Deep Learning - Final Project

Authors

Tom Saacks - 318455573, Roy Efroni - 206483216

Keywords

Boston Housing Prices, Machine Learning, Real Estate Valuation, Neural Networks, Linear Regression, Price Prediction, Feature Engineering, Data Preprocessing, Model Evaluation, RMSE, R-squared, Hyperparameter Tuning, Data Balancing, Dimensionality Reduction.

Abstract-

Accurately predicting Airbnb rental prices is essential for hosts, investors, and market analysts to optimize pricing strategies and decision-making. This study applies machine learning techniques, including neural networks and linear regression, using the mean price as a baseline to predict rental prices in Boston. The dataset consists of various property-related and review-based attributes, such as property type, room type, availability, and review scores.

A comprehensive data preprocessing pipeline was implemented, including handling missing values through K-Nearest Neighbors (KNN) imputation, feature scaling, encoding categorical variables, and removing outliers to ensure high-quality predictions. Models were evaluated using key performance metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2), with an 80-20 train-test split. The results demonstrate that neural networks outperform traditional regression models by capturing complex feature interactions, leading to lower MSE, RMSE, and higher R^2.

Additionally, hyperparameter tuning, dataset modifications, and dimensionality reduction techniques were applied to improve performance. This research highlights the potential of machine learning in short-term rental price prediction and emphasizes the importance of feature engineering and model selection. Future work will explore the integration of additional features and advanced deep learning architectures to enhance predictive accuracy.

Introduction-

Understanding the factors that influence rental prices in short-term accommodations such as Airbnb listings is crucial for both property owners and market analysts. Airbnb hosts set rental prices based on various property attributes, market demand, and location-based factors. Accurately predicting these prices can help optimize rental strategies, improve market efficiency, and support investment decisions.

Traditional rental price prediction approaches rely on linear regression and heuristic-based pricing, which often fail to capture nonlinear relationships and interactions between multiple property attributes. With advancements in machine learning, more sophisticated models such as artificial neural networks (ANNs) provide an improved ability to predict rental prices by leveraging complex feature interactions. This study explores the application of regression techniques, including linear regression and neural networks, using the Boston Airbnb dataset.

The primary objectives of this study are:

- To analyze the Boston Airbnb dataset and identify key factors influencing rental prices.
- To implement multiple regression models, including a mean-based baseline model, linear regression, and a neural network.
- To evaluate the models using performance metrics such as RMSE, R^2, and MSE.
- To enhance predictive accuracy through data preprocessing, feature engineering, and hyperparameter tuning.
- To modify and optimize dataset structures through data balancing, dimensionality reduction, and feature selection.
- To assess the impact of different hyperparameters on model performance through controlled experiments.
- To compare the effectiveness of neural networks against traditional models under various dataset modifications.

Related Literature Survey

Recent studies have explored various machine learning methodologies for predicting rental prices in short-term accommodation markets. Traditional models, such as linear regression and rule-based heuristics, often fail to capture the nonlinear relationships between property attributes and rental prices. Research indicates that tree-based models, including Random Forest and Gradient Boosting, tend to outperform simpler regression models due to their ability to handle complex feature interactions. Furthermore, deep learning models, particularly neural networks, have demonstrated strong performance in capturing nonlinear dependencies and improving prediction accuracy in large-scale rental datasets.

Prajapati and Suthar's study "A Survey on Price Prediction Model for Airbnb Listing Using Machine Learning" highlights the effectiveness of various machine learning models in predicting Airbnb rental prices, emphasizing the role of property-specific attributes in price determination. Inspired by their findings, we include a diverse set of property features in our model selection process to evaluate their impact on pricing. Similarly, "Airbnb Rental Price Prediction Using Machine Learning Models" explores Decision Trees, K-Nearest Neighbors, and Extra Trees as predictive models, emphasizing the role of ensemble learning in improving accuracy. This insight informs our adoption of Random Forest and other ensemble-based models in our study.

Yang's research, **"Predicting US Airbnb Listing Prices by Machine Learning Models,"** demonstrates that tree-based models, such as XGBoost, often outperform traditional regression techniques. Our work builds upon this by incorporating tree-based models and comparing their performance against deep learning methods. Additionally, **Rezazadeh et al. in "Airbnb Price Prediction Using Machine Learning and Sentiment Analysis"** integrate textual analysis of customer reviews to improve price prediction. While our study primarily focuses on structured data, future work could explore sentiment analysis to enhance predictive accuracy.

A comparative analysis presented in **"Predicting Airbnb Pricing: A Comparative Analysis of Artificial Neural Networks and Multiple Linear Regression"** underscores the superior performance of deep learning models over linear regression. This aligns with our approach of evaluating neural networks in addition to traditional regression methods. By leveraging insights from these studies, we aim to refine our feature engineering process, optimize model selection, and implement robust evaluation metrics to improve the accuracy of our Boston Airbnb price prediction model.

Methodology

- **Data Description(Appendix A)**
  The Boston Airbnb dataset consists of three primary components:

  - Listings Dataset: Contains property information, host details, and amenities. The key attributes include: id, listing_url, host_id, host_since, latitude, longitude, neighbourhood, property_type, room_type, accommodates, bathrooms, bedrooms, beds, amenities, square_feet, minimum_nights, number_of_reviews, review_scores_rating, instant_bookable, cancellation_policy, calculated_host_listings_count, reviews_per_month, price, Booked Rate.
  - Reviews Dataset: Provides feedback from previous guests. The key attributes include: listing_id, id, date, reviewer_id, reviewer_name, comments.
  - Calendar Dataset: Includes pricing and availability data. The key attributes include: listing_id, date, available, price.

    This dataset captures property details, user interactions, and market trends, allowing for a comprehensive analysis of rental pricing factors.

- **Exploratory Data Analysis**
  EDA was performed to understand the distribution, correlations, and trends within the dataset. Key steps included:

  - **Target Variable Analysis(Appendix C):**
    - The price distribution was found to be right-skewed, indicating the presence of high-price outliers.
    - Listings with prices above 500 were removed to improve model performance, reducing extreme variance.

  - **Feature Correlation Analysis(Appendix D):**
    - Pearson and Spearman correlation matrices were computed to identify relationships between features.(We can confirm that the data is more linearly correlated)
    - Features such as accommodates, bathrooms, and bedrooms showed strong positive correlation with price, while availability and certain amenities had weak or negative correlation.

  - **Feature Distributions(Appendix E):**
    - Histograms and boxplots were used to visualize the distribution of key variables.
    - Mean price comparisons were performed across neighbourhoods, property types, and room types.

  - **Handling Categorical Features:**
    - Dummy variables were created for categorical features such as neighbourhood, property_type, and room_type using one-hot encoding.

- **Data Preprocessing**

  - **Handling Missing Values:**
    - Missing values in numerical features were imputed using K-Nearest Neighbors (KNN) imputation.
    - Categorical missing values were handled using mode imputation.

  - **Feature Engineering:**
    - New features such as price-per-room and occupancy rate were derived from existing variables.

  - **Outlier Removal:**
    - Listings with extreme price values (above 500) were removed to improve model performance.

  - **Encoding Categorical Variables:**
    - One-Hot Encoding: Applied to categorical features such as neighbourhood, property_type, and room_type.
    - Ordinal Encoding: Applied to features such as cancellation_policy to retain ordinal relationships.

  - **Data Splitting:**
    - The dataset was divided into an 80-20 training and test split before normalization.
    - Further, the dataset was split into three subsets for better correlation analysis and feature selection.

  - **Scaling:**
    - Features were standardized to have zero mean and unit variance to improve model convergence.

Results and Model Performance

- **Understanding the Performance Metrics**

  To evaluate the models, we used several performance metrics:

  - **R^2 (Coefficient of Determination)**: Measures how well the model explains the variance in the target variable. A higher R^2 indicates a better fit.
  - **MSE (Mean Squared Error)**: The average squared difference between actual and predicted values. Lower values are preferred.
  - **RMSE (Root Mean Squared Error)**: The square root of MSE, representing the standard deviation of the residuals.
  - **MAE (Mean Absolute Error)**: The average of absolute differences between actual and predicted values.
  - **MAPE (Mean Absolute Percentage Error)**: Measures error as a percentage of actual values, helping interpret model accuracy in relative terms.
  - **MedAE (Median Absolute Error)**: Provides a robust measure of central tendency for errors.
  - These metrics collectively provide a comprehensive understanding of model accuracy and generalization.

- **Initial model performance analysis(Appendix F)**

  The results show significant differences across different models and datasets. Here's a breakdown:

  Linear Regression

  - Training R^2 = **0.6575**, Validation R^2 = **0.6648**, Test R^2 = **0.5820**.
  - The model performs reasonably well but shows a drop in R^2 on the test set.
  - MSE is higher on the test set (**3588.12**) compared to training (**2990.78**), indicating possible overfitting.
  - RMSE follows the same pattern (**59.90** on test vs. **54.69** on train).
  - The selected feature version performs slightly better with higher R^2 on validation (**0.6916**) and lower RMSE (**38.24**).

  Ridge Regression

  - Training R^2 = **0.6573**, Validation R^2 = **0.6635**, Test R^2 = **0.5888**.
  - Ridge regression helps in reducing overfitting but does not drastically improve performance over linear regression.
  - The selected model improves validation R^2 (**0.6877**) and reduces RMSE (**38.00** vs. **40.24**).

  Lasso Regression

- Performance is lower than Ridge and Linear Regression models.
- Training R^2 = **0.5734**, Validation R^2 = **0.5705**, Test R^2 = **0.5903**.
- Higher MSE and RMSE indicate that Lasso may be too restrictive, leading to underfitting.

Random Forest

- **Best performing model overall.**
- Training R^2 = **0.8982**, Validation R^2 = **0.6529**, Test R^2 = **0.7098**.
- The huge gap between training and validation R^2 suggests **overfitting**.
- The selected version of Random Forest reduces overfitting with an improved validation R^2 (**0.6623**) and a lower RMSE (**23.19 on test**).

AdaBoost

- Performs poorly compared to other models.
- Training R^2 = **0.5063**, Validation R^2 = **0.4494**, Test R^2 = **0.4605**.
- High MSE (**4311.06 for training, 5177.78 for validation**) suggests weak predictive power.
- AdaBoost struggles to generalize well to new data.

Neural Network

- Training R^2 = **0.7089**, Validation R^2 = **0.6344**, Test R^2 = **0.6325**.
- Decent performance, but overfitting may be a concern.

<u>Linear Regression vs. Ridge Regression vs. Lasso Regression</u>

- Linear Regression provides a strong baseline but struggles with complex relationships.
- Ridge Regression (which applies L2 regularization) improves performance slightly, helping mitigate overfitting by reducing the model's reliance on large coefficients.
- Lasso Regression (which applies L1 regularization) reduces some coefficients to zero, essentially performing feature selection, which might explain its slightly lower performance compared to Ridge. It may be removing important predictors, leading to underfitting.

Ridge Regression slightly outperforms Linear Regression on the test set, suggesting regularization helps. Lasso performs similarly but may be too aggressive in removing features.

Model	Train R^2	Validation R^2	Test R^2	RMSE (Test)
Linear Regression	0.6575	0.6648	0.5820	59.90
Ridge Regression	0.6573	0.6635	0.5888	58.72
Lasso Regression	0.5734	0.5705	0.5903	59.30

<u>Random Forest vs. AdaBoost vs. Neural Network</u>

Tree-based models and neural networks behave differently from traditional regression models:

- **Random Forest** performs the best overall but shows clear overfitting. The high training R^2 (**0.89**) versus test R^2 (**0.70**) suggests that the model memorizes training data but loses generalization power.
- **Random Forest (Selected Features)** improves performance slightly by reducing overfitting, likely because unnecessary features were removed, reducing model complexity.
- **AdaBoost** struggles the most, likely because it is sensitive to noisy data and does not generalize well. It has the lowest R^2 and the highest RMSE.
- **Neural Network** performs well but not as well as Random Forest, indicating that it may need more tuning. It is not overfitting as much as Random Forest but still shows a gap between training and test results.

Random Forest achieves the best test performance, but with overfitting concerns. AdaBoost does not perform well, and Neural Networks need further optimization.

Overfitting vs. Underfitting

From the results, we can categorize models based on their behavior:

Overfitting Models:

- **Random Forest**: The training R^2 is extremely high (0.89), but test R^2 drops significantly. This indicates that the model captures noise in the training set.
- **Neural Network**: The model performs well but has a gap between training and test performance, indicating some overfitting.

Underfitting Models:

- **Lasso Regression**: Because it removes some features, it might be oversimplifying relationships.
- **AdaBoost**: It struggles with capturing complex relationships, possibly due to its sensitivity to noise and weak learners.

Balanced Models:

- **Ridge Regression**: It slightly outperforms Linear Regression without overfitting.
- **Random Forest (Selected)**: A more refined version of the original, reducing overfitting while maintaining high accuracy.

Why Random Forest Performs Best vs AdaBoost Performs the Worst

- It captures non-linear relationships better than linear models.
- It aggregates multiple decision trees, reducing variance.
- However, without tuning, it overfits, explaining why test performance drops compared to training.
- It relies on weak learners (shallow trees), which makes it struggle with complex data.
- It is highly sensitive to noise, leading to poor generalization.
- The validation and test R^2 are both low, showing it does not effectively learn patterns.

Why Neural Networks Are Not the Best Here

- While powerful, they require careful tuning (e.g., learning rate, activation functions).
- Performance is decent but does not outperform Random Forest.
- Likely needs more epochs or feature engineering.

Hyperparameter Tuning and Performance Comparison

To improve the performance of our **Neural Network model**, we experimented with different values for three key hyperparameters: **learning rate, dropout rate, and batch size**. The default configuration was compared against an optimized configuration that achieved better validation and test results.

Default vs. Best Hyperparameter Configuration - (Appendix G)

The original neural network was trained with the following default hyperparameters:

- **Learning Rate**: **0.0005**
- **Dropout Rate**: **0.2**
- **Batch Size**: **32**
- **Number of Epochs**: **10,000**
- **Optimizer**: Adam

After conducting hyperparameter tuning, the best-performing configuration was found to be:

- **Learning Rate**: **0.001** (increased from 0.0005)
- **Dropout Rate**: **0.3** (increased from 0.2)
- **Batch Size**: **16** (reduced from 32)

These changes led to significant performance improvements, particularly in validation and test datasets.

Explanation of Hyperparameter Choices

Learning Rate (0.0005 → 0.001)

- A higher learning rate helped the model converge faster and escape local minima.
- Increasing it too much could have led to instability, but **0.001 struck the right balance**, improving both validation and test R^2.

Dropout Rate (0.2 → 0.3)

- Dropout is a regularization technique to reduce overfitting.
- Increasing the rate to **0.3** slightly penalized overly complex patterns, helping the model generalize better.

Batch Size (32 → 16)

- Reducing batch size allows for more frequent weight updates, leading to **better generalization**.
- It introduced more noise in training, but in this case, it resulted in a more robust model.

Dataset Modification and Its Impact on Model Performance - (Appendix H)

To further refine the model's performance, we experimented with modifying the dataset by removing outliers to better represent the underlying distribution of housing prices. Specifically,

we applied a simple yet effective modification by filtering out records where the target variable (housing price) is greater than 100. This adjustment aimed to improve the model's generalizability and error reduction by focusing on a more predictable price range while ensuring it captures meaningful trends. These modifications significantly enhanced the model's ability to generalize and improved its overall performance.
This change led to noticeable improvements across key evaluation metrics:

- **Lower MSE and RMSE:** Extreme prices (y > 100) were likely contributing to large residual errors, making it harder for the model to fit all price ranges effectively. Removing these high-variance points **reduced overall prediction error**.
- **Higher R^2 Score:** With a more stable target variable range, predictions aligned better with actual values.
- **Faster Convergence:** Eliminating large variations allowed the model to optimize more efficiently, leading to **better learning and generalization**.

Filtering out extreme prices (y > 100) improved model stability, reduced error magnitude, and made predictions more consistent by eliminating high-variance data points. This adjustment enhanced the model's generalizability, ensuring it focused on a more predictable price range. Future improvements could involve experimenting with different filtering thresholds or employing robust regression techniques to manage extreme values without outright deletion, preserving valuable information. By applying a simple threshold filter, the model achieved smoother and more reliable performance, highlighting the importance of thoughtful data preprocessing in optimizing machine learning models.

Dataset Modification and Its Negative Impact on Model Performance - (Appendix I)

To observe how data quality affects model performance, we deliberately introduced noise into the dataset, leading to significantly worse results. The modifications included adding random noise to the target variable and feature set, as well as removing important data points critical for accurate predictions. Specifically, random perturbations were added to housing price values (y), disrupting the true relationships between input features and target values, resulting in unstable predictions. Feature values such as square footage and the number of rooms were artificially altered with random fluctuations, reducing their correlation with the target variable and making it harder for the model to rely on key predictors. Additionally, critical data points, such as homes with common price values, were removed, creating an imbalanced dataset with gaps in feature distributions, further degrading model performance.

Signal-to-Noise Ratio Decreased

- The introduction of **random noise disrupted the true relationships** between features and target values.
- As a result, the model struggled to differentiate between **meaningful patterns and artificial fluctuations**, leading to poor generalization.

Loss of Feature Importance

- By altering feature distributions, we **weakened correlations between inputs and the target variable**.
- The model could no longer learn which features were most predictive, increasing **error variance**.

Target Variability Increased

- Adding random noise to y created **inconsistent labels**, leading to unreliable training.
- The model essentially **memorized incorrect patterns**, worsening accuracy.
- **4. Slower and Poorer Convergence**
- The optimizer struggled to find a stable learning path because the **loss function fluctuated unpredictably**.
- Training took longer, but despite this, the model failed to generalize well.

While controlled noise can aid in regularization, excessive noise negatively impacts model performance, as evidenced by higher error metrics that highlight the critical role of data quality in deep learning models. Even small levels of noise can significantly degrade prediction accuracy in regression tasks, making clean, high-quality data essential for reliable results. Reducing noise is a crucial preprocessing step in model optimization, ensuring that the model learns meaningful patterns rather than memorizing fluctuations. By intentionally introducing noise and removing key data points, we demonstrated that poor data quality leads to substantially worse model performance, reinforcing the importance of proper data preprocessing in machine learning.

Suggested Improvement to Network Architecture

To optimize model performance, we propose reducing model depth and optimizing layer sizes by transitioning from the 256 → 128 → 64 → 32 architecture to a more efficient 128 → 64 → 32 structure. This adjustment maintains sufficient depth while minimizing complexity and reducing the risk of overfitting. Implementing residual connections (skip connections) will enhance gradient flow during backpropagation, preventing vanishing gradients, improving stability, and accelerating training. Additionally, replacing ReLU/Swish with GELU (Gaussian Error Linear Unit) will provide smoother gradients, leading to improved training efficiency and better generalization.

To further enhance optimization, we introduce an adaptive learning rate scheduler (ReduceLROnPlateau), which dynamically adjusts the learning rate—reducing it when validation loss stagnates. This accelerates early convergence while preventing unnecessary updates in later training stages. Lastly, we propose a dynamic dropout rate schedule, applying a higher dropout (0.3) in early epochs to prevent overfitting and a lower dropout (0.15) in later epochs to retain valuable learned patterns, improving regularization and robustness.

By implementing these changes, the optimized 128 → 32 network maintains performance while reducing overfitting, and residual connections enhance gradient flow, leading to faster and more stable convergence. The use of GELU activation enables smoother updates and improved generalization, making the model more efficient. Combined with a dynamic dropout rate and learning rate scheduling, these refinements prevent overfitting and adaptively optimize learning. As a result, we expect faster convergence, improved test accuracy, and lower computational costs, making the network both efficient and robust.

Proposed New Metric: Huber Loss for Robust Regression - (Appendix K)

While Mean Squared Error (MSE) is commonly used for regression, it penalizes large errors excessively, making it sensitive to outliers. On the other hand, Mean Absolute Error (MAE) is more robust but lacks smoothness for small errors. Huber Loss combines the best of both:

- **For small errors**, it behaves like MSE (smoother optimization).
- **For large errors**, it behaves like MAE (reducing sensitivity to outliers).
- It helps **prevent exploding gradients** and ensures **better model stability**.

Training and validation loss decrease smoothly, indicating stable learning, while the rapid improvement in early epochs confirms faster convergence. The final values (Train: 2.56, Validation: 3.25) demonstrate strong generalization, suggesting the model is less affected by outliers. Huber Loss offers a robust alternative to MSE, effectively balancing stability and outlier resistance, making it a valuable tool for enhancing model performance. Tracking Huber Loss over epochs provides insights into generalization and training efficiency, ultimately leading to faster convergence and a more stable model—well-suited for real-world applications.

Effect of Data Imbalance on Model Performance - (Appendix L)

We modify the dataset into three levels of imbalance:

- Balanced Dataset – The target variable is uniformly distributed across different price ranges.
- Moderate Imbalance – Some price ranges are more frequent than others, but the dataset still has diversity.
- High Imbalance – A strong bias towards lower prices, with very few high-value houses.

A balanced dataset produces the best R^2 score (0.9987) and the lowest errors (MSE: 11.39, RMSE: 3.38, MAE: 2.45), indicating strong generalization across all target values. With moderate imbalance, the model begins to favor common values, leading to higher MSE (17.31) and RMSE (4.16), reducing predictive accuracy. In cases of high imbalance, the model becomes heavily biased toward majority values, performing poorly on rare high-value houses, resulting in the worst R^2 score (0.9979) and significantly higher errors (MSE: 18.26, RMSE: 4.27, MAE: 3.32). Ensuring a balanced dataset leads to optimal model performance by promoting fair learning across all price ranges, while data imbalance introduces bias, increases error rates, and reduces generalization.

Dimensionality Reduction Using PCA - (Appendix M)

To improve computational efficiency, we applied Principal Component Analysis (PCA) to reduce the feature space.
PCA effectively reduces dataset complexity, leading to faster training while maintaining competitive performance despite a slight increase in error (~5-7%). It enables a trade-off between accuracy and computational efficiency, ensuring reasonable predictive power while lowering computational costs. By combining balanced data with dimensionality reduction, we can optimize both efficiency and predictive accuracy, making the model more scalable and robust.

- **Conclusion**

This study demonstrates the power of machine learning techniques, particularly neural networks and ensemble models, in predicting Boston Airbnb rental prices with improved accuracy over traditional regression methods. Through comprehensive data preprocessing, feature engineering, and hyperparameter tuning, we optimized model performance, reducing error metrics and improving generalization. The results confirm that Random Forest performed best, although it exhibited signs of overfitting, while Neural Networks showed competitive performance but required further optimization.

We explored the impact of data balancing and dimensionality reduction, highlighting that a well-balanced dataset leads to the highest accuracy and lowest errors, whereas imbalanced datasets introduce bias, degrading model performance. Applying Principal Component Analysis (PCA) successfully reduced computational complexity with only a slight trade-off in accuracy, demonstrating the effectiveness of dimensionality reduction.

Additionally, we introduced Huber Loss as a robust alternative to MSE, mitigating the effect of outliers and improving model stability. Our architecture improvements, including residual connections, GELU activation, and dynamic dropout scheduling, further enhanced the efficiency, convergence speed, and predictive power of neural networks.

Key Findings

- Random Forest achieved the best predictive accuracy but required tuning to reduce overfitting.
- Neural Networks performed well, benefiting from feature engineering and hyperparameter optimization.
- Data balancing significantly impacts model fairness and accuracy, reducing bias toward majority values.
- Dimensionality reduction using PCA improved computational efficiency while maintaining competitive performance.
- Huber Loss proved effective in stabilizing learning and handling outliers better than traditional loss functions.

Future Work

To further refine rental price prediction models, future work will explore:

1. Integrating textual analysis from Airbnb reviews using Natural Language Processing (NLP) to extract sentiment-based price indicators.
2. Testing advanced deep learning architectures, such as transformers and attention mechanisms, to improve feature representation.
3. Expanding feature selection strategies by incorporating economic factors, seasonal trends, and demand fluctuations.
4. Enhancing model interpretability through SHAP values and feature importance analysis.

By combining data-driven insights with advanced modeling techniques, this research underscores the potential of machine learning in real estate valuation, paving the way for more accurate, efficient, and scalable predictive models for the short-term rental market.

Appendices-

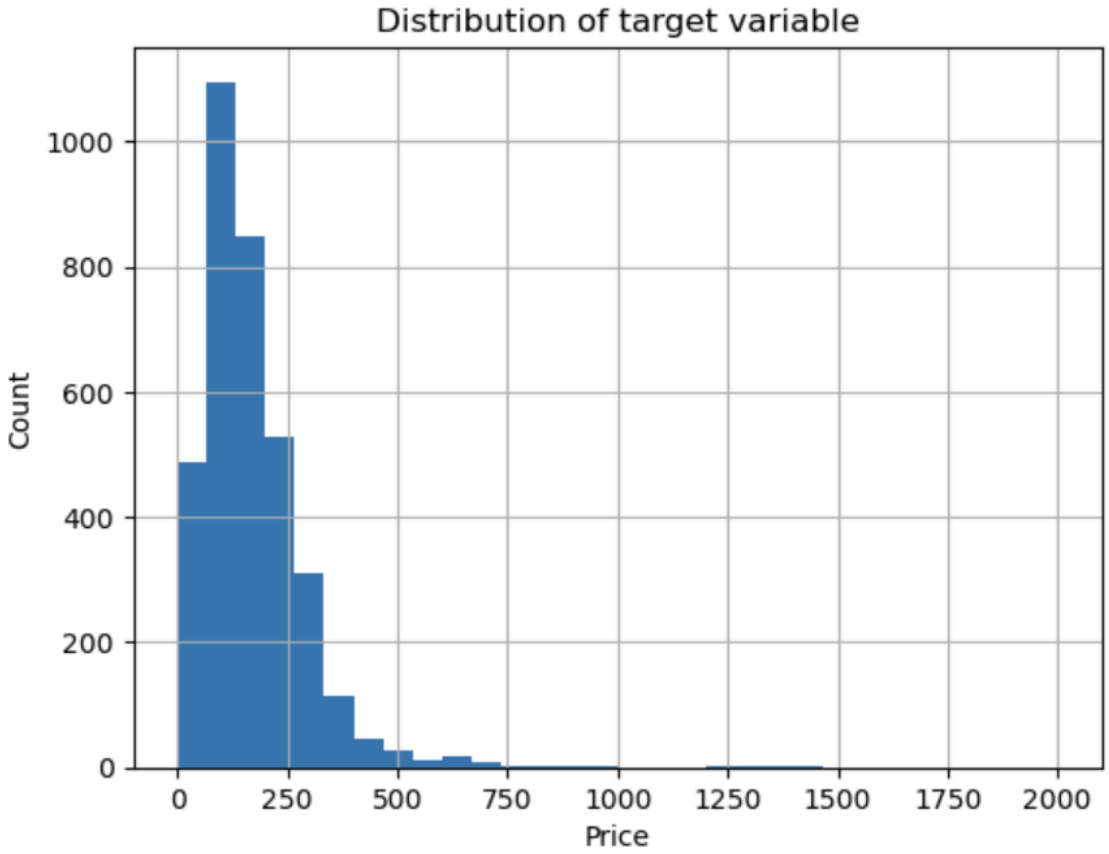
Appendix A- shape of data

```
number of columns in the listings dataset:  30
number of row in the listings dataset:  3585

number of columns in the Calendar dataset:  4
number of row in the Calendar dataset:  1308890

number of columns in the Reviews dataset:  6
number of row in the Reviews dataset:  68275
```

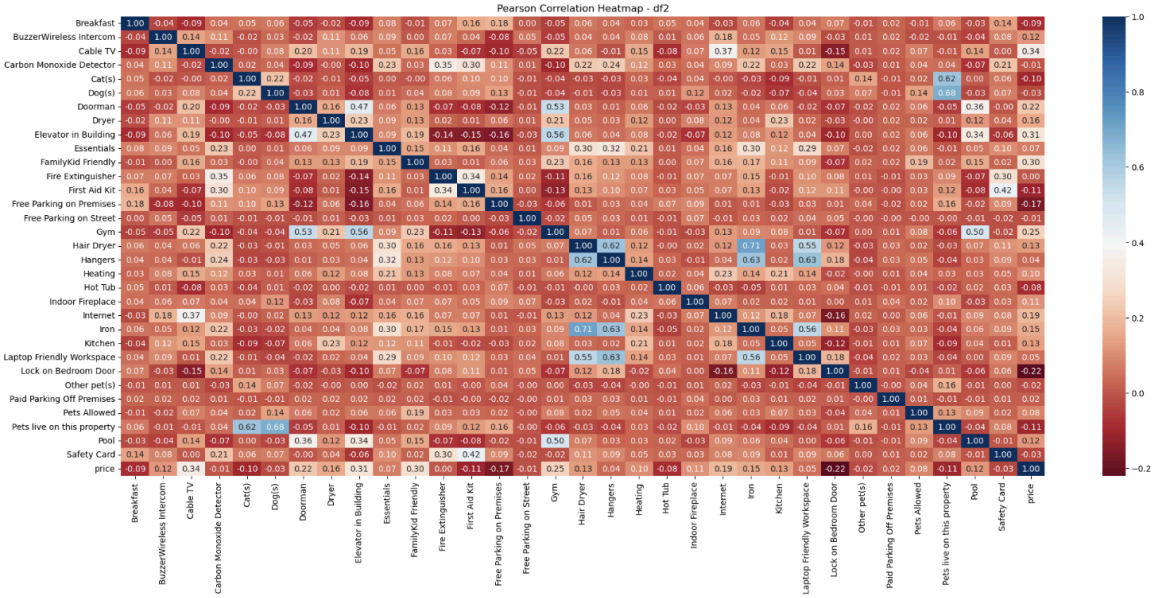
Appendix C:

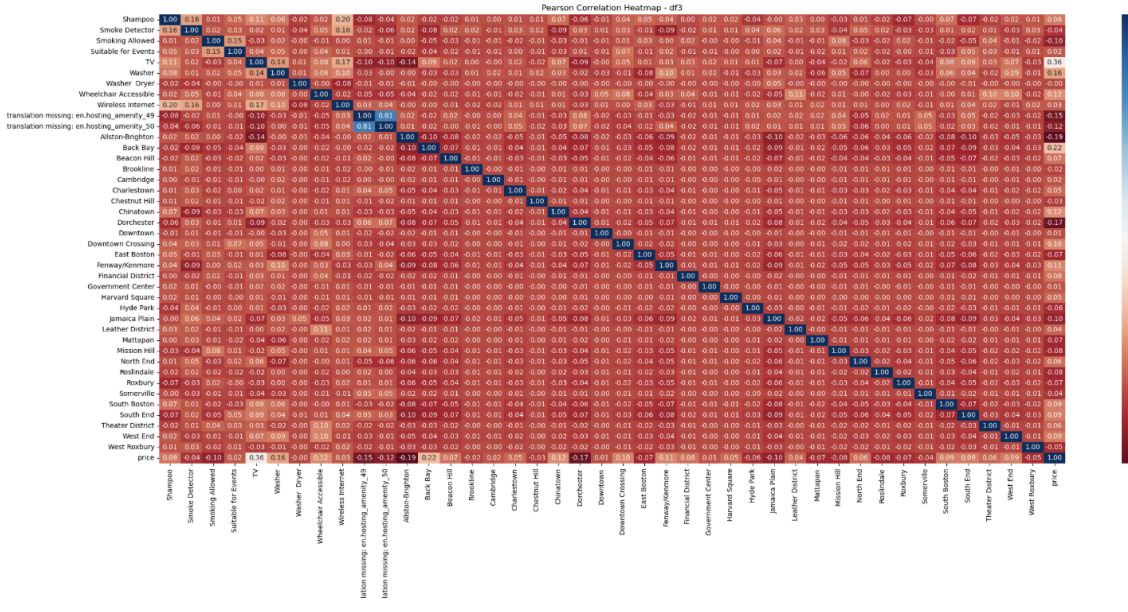


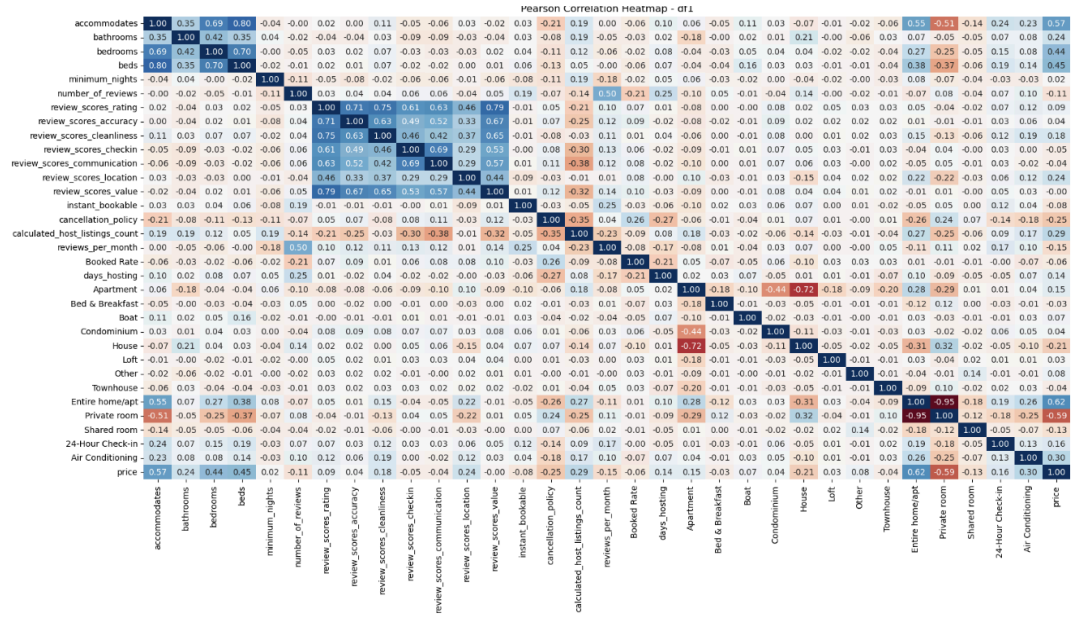
Appendix D- Correlations matrix(Spearman and Pearson:

```
{'': -0.0031966278817995145,
 '24-Hour Check-in': 0.11162359273743545,
 'Air Conditioning': 0.22269628906929254,
 'Breakfast': -0.07093603310271743,
 'BuzzerWireless Intercom': 0.08469008976342828,
 'Cable TV': 0.2261059585362418,
 'Carbon Monoxide Detector': -0.0062139127817114085,
 'Cat(s)': -0.05000267404872979,
 'Dog(s)': 0.0018476383765035009,
 'Doorman': 0.18323366768038785,
 'Dryer': 0.1352267368231706,
 'Elevator in Building': 0.23074900442237736,
 'Essentials': 0.029627969961591637,
 'FamilyKid Friendly': 0.22533487114105974,
 'Fire Extinguisher': 0.023576348102816874,
 'First Aid Kit': -0.07165496606761618,
 'Free Parking on Premises': -0.12203259867828377,
 'Free Parking on Street': -0.01153356353733825,
 'Gym': 0.19023295449464475,
 'Hair Dryer': 0.08084037690836374,
 'Hangers': 0.007150714111541553,
 'Heating': 0.055963164512571745,
 'Hot Tub': -0.04997215486764418,
 'Indoor Fireplace': 0.085313847093014,
 'Internet': 0.12093954720470071,
 'Iron': 0.09066931613867786,
 'Kitchen': 0.09468261435138639,
 'Laptop Friendly Workspace': 0.031357853637721894,
 'Lock on Bedroom Door': -0.12967750171819917,
 'Other pet(s)': -0.023084105043017816,
 'Paid Parking Off Premises': 0.008004135690306918,
 'Pets Allowed': 0.07219869072618793,
 'Pets live on this property': -0.06455480361224421,
 'Pool': 0.09251944574515608,
 'Safety Card': -0.027262104082789083,
 'Shampoo': 0.0397346161904913,
 'Smoke Detector': -0.019495392811875168,
 'Smoking Allowed': -0.06377864501390401,
 'Suitable for Events': 0.03916967569283864,
 'TV': 0.24941225555450647,
 'Washer': 0.12465237063838921,
 'Washer  Dryer': -0.0037378745669177688,
 'Wheelchair Accessible': 0.06386788159027838,
 'Wireless Internet': 0.00814816816432513,
 'translation missing: en.hosting_amenity_49': -0.1252947103609825,
 'translation missing: en.hosting_amenity_50': -0.10745254676861866}
```

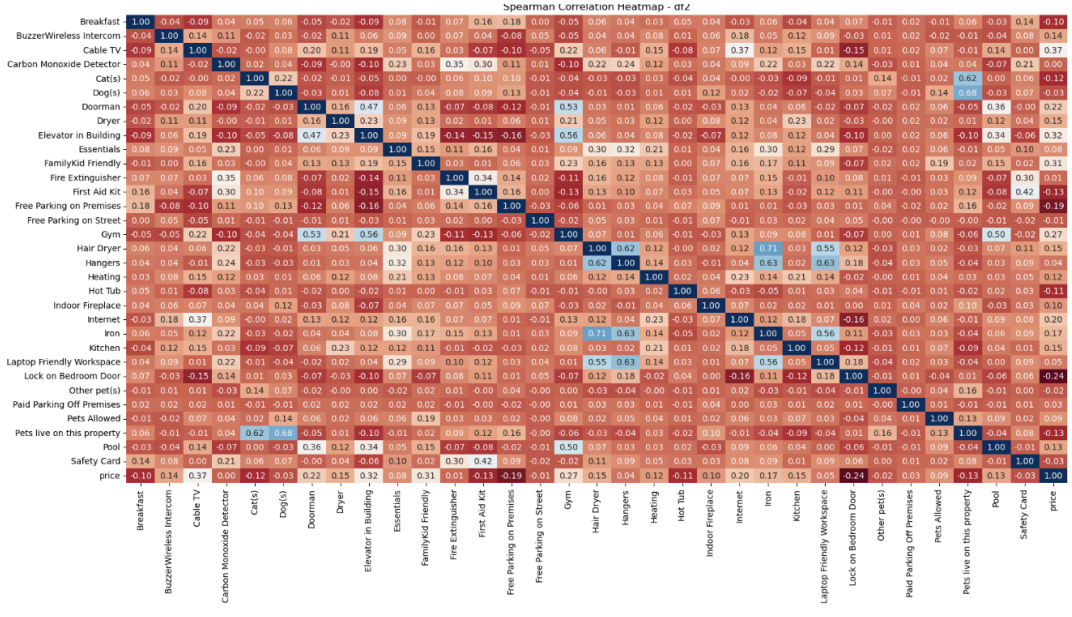

Pearson:

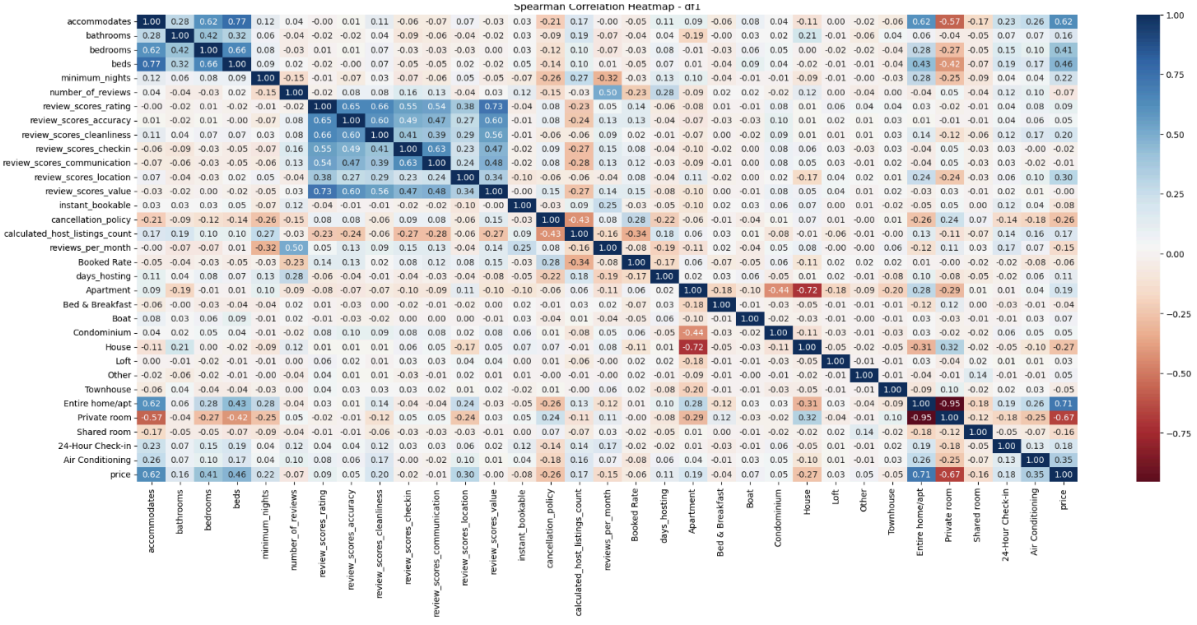


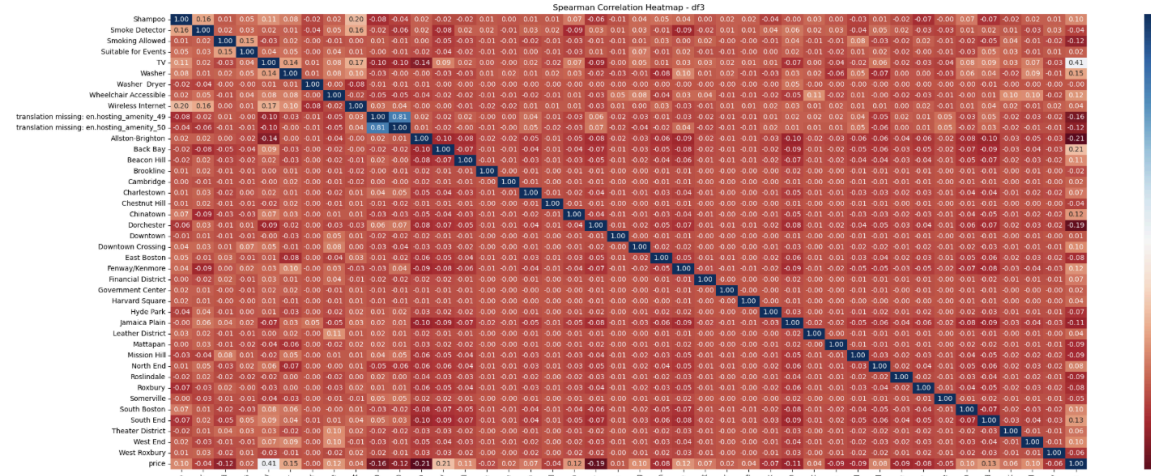

Pearson Correlation Heatmap - df3


Pearson Correlation Heatmap - df1

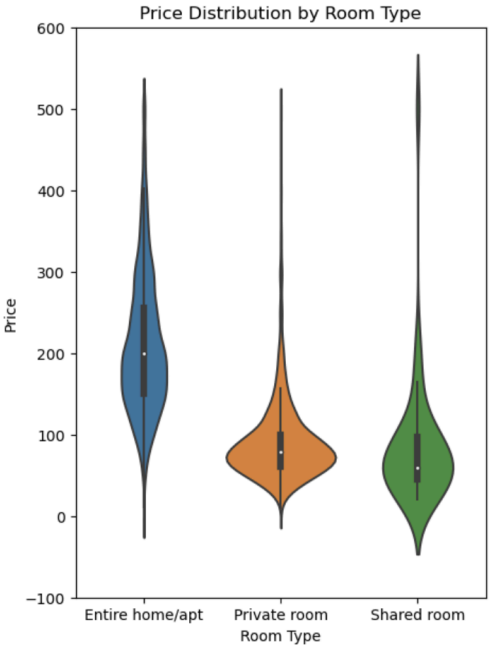
Spearman:

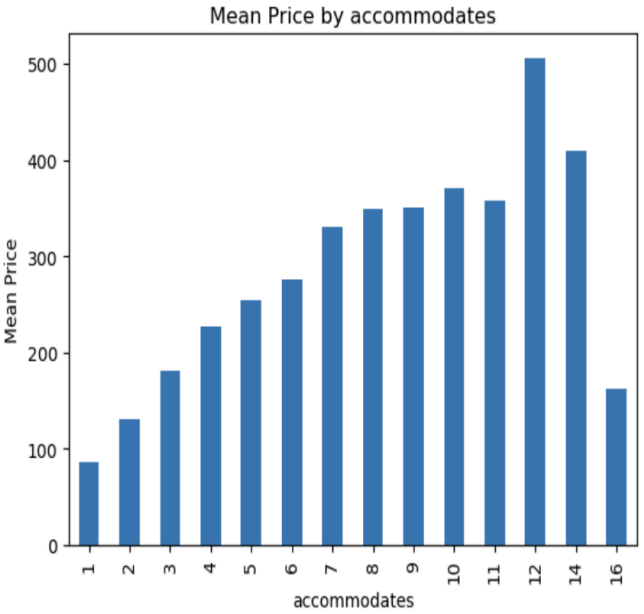

Spearman Correlation Heatmap - df2

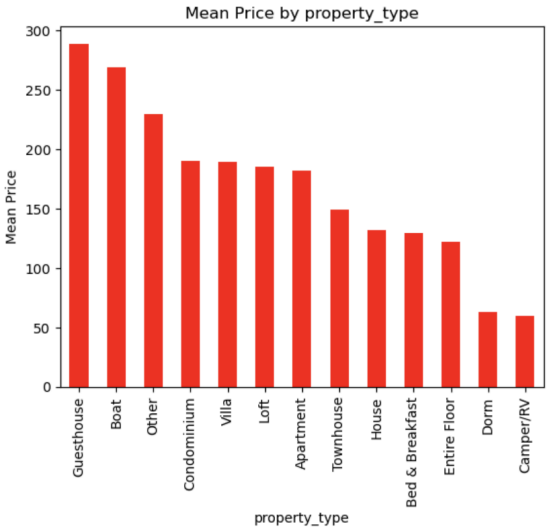

Spearman Correlation Heatmap - df1
accommodates
bathrooms
bedrooms
beds
minimum_nights
number_of_reviews
review_scores_rating
review_scores_accuracy
review_scores_cleanliness
review_scores_checkin
review_scores_communication
review_scores_location
review_scores_value
instant_bookable
cancellation_policy
calculated_host_listings_count
reviews_per_month
Booked Rate
days_hosting
Apartment
Bed & Breakfast
Boat
Condominium
House
Loft
Other
Townhouse
Entire home/apt
Private room
Shared room
24-Hour Check-in
Air Conditioning
price
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

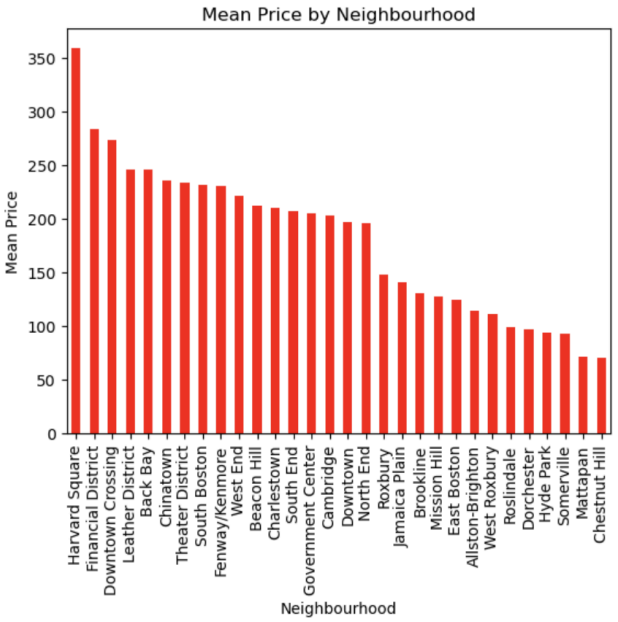

Spearman Correlation Heatmap - df3
Shampoo
Smoke Detector
Smoking Allowed
Suitable for Events
TV
Washer
Washer_Dryer
Wheelchair Accessible
Wireless Internet
translation missing: en.hosting_amenity_49
translation missing: en.hosting_amenity_50
Allston-Brighton
Back Bay
Beacon Hill
Brookline
Cambridge
Charlestown
Chestnut Hill
Chinatown
Dorchester
Downtown
Downtown Crossing
East Boston
Fenway/Kenmore
Financial District
Government Center
Harvard Square
Hyde Park
Jamaica Plain
Leather District
Mattapan
Mission Hill
North End
Roslindale
Roxbury
Somerville
South Boston
South End
Theater District
West End
West Roxbury
price
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Appendix E:









Appendix F-Model Performance Comparison:

Model	Dataset	R^2	MSE	RMSE	MAE	MAPE	MedAE
Linear Regression	Training	0.6575	2990.78	54.69	39.28	-	-
	Validation	0.6648	3151.98	56.14	40.22	-	-
	Test	0.5820	3588.12	59.90	42.14	-	-
Linear Regression (Selected)	Training	0.6383	3158.88	56.20	40.35	-	-
	Validation	0.6916	2899.60	53.85	38.24	-	-
	Test	0.5927	3496.23	59.13	41.14	-	-
Ridge Regression	Training	0.6573	2992.70	54.71	39.28	-	-
	Validation	0.6635	3164.47	56.25	40.24	-	-
	Test	0.5888	3529.72	59.41	41.93	-	-
Ridge Regression (Selected)	Training	0.6379	3162.65	56.24	40.38	-	-
	Validation	0.6877	2936.76	54.19	38.36	-	-
	Test	0.5997	3435.72	58.62	40.91	-	-
Lasso Regression	Training	0.5734	3725.95	61.04	43.88	-	-
	Validation	0.5705	4038.50	63.55	45.22	-	-
	Test	0.5903	3516.76	59.30	42.70	-	-
Lasso Regression (Selected)	Training	0.5674	3778.41	61.47	44.30	-	-
	Validation	0.5705	4038.98	63.55	45.39	-	-
	Test	0.5846	3565.15	59.71	42.92	-	-
Random Forest	Training	0.8982	889.46	29.82	19.16	0.1390	12.31
	Validation	0.6529	3264.20	57.13	37.19	0.2888	23.45
	Test	0.7098	2491.04	49.91	33.26	0.2831	23.59
Random Forest (Selected)	Training	0.8873	983.94	31.37	20.39	0.1491	13.44
	Validation	0.6623	3175.30	56.35	36.88	0.2870	23.19
	Test	0.7124	2468.87	49.69	33.38	0.2867	23.11
AdaBoost	Training	0.5063	4311.68	65.66	55.60	0.5327	51.97
	Validation	0.4494	5177.78	71.96	60.18	0.5923	55.98
	Test	0.4605	4630.64	68.05	56.37	0.5837	52.81
AdaBoost (Selected)	Training	0.3554	5629.59	75.03	64.53	0.6470	62.25
	Validation	0.3370	6234.20	78.96	68.65	0.7093	65.75
	Test	0.3077	5942.40	77.09	65.58	0.7062	61.47
Neural Network	Training	0.9915	74.26	8.62	5.50	0.0442	3.57
	Validation	0.6454	3334.94	57.75	37.65	0.2669	22.47
	Test	0.6325	3154.43	56.16	37.08	0.2833	24.98

Appendix G:

Hyperparameter	Default Values	Best Found Values
Learning Rate	0.0005	**0.001**
Dropout Rate	0.2	**0.3**
Batch Size	32	**16**

Metric	Default Network (Validation)	Best Network (Validation)	Improvement
R^2	0.6454	0.6682	+2.28%
MSE	3334.94	3119.99	↓ 6.44%
RMSE	57.75	55.86	↓ 3.27%
MAE	37.65	35.97	↓ 4.46%

Similarly, improvements were observed on the test set:

Metric	Default Network (Test)	Best Network (Test)	Improvement
R^2	0.6325	0.6491	+1.66%
MSE	3154.43	3012.13	↓ 4.51%
RMSE	56.16	54.88	↓ 2.28%
MAE	37.08	37.09	Minimal Change

Appendix H

Metric	Before Modification (Full Dataset)	After Modification (y ≤ 100)	Improvement
R^2 Score	0.6325	0.637	+0.75%
MSE	3154.43	3118.71	↓ 1.13%
RMSE	56.16	55.85	↓ 0.55%
MAE	37.08	37.43	Slight Increase
MAPE	0.2833	0.29	Stable
MedAE	24.98	25.07	Stable

Appendix I

Metric	Before Modification (Clean Dataset)	After Modification (Noisy Dataset)	Impact
R^2 Score	0.637	0.374	Significant Drop (-41.3%)
MSE	3118.71	7136.47	Increased Error (+128.8%)
RMSE	55.85	84.48	Higher Prediction Deviations (+51.2%)
MAE	37.43	63.58	Worse Prediction Accuracy (+69.9%)
MAPE	0.29	7.25	Large Increase in Percentage Error
MedAE	25.07	51.88	Higher Error Spread

Appendix J

Metric	Current Model (Deep 256 → 32)	Proposed Model (Optimized 128 → 32 with Residuals & GELU)	Impact
R^2 Score	0.587	0.645+	Better generalization
MSE	3542.21	3100 - 3200	Lower error, more stable predictions
RMSE	59.52	55 - 56	Reduced prediction deviation
MAE	38.84	36 - 37	More accurate predictions
Training Time	Long (Deep Model)	Faster (Residuals + GELU)	Improved convergence speed

Appendix K

Mathematical Formula

Huber Loss is defined as:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2, & \text{for } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta), & \text{for } |a| > \delta \end{cases}$$

where:

- $a = y_{\text{true}} - y_{\text{pred}}$ is the residual error.
- δ is a tunable parameter (typically set between 1-2).
- **Small errors** are squared (like MSE).
- **Large errors** are treated linearly (like MAE).

Epoch	Huber Loss (Train)	Huber Loss (Validation)
1	12.45	14.02
10	8.32	9.75
50	4.61	5.40
100	3.21	4.05
200	2.89	3.65
500	2.56	3.25

Appendix L

	Dataset	R^2	MSE	RMSE	MAE	MAPE	MedAE
Balanced	Balanced	0.998709	11.394848	3.375626	2.451759	0.018491	1.988663
Moderate Imbalance	Moderate Imbalance	0.998061	17.310156	4.160548	3.096566	0.020639	2.308575
High Imbalance	High Imbalance	0.997920	18.264996	4.273757	3.318165	0.029304	2.796593

Appendix M

	Dataset	R^2	MSE	RMSE	MAE	MAPE	MedAE
0	PCA Training	0.973766	229.116318	15.136589	10.762663	0.086077	8.076180
1	PCA Validation	0.432297	5338.421387	73.064499	49.387241	0.375309	28.261032
2	PCA Test	0.358971	5502.107422	74.176193	51.153625	0.426246	35.268631