Part A: Data Preprocessing and Exploratory Data Analysis

1. Dataset Structure

The training dataset consists of **41,348 samples** and 6 input features, along with 1 target variable (price_class).

The features can be categorized as follows:

Categorical Features

- neighbourhood_group: Represents the borough of the Airbnb listing
- room_type: Type of accommodation offered

Both features are stored as object data types.

Numerical Features

- minimum_nights (float64): Minimum number of nights required for booking
- amenity_score (float64): Listing quality and amenity score
- number_of_reviews (float64): Total number of reviews received
- availability_365 (float64): Number of available days in the next year

---

Target Variable

- price_class (int64): Categorical variable with four classes:

Class	Meaning
0	**Budget**
1	**Moderate**
2	**Premium**
3	**Luxury**

This confirms that the problem is a **multi-class classification** task.

---

2. Missing Values Analysis

Missing value analysis revealed that several features contained missing values.

The number of missing values in each feature before imputation is shown below:

Feature	Missing Values
neighbourhood_group	839
room_type	611
minimum_nights	1322
amenity_score	916
number_of_reviews	1123
availability_365	595
price_class	0

Since missing values were present in both categorical and numerical features, appropriate imputation strategies were applied.

---

3. Missing Value Handling Strategy

The following imputation methods were used:

Categorical Features

Missing values in:

- neighbourhood_group
- room_type

were filled using the mode (most frequent category).

Justification:

Mode imputation preserves the most common category and does not introduce unrealistic or artificial categories.

Numerical Features

Missing values in:

- **minimum_nights**
- **amenity_score**
- **number_of_reviews**
- **availability_365**

were filled using the median value.

Justification:

Median was preferred over mean because it is robust to outliers and better represents the central tendency of skewed distributions, which are common in real-world tabular data.

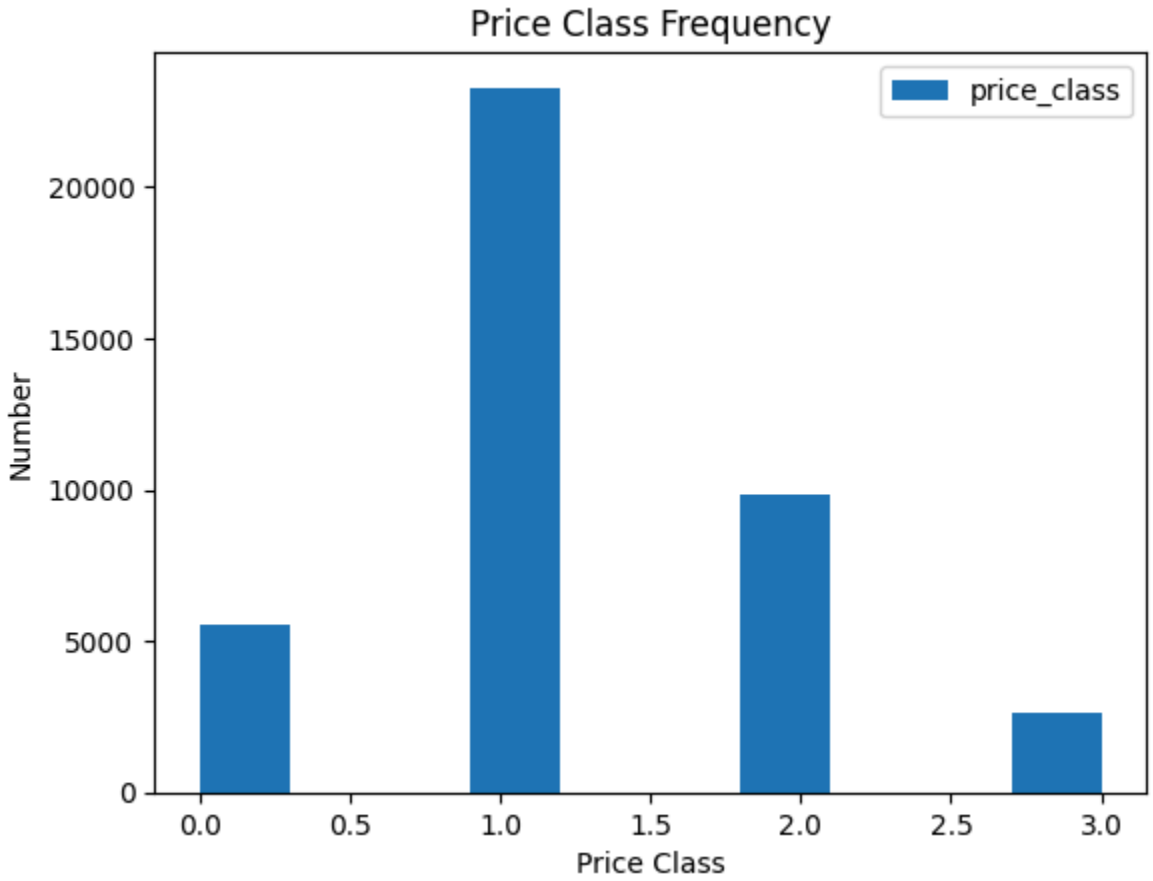
Final Verification

After imputation, no missing values remained in the dataset.

---

4. Class Distribution Analysis

The class distribution of the target variable (price_class) was analyzed using a bar chart (Figure X).



)

The dataset contains four price categories:

- **Budget**
- **Moderate**
- **Premium**
- **Luxury**

Observation

The dataset shows class imbalance, where some price classes have significantly more samples than others. Moderate class has the maximum samples while Luxury has the least samples.

This imbalance may cause the neural network to become biased toward predicting majority classes more frequently, potentially reducing performance on under-represented classes such as Luxury listings.

Class imbalance is an important factor that can influence model generalization.

---

5. Encoding of Categorical Variables

The categorical variables:

- neighbourhood_group
- room_type

were encoded using One-Hot Encoding.

Justification

One-hot encoding was chosen because categorical features do not have any natural ordinal relationship.

Using label encoding would introduce artificial numerical ordering, which could mislead the neural network during training.

One-hot encoding allows the model to treat each category independently without introducing false relationships.

6.Feature Standardization

All numerical features were standardized before training the neural network.

Standardization transforms each feature using the following formula using Standard Scaler from Scikit learn.

Justification

Standardization was chosen because neural networks are sensitive to the scale of input features.

Without standardization:

- Features with larger numerical ranges would dominate gradient updates
- Optimization would become unstable
- Training would be slower

Standardization improves:

- Gradient stability
- Convergence speed
- Overall model performance

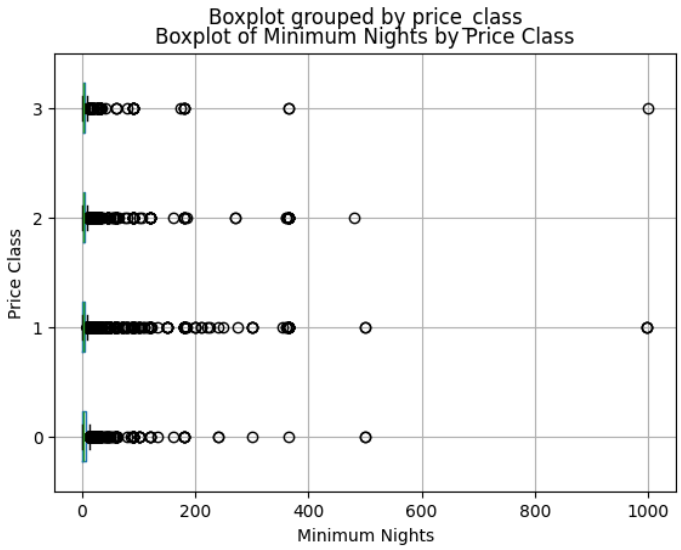
Why not normalization?

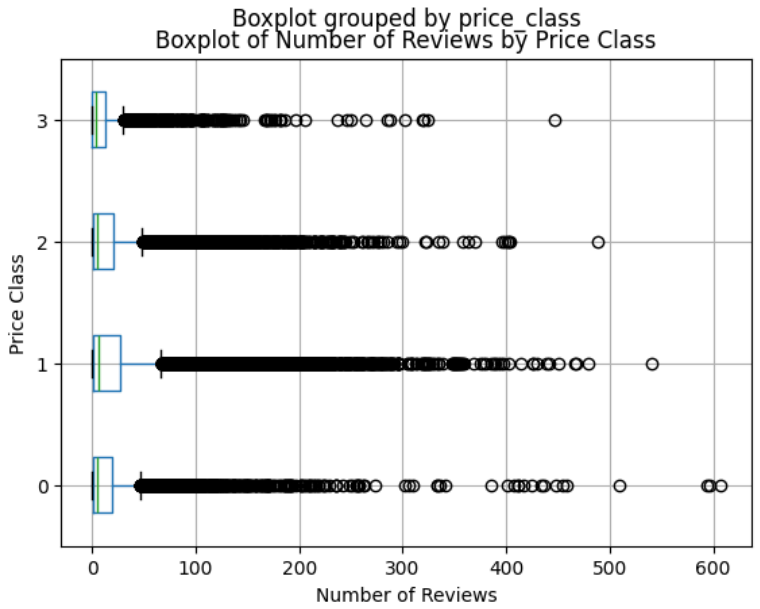
Standardization was preferred over min–max normalization because it preserves the effect of outliers while still ensuring consistent feature scale, which is suitable for gradient-based optimization.

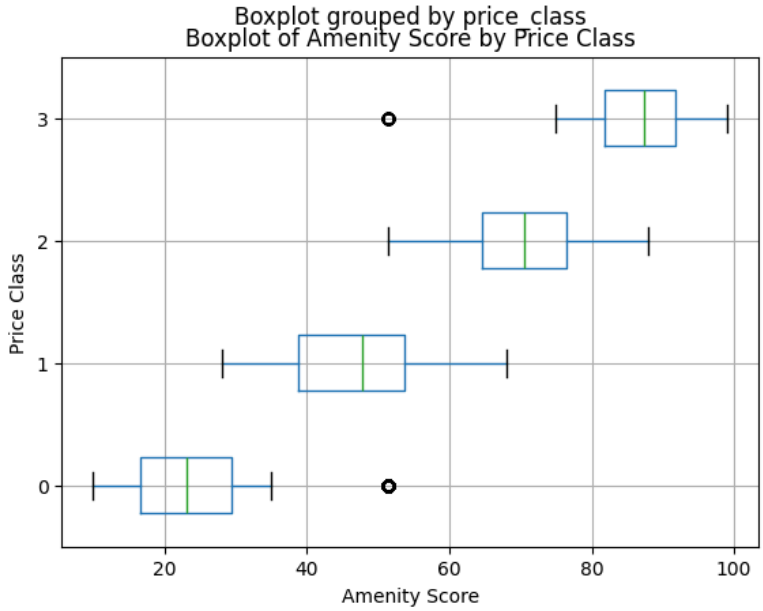
---

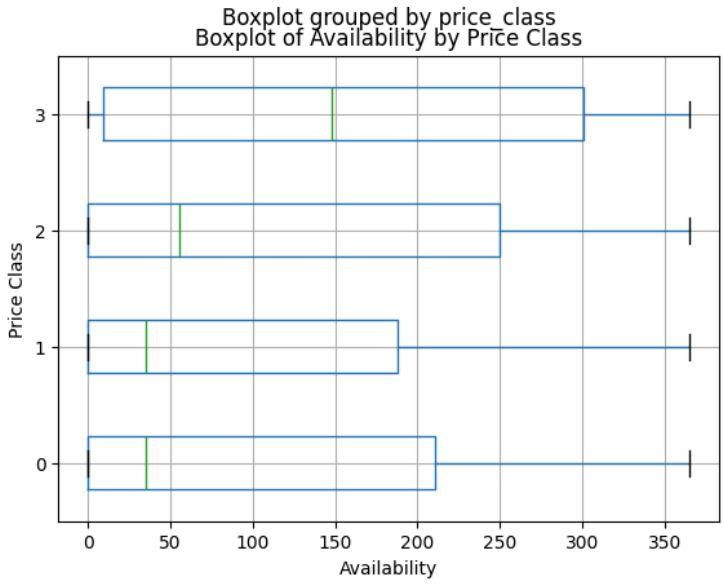
7. Feature vs Target Relationship Analysis

Feature distributions were analyzed with respect to price_class using boxplots.




Boxplot grouped by price_class
Boxplot of Number of Reviews by Price Class
Price Class
3
2
1
0
0
100
200
300
400
500
600
Number of Reviews


Boxplot grouped by price_class
Boxplot of Amenity Score by Price Class
Price Class
3
2
1
0
20
40
60
80
100
Amenity Score



Key Observations

Amenity score showed strong separation between different price classes.

Higher price classes generally had higher amenity scores, indicating that amenity_score is likely an important predictive feature.

Minimum nights showed moderate variation across price classes.

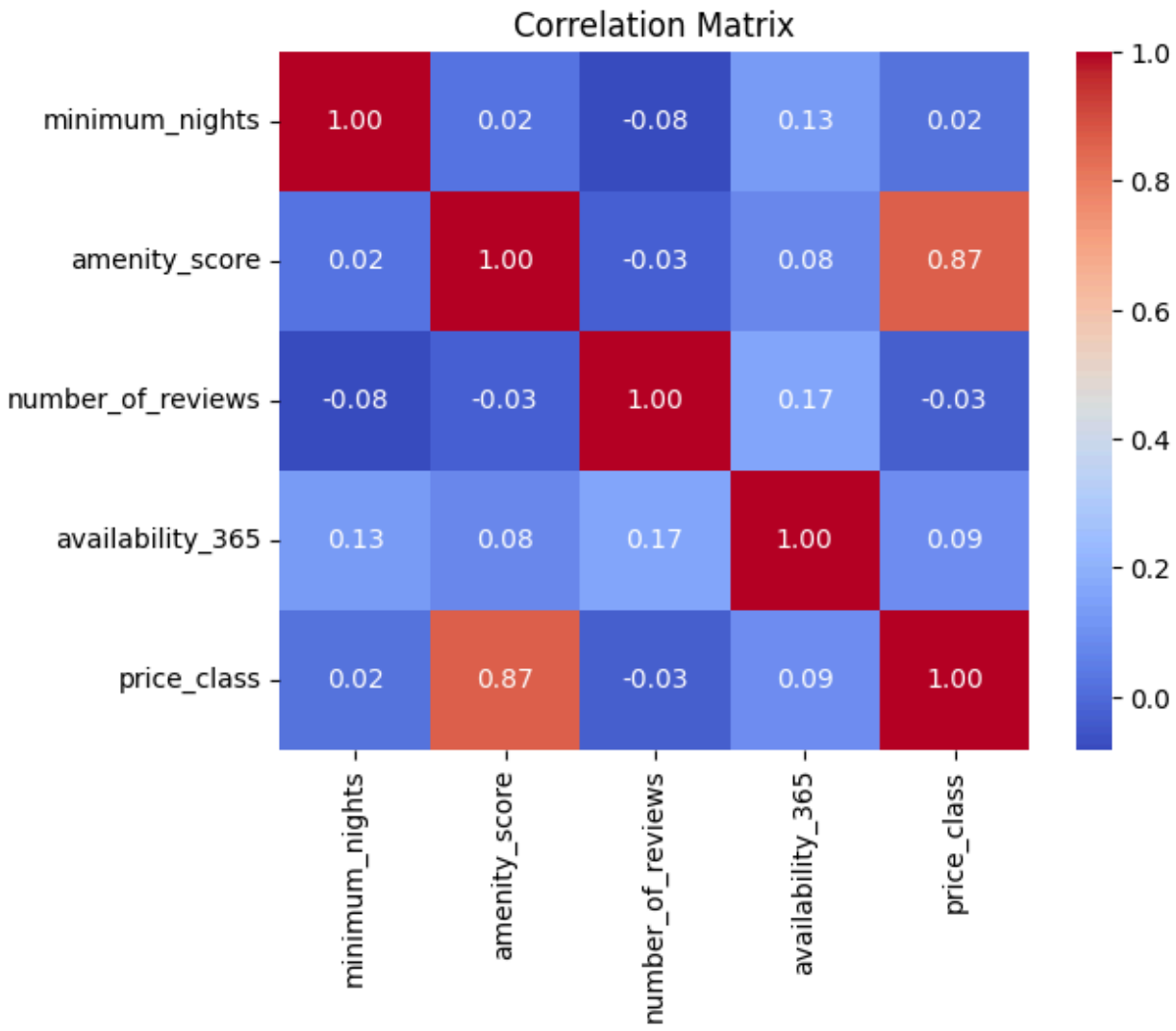
Number of reviews and availability showed weaker separation, indicating relatively lower predictive strength.

These observations suggest that amenity_score is likely to be one of the most influential features for prediction.

---

8. Correlation Analysis

A correlation matrix was computed to analyze relationships between numerical features.



Observations

Most features showed low to moderate correlation with each other.

No extremely high correlations were observed except price_class and amenity_score. It indicates that features provide relatively independent information.

This is beneficial for neural network training because redundant features can negatively affect learning.

---

9. Most Influential Features

Based on exploratory analysis, the most influential feature is:

Amenity Score

This conclusion is supported by:

- Clear separation across price classes
- Logical relationship with listing quality
- Strong visual evidence in boxplots

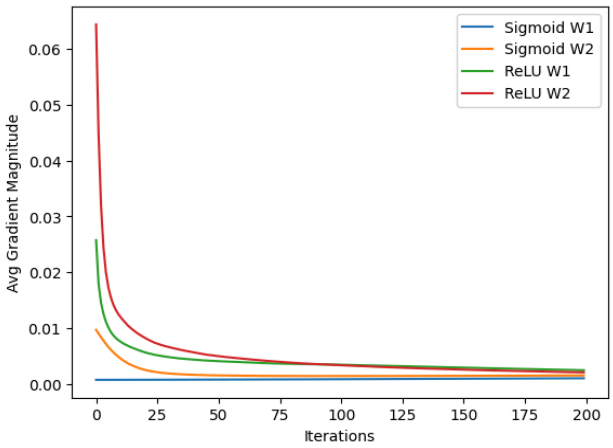
---

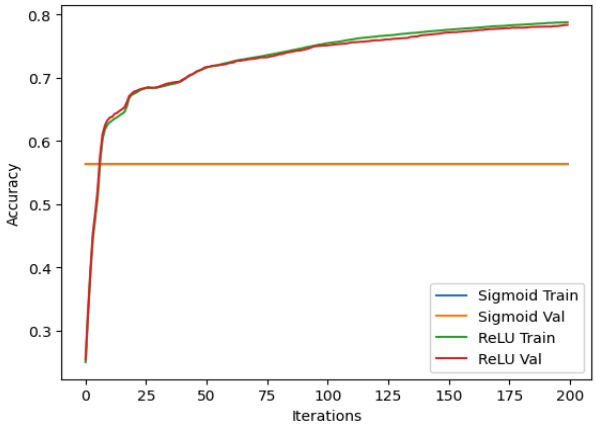
10. Suspicious or Dominant Features

Amenity score appears highly predictive.

While this is expected because higher quality listings tend to be more expensive, it may cause the model to rely heavily on this single feature.

This could potentially reduce generalization if the relationship changes in the test data.





Part B(a): Two-Layer Perceptron Implemented from Scratch

Model Architecture and Training Setup

A fully connected **feed-forward neural network** with **two hidden layers** was implemented using only **NumPy** operations. No automatic differentiation libraries were used; all gradients were derived manually via backpropagation equations discussed in class.

Network Architecture:

`Input → Hidden Layer 1 → Hidden Layer 2 → Output Layer`

- **Hidden Layers:** Contain learnable weights W1,W2W_1, W_2W1,W2 and biases.
- **Output Layer:** Uses **softmax activation** for multi-class classification.
- **Loss Function:** Cross-Entropy Loss
- **Optimization:** Vanilla Batch Gradient Descent
- **Training Iterations:** 200

Two activation functions were evaluated: **Sigmoid** and **ReLU**.

Accuracy Comparison Across Iterations

Training and validation accuracy curves for both activations are shown in **Figure 1**.

Observations:

- **ReLU:**
    - Converges faster during early iterations.
    - Accuracy rises sharply within the first 20–30 iterations due to efficient gradient propagation.
- **Sigmoid:**
    - Improves more gradually with a smoother learning curve.

- Early training shows a flat validation line due to weak gradients and saturation effects.

Final Performance:

Activation	Training Accuracy	Validation Accuracy
ReLU	≈ 0.79	≈ 0.78–0.79
Sigmoid	Slightly lower but comparable	Similar to training

Validation closely follows training, indicating **stable optimization** and **no severe overfitting**.

Effect of Activation Function on Optimization

Sigmoid Activation:

- Derivative bounded by 0.25 → **small gradients**.
- Saturation for large positive/negative inputs → **vanishing gradients**, slowing learning in early layers.

ReLU Activation:

- Gradients remain unsaturated for positive activations.
- Enables **stronger weight updates**, faster convergence, and better scaling to deeper networks.

Part B(b): Gradient Magnitude Comparison Across Layers

To examine gradient flow, the **average magnitude of gradients** for:

- **First hidden layer weights W1W_1W1**
- **Second hidden layer weights W2W_2W2**

was tracked across training iterations (**Figure 2**).

Observed Gradient Behavior:

1. **Sigmoid Network:**
    - Gradients in Layer 1 are extremely small throughout training.
    - Gradient magnitude decays sharply from Layer 2 to Layer 1
    - Confirms the **vanishing gradient phenomenon**, limiting the representational power of early layers.
2. **ReLU Network:**
    - Gradients are significantly larger, especially during early iterations.
    - Better balance between Layer 1 and Layer 2 gradients.
    - Gradients decay smoothly instead of collapsing, supporting **faster and more stable learning**.

Conclusion:
The experiments demonstrate that **ReLU activation** provides more stable and efficient gradient flow compared to Sigmoid. While both activations achieve similar final accuracy, ReLU converges faster and avoids vanishing gradients, making it preferable for deeper networks.

PART C (a)

LUMS

Part C (a)

1- Forward Pass :-

$$z_1 = W_1 x + b_1$$
$$a_1 = g(z_1)$$
$$z_2 = W_2 a_1 + b_2$$
$$a_2 = g(z_2)$$
$$z_3 = W_3 a_2 + b_3$$
$$\hat{y} = \text{softmax}(z_3)$$
$$L = \text{CrossEntropy}(\hat{y}, y)$$

2- Standard Back Propagation :-

$$\delta_3 = \partial L / \partial z_3$$
$$\delta_2 = W_3^T \delta_3 \odot g'(z_2)$$
$$\delta_1 = W_2^T \delta_2 \odot g'(z_1)$$

- Extend Back Propagation to I/P

$$z_1 = W_1 x + b_1$$
$$\partial L / \partial x = \partial L / \partial z_1 \cdot \partial z_1 / \partial x$$
$$\partial L / \partial z_1 = \delta_1$$
$$\partial z_1 / \partial x = W_1$$

$$\boxed{\frac{\partial L}{\partial x} = W_1^T \delta_1}$$

4- Expanded Expression:

$$\frac{\partial L}{\partial x} = W_1^T \left[(W_2^T (W_3^T \delta \odot g'(z_2)) \odot g'(z_1) \right]$$

5- Feature Attribution Definition:

$$I_i = \left| \frac{\partial L}{\partial x_i} \right|$$

Average over dataset

$$\text{Importance}(i) = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{\partial L^{(n)}}{\partial x_i} \right|$$

The gradient $\partial L / \partial x_i$ measures how sensitive the loss is to small changes in feature x_i.

If a small change in a feature causes a large change in loss, the model relies heavily on that feature

Therefore the gradient magnitude can be used to rank feature importance by measuring each I/P's influence on Prediction.

PART C (b)

Rank	Feature	Importance
1	amenity_score	0.000021
2	room_type_Entire home/apt	0.000007
3	neighbourhood_group_Staten Island	0.000005
4	availability_365	0.000005
5	neighbourhood_group_Bronx	0.000005
6	neighbourhood_group_Manhattan	0.000005
7	room_type_Private room	0.000004
8	neighbourhood_group_Brooklyn	0.000004
9	room_type_Shared room	0.000003
10	number_of_reviews	0.000003

11	**neighbourhood_group_Queens**	**0.000003**
12	**minimum_nights**	**0.000002**

Key Observations

1. Amenity Score is the Most Influential Feature
   It has the highest gradient magnitude, meaning the model's predictions are most sensitive to changes in listing quality.
   Small variations in amenities significantly affect the predicted price class.

2. Room Type Plays a Strong Secondary Role
   Features such as *Entire home/apt* and *Private room* rank highly, indicating that the type of accommodation strongly influences pricing decisions.

3. Location Features Have Moderate Impact
   Encoded neighbourhood variables contribute, but none dominate individually.
   This suggests that while location matters, it is less decisive than property characteristics.

4. Minimum Nights and Review Count Have Low Influence
   These features produced the smallest gradients, meaning they have limited effect on the model's classification decision.

Conclusion

The gradient-based feature attribution method successfully identifies which inputs drive the neural network's predictions.
The model relies primarily on **amenity-related and property-type features**, while temporal or review-based variables contribute less.
This demonstrates how backpropagation can be repurposed not only for learning but also for interpreting model behavior.

PART D

The model achieved a **training accuracy of 80.25%** and a **validation accuracy of 79.44%**, indicating stable learning and no clear signs of classical overfitting. However, **the test accuracy dropped significantly to 44.22%**, showing a major generalization failure. Since training and validation performances are very close, the issue is not poor optimization or memorization, but rather a distribution mismatch between the training/validation data and the test data.

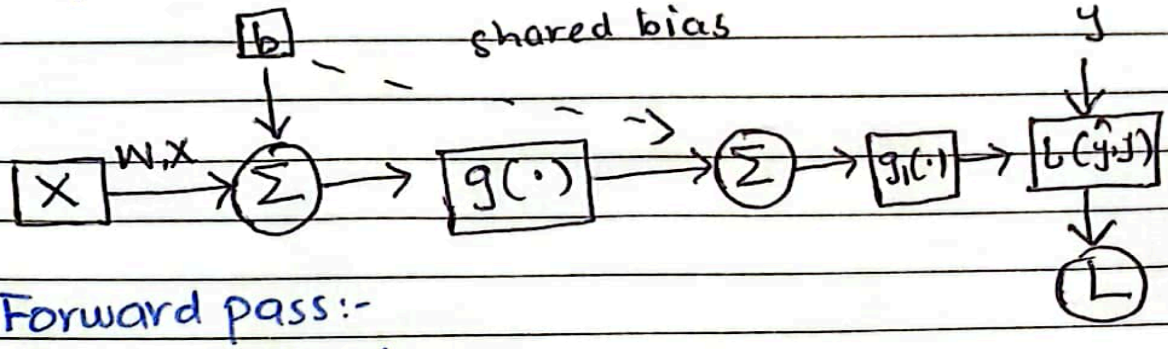
The gradient-based feature attribution results reveal that amenity_score has a much higher importance compared to all other features. This indicates that the model relies heavily on this single feature to make predictions. If the distribution or predictive relationship of amenity_score changes in the test set, the learned decision boundaries become unreliable, leading to a sharp drop in accuracy.

Therefore, the primary cause of poor test performance is feature over-reliance combined with distribution shift. To mitigate this issue, regularization techniques such as weight decay or dropout could be applied to reduce dominance of a single feature, and more diverse training data could be used to improve robustness.

QUESTION 2

LUMS
A Not-for-Profit University

Q NO 2



Forward pass:-

$z_1 = Wx + b$

$h_1 = g(z_1)$

$z_2 = Uh_1 + b$

$\hat{y} = g_1(z_2)$

$L = L(\hat{y}, y)$

∵ b appears both in z_1 and z_2 so

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial b} + \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial b}$$

step # 01

$$\partial L/\partial \hat{y} = \partial L(\hat{y}, y)/\partial \hat{y}$$

step # 02

$$\partial L/\partial z_2 = \partial L/\partial \hat{y} \cdot \partial \hat{y}/\partial z_2$$

$$\partial \hat{y}/\partial z_2 = g_1'(z_2)$$

$$\boxed{\delta_2 = \frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial \hat{y}} \odot g_1'(z_2)}$$

Step # 03

$$z_2 = Uh_1 + b$$

$$\frac{\partial z_2}{\partial b} = I$$

Path 1 $b \rightarrow z_2 \rightarrow y \rightarrow L$

$$\partial L/\partial b \,|_{path1} = \partial L/\partial z_2 \cdot \frac{\partial z_2}{\partial b} = \delta_2 . I = \delta_2$$

Step # 04

$$\partial L/\partial h_1 = \partial L/\partial z_2 \; \partial z_2/\partial h_1$$

$$\partial z_2/\partial h_1 = U$$

$$\boxed{\partial L/\partial h_1 = U^T \delta_2}$$

Step # 05

$$\partial L/\partial z_1 = \partial L/\partial h_1 \; \partial h_1/\partial z_1$$

$$\partial h_1/\partial z_1 = g'(z_1)$$

$$\boxed{\delta_1 = \partial L/\partial z_1 = (U^T \delta_2) \odot g'(z_1)}$$

Step # 06

$$z_1 = Wx + b$$

$$\partial z_1 / \partial b = I$$

Path #2 $b \to z_1 \to h_1 \to z_2 \to \hat{y} \to L$

$$\partial L / \partial b \Big|_{path_2} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial b} = \delta_1 \cdot I = \delta_1$$

Step # 07

$$\partial L / \partial b = \delta_1 + \delta_2 = \underbrace{(U^T \delta_2) \cdot g'(z_1)}_{1^{st} \text{ layer}} + \underbrace{\partial L / \partial \hat{y} \odot g'(z_2)}_{2^{nd} \text{ layer}}$$

Since b is a shared parameter appearing in both transformations, by the total derivative rule its gradient is the sum of partial derivatives along all computational paths through which it influences the loss.

$$\boxed{\frac{\partial L}{\partial b} = \delta_1 + \delta_2}$$

USE OF GEN AI:

PROMPTS:

1- Please format the given text in a more elaborate way

2- Give me the steps of EDA that I need to do not write the code. Give me only the steps

Here are the exact EDA steps your professor expects, in the correct order, without code and without extra theory — only actionable steps. Step 1: Load and Inspect Dataset Do the following: Load train.csv Check number of rows and columns View first few rows Check data types of each feature Goal: Understand dataset structure. Step 2: Identify Feature Types Separate features into: Numerical features: minimum_nights number_of_reviews availability_365 amenity_score Categorical features: neighbourhood_group room_type Target variable: price_class Goal: Know which preprocessing applies where. Step 3: Missing Value Analysis Check missing values in each column. Then: Handle missing values: Numerical → fill with median Categorical → fill with mode Finally: Verify no missing values remain. Goal: Clean dataset. Step 4: Class Distribution Analysis Analyze target variable: price_class Do: Count number of samples in each class Create bar chart Goal: Identify class imbalance. Step 5: Analyze Feature Distributions For each numerical feature: minimum_nights number_of_reviews availability_365 amenity_score Create plots showing relationship with price_class Use: boxplots OR violin plots Goal: See how features influence target. Step 6: Correlation Analysis Select only numerical features. Create correlation matrix heatmap. Goal: Identify: highly correlated features redundant features Step 7: Identify Most Influential Features Based on: boxplots distributions correlation Determine: Which features appear most predictive. Goal: Feature importance intuition. Step 8: Encode Categorical Variables Apply one-hot encoding to: neighbourhood_group room_type Goal: Convert categorical → numerical. Step 9: Standardize Numerical Features Apply standardization to: minimum_nights number_of_reviews availability_365 amenity_score Goal: Prepare for neural network. Step 10: Prepare Final Dataset Combine: Standardized numerical features Encoded categorical features Separate: X → input features y → target Goal: Dataset ready for training. Step 11: Split Dataset Split into: Training set Validation set Goal: Evaluate model properly. Final Summary Checklist You must include in report: Dataset structure Missing value analysis Class distribution plot Feature vs target plots Correlation heatmap Encoding explanation Standardization explanation Most influential feature analysis These are EXACT steps your professor expects. If you want, I can now give you: EDA report template where you just fill values Say: "Give fill-in-the-blank EDA template"