Laptop Price Prediction Notebook

Laptop Price Prediction

This notebook performs a full analysis and regression model pipeline for laptop price prediction.

Contents:

- Data loading
- Beginner, Intermediate, Advanced, Expert analyses
- Feature engineering
- Regression model with evaluation and feature importance

Dataset source:
`https://raw.githubusercontent.com/jhhalls/Data-Analysis/main/Datasets/laptop_data.csv`

Run each cell in order.

```
# Imports and settings
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, OrdinalEncoder
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import joblib
print('Imports done')
```

```
# Load dataset from GitHub raw URL
url = 'https://raw.githubusercontent.com/jhhalls/Data-Analysis/main/Datasets/laptop_
data.csv'
try:
df = pd.read_csv(url)
print('Loaded data from GitHub raw URL:', url)
except Exception as e:
print('Could not download from internet in this environment.\nPlease place the CSV
in the working directory as `laptop_data.csv`.')
# fallback: try local file
df = pd.read_csv('laptop_data.csv')
print('Shape:', df.shape)
df.head()
```

Beginner Level — Basic Data Understanding

Answer the beginner tasks with code and short commentary.

```
# Basic overview
df.info(), df.describe(include='all').T.head()
```

```
# 1. Unique laptop companies and company with highest number of models
company_counts = df['Company'].value_counts()
num_unique_companies = df['Company'].nunique()
top_company = company_counts.idxmax()
top_company_count = company_counts.max()
num_unique_companies, top_company, top_company_count
```

```
# 2. Average price overall and by company
avg_price_overall = df['Price'].mean()
avg_price_by_company =
df.groupby('Company')['Price'].mean().sort_values(ascending=False)
```

```
avg_price_overall, avg_price_by_company.head(10)

# 3. Most common TypeName and its average price
most_common_type = df['TypeName'].value_counts().idxmax()
avg_price_most_common_type = df[df['TypeName']==most_common_type]['Price'].mean()
most_common_type, avg_price_most_common_type

# 4. Distribution of screen sizes
inches_counts = df['Inches'].value_counts().sort_index()
inches_counts, inches_counts.idxmax()

# 5. Min, max, avg laptop weight and companies with heaviest/lightest
weight_stats = df['Weight'].agg(['min','max','mean'])
# convert weight to numeric if string contains 'kg'
def to_kg(x):
try:
return float(str(x).replace('kg','').strip())
except:
return np.nan
df['Weight_num']=df['Weight'].apply(to_kg)
company_heaviest = df.loc[df['Weight_num'].idxmax(),'Company']
company_lightest = df.loc[df['Weight_num'].idxmin(),'Company']
weight_stats, company_heaviest, company_lightest
```

Intermediate Level — Comparative Insights

```
# Average price by Operating System
avg_price_by_os = df.groupby('OpSys')['Price'].mean().sort_values(ascending=False)
avg_price_by_os

# Analyze how RAM affects price
# Normalize RAM column (it may be like '8GB' or numeric). We'll extract numbers.
def parse_ram(x):
if pd.isna(x): return np.nan
s = str(x)
s = s.replace('GB','').replace('gb','').strip()
try:
return int(s)
except:
try:
return int(float(s))
except:
return np.nan

# If RAM is already numeric, keep it
if df['RAM'].dtype == object:
df['RAM_num'] = df['RAM'].apply(parse_ram)
else:
df['RAM_num'] = df['RAM'].astype(int)

ram_price = df.groupby('RAM_num')['Price'].mean().sort_index()
ram_price

# ScreenResolution influence
res_price =
df.groupby('ScreenResolution')['Price'].mean().sort_values(ascending=False)
res_price.head(10)

# CPU brands dominance
# Extract brand from CPU string (e.g., 'Intel Core i5' -> 'Intel')
def cpu_brand(s):
if pd.isna(s): return 'Unknown'
s = s.lower()
if 'intel' in s: return 'Intel'
if 'amd' in s: return 'AMD'
if 'apple' in s: return 'Apple'
return s.split()[0].title()

df['CPU_brand'] = df['CPU'].apply(cpu_brand)
cpu_counts = df['CPU_brand'].value_counts()
cpu_avg_price = df.groupby('CPU_brand')['Price'].mean().sort_values(ascending=False)
cpu_counts, cpu_avg_price.head(10)
```

```
# Top 5 most expensive laptop models
top5 = df.sort_values('Price', ascending=False).head(5)
top5[['Company','TypeName','Inches','CPU','GPU','RAM','Memory','OpSys','Weight','Pri
ce']]
```

Advanced Level — Deeper Analytical Insights

```
# Correlation between numeric variables
# Ensure numeric columns
for c in ['Inches','RAM_num','Weight_num','Price']:
print(c, df[c].dtype)
corr_df = df[['Inches','RAM_num','Weight_num','Price']].dropna()
corr = corr_df.corr()
corr
```

```
# Extract screen quality categories from ScreenResolution
# Heuristic: look for '4k', '3840', '2k', '2560', 'full hd', 'fhd', 'hd'
def screen_quality(s):
s = str(s).lower()
if '4k' in s or '3840' in s: return '4K'
if '2k' in s or '2560' in s: return '2K'
if 'full hd' in s or 'fhd' in s or '1920' in s: return 'Full HD'
if 'hd' in s or '1366' in s: return 'HD'
return 'Other'

df['Screen_Quality'] = df['ScreenResolution'].apply(screen_quality)
df.groupby('Screen_Quality')['Price'].agg(['count','mean']).sort_values('mean',ascen
ding=False)
```

```
# Gaming vs Non-gaming based on GPU (heuristic: if GPU string contains 'intel' or
'integrated' -> non-gaming)
def is_dedicated_gpu(g):
if pd.isna(g): return False
s = str(g).lower()
if 'intel' in s or 'integrated' in s or 'uhd' in s or 'hd graphics' in s: return
False
return True

df['Dedicated_GPU'] = df['GPU'].apply(is_dedicated_gpu)
df.groupby('Dedicated_GPU')['Price'].agg(['count','mean'])
```

```
# Lightweight vs heavier laptops average price (<1.5 kg)
df['Lightweight'] = df['Weight_num'] < 1.5
df.groupby('Lightweight')['Price'].agg(['count','mean'])
```

```
# Identify best performance-to-price using simple proxy: (CPU_rank + GPU_rank +
RAM)/Price
# We'll create rough ranks: CPU_brand preference + GPU dedicated + RAM
# CPU rank: Intel/AMD/Apple heuristics, fallback 0
cpu_rank_map = {'Apple':3, 'Intel':2, 'AMD':2}
df['CPU_rank'] = df['CPU_brand'].map(cpu_rank_map).fillna(1)
df['GPU_rank'] = df['GPU'].apply(lambda g: 2 if is_dedicated_gpu(g) else 1)
df['Perf_Score'] = df['CPU_rank'] + df['GPU_rank'] + (df['RAM_num'].fillna(0)/4)
df['Perf_per_Price'] = df['Perf_Score'] / df['Price']
df.sort_values('Perf_per_Price',
ascending=False).head(10)[['Company','CPU','GPU','RAM','Price','Perf_per_Price']]
```

Expert Level — Predictive Modeling

```
# Quick preprocessing and modeling pipeline
# We'll engineer a few useful features and train RandomForest or GradientBoosting.
df_model = df.copy()

# Feature engineering helper functions
def parse_memory(mem):
# examples: '256GB SSD', '1TB HDD', '256GB SSD + 1TB HDD'
if pd.isna(mem): return ''
s = str(mem).lower()
# count total GB
total_gb = 0
parts = s.replace('tb',' TB').replace('gb',' GB').split('+')
for p in parts:
p = p.strip()
if 'tb' in p:
```

```
try:
total_gb += float(p.split('tb')[0].strip())*1024
except:
pass
elif 'gb' in p:
try:
total_gb += float(p.split('gb')[0].strip())
except:
pass
return total_gb

df_model['Memory_GB'] = df_model['Memory'].apply(parse_memory)
# Keep numeric RAM, weight
df_model['RAM_num'] = df_model['RAM_num'].fillna(df_model['RAM'].apply(lambda x:
parse_ram(x) if pd.notna(x) else np.nan))
df_model['Weight_num'] = df_model['Weight_num']
# Create simple target
y = df_model['Price']

# Select features
features = ['Company','TypeName','Inches','Screen_Quality','CPU_brand','GPU','RAM_nu
m','Memory_GB','OpSys','Weight_num','Dedicated_GPU']
X = df_model[features].copy()

# Simple train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
print('Train shape:', X_train.shape, 'Test shape:', X_test.shape)


# Build preprocessing
numeric_features = ['Inches','RAM_num','Memory_GB','Weight_num']
numeric_transformer = Pipeline(steps=[
('imputer', SimpleImputer(strategy='median')),
('scaler', StandardScaler())
])

categorical_features =
['Company','TypeName','Screen_Quality','CPU_brand','OpSys','Dedicated_GPU']
categorical_transformer = Pipeline(steps=[
('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
('onehot', OneHotEncoder(handle_unknown='ignore', sparse=False))
])

# For GPU we will do simple label encoding via target frequency -> we'll convert to
top N GPUs and 'other'
top_gpus = X_train['GPU'].value_counts().nlargest(20).index.tolist()
X_train['GPU_simple'] = X_train['GPU'].apply(lambda g: g if g in top_gpus else
'Other')
X_test['GPU_simple'] = X_test['GPU'].apply(lambda g: g if g in top_gpus else
'Other')

gpu_transformer = Pipeline(steps=[
('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
('onehot', OneHotEncoder(handle_unknown='ignore', sparse=False))
])

preprocessor = ColumnTransformer(
transformers=[
('num', numeric_transformer, numeric_features),
('cat', categorical_transformer, categorical_features),
('gpu', gpu_transformer, ['GPU_simple'])
], remainder='drop'
)

# Model pipeline
model = RandomForestRegressor(n_estimators=200, random_state=42, n_jobs=-1)
pipe = Pipeline(steps=[('preprocessor', preprocessor),
('model', model)])


# Fit model (this may take time depending on data size)
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2, mae, rmse
```

```
# Feature importance: approximate via permutation importance (scikit-learn)
from sklearn.inspection import permutation_importance
res = permutation_importance(pipe, X_test, y_test, n_repeats=10, random_state=42,
n_jobs=-1)
# map importance back to feature names from preprocessor
# get column names
ohe_cols = list(pipe.named_steps['preprocessor'].transformers_[1][1].named_steps['on
ehot'].get_feature_names_out(categorical_features))
gpu_cols = list(pipe.named_steps['preprocessor'].transformers_[2][1].named_steps['on
ehot'].get_feature_names_out(['GPU_simple']))
all_feature_names = numeric_features + ohe_cols + gpu_cols
importances = pd.Series(res.importances_mean,
index=all_feature_names).sort_values(ascending=False)
importances.head(20)

# Save model for later use
joblib.dump(pipe, 'laptop_price_model.joblib')
print('Model saved to laptop_price_model.joblib')
```

Conclusion & Next steps

- The notebook performs EDA, feature engineering, and trains a RandomForest model.
- If you need higher R^2, try Gradient Boosting, LightGBM/XGBoost, hyperparameter tuning, target transformation (log), or more feature engineering.

Files created: `/mnt/data/Laptop_price_prediction.ipynb`, `laptop_price_model.joblib` (model saved after training when you run the notebook).