## Laptop Price Prediction for SmartTech Co

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

import warnings
warnings.filterwarnings('ignore')
```

Data Exploration and Understanding:

```
df=pd.read_csv(r"D:\ml\laptop.csv") # reading dataset

df.sample(5) # displays 5 random rows

      Unnamed: 0.1  Unnamed: 0 Company   TypeName Inches  \
763            763       763.0    Asus  Ultrabook   13.3
466            466       466.0    Acer   Notebook   15.6
705            705       705.0    Dell   Notebook   15.6
1240          1240      1240.0  Lenovo   Notebook   15.6
634            634       634.0    Asus   Notebook   15.6

                  ScreenResolution
Cpu  Ram  \
763   IPS Panel Quad HD+ 3200x1800            Intel Core i5 7200U
2.5GHz  8GB
466                       1366x768              Intel Core i3 6006U
2GHz  4GB
705              Full HD 1920x1080            Intel Core i5 7200U
2.5GHz  8GB
1240             Full HD 1920x1080           AMD A12-Series 9720P
3.6GHz  6GB
634                       1366x768  Intel Celeron Dual Core N3350
1.1GHz  8GB

         Memory                       Gpu       OpSys  Weight
Price
763   256GB SSD     Intel HD Graphics 620  Windows 10    1.2kg
60153.1200
466   500GB HDD  Nvidia GeForce GTX 940MX  Windows 10    2.2kg
24988.3200
705   256GB SSD     Intel HD Graphics 620  Windows 10   2.18kg
42357.6000
1240  256GB SSD            AMD Radeon 530  Windows 10    2.2kg
31838.5296
634     1TB HDD     Intel HD Graphics 500  Windows 10      2kg
21258.7200

df.columns
```

```
Index(['Unnamed: 0.1', 'Unnamed: 0', 'Company', 'TypeName', 'Inches',
       'ScreenResolution', 'Cpu', 'Ram', 'Memory', 'Gpu', 'OpSys',
'Weight',
       'Price'],
      dtype='object')
```

```
# Removing Unwanted columns
df.drop(columns=["Unnamed: 0.1", 'Unnamed: 0'], inplace=True)
```

```
df.columns
```

```
Index(['Company', 'TypeName', 'Inches', 'ScreenResolution', 'Cpu',
'Ram',
       'Memory', 'Gpu', 'OpSys', 'Weight', 'Price'],
      dtype='object')
```

```
print(f"{df.shape}") # There are 1303 rows and 11 columns
```

```
(1303, 11)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Company           1273 non-null   object
 1   TypeName          1273 non-null   object
 2   Inches            1273 non-null   object
 3   ScreenResolution  1273 non-null   object
 4   Cpu               1273 non-null   object
 5   Ram               1273 non-null   object
 6   Memory            1273 non-null   object
 7   Gpu               1273 non-null   object
 8   OpSys             1273 non-null   object
 9   Weight            1273 non-null   object
 10  Price             1273 non-null   float64
dtypes: float64(1), object(10)
memory usage: 112.1+ KB
```

```
df.isnull().sum() # there are 30 null values in each row
```

```
Company             30
TypeName            30
Inches              30
ScreenResolution    30
Cpu                 30
Ram                 30
Memory              30
Gpu                 30
```

```
OpSys               30
Weight              30
Price               30
dtype: int64
```

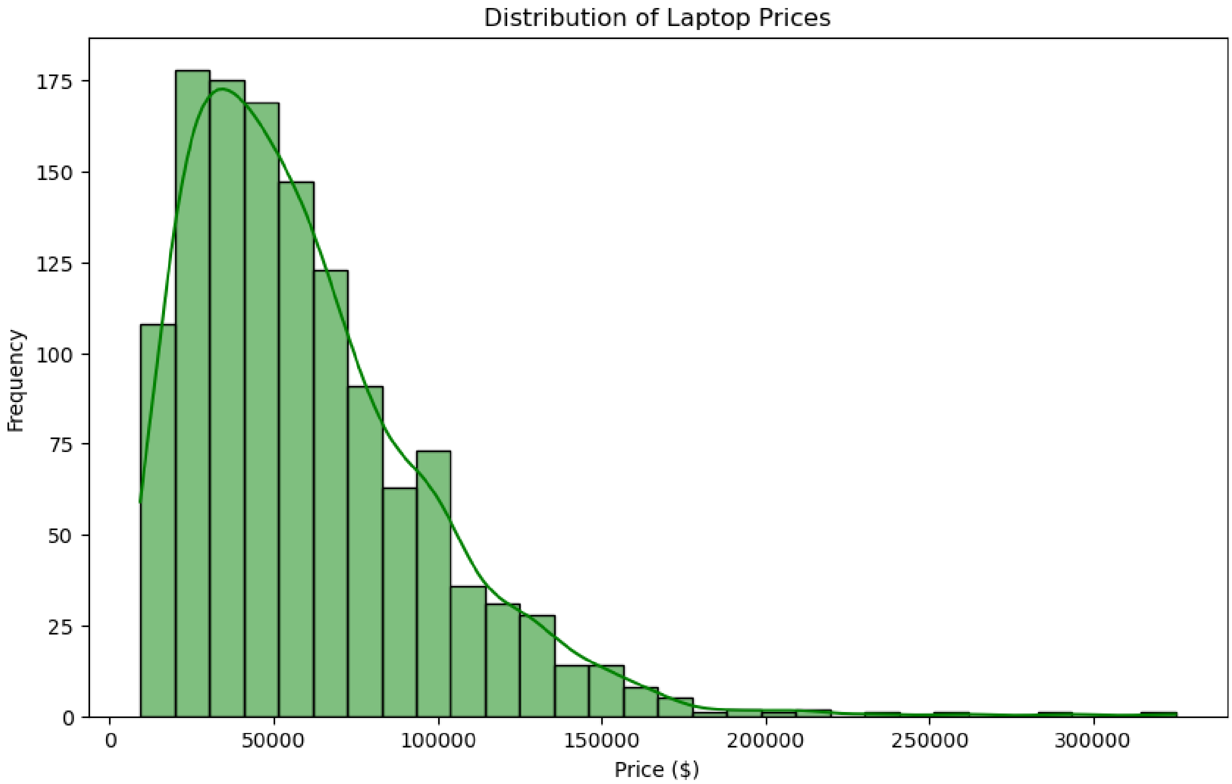
```python
# As there are equal null values in each column Droping the rows rows
which contails null values
df.dropna(inplace=True)

df.isnull().sum() # There are no null values
```

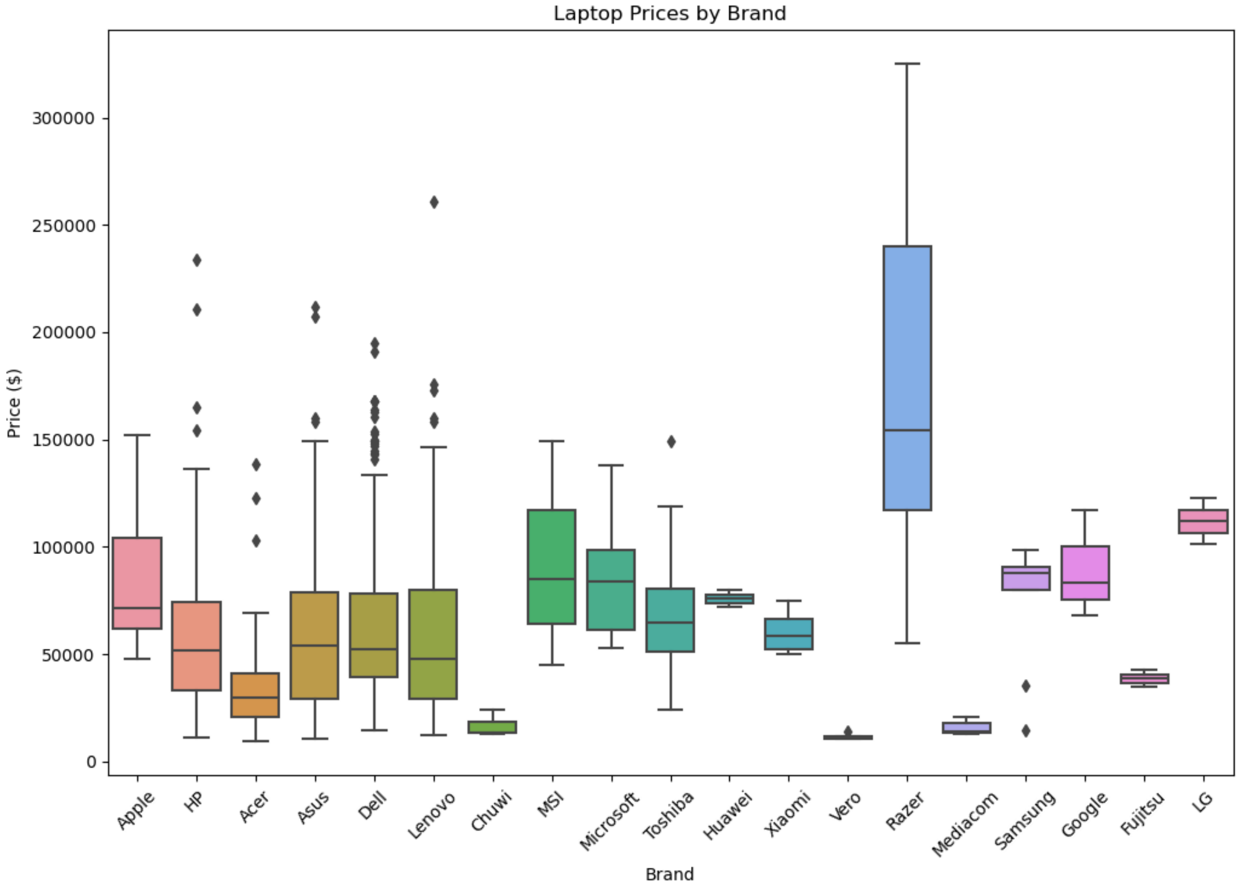
```
Company             0
TypeName            0
Inches              0
ScreenResolution    0
Cpu                 0
Ram                 0
Memory              0
Gpu                 0
OpSys               0
Weight              0
Price               0
dtype: int64
```

Data Visualization

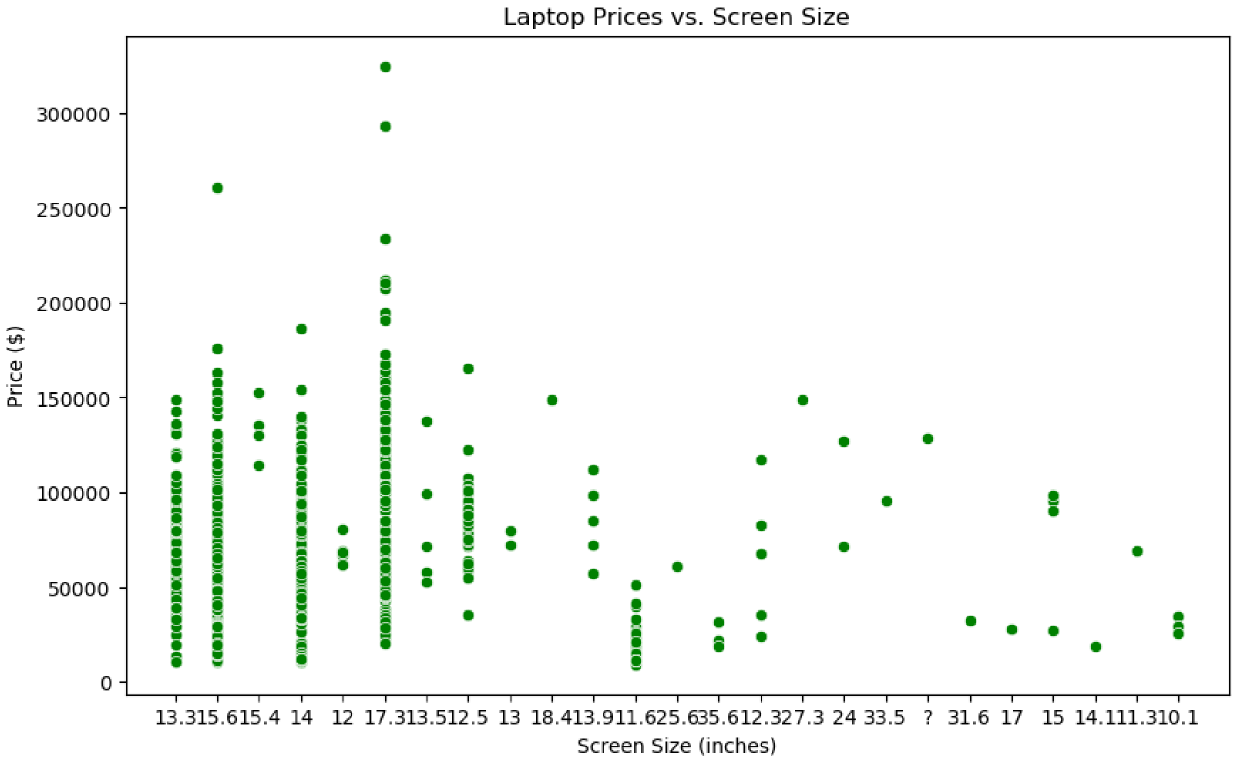
```python
# Visualize trends in laptop prices
plt.figure(figsize=(10, 6))
sns.histplot(df['Price'], bins=30, kde=True, color='green')
plt.title('Distribution of Laptop Prices')
plt.xlabel('Price ($)')
plt.ylabel('Frequency')
plt.show()
```



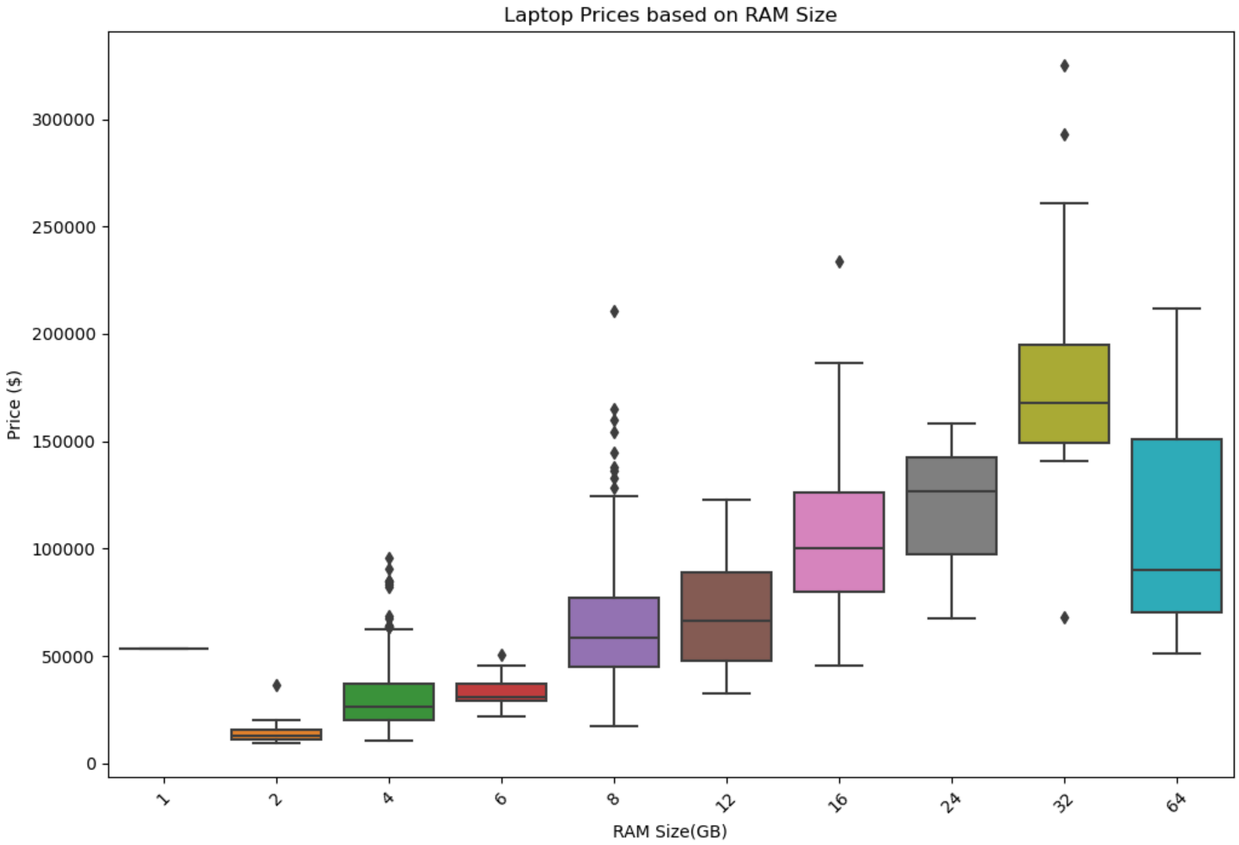
```
# Boxplot of laptop prices by brand
plt.figure(figsize=(12, 8))
sns.boxplot(x='Company', y='Price', data=df)
plt.title('Laptop Prices by Brand')
plt.xlabel('Brand')
plt.ylabel('Price ($)')
plt.xticks(rotation=45)
plt.show()
```



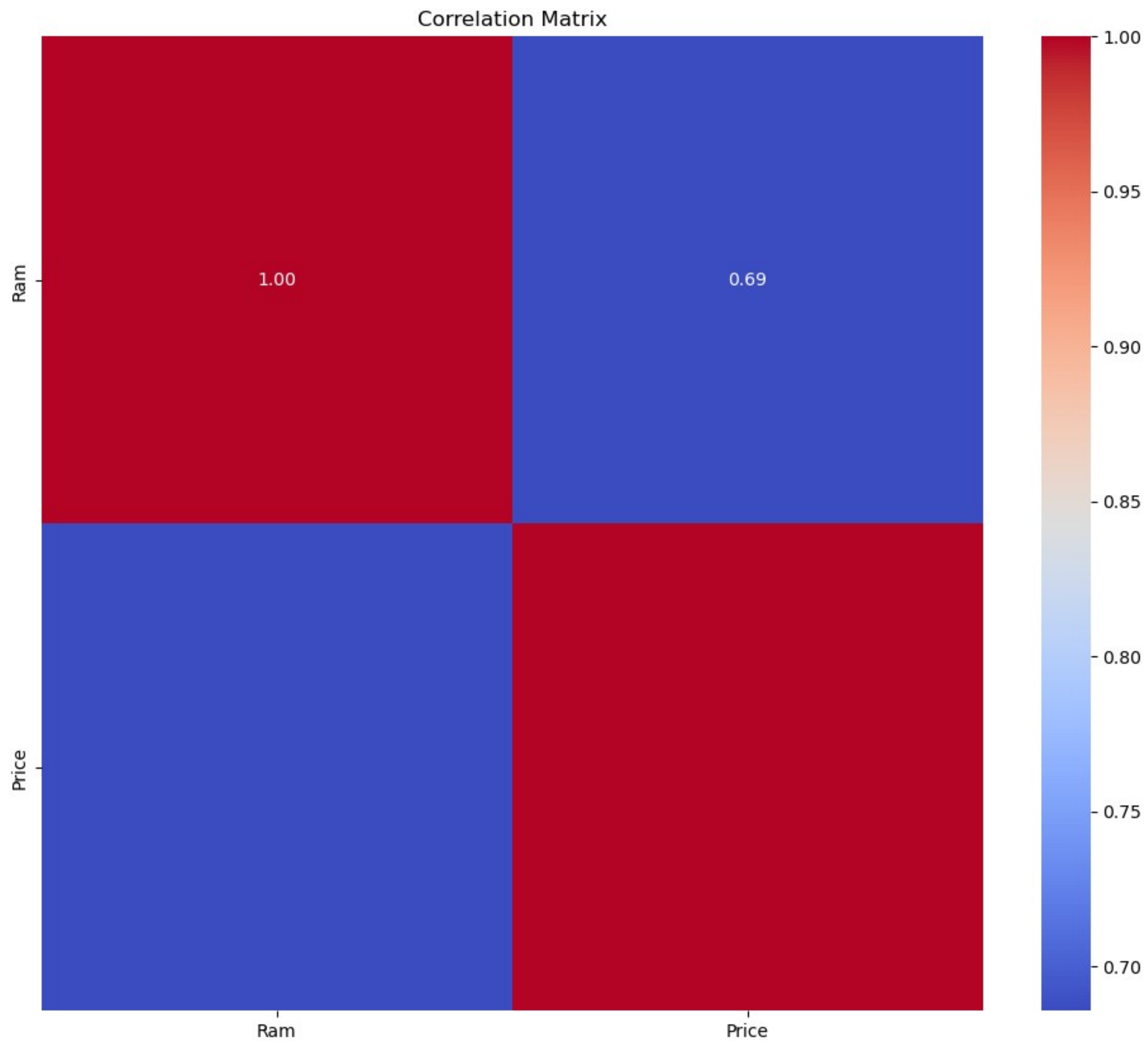
```
# Scatter plot of laptop prices against screen size
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Inches', y='Price', data=df, color='green')
plt.title('Laptop Prices vs. Screen Size')
plt.xlabel('Screen Size (inches)')
plt.ylabel('Price ($)')
plt.show()
```



```
# Clean RAM Column
# Visualization of RAM
df['Ram'] = df['Ram'].str.replace('GB', '').astype(int)
plt.figure(figsize=(12, 8))
sns.boxplot(x='Ram', y='Price', data=df)
plt.title('Laptop Prices based on RAM Size')
plt.xlabel('RAM Size(GB)')
plt.ylabel('Price ($)')
plt.xticks(rotation=45)
plt.show()
```



```
# Visualize correlation matrix - Have to select only Numerical columns
for Correlation
numeric = df.select_dtypes(include=[float,int]).columns
plt.figure(figsize=(12, 10))
sns.heatmap(df[numeric].corr(), annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



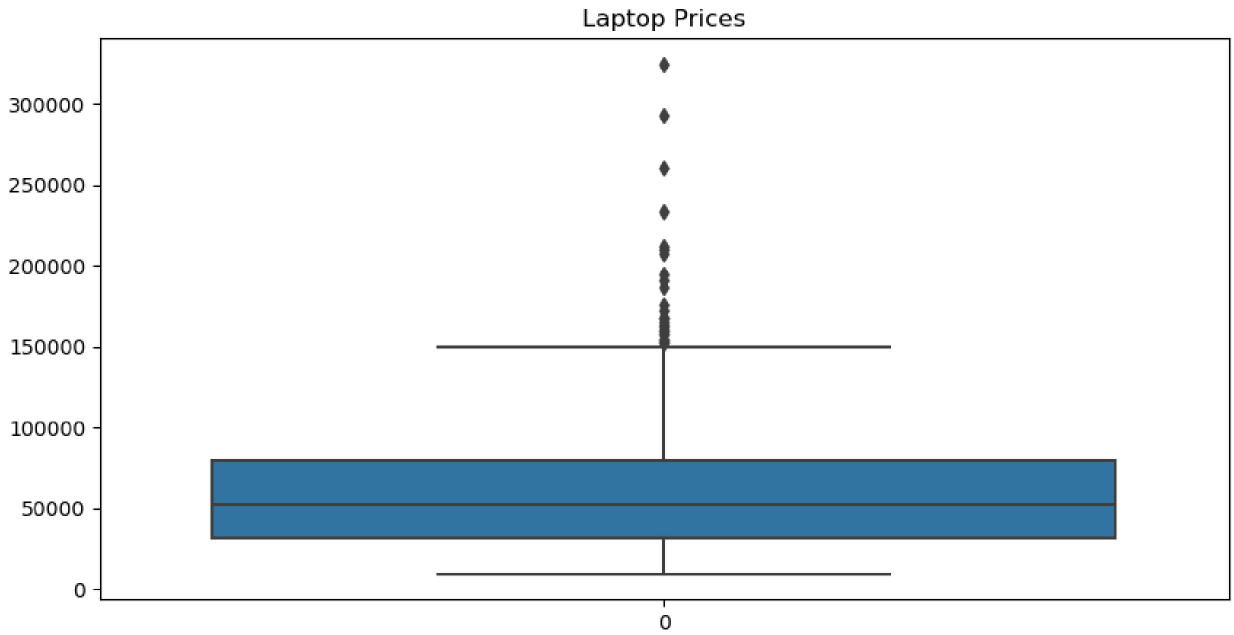
Data Preprocessing:

```
# Missing Values
print(f"Missing Values:\n {df.isnull().sum()}")

Missing Values:
 Company             0
TypeName            0
Inches              0
ScreenResolution    0
Cpu                 0
Ram                 0
Memory              0
Gpu                 0
OpSys               0
Weight              0
```

```
Price               0
dtype: int64
```

```
#  Identify Outlier in Price column using Box Plot
plt.figure(figsize=(10, 5))
sns.boxplot(df['Price'])
plt.title('Laptop Prices')
plt.show()
```



```
# Handling Outliers with IQR Method
def remove_outliers(df, column):
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        return df[(df[column] >= lower_bound) & (df[column] <=
upper_bound)]

df = remove_outliers(df, 'Price')

print(f"Shape of dataset after handling Outliers {df.shape}")
```

```
Shape of dataset after handling Outliers (1245, 11)
```

Encode categorical variables

```
categorical = df.select_dtypes(include=[object]).columns
print(f"Caltegorical Columns {categorical}")
```

```
Caltegorical Columns Index(['Company', 'TypeName', 'Inches',
'ScreenResolution', 'Cpu', 'Memory',
       'Gpu', 'OpSys', 'Weight'],
      dtype='object')

# Performing One-hot encoding on Categorica Variables
encoding = pd.get_dummies(df,columns=categorical,drop_first=True)

encoding.head()

   Ram        Price  Company_Apple  Company_Asus  Company_Chuwi
Company_Dell  \
0    8   71378.6832           True         False          False
False
1    8   47895.5232           True         False          False
False
2    8   30636.0000          False         False          False
False
3   16  135195.3360           True         False          False
False
4    8   96095.8080           True         False          False
False

   Company_Fujitsu  Company_Google  Company_HP  Company_Huawei  ...  \
0            False           False       False           False  ...
1            False           False       False           False  ...
2            False           False        True           False  ...
3            False           False       False           False  ...
4            False           False       False           False  ...

   Weight_4.5kg  Weight_4.6kg  Weight_4kg  Weight_5.4kg  Weight_5.8kg
\
0         False         False       False         False         False

1         False         False       False         False         False

2         False         False       False         False         False

3         False         False       False         False         False

4         False         False       False         False         False


   Weight_6.2kg  Weight_7.2kg  Weight_8.23kg  Weight_8.4kg  Weight_?
0         False         False          False         False     False
1         False         False          False         False     False
2         False         False          False         False     False
3         False         False          False         False     False
4         False         False          False         False     False

[5 rows x 531 columns]
```

Ensure the dataset is ready for model training

```
# Seperating the features(X) and target(Y) Variables
X = encoding.drop(columns=["Price"])
y = encoding["Price"]

X.shape

(1245, 530)

y.shape

(1245,)
```

Feature Engineering:

Extract meaningful features to enhance model performance

```
# Extracting CPU brand
df['CpuBrand'] = df['Cpu'].str.split().str[0]
df['CpuBrand'].unique()

array(['Intel', 'AMD', 'Samsung'], dtype=object)

# Extracting screen height and width from the screenresolution column
df['ScreenWidth'] = df['ScreenResolution'].str.extract(r'(\d+)x')
[0].astype(int)
df['ScreenHeight'] = df['ScreenResolution'].str.extract(r'(\d+)x')
[0].astype(int)

df[['ScreenResolution', 'ScreenWidth', 'ScreenHeight']].head()

                     ScreenResolution  ScreenWidth  ScreenHeight
0  IPS Panel Retina Display 2560x1600         2560          2560
1                            1440x900         1440          1440
2                   Full HD 1920x1080         1920          1920
3  IPS Panel Retina Display 2880x1800         2880          2880
4  IPS Panel Retina Display 2560x1600         2560          2560
```

Creation of New features

```
# Inches column should ne Numeric and Have to handle conversion issues
df['Inches'] = pd.to_numeric(df['Inches'].str.replace('"', ''),
errors='coerce')
df.dropna(subset=['Inches'], inplace=True)
```

Encode Categorical Features

```
categorical_columns = df.select_dtypes(include=['object']).columns #
categorical columns
categorical_columns
```

```
Index(['Company', 'TypeName', 'ScreenResolution', 'Cpu', 'Memory',
'Gpu',
       'OpSys', 'Weight', 'CpuBrand'],
      dtype='object')
```

```
encoding =
pd.get_dummies(df,columns=categorical_columns,drop_first=True)
encoding.head()
```

```
   Inches  Ram        Price  ScreenWidth  ScreenHeight  Company_Apple
\
0    13.3    8   71378.6832         2560          2560           True

1    13.3    8   47895.5232         1440          1440           True

2    15.6    8   30636.0000         1920          1920          False

3    15.4   16  135195.3360         2880          2880           True

4    13.3    8   96095.8080         2560          2560           True


   Company_Asus  Company_Chuwi  Company_Dell  Company_Fujitsu  ...  \
0         False          False         False            False  ...
1         False          False         False            False  ...
2         False          False         False            False  ...
3         False          False         False            False  ...
4         False          False         False            False  ...

   Weight_4kg  Weight_5.4kg  Weight_5.8kg  Weight_6.2kg  Weight_7.2kg
\
0       False         False         False         False         False

1       False         False         False         False         False

2       False         False         False         False         False

3       False         False         False         False         False

4       False         False         False         False         False


   Weight_8.23kg  Weight_8.4kg  Weight_?  CpuBrand_Intel
CpuBrand_Samsung
0          False         False     False            True
False
1          False         False     False            True
```

```
False
2          False         False     False                True
False
3          False         False     False                True
False
4          False         False     False                True
False

[5 rows x 512 columns]
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV

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

def evaluate_model(model, X_test, y_test, X_train, y_train):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    abs = mean_absolute_error(y_test, y_pred)
    sqr = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"Model: {model.__class__.__name__}")
    print(f" Mean Absolute Error: {abs: .2f}")
    print(f" Mean Square Error: {sqr: .2f}")
    print(f" R-Squared: {r2: .2f}")
    print('_' * 30)

    return abs, sqr, r2
```

Linear Regression

```
lr = LinearRegression()
evaluate_model(lr, X_test, y_test, X_train, y_train)
```

```
Model: LinearRegression
 Mean Absolute Error:  38418054892521.20
 Mean Square Error:  29241147447119535695990882304.00
 R-Squared: -26501020985635262464.00
______________________________

(38418054892521.195, 2.9241147447119536e+28, -2.6501020985635262e+19)
```

Random Forest

```
rf = RandomForestRegressor(random_state=42)
evaluate_model(rf, X_test, y_test, X_train, y_train)
```

```
Model: RandomForestRegressor
 Mean Absolute Error:  9679.64
 Mean Square Error:  183796936.77
 R-Squared:  0.83
______________________________

(9679.636134245782, 183796936.7715833, 0.8334262878265054)
```

```
gb = GradientBoostingRegressor(random_state=42)
evaluate_model(gb, X_test, y_test, X_train, y_train)
```

```
Model: GradientBoostingRegressor
 Mean Absolute Error:  11184.26
 Mean Square Error:  226761900.16
 R-Squared:  0.79
______________________________

(11184.256581997686, 226761900.16170156, 0.7944874808420094)
```

Hyperparameter Tuning

Hyperparameter Tuning for Random Forest

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best CV Score: {-grid_search.best_score_}")
best_model = grid_search.best_estimator_
evaluate_model(best_model, X_test, y_test, X_train, y_train)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

Hyperparameter Tuning for Gradient Boosting

```
param_grid_gb = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 4, 5],
    'subsample': [0.8, 0.9, 1.0]
}

# Setup the GridSearchCV
grid_search = GridSearchCV(estimator=gb, param_grid_gb=param_grid_gb, 
cv=5, scoring='neg_mean_squared_error', n_jobs=-1, verbose=2)

# Fit the model
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best CV Score: {-grid_search.best_score_}")

# Evaluate the best model
best_model = grid_search.best_estimator_
evaluate_model(best_model, X_test, y_test, X_train, y_train)
```

Real-time Predictions (Assuming Flask API):

```
from flask import Flask, request, jsonify
import joblib

app = Flask(__name__)
model = joblib.load('best_rf_model.pkl')

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    features = [data['RAM'], data['Storage'], data['Weight'], 
data['Brand_Dell'], data['Brand_HP'], data['Brand_Lenovo']]
    features_scaled = scaler.transform([features])
    prediction = model.predict(features_scaled)[0]
    return jsonify({'predicted_price': prediction})

if __name__ == '__main__':
    app.run(debug=True)
```

```
##  Interpretability and Insights

shap.initjs()
explainer = shap.Explainer(best_rf_model)
shap_values = explainer.shap_values(X_test_scaled)

shap.summary_plot(shap_values, X_test_scaled, plot_type='bar',
show=False)
plt.title('SHAP Values for Feature Importance')
plt.show()

----------------------------------------------------------------------
-----
ModuleNotFoundError                       Traceback (most recent call
last)
Cell In[1], line 1
----> 1 import shap
      3 shap.initjs()
      4 explainer = shap.Explainer(best_rf_model)

ModuleNotFoundError: No module named 'shap'

##  Client Presentation (Using Plotly for Interactive Dashboards)

import plotly.express as px

# Create interactive dashboards
fig = px.scatter(df, x='RAM', y='Price', color='Brand',
trendline='ols', title='RAM vs. Price by Brand')
fig.show()
```