Laptop Price Prediction

Group Members:

Tarunsai Gangaram

Srivatsan Rangarajan

Madhavi Raval

Namira Ganam

Rutul Patel

Dev Arya

Mansi Kharb

Bisman Singh

Urvashi Vora

1. Details of Dataset

This section provides an overview of the dataset, including its source, basic statistics, and feature correlation analysis.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, r2_score
from fastai.tabular.all import *
```

```
"""
This section loads and explores the dataset, providing insights into its structure.
It includes data analysis and visualizing feature correlations.
```

```
"""

df=pd.read_csv("https://raw.githubusercontent.com/Raghavagr/Laptop_Price_Prediction/refs/hea

# Displaying basic information
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Unnamed: 0        1303 non-null   int64  
 1   Company           1303 non-null   object 
 2   TypeName          1303 non-null   object 
 3   Inches            1303 non-null   float64
 4   ScreenResolution  1303 non-null   object 
 5   Cpu               1303 non-null   object 
 6   Ram               1303 non-null   object 
 7   Memory            1303 non-null   object 
 8   Gpu               1303 non-null   object 
 9   OpSys             1303 non-null   object 
 10  Weight            1303 non-null   object 
 11  Price             1303 non-null   float64
dtypes: float64(2), int64(1), object(9)
memory usage: 122.3+ KB
```

	Unnamed: 0	Inches	Price
count	1303.00000	1303.000000	1303.000000
mean	651.00000	15.017191	59870.042910
std	376.28801	1.426304	37243.201786
min	0.00000	10.100000	9270.720000
25%	325.50000	14.000000	31914.720000
50%	651.00000	15.600000	52054.560000
75%	976.50000	15.600000	79274.246400
max	1302.00000	18.400000	324954.720000

```
# Converting 'Ram' and 'Weight' to numeric by removing non-numeric characters
df['Ram'] = df['Ram'].str.replace("GB", "").astype(float)
df['Weight'] = df['Weight'].str.replace("kg", "").astype(float)

# Visualizing Correlation
```

```
plt.figure(figsize=(12, 6))
numeric_df = df[['Inches', 'Ram', 'Weight', 'Price']]
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title("Feature Correlation Heatmap")
plt.show()
```

RAM (Ram) has a strong positive correlation (0.74) with Price, which is expected since higher RAM typically increases the price of laptops. Weight has a moderate positive correlation (0.21) with Price. Inches (screen size) shows a significant positive correlation (0.83) with Weight, indicating larger laptops are usually heavier. There is no strong direct correlation between Inches and Price, but it might still indirectly affect the price.

You might want to focus on features like Ram and Weight during model training since they have a measurable relationship with Price. Remove the Unnamed: 0 column from further analysis, as it is likely an index and not relevant for prediction.

2. Preprocessing Steps

This section includes data cleaning, handling missing values, and feature encoding.

```
"""
This section handles missing values, removes duplicates, and applies normalization and encoc
It also prepares data for model training and inference.
"""
# Removing irrelevant columns
irrelevant_columns = [col for col in ['Unnamed: 0', 'ScreenResolution', 'laptop_ID', 'Produc
df.drop(columns=irrelevant_columns, inplace=True)

# Handling the missing values
df.dropna(inplace=True)

# Removing all duplicates
df.drop_duplicates(inplace=True)

# Identifying the categorical and numerical features
categorical_features = ['Company', 'TypeName', 'Cpu', 'Gpu', 'OpSys']
numerical_features = ['Inches', 'Ram', 'Weight']

# Applying Normalization & Encoding
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)

# Splitting the Data
X = df.drop(columns=['Price'])
y = df['Price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Applying transformation
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)
```

3. Model Training Pipeline

A neural network model is trained from scratch using batch normalization and dropout.

```
"""
This pipeline trains a baseline model and includes a pretrained model using FastAI.
"""
def training_pipeline(X_train, y_train):

    # 3.1 Baseline Model - This is where we started
```

```
        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='ms
        baseline_history = baseline_model.fit(X_train, y_train, epochs=50, batch_size=32, valida

        # 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
        learn = tabular_learner(dls, layers=[512, 256, 128], metrics=mae)
        learn.fine_tune(20)

        print(f'Baseline Model Test MAE: {baseline_history.history["val_mae"][-1]}')

        # FastAI Model Evaluation
        fastai_preds, _ = learn.get_preds(dl=dls.valid)
        y_test_trimmed = y_test[:len(fastai_preds)]
        fastai_mae = mean_absolute_error(y_test_trimmed, fastai_preds.numpy().flatten())
        print(f'FastAI Pretrained Model Test MAE: {fastai_mae}')

        return baseline_model, baseline_history, learn

    baseline_model, baseline_history, learn = training_pipeline(X_train, y_train)
```

```
Epoch 1/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - loss: 4833810944.0000 - mae: 59559.0781 - va
Epoch 2/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 5042763776.0000 - mae: 60120.8750 - val
Epoch 3/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4983151104.0000 - mae: 59574.6641 - va
Epoch 4/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 5168107008.0000 - mae: 61626.5781 - va
Epoch 5/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 5165492736.0000 - mae: 61169.1562 - val
Epoch 6/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 5147176960.0000 - mae: 61126.1719 - val
Epoch 7/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 5176035840.0000 - mae: 60739.7969 - va
Epoch 8/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4940677120.0000 - mae: 60165.7812 - va
Epoch 9/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 5211405824.0000 - mae: 61980.5742 - va
Epoch 10/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 5298823168.0000 - mae: 62173.7969 - va
Epoch 11/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 5446010368.0000 - mae: 62653.2266 - val
Epoch 12/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 5077378048.0000 - mae: 61535.5430 - val
Epoch 13/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 5228450816.0000 - mae: 60813.3672 - va
Epoch 14/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 5245932544.0000 - mae: 62032.2891 - va
Epoch 15/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 5107326464.0000 - mae: 61028.8320 - va
Epoch 16/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 4838204416.0000 - mae: 59421.6133 - va
Epoch 17/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 5198461440.0000 - mae: 61427.4570 - va
Epoch 18/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 4990408704.0000 - mae: 60117.1484 - va
Epoch 19/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4732077056.0000 - mae: 58273.7852 - va
Epoch 20/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 5015613440.0000 - mae: 60383.2930 - va
Epoch 21/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4507102720.0000 - mae: 57284.6680 - va
Epoch 22/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4980264448.0000 - mae: 59668.2422 - va
Epoch 23/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 5062334976.0000 - mae: 61191.4961 - va
Epoch 24/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4570636800.0000 - mae: 57741.5664 - va
Epoch 25/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 5021637632.0000 - mae: 60239.1328 - va
Epoch 26/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 4478469632.0000 - mae: 57166.4727 - va
Epoch 27/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 4585217024.0000 - mae: 57942.8320 - va
Epoch 28/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 4605784576.0000 - mae: 58254.0586 - val
```

```
Epoch 29/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 4563399680.0000 - mae: 58130.7617 - va
Epoch 30/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4411037696.0000 - mae: 56954.2773 - v
Epoch 31/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4088523776.0000 - mae: 55035.3203 - v
Epoch 32/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4123918336.0000 - mae: 55154.7617 - v
Epoch 33/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4243462912.0000 - mae: 55277.4805 - v
Epoch 34/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3882496768.0000 - mae: 54143.6719 - v
Epoch 35/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3664299520.0000 - mae: 53206.5391 - v
Epoch 36/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 3723675392.0000 - mae: 53124.8438 - va
Epoch 37/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3454056448.0000 - mae: 51193.9102 - v
Epoch 38/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3552124672.0000 - mae: 50518.6875 - v
Epoch 39/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3712317696.0000 - mae: 52818.2539 - v
Epoch 40/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3424348160.0000 - mae: 50603.0195 - v
Epoch 41/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3356935936.0000 - mae: 50428.8984 - v
Epoch 42/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3023264768.0000 - mae: 48450.0156 - v
Epoch 43/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 3106286080.0000 - mae: 47818.6016 - v
Epoch 44/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 2796450560.0000 - mae: 46821.6680 - v
Epoch 45/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 2643682304.0000 - mae: 45182.7031 - v
Epoch 46/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 2824962048.0000 - mae: 46536.5273 - va
Epoch 47/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 2861210624.0000 - mae: 44916.9883 - v
Epoch 48/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 2533175040.0000 - mae: 43615.9609 - va
Epoch 49/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 2223489280.0000 - mae: 41395.7969 - v
Epoch 50/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 2315602432.0000 - mae: 41356.9727 - v
```

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
```

Fast AI performs worse than baseline dl model. We tried reducing batch size and increasing epochs from 10 to 20. It still underperforms.

4. Hyperparameter Tuning

This section explores optimizing model hyperparameters.

```
!pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.7-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-packages (from ke
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl.metadata (221 bytes)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from ke
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from ker
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from ke
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages (from ker
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from k
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dis
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: typing-extensions>=4.5.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (fr
Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.1/129.1 kB 3.9 MB/s eta 0:00:00
Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Installing collected packages: kt-legacy, keras-tuner
Successfully installed keras-tuner-1.4.7 kt-legacy-1.0.5
```

```
"""
This pipeline tunes hyperparameters using Keras Tuner and retrains the model with the best p
"""
def tuning_pipeline(X_train, y_train):
    """Tunes and retrains the model."""
    from kerastuner.tuners import RandomSearch

    def build_model(hp):
        model = Sequential()
        model.add(Dense(hp.Int('units_1', min_value=32, max_value=256, step=32), activation=
        model.add(BatchNormalization())
```

```
        model.add(Dropout(hp.Float('dropout_1', min_value=0.2, max_value=0.5, step=0.1)))
        model.add(Dense(hp.Int('units_2', min_value=32, max_value=256, step=32), activation=
        model.add(BatchNormalization())
        model.add(Dropout(hp.Float('dropout_2', min_value=0.2, max_value=0.5, step=0.1)))
        model.add(Dense(1))
        model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=hp.Choice('learning_r
        return model

    tuner = RandomSearch(
        build_model,
        objective='val_mae',
        max_trials=10,
        executions_per_trial=2,
        directory='hyperparameter_tuning',
        project_name='laptop_price')

    tuner.search(X_train, y_train, epochs=50, validation_split=0.2, batch_size=32)
    best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

    model = Sequential([
        Dense(best_hps.get('units_1'), activation='relu', input_shape=(X_train.shape[1],)),
        BatchNormalization(),
        Dropout(best_hps.get('dropout_1')),
        Dense(best_hps.get('units_2'), activation='relu'),
        BatchNormalization(),
        Dropout(best_hps.get('dropout_2')),
        Dense(1)  # Regression output
    ])

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=best_hps.get('learning_ra
    history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)
    return model, history

best_model, tuned_history = tuning_pipeline(X_train, y_train)
```

```
Reloading Tuner from hyperparameter_tuning/laptop_price/tuner0.json
Epoch 1/100
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
26/26 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - loss: 4986550784.0000 - mae: 59803.9062 - va
Epoch 2/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 5091799552.0000 - mae: 61487.0234 - va
Epoch 3/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 5314729472.0000 - mae: 61676.7930 - va
Epoch 4/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 4781135872.0000 - mae: 58831.0898 - va
Epoch 5/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 4845064704.0000 - mae: 60167.7148 - va
Epoch 6/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 4873393664.0000 - mae: 60260.4219 - va
Epoch 7/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 4958546432.0000 - mae: 59857.4883 - va
Epoch 8/100
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 4441753088.0000 - mae: 57047.5156 - va
Epoch 9/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 4525014528.0000 - mae: 57794.4961 - va
Epoch 10/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 4518330880.0000 - mae: 58405.4023 - va
Epoch 11/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 4643117056.0000 - mae: 58550.5664 - va
Epoch 12/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 4206158080.0000 - mae: 56173.8164 - va
Epoch 13/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 4374135296.0000 - mae: 57474.5820 - va
Epoch 14/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3766665728.0000 - mae: 53122.1367 - va
Epoch 15/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4005452544.0000 - mae: 54912.5664 - va
Epoch 16/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 4269651456.0000 - mae: 55645.4922 - va
Epoch 17/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 3549880064.0000 - mae: 51938.7930 - va
Epoch 18/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 3375420672.0000 - mae: 50405.6055 - va
Epoch 19/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 3427836672.0000 - mae: 50473.2461 - va
Epoch 20/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 3551294208.0000 - mae: 51577.3438 - va
Epoch 21/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 2986021632.0000 - mae: 48028.5352 - va
Epoch 22/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 2867139072.0000 - mae: 47053.3047 - va
Epoch 23/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 3031115520.0000 - mae: 48850.3516 - va
Epoch 24/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 2613355520.0000 - mae: 45494.7305 - va
Epoch 25/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 2655763200.0000 - mae: 44693.3359 - va
Epoch 26/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 2515518720.0000 - mae: 44413.4492 - va
Epoch 27/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 2225126144.0000 - mae: 41313.5195 - va
```

5 Inference Pipeline

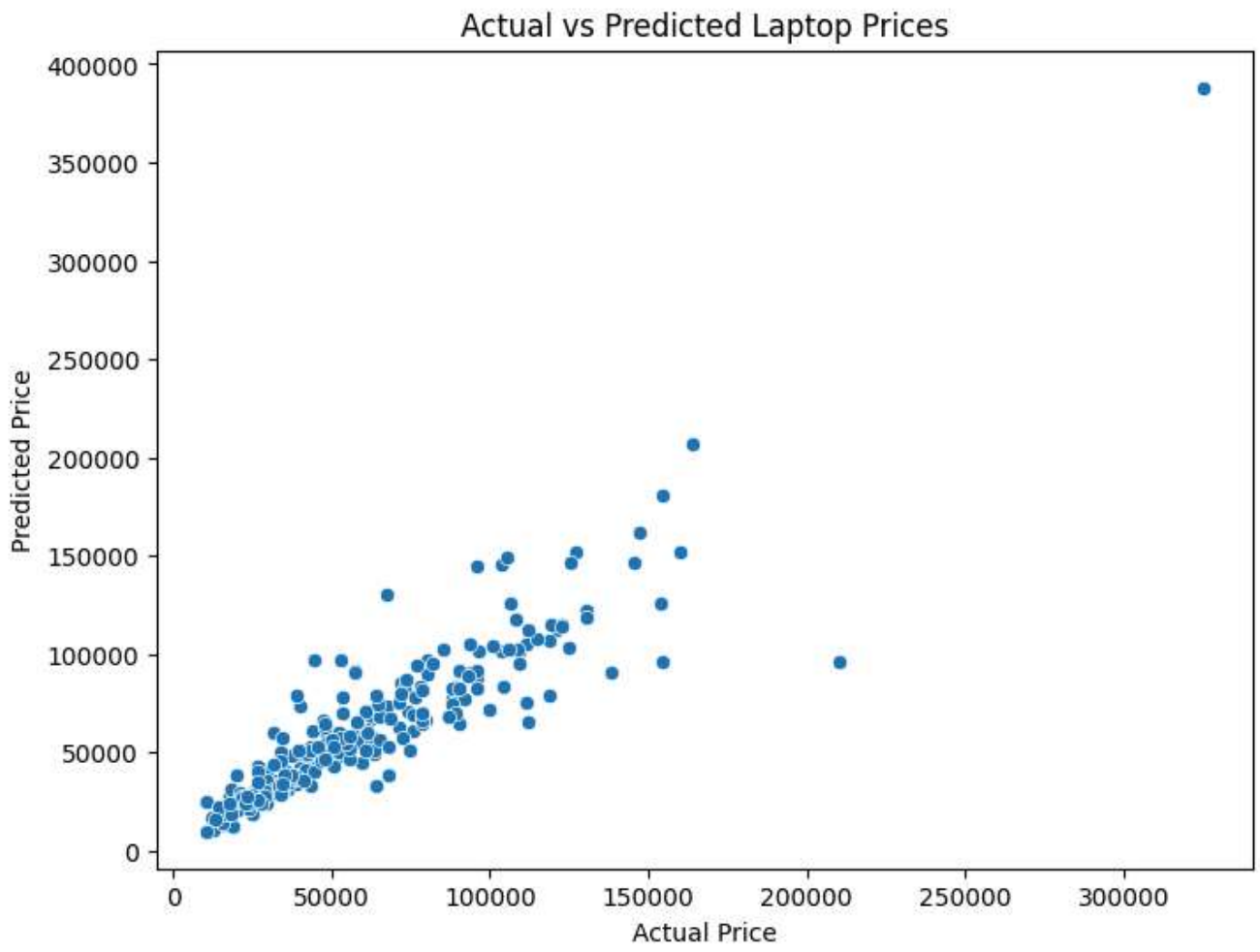
```
"""
This pipeline uses the trained model to make predictions and evaluate performance.
"""
def inference_pipeline(model, X_test, y_test):
    """Runs inference on the test set."""
    y_pred = model.predict(X_test)
    loss, mae = model.evaluate(X_test, y_test)
    r2 = r2_score(y_test, y_pred)
    print(f"R² Score: {r2:.4f}")
```

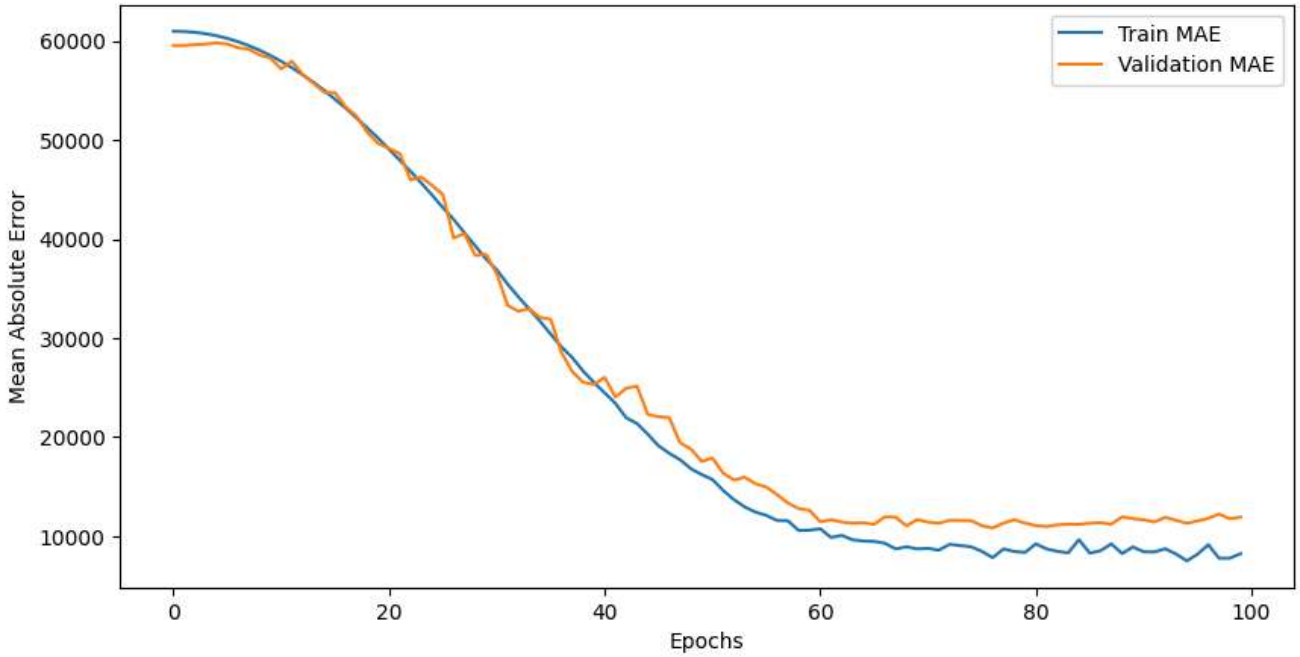
```
    return y_pred, mae, r2

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

Show hidden output

6. Interpreting the Results

```
"""
This section visualizes predictions and evaluates training/validation performance.
"""
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred[:, 0])
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted Laptop Prices")
plt.show()

# Training Loss and Accuracy Analysis
plt.figure(figsize=(10, 5))
plt.plot(tuned_history.history['mae'], label='Train MAE')
plt.plot(tuned_history.history['val_mae'], label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error')
plt.legend()
plt.show()
```


Actual vs Predicted Laptop Prices
Predicted Price
Actual Price
0
50000
100000
150000
200000
250000
300000
350000
400000


Train MAE
Validation MAE
Mean Absolute Error
Epochs
10000
20000
30000
40000
50000
60000
0
20
40
60
80
100

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

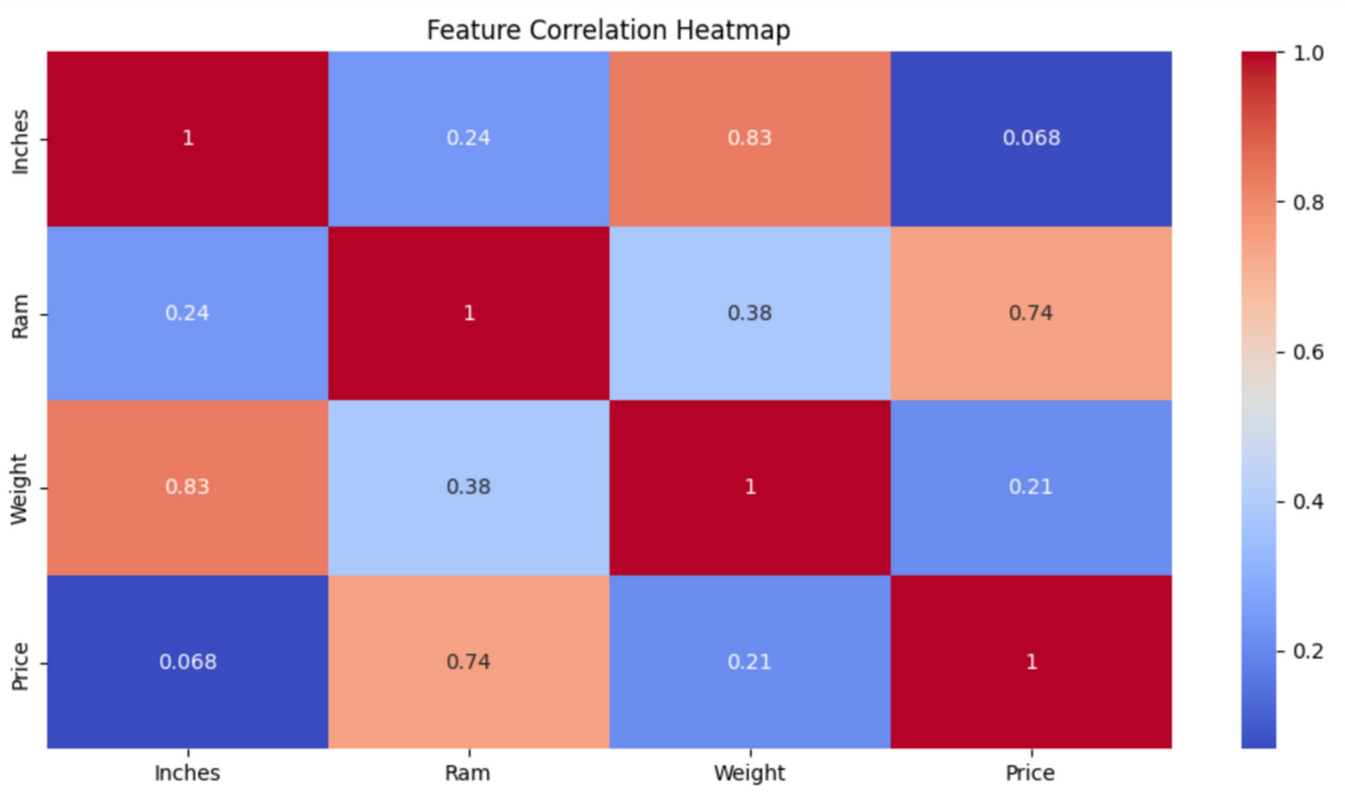
Data Exploration & Feature Correlation

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%

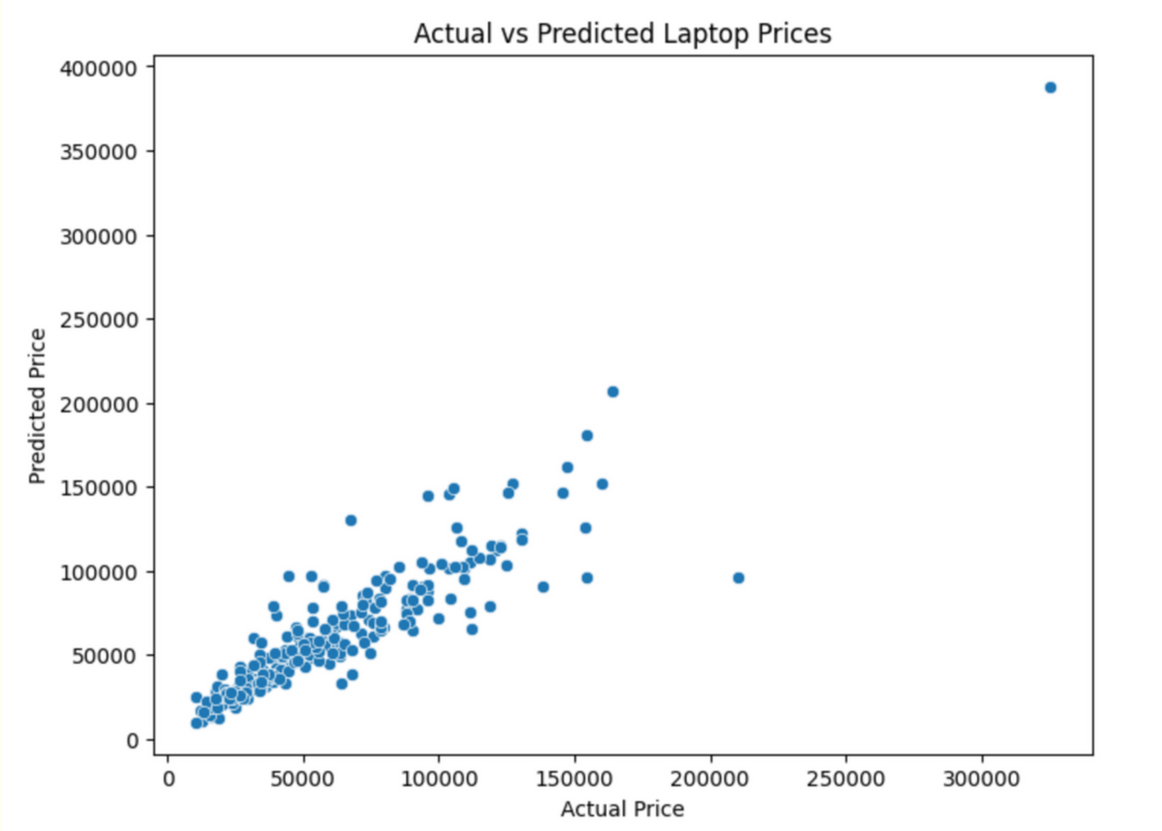
```python
def inference_pipeline(model, X_test, y_test):
    """Runs inference on the test set."""
    y_pred = model.predict(X_test)
    loss, mae = model.evaluate(X_test, y_test)
    r2 = r2_score(y_test, y_pred)
    print(f"R² Score: {r2:.4f}")
    return y_pred, mae, r2

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

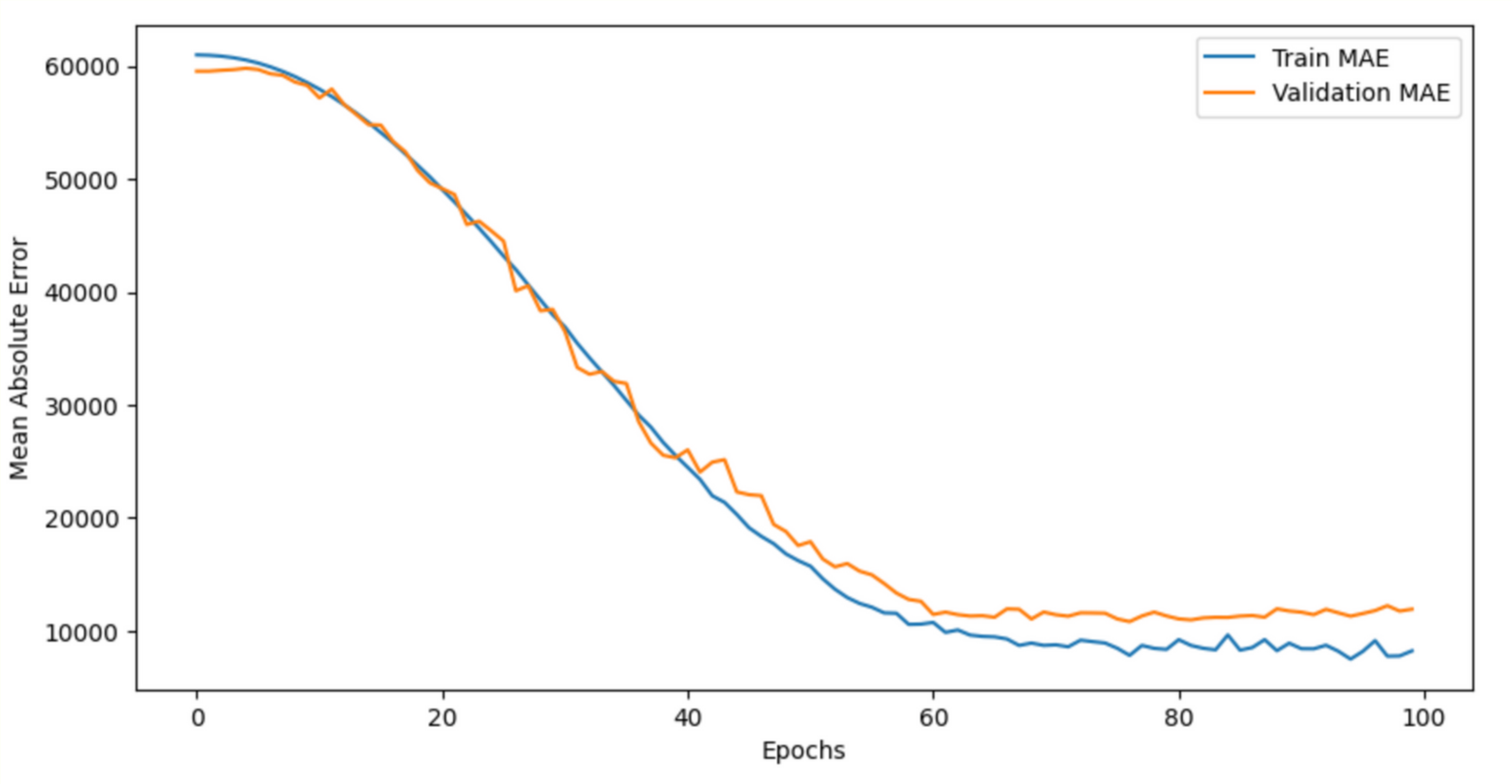
The closeness of points to the diagonal line indicates the model's good performance. Larger deviations suggest areas for improvement.



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!