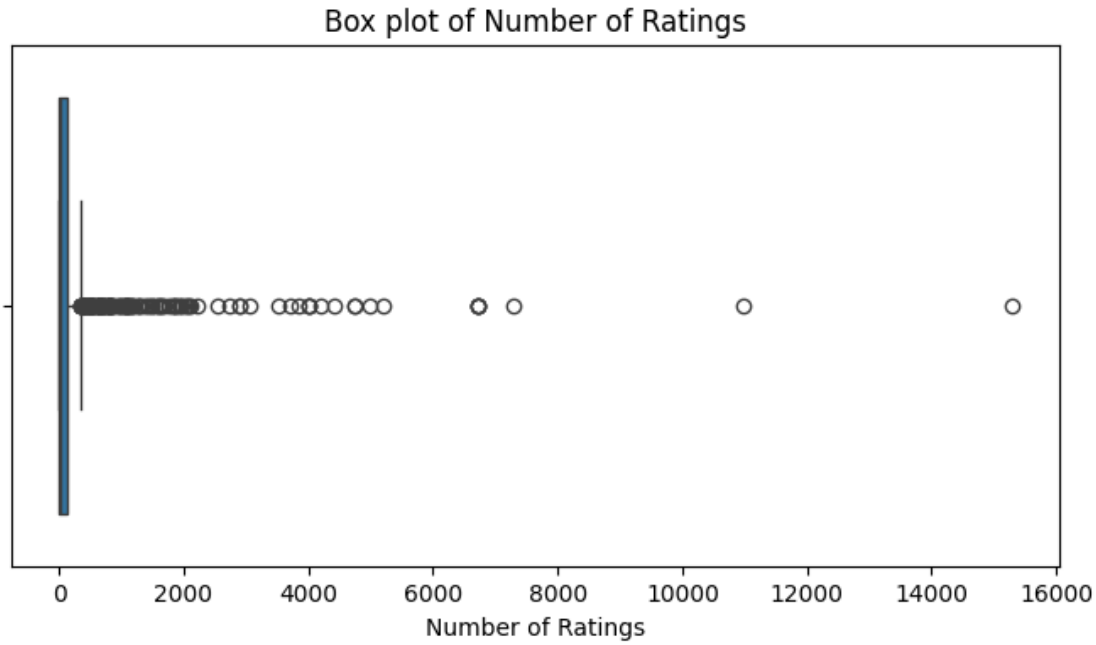

Distribution of msoffice
No
Yes
msoffice
0
100
200
300
400
500
Count

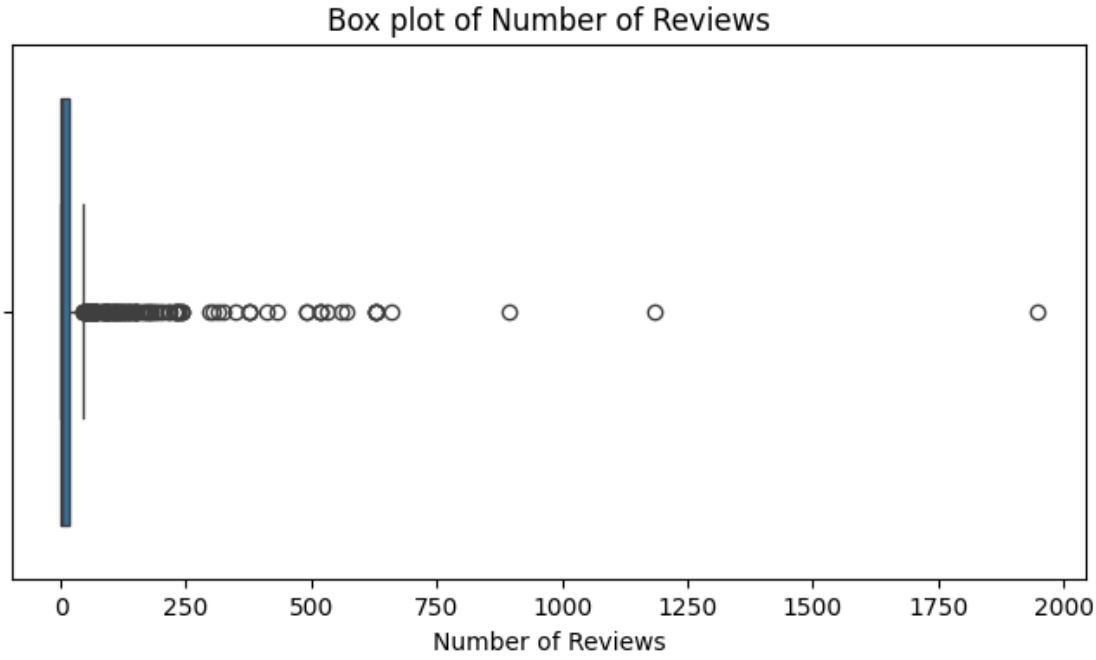

Price by msoffice
No
Yes
msoffice
0
100000
200000
300000
400000
Price

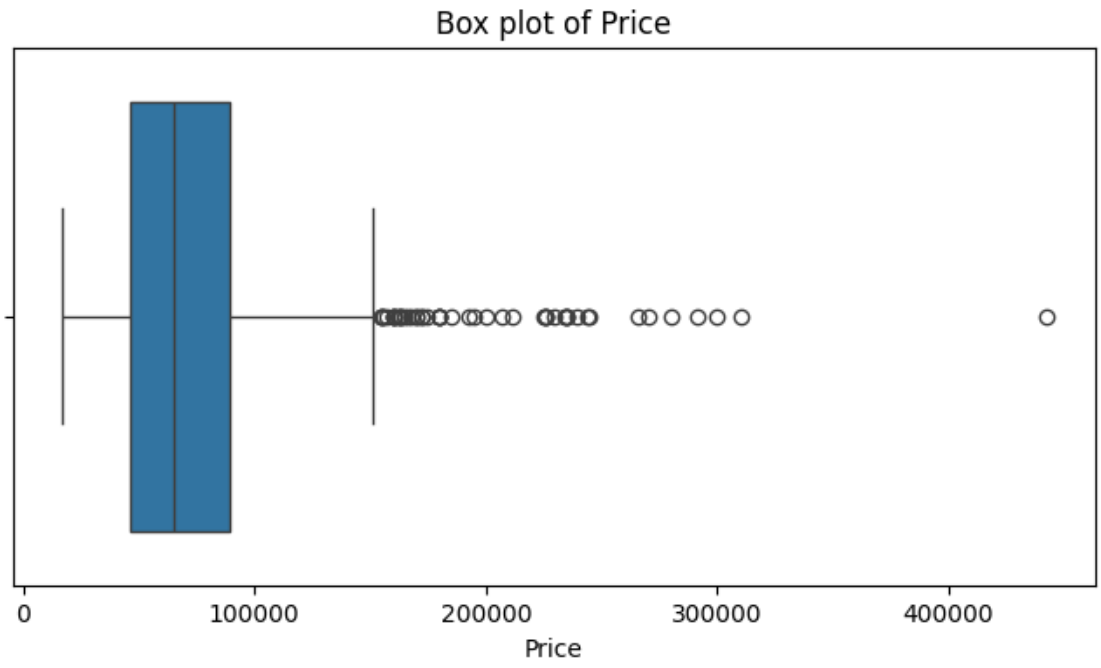

Distribution of rating
4 stars
3 stars
5 stars
2 stars
1 star
rating
0
100
200
300
400
Count




```
brand                0
processor_brand      0
processor_name       0
processor_gnrtn      0
ram_gb               0
ram_type             0
ssd                  0
hdd                  0
os                   0
os_bit               0
graphic_card_gb      0
weight               0
warranty             0
Touchscreen          0
msoffice             0
Price                0
rating               0
Number of Ratings    0
Number of Reviews    0
dtype: int64
```


Box plot of Number of Ratings
0
2000
4000
6000
8000
10000
12000
14000
16000
Number of Ratings


Box plot of Number of Reviews
0
250
500
750
1000
1250
1500
1750
2000
Number of Reviews



Feature Engineering:

[ ]:
```
# One-Hot Encode categorical features
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Example of creating an interaction feature (replace with relevant columns if␣
 ↪needed)
# This is a placeholder and you might want to choose features based on your␣
 ↪analysis
# df['Inches_x_Weight'] = df['Inches'] * df['Weight']

# Display the first few rows of the dataframe with new features
display(df.head())
```

```
   Price  Number of Ratings  Number of Reviews  brand_ASUS  brand_Avita  \
0  34649                  3                  0        True        False
1  38999                 65                  5       False        False
2  39999                  8                  1       False        False
3  69990                  0                  0        True        False
4  26990                  0                  0        True        False

   brand_DELL  brand_HP  brand_Lenovo  brand_MSI  brand_acer  …  \
0       False     False         False      False       False  …
1       False     False          True      False       False  …
2       False     False          True      False       False  …
3       False     False         False      False       False  …
```

```
4       False     False         False      False       False  …

   weight_ThinNlight  warranty_2 years  warranty_3 years  \
0              False             False             False
1              False             False             False
2              False             False             False
3              False             False             False
4              False             False             False

   warranty_No warranty  Touchscreen_Yes  msoffice_Yes  rating_2 stars  \
0                  True            False         False            True
1                  True            False         False           False
2                  True            False         False           False
3                  True            False         False           False
4                  True            False         False           False

   rating_3 stars  rating_4 stars  rating_5 stars
0           False           False           False
1            True           False           False
2            True           False           False
3            True           False           False
4            True           False           False

[5 rows x 64 columns]
```

Feature Engineering

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score

# Define features (X) and target variable (y)
X = df.drop('Price', axis=1)
y = df['Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize regression models
# (Note: The imports and variable definitions are repeated below,
# which is redundant but won't cause the NameError.
# You might want to clean this up later.)
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score

# Define features (X) and target variable (y)
```

```
X = df.drop('Price', axis=1)
y = df['Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize regression models
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)  # Make sure you did this
lr_prediction = lr_model.predict(X_test)
ridge_model = Ridge(alpha=1.0)  # You can tune the alpha parameter
lasso_model = Lasso(alpha=1.0)  # You can tune the alpha parameter

# Train the models
lr_model.fit(X_train, y_train)
ridge_model.fit(X_train, y_train)
lasso_model.fit(X_train, y_train)

print("Models trained successfully!")

# Train the models
lr_model.fit(X_train, y_train)
ridge_model.fit(X_train, y_train)
lasso_model.fit(X_train, y_train)

print("Models trained successfully!")
```

```
Models trained successfully!
Models trained successfully!
```

```
/usr/local/lib/python3.11/dist-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.049e+11, tolerance: 1.350e+08
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.11/dist-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.049e+11, tolerance: 1.350e+08
  model = cd_fast.enet_coordinate_descent(
```

Model Selection and Training:

Model Evaluation:

[ ]:
```
# Make predictions on the test set
lr_predictions = lr_model.predict(X_test)
ridge_predictions = ridge_model.predict(X_test)
lasso_predictions = lasso_model.predict(X_test)

# Import necessary metrics and numpy
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np

# Evaluate Linear Regression
lr_mae = mean_absolute_error(y_test, lr_predictions)
lr_mse = mean_squared_error(y_test, lr_predictions) # Calculate Mean Squared␣
 ↪Error
lr_rmse = np.sqrt(lr_mse) # Manually calculate RMSE
lr_r2 = r2_score(y_test, lr_predictions)

print("Linear Regression Evaluation:")
print(f"MAE: {lr_mae:.2f}")
print(f"RMSE: {lr_rmse:.2f}")
print(f"R-squared: {lr_r2:.2f}")
print("-" * 30)

# Evaluate Ridge Regression
ridge_mae = mean_absolute_error(y_test, ridge_predictions)
ridge_mse = mean_squared_error(y_test, ridge_predictions) # Calculate Mean␣
 ↪Squared Error
ridge_rmse = np.sqrt(ridge_mse) # Manually calculate RMSE
ridge_r2 = r2_score(y_test, ridge_predictions)

print("Ridge Regression Evaluation:")
print(f"MAE: {ridge_mae:.2f}")
print(f"RMSE: {ridge_rmse:.2f}")
print(f"R-squared: {ridge_r2:.2f}")
print("-" * 30)

# Evaluate Lasso Regression
lasso_mae = mean_absolute_error(y_test, lasso_predictions)
lasso_mse = mean_squared_error(y_test, lasso_predictions) # Calculate Mean␣
 ↪Squared Error
lasso_rmse = np.sqrt(lasso_mse) # Manually calculate RMSE
lasso_r2 = r2_score(y_test, lasso_predictions)

print("Lasso Regression Evaluation:")
print(f"MAE: {lasso_mae:.2f}")
print(f"RMSE: {lasso_rmse:.2f}")
print(f"R-squared: {lasso_r2:.2f}")
```

```python
# Import necessary metrics and numpy
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np

# Evaluate Linear Regression
lr_mae = mean_absolute_error(y_test, lr_predictions)
lr_mse = mean_squared_error(y_test, lr_predictions) # Calculate Mean Squared␣
 ↪Error
lr_rmse = np.sqrt(lr_mse) # Manually calculate RMSE
lr_r2 = r2_score(y_test, lr_predictions)

print("Linear Regression Evaluation:")
print(f"MAE: {lr_mae:.2f}")
print(f"RMSE: {lr_rmse:.2f}")
print(f"R-squared: {lr_r2:.2f}")
print("-" * 30)

# Evaluate Ridge Regression
ridge_mae = mean_absolute_error(y_test, ridge_predictions)
ridge_mse = mean_squared_error(y_test, ridge_predictions) # Calculate Mean␣
 ↪Squared Error
ridge_rmse = np.sqrt(ridge_mse) # Manually calculate RMSE
ridge_r2 = r2_score(y_test, ridge_predictions)

print("Ridge Regression Evaluation:")
print(f"MAE: {ridge_mae:.2f}")
print(f"RMSE: {ridge_rmse:.2f}")
print(f"R-squared: {ridge_r2:.2f}")
print("-" * 30)

# Evaluate Lasso Regression
lasso_mae = mean_absolute_error(y_test, lasso_predictions)
lasso_mse = mean_squared_error(y_test, lasso_predictions) # Calculate Mean␣
 ↪Squared Error
lasso_rmse = np.sqrt(lasso_mse) # Manually calculate RMSE
lasso_r2 = r2_score(y_test, lasso_predictions)

print("Lasso Regression Evaluation:")
print(f"MAE: {lasso_mae:.2f}")
print(f"RMSE: {lasso_rmse:.2f}")
print(f"R-squared: {lasso_r2:.2f}")
```

```
Linear Regression Evaluation:
MAE: 14654.81
RMSE: 24008.54
R-squared: 0.70
------------------------------
```

```
Ridge Regression Evaluation:
MAE: 14440.68
RMSE: 23572.80
R-squared: 0.71
------------------------------
Lasso Regression Evaluation:
MAE: 14604.09
RMSE: 23931.15
R-squared: 0.71
Linear Regression Evaluation:
MAE: 14654.81
RMSE: 24008.54
R-squared: 0.70
------------------------------
Ridge Regression Evaluation:
MAE: 14440.68
RMSE: 23572.80
R-squared: 0.71
------------------------------
Lasso Regression Evaluation:
MAE: 14604.09
RMSE: 23931.15
R-squared: 0.71
```

[ ]:
```python
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor # Import ensemble models
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np
import pandas as pd # Ensure pandas is imported if you haven't already

# Assuming df, X, y, X_train, X_test, y_train, y_test are already defined
# by running the previous cells in your notebook.

print("--- Ensemble Models ---")

# Initialize ensemble regression models
rf_model = RandomForestRegressor(n_estimators=100, random_state=42) # n_estimators is the number of trees
gbm_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42) # Parameters for GB

# Train the ensemble models
print("Training Random Forest Regressor...")
rf_model.fit(X_train, y_train)
print("Random Forest Regressor trained.")
```

```
print("Training Gradient Boosting Regressor...")
gbm_model.fit(X_train, y_train)
print("Gradient Boosting Regressor trained.")

# Make predictions with ensemble models
rf_predictions = rf_model.predict(X_test)
gbm_predictions = gbm_model.predict(X_test)

# Evaluate Random Forest Regressor
rf_mae = mean_absolute_error(y_test, rf_predictions)
rf_mse = mean_squared_error(y_test, rf_predictions)
rf_rmse = np.sqrt(rf_mse)
rf_r2 = r2_score(y_test, rf_predictions)

print("\nRandom Forest Regression Evaluation:")
print(f"MAE: {rf_mae:.2f}")
print(f"RMSE: {rf_rmse:.2f}")
print(f"R-squared: {rf_r2:.2f}")
print("-" * 30)

# Evaluate Gradient Boosting Regressor
gbm_mae = mean_absolute_error(y_test, gbm_predictions)
gbm_mse = mean_squared_error(y_test, gbm_predictions)
gbm_rmse = np.sqrt(gbm_mse)
gbm_r2 = r2_score(y_test, gbm_predictions)

print("Gradient Boosting Regression Evaluation:")
print(f"MAE: {gbm_mae:.2f}")
print(f"RMSE: {gbm_rmse:.2f}")
print(f"R-squared: {gbm_r2:.2f}")
print("-" * 30)


print("\n--- Cross-Validation ---")

# Example of using cross-validation with Linear Regression
# We will use K-Fold cross-validation
n_splits = 5 # Number of folds
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

# Perform cross-validation for Mean Squared Error (negative MSE because␣
 ↪cross_val_score minimizes)
# Scoring options: 'neg_mean_squared_error', 'neg_mean_absolute_error', 'r2'
lr_cv_scores_mse = cross_val_score(lr_model, X, y, cv=kf,␣
 ↪scoring='neg_mean_squared_error')
```

```python
# Convert negative MSE scores to positive MSE and then to RMSE
lr_cv_rmse_scores = np.sqrt(-lr_cv_scores_mse)

# Perform cross-validation for R-squared
lr_cv_scores_r2 = cross_val_score(lr_model, X, y, cv=kf, scoring='r2')

print(f"Linear Regression {n_splits}-Fold Cross-Validation Results:")
print(f"Mean RMSE across folds: {lr_cv_rmse_scores.mean():.2f} (+/-␣
 ↪{lr_cv_rmse_scores.std():.2f})")
print(f"Mean R-squared across folds: {lr_cv_scores_r2.mean():.2f} (+/-␣
 ↪{lr_cv_scores_r2.std():.2f})")
print("-" * 30)

# You can repeat the cross-validation process for Ridge, Lasso, Random Forest,␣
 ↪and Gradient Boosting
# For example, for Random Forest:
# rf_cv_scores_rmse = np.sqrt(-cross_val_score(rf_model, X, y, cv=kf,␣
 ↪scoring='neg_mean_squared_error'))
# rf_cv_scores_r2 = cross_val_score(rf_model, X, y, cv=kf, scoring='r2')
# print(f"Random Forest Regression {n_splits}-Fold Cross-Validation Results:")
# print(f"Mean RMSE across folds: {rf_cv_scores_rmse.mean():.2f} (+/-␣
 ↪{rf_cv_scores_rmse.std():.2f})")
# print(f"Mean R-squared across folds: {rf_cv_scores_r2.mean():.2f} (+/-␣
 ↪{rf_cv_scores_r2.std():.2f})")
# print("-" * 30)
```

```
--- Ensemble Models ---
Training Random Forest Regressor…
Random Forest Regressor trained.
Training Gradient Boosting Regressor…
Gradient Boosting Regressor trained.

Random Forest Regression Evaluation:
MAE: 12497.34
RMSE: 22259.85
R-squared: 0.75
------------------------------
Gradient Boosting Regression Evaluation:
MAE: 12963.89
RMSE: 22644.15
R-squared: 0.74
------------------------------

--- Cross-Validation ---
Linear Regression 5-Fold Cross-Validation Results:
Mean RMSE across folds: 23830.97 (+/- 4856.56)
Mean R-squared across folds: 0.72 (+/- 0.05)
```

---

[3]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
import gradio as gr



# Load dataset
df = pd.read_csv("laptopPrice.csv")

# Feature columns
features = [
    "brand", "processor_brand", "processor_name", "processor_gnrtn",
    "ram_gb", "ram_type", "ssd", "hdd", "os", "os_bit",
    "graphic_card_gb", "Touchscreen", "msoffice"
]
target = "Price"

X = df[features]
y = df[target]

# Preprocessing: encode categorical columns
categorical_cols = X.select_dtypes(include="object").columns.tolist()
preprocessor = ColumnTransformer([
    ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_cols)
])

# Model pipeline
pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("regressor", RandomForestRegressor(n_estimators=100, random_state=42))
])

# Train/test split and fit
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
pipeline.fit(X_train, y_train)

# Prediction function
def predict_price(brand, processor_brand, processor_name, processor_gnrtn,
                  ram_gb, ram_type, ssd, hdd, os, os_bit,
                  graphic_card_gb, Touchscreen, msoffice):
```

```
    input_df = pd.DataFrame([[brand, processor_brand, processor_name,␣
 ↪processor_gnrtn,
                              ram_gb, ram_type, ssd, hdd, os, os_bit,
                              graphic_card_gb, Touchscreen, msoffice]],
                            columns=features)
    price = pipeline.predict(input_df)[0]
    return f" Estimated Laptop Price: {int(price):,}"

# Get dropdown options
options = {col: sorted(df[col].dropna().unique().tolist()) for col in features}

# Gradio UI
inputs = [
    gr.Dropdown(choices=options["brand"], label="Brand"),
    gr.Dropdown(choices=options["processor_brand"], label="Processor Brand"),
    gr.Dropdown(choices=options["processor_name"], label="Processor Name"),
    gr.Dropdown(choices=options["processor_gnrtn"], label="Processor␣
 ↪Generation"),
    gr.Dropdown(choices=options["ram_gb"], label="RAM"),
    gr.Dropdown(choices=options["ram_type"], label="RAM Type"),
    gr.Dropdown(choices=options["ssd"], label="SSD"),
    gr.Dropdown(choices=options["hdd"], label="HDD"),
    gr.Dropdown(choices=options["os"], label="Operating System"),
    gr.Dropdown(choices=options["os_bit"], label="OS Bit"),
    gr.Dropdown(choices=options["graphic_card_gb"], label="Graphic Card"),
    gr.Dropdown(choices=options["Touchscreen"], label="Touchscreen"),
    gr.Dropdown(choices=options["msoffice"], label="MS Office")
]

gr.Interface(
    fn=predict_price,
    inputs=inputs,
    outputs="text",
    title=" Laptop Price Prediction App"
).launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the
Gradio app to work, sharing must be enabled. Automatically setting `share=True`
(you can turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://c37619ea69b41b2551.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades,
run `gradio deploy` from the terminal in the working directory to deploy to
Hugging Face Spaces (https://huggingface.co/spaces)
```

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

[3]: