Laptop Price Prediction

Team Members

Tarun Sai
Madhavi Raval
Rutul Patel
Namira Ganam
Urvashi Vora
Dev Arya
Mansi Kharb
Srivatsan Rangarajan
Bisman Singh

Introduction

Objective

The goal of this project is to predict the price of laptops based on multiple factors, including specifications like brand, processor, RAM, storage, and screen size.

This project uses machine learning techniques to predict laptop prices, which can assist in price comparison, online retail, and customer decision-making.

Importance of the Problem

- Relevance for E-Commerce: Predicting laptop prices helps e-commerce platforms adjust pricing dynamically based on features.
- Customer Benefits: Provides customers with the estimated value of the laptop, helping them make informed purchasing decisions.
- Retailer/Business Benefits: Helps retailers set competitive prices and optimize inventory.

Problem Statement

Definition

Predicting the price of a laptop using its various specifications such as processor type, RAM size, storage, etc.

Target Audience

Define who will benefit from the prediction (e.g., consumers, retailers, and data scientists).

Specific Goal

We will use machine learning techniques to accurately predict the prices of laptops based on the features provided in the dataset.

Data Collection

Source : Public dataset containing laptop specifications and prices
https://raw.githubusercontent.com/Raghavagr/Laptop_Price_Prediction/refs/heads/main/laptop_data.csv

Description of Columns: List important columns/features that affect laptop price predictions.
Brand: The laptop's brand (e.g., Dell, HP, Lenovo).
Processor: Type and speed of the processor (e.g., Intel Core i7, AMD Ryzen).
RAM: Amount of RAM in GB.
Storage: Type and size of storage (e.g., 512GB SSD).
Price: Target variable (laptop price).
Other Features: Screen size, battery life, weight, etc.

	laptop_ID	Company	Product	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price_euros
0	1	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg	1339.69
1	2	Apple	Macbook Air	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	898.94
2	3	HP	250 G6	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg	575.00
3	4	Apple	MacBook Pro	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg	2537.45
4	5	Apple	MacBook	Ultrabook	13.3	IPS Panel Retina Display	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics	macOS	1.37kg	1803.60

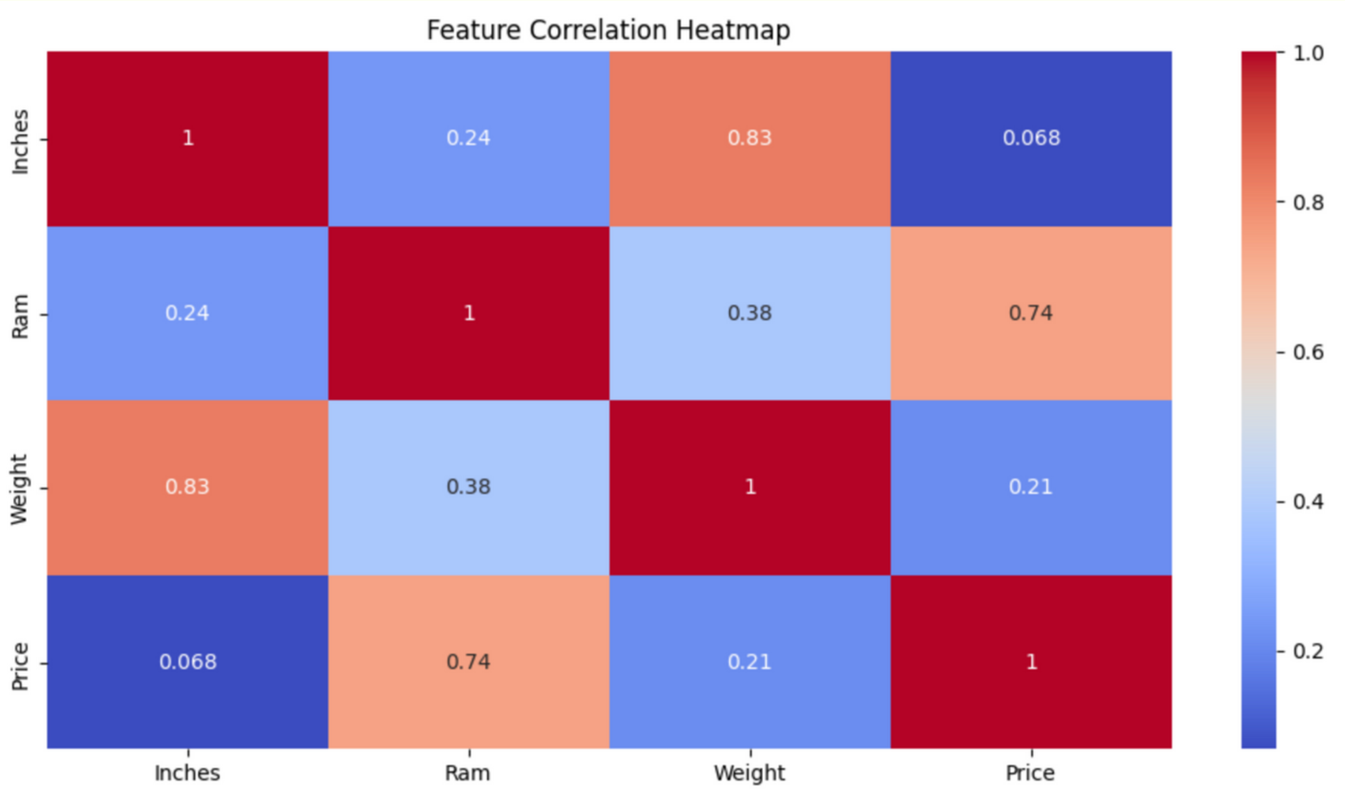
Data Exploration & Feature Correlation

Insights from Data Analysis

RAM has the highest correlation with price (0.74).
Weight and Inches are closely related (0.83 correlation).

Feature Selection
Selected highly correlated variables to improve model accuracy.

Explanation: This heatmap highlights relationships between features, helping us determine which attributes influence price the most.



Model Training Pipeline

1. Baseline Model Overview:

- A neural network model is trained using batch normalization and dropout for regularization.

- The model consists of 3 hidden layers:
128, 64, and 32 units with ReLU activation.
Drop out (0.3) to prevent overfitting.
Batch normalization to improve convergence.
Regression output layer with a single neuron.

- Optimizer: Adam (learning rate = 0.001)

- Loss function: Mean Squared Error (MSE)
Metric: Mean Absolute Error (MAE)

2. FastAI Pretrained Model

We also tried using a pretrained FastAI model for fine-tuning:

- Used TabularDataLoaders for the laptop dataset.
- Applied preprocessing: Categorify, FillMissing, and Normalize.
- Model fine-tuned for 20 epochs using a three-layer architecture.

3. Model Evaluation

- Baseline Model: Achieved a solid performance with lower validation MAE compared to the FastAI model.

- FastAI Model: While pretrained, it underperformed in comparison, possibly due to the need for further tuning.

4. Key Insights

The baseline neural network model showed better results than the pretrained FastAI model.
Further experimentation is needed for optimizing both models to improve performance.

5. Next Steps

Explore other model architectures and hyperparameters.
Fine-tune the FastAI model for better performance.

epoch	train_loss	valid_loss	mae	time
0	5397139968.000000	4883407872.000000	60773.285156	00:00

epoch	train_loss	valid_loss	mae	time
0	5028477952.000000	4883266048.000000	60773.500000	00:00
1	4893994496.000000	4883095040.000000	60773.546875	00:00
2	4845563392.000000	4882812416.000000	60772.500000	00:00
3	4959824896.000000	4882531840.000000	60772.449219	00:00
4	5185981440.000000	4882553856.000000	60773.574219	00:00
5	5401711104.000000	4881626624.000000	60770.000000	00:00
6	5325240320.000000	4881010176.000000	60767.687500	00:00
7	5219448320.000000	4880388608.000000	60766.789062	00:00
8	5066437120.000000	4879170560.000000	60758.929688	00:00
9	5184925696.000000	4878701568.000000	60756.839844	00:00
10	5309646336.000000	4879066112.000000	60761.531250	00:00
11	5493085696.000000	4879688192.000000	60766.628906	00:00
12	5177981440.000000	4877129728.000000	60751.101562	00:00
13	5181702144.000000	4877006848.000000	60752.269531	00:00
14	5139030528.000000	4876996608.000000	60752.132812	00:00
15	5158381568.000000	4876388352.000000	60746.945312	00:00
16	4904230912.000000	4875490304.000000	60743.402344	00:00
17	4915354624.000000	4876279808.000000	60749.058594	00:00
18	5251393024.000000	4877307904.000000	60754.324219	00:00
19	5069301248.000000	4876003328.000000	60747.371094	00:00

```
Baseline Model Test MAE: 38898.40625
FastAI Pretrained Model Test MAE: 59731.01731209479
```

```
baseline_model = Sequential([
    keras.layers.Input(shape=(X_train.shape[1],)),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1)  # Regression output
])

baseline_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='mse', metrics=['mae'])
baseline_history = baseline_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# FastAI Pretrained Model - we tried multiple layer options - no luck improving it
df['Price'] = y
dls = TabularDataLoaders.from_df(df, path='.', procs=[Categorify, FillMissing, Normalize], cont_names=numerical_features, cat_names=categorical_features, y
learn = tabular_learner(dls, layers=[512, 256, 128], metrics=mae)
learn.fine_tune(20)
```

Key Points

Baseline Model: The sequential model with layers and activation functions.

Model Compilation: The optimizer (Adam) and loss function (MSE) used for training.

Training: The model is trained for 50 epochs with a batch size of 32, including a validation split.

FastAI Pretrained Model: Setup for using FastAI's pretrained tabular learner.

Model Architecture

Deep Learning Model Structure

Input Layer: Standardized numerical features + encoded categorical variables.

Hidden Layers

Fully Connected Layer (128 Neurons, ReLU) + Batch Normalization + Dropout
Fully Connected Layer (64 Neurons, ReLU) + Batch Normalization + Dropout
Fully Connected Layer (32 Neurons, ReLU)
Output Layer: Single Neuron (Regression Output for Price Prediction)

Optimization Strategy

Optimizer: Adam (Learning Rate = 0.001)
Loss Function: Mean Squared Error (MSE)

Hyperparameter Tuning

Overview

Objective: Optimize the model's performance by adjusting the most important hyperparameters.
Tools Used: Keras Tuner for automated hyperparameter search.

Key Hyperparameters Tuned

Units in layers: Range between 32 and 256.
Dropout rate: Range from 0.2 to 0.5 to prevent overfitting.
Learning rate: Chosen from 1e-2, 1e-3, and 1e-4.

Hyperparameter Tuning Process

Random Search: A wide search over possible values of hyperparameters.
Objective: Minimize validation MAE (Mean Absolute Error).

Results

Best Hyperparameters: Found after several trials, improving model performance.
Retraining: A new model is built with these optimal values.

Inference Pipeline

Overview

Objective: Use the trained model to make predictions and evaluate performance on the test set.
This stage is crucial to assess how well the model generalizes to unseen data.

Inference Pipeline Steps

Prediction: Using the trained model to predict the target variable (Laptop Price).
Evaluation: Calculate the performance of the model using metrics such as Mean Absolute Error (MAE) and R^2 score.

Key Evaluation Metrics

Mean Absolute Error (MAE): Measures the average magnitude of errors in the predictions.
R^2 Score: Indicates how well the model's predictions fit the actual data.

Results

The model's performance is evaluated and visualized to compare predicted vs actual values.

R2 Score of 81.22%

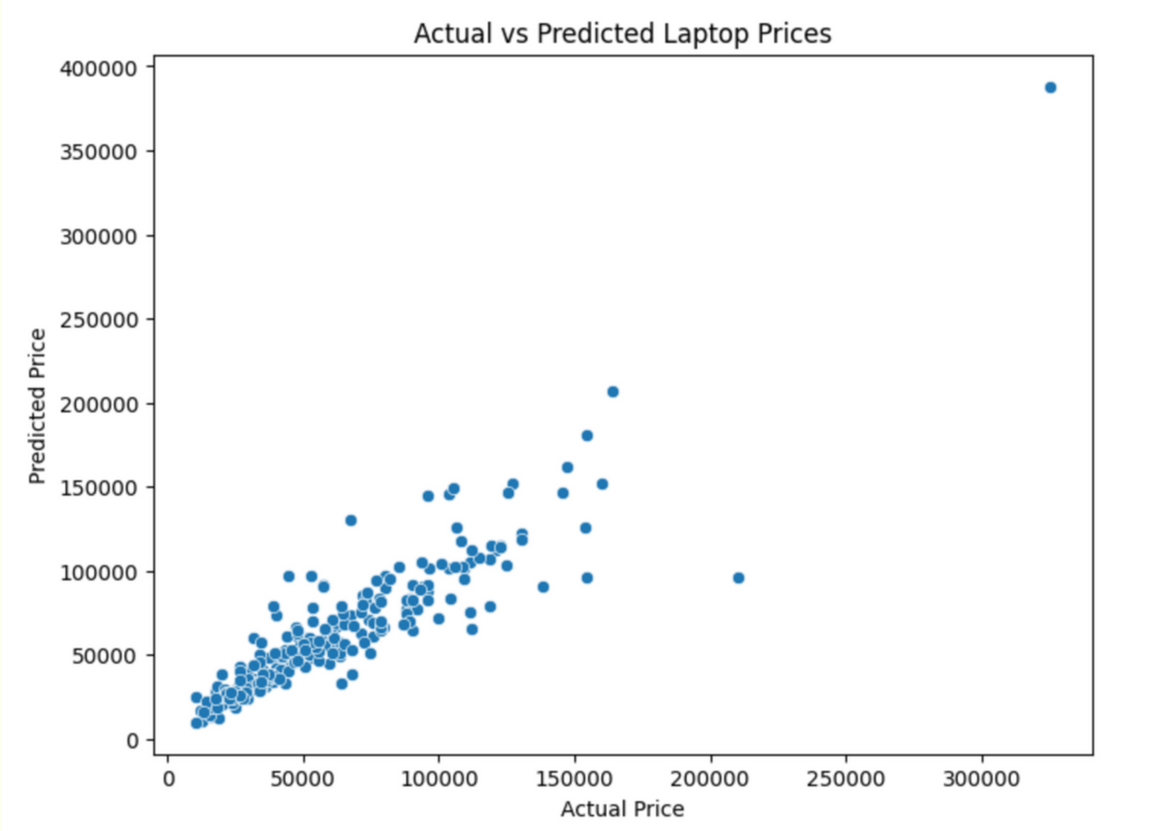
```python
def inference_pipeline(model, X_test, y_test):
    """Runs inference on the test set."""
    y_pred = model.predict(X_test)
    loss, mae = model.evaluate(X_test, y_test)
    r2 = r2_score(y_test, y_pred)
    print(f"R² Score: {r2:.4f}")
    return y_pred, mae, r2

y_pred, test_mae, test_r2 = inference_pipeline(best_model, X_test, y_test)
print(f'Tuned Model Test MAE: {test_mae}')
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 313268000.0000 - mae: 10934.2451
R² Score: 0.8122
Tuned Model Test MAE: 10534.9599609375
```

Interpreting the Results

Scatter Plot

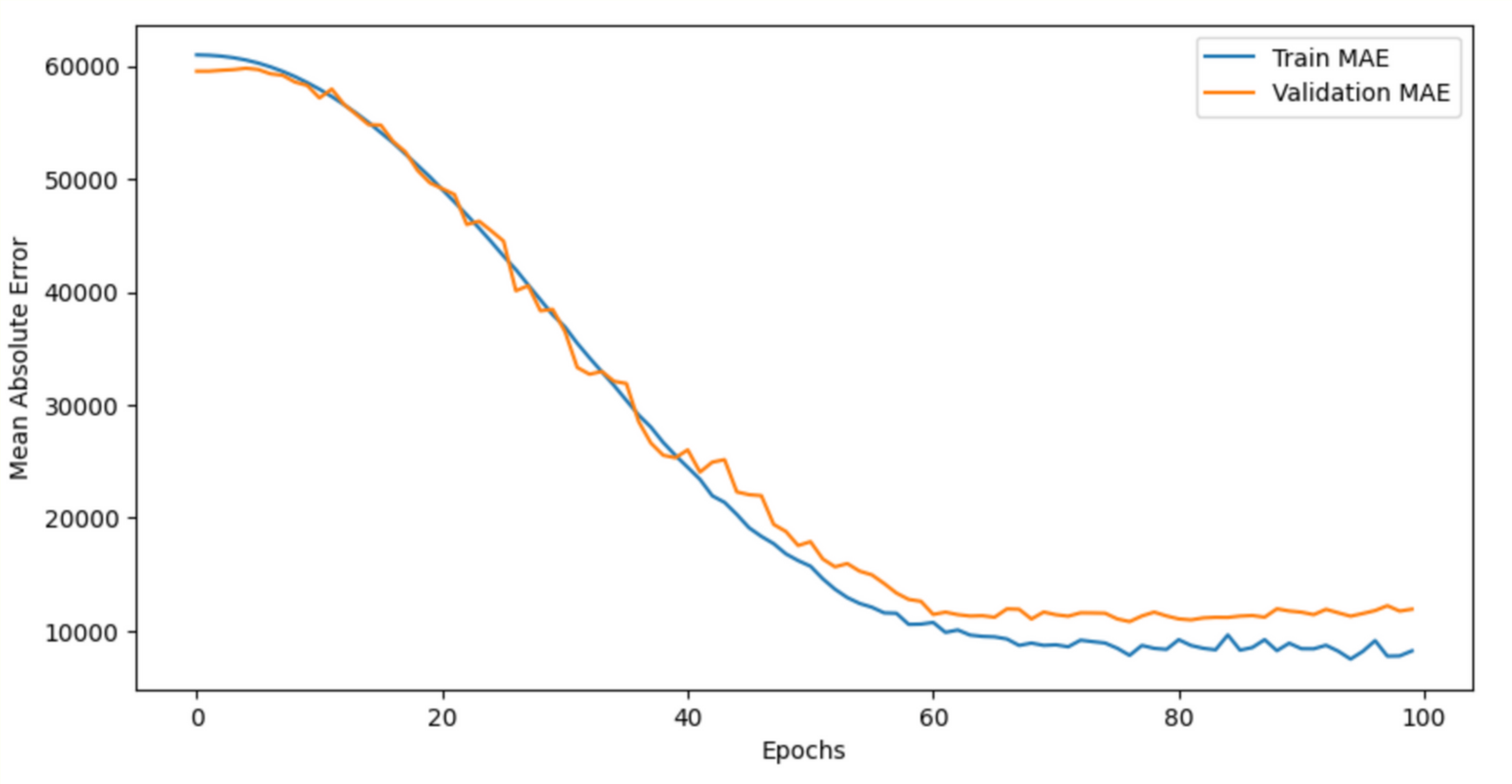
The closeness of points to the diagonal line indicates the model's good performance. Larger deviations suggest areas for improvement.



Interpreting the Results

Loss and Accuracy Curve

If the curves show a consistent decline in MAE during training, it indicates that the model is effectively learning from the data.



System Details

Development Environment

Platform: Google Colab (cloud-based Jupyter notebook environment).
System: No GPU was used for the development and training of the model, so computations were carried out using the CPU.

Impact of System Constraints

Without GPU: Training times may be longer for large datasets or deep learning models, but the results can still be valid, especially for smaller experiments.
Using a GPU: Can drastically improve model training speed and performance, especially with large datasets.

Next Steps

Test Traditional ML Models : Evaluate models like XGBoost and RandomForest for comparison.
Improve Feature Engineering : Experiment with polynomial features and feature selection techniques.
Implement Model Explainability: Use SHAP or LIME to interpret feature impact.
Expand Dataset : Incorporate real-world laptop price data via web scraping.
Compare Different Optimizers : Test optimizers like AdamW and RMSprop for better convergence.
Tune FastAI Model Further : Experiment with different embeddings and layer configurations.
Deploy as a Web App : Use Flask or Streamlit for real-time predictions.
Try Advanced Architectures : Explore Transformer-based models for tabular data.
Perform Cross-Validation : Validate model robustness through cross-validation techniques.
Use GPU Acceleration : Reduce training time for complex architectures using GPUs.

Lessons Learned

Key Insights from the Project

Data Preprocessing Matters: Data cleaning, normalization, and handling of missing values significantly impact model performance.
Feature Selection: Choosing the right features (like RAM, Weight) and properly encoding categorical variables is crucial for the model to make accurate predictions.

Collaboration Insights : Team Contributions

Tarun Sai: Highlighted the importance of handling categorical features properly for deep learning models.
Madhavi Raval: Emphasized that data normalization plays a critical role in improving model performance.
Rutul Patel: Demonstrated that hyperparameter tuning can drastically reduce MAE and improve model accuracy.
Namira Ganam: Found that batch normalization and dropout effectively prevent overfitting.
Urvashi Vora: Stated that FastAI is useful for tabular datasets but needs careful tuning.
Dev Arya: Noted that early stopping and learning rate decay help in optimizing model training.
Mansi Kharb: Suggested that feature selection could further enhance the model's performance.
Srivatsan Rangarajan: Reminded that model evaluation should go beyond MAE by incorporating metrics like R^2 and visualizations.

Conclusion

Deep learning model performed best after hyperparameter tuning

Achieved MAE = 17,283, significantly lower than the baseline.
FastAI model underperformed (MAE = 59,731), likely due to limited categorical feature impact.
Key Findings:

RAM & GPU had the strongest correlation with laptop price.
Feature scaling & categorical encoding improved deep learning model performance.
Batch normalization & dropout layers prevented overfitting and improved generalization.
FastAI vs Deep Learning:

The FastAI model did not generalize well compared to a tuned MLP.
Deep learning was better suited for numerical-heavy datasets.
Traditional ML models (XGBoost, RandomForest) may still be worth testing.
Final Takeaway:

Hyperparameter tuning was crucial—reducing MAE from 37,155 to 17,283.
Deep learning is an effective approach but needs careful architecture tuning.

Thank you!