Predicting the Cost of Subcutaneous Tissue Debridement Across U.S. Hospitals Using Machine Learning

SML301 Semester Project Report

Brian Mmari, Gabriel Vargas, Aphia Ishimwe

Abstract

Subcutaneous tissue debridement, a common outpatient procedure used to remove damaged skin and tissue, is often subject to wide pricing variability across hospitals in the United States. Leveraging data made available through the 2021 Hospital Price Transparency Rule, we developed a machine learning pipeline to predict the cost of this procedure. After data cleaning, feature engineering, and log transformation, we trained linear, quadratic, decision tree, and neural network models. The neural network performed best ($R^2 \approx 0.69$, MAE ≈ $0.42). Stratified analysis revealed that CASH payers in Michigan face the highest prices, while Florida showed the greatest price variability. These insights highlight the potential of predictive modeling in uncovering systemic pricing disparities in healthcare

1. Introduction

Healthcare spending in the United States is enormous and growing, yet patients often face unpredictable and unaffordable costs. In 2023, U.S. health expenditures reached about $4.9 trillion, more than triple the amount spent in 2000[1]. On a per-capita basis, this is equated to roughly $14,570 per person per year, much higher than in most other developed nations. Despite this high spending, price transparency and affordability remain major challenges. Patients frequently cannot determine in advance what a given test, procedure, or hospital visit will cost them. Nearly 20% of insured patients undergoing in-network hospital surgeries or childbirth received surprise out-of-network bills, costing them thousands of dollars. Unsurprisingly, the high cost of care leads many Americans to delay or even skip treatment. Almost half of U.S.

adults reported avoiding medical care in the past to avoid the expense burden or due to a lack of financial resources[3]. Such unpredictability and lack of transparency in pricing undermine public trust and can result in significant detrimental effects when people forgo needed care due to cost concerns.

This opaque pricing environment has prompted calls for both policy solutions and technological innovations to better predict the cost of medical procedures. Recent policies have made some initial steps: the *No Surprises Act* (2022) curtails unexpected out-of-network bills, and *Hospital Price Transparency Rules* (2021) requires hospitals to post prices for services. Compliance, however, are ongoing challenges. Even when published, raw price lists are hard for consumers to interpret, and many services, such as emergency care, do not provide cost estimates in advance[2,4]. Thus, accurate healthcare cost prediction is crucial to improve access to medical care and provide patients with accurate cost estimates tailored to their specific needs.

Here, we leverage a large dataset obtained as a result of the *Hospital Price Transparency Rule* (2021), providing extensive price reports for a wide range of procedures throughout the U.S., to develop a machine learning algorithm to predict the cost of subcutaneous tissue debridement across the U.S. We focus on subcutaneous tissue debridement due to the nicheness of the procedure and extensive representation in our data. Our model performs particularly well in our testing dataset, and it can be adapted to predict a wider range of procedures. To model pricing behavior, we applied three supervised regression techniques: Linear Regression, Decision Trees, and a Feedforward Neural Network**.** We trained and tested these models using features such as payer type, state location, and encoded metadata, while targeting the log-transformed procedure cost for more robust learning. Among the models, the neural network achieved the best generalization on unseen data with an R^2 score of 0.69 and a mean absolute error (MAE) of \$0.42, capturing nonlinear interactions more effectively than linear baselines. The decision tree achieved nearly perfect performance ($R^2 \approx 0.999$), but showed signs of overfitting. A quadratic regression model further improved over the linear baseline by modeling diminishing returns in price effects. We also find that payer type and state location significantly influence pricing, with patients in Florida and those billed under the category "OTHER" experiencing the greatest price variance. Specifically, price × payer interactions (e.g., AETNA and HUMANA) were among the most influential terms in the quadratic model. This pipeline demonstrates the feasibility of using

machine learning to uncover regional and insurer-specific cost dynamics. While our focus was on a single procedure, the framework can be adapted to model a broader range of services, ultimately supporting more transparent and equitable healthcare pricing.

2. Data Description

The dataset used in this project is titled "Hospital Price transparency" that can be found on this website:https://www.dolthub.com/repositories/dolthub/hospital-price-transparency. This repository is hosted on the Dolthub database after a January 1, 2021 US law was passed forcing hospitals to publish their prices in human and machine readable format. The dataset consists of three tables: cpt_hcpcs, hospitals, and prices. The dataset consists of three main tables: **cpt_hcpcs, hospital, and prices.**

1. **`cpt_hcpcs` Table**:

   This table includes three columns: **`code`, `short_description`,** and **`long_description`**.

   1. **`code`** column contains either CPT (Current Procedural Terminology) or HCPCS (Healthcare Common Procedure Coding System) codes. CPT codes are maintained by the American Medical Association, while HCPCS codes are used primarily for Medicare billing. While most hospitals are expected to publish prices using CPT codes, HCPCS codes may appear in cases where pricing data was published under that standard.
   2. **short_description:** A concise, standardized label for the medical procedure or service, often abbreviated for quick reference or display. This field is useful for filtering or categorizing procedures without excessive detail.

      Example*:* "HC REMOVAL OF DAMAGED SKIN AND UNDERLYING TISSUE"
   3. **long_description:** A more detailed textual explanation of the procedure, providing clarity on what the service entails. It may include medical context, conditions for billing, or technical terms aligned with CPT/HCPCS standards.

Example*:* "Debridement, subcutaneous tissue (includes epidermis and dermis, if performed); first 20 sq cm or less"

2. **Hospital Table**:

   This table contains the columns: **npi_number, name, url, street_address, city, state,** and **zip_code.**

   1. **npi_number** is a 10-digit National Provider Identifier used to uniquely identify hospitals.
   2. **name** refers to the hospital's name.
   3. The **url** column is intended to store the link to the hospital's publicly available pricing data.

3. **Prices Table**:

   This table captures the core pricing information and includes:

   1. **code**: A foreign key referencing the **cpt_hcpcs** table.
   2. **npi_number**: A foreign key referencing the **hospital** table.
   3. **payer**: The name of the insurer, or 'CASH' if the individual is self-paying.
   4. **price**: The cost of the medical procedure in USD.

3. Literature

Predicting individual healthcare costs remains a highly complex challenge due to the multitude of influencing variables and the heavily skewed distribution of expenditures—where a small fraction of patients incur disproportionately high costs. Traditional actuarial models have relied on linear regression and diagnosis-based risk scores, but these often fall short in capturing the non-linear interactions present in real-world healthcare data. In response, recent studies have employed supervised machine learning (ML) approaches—primarily regression to estimate cost values or classification to identify high-cost patients—using features such as demographics, clinical history, prior utilization, and sometimes social determinants of health. Though a few studies have explored unsupervised methods (e.g., clustering cost trajectories), the bulk of recent work has centered on supervised models aimed at improving accuracy and insight.

Rose (2016) developed an ML-based risk adjustment model using insurance claims data, comparing traditional linear models against penalized regression, decision trees, neural networks, and an ensemble “super learner” that optimized performance via cross-validation. Their results showed that the ensemble model outperformed standard approaches in terms of R^2, and importantly, the ML-based process selected a simplified feature set that preserved most of the predictive power. This suggested that ML can not only enhance actuarial models but also simplify them while mitigating risks such as diagnosis upcoding. However, this work focused on insurer payments, limiting generalizability to broader cost prediction tasks.

In a more comprehensive evaluation, Morid et al. (2017) analyzed data from approximately 90,000 individuals encompassing over 6 million medical and 1.2 million pharmacy claims. They compared gradient boosting machines (GBMs), random forests, neural networks, and ridge regression, finding that GBMs provided the best overall predictive accuracy for the general population, while ridge regression and neural networks performed better for identifying high-cost patients. This emphasized the importance of matching algorithm choice to the cost distribution segment being modeled. While the dataset was extensive and the comparison rigorous, even the best models struggled with the volatility of outlier cases, underscoring the need for hybrid methods.

Tamang et al. (2017) tackled the problem of identifying "cost bloomers"—patients who experience dramatic increases in healthcare costs from one year to the next. Using a rich dataset comprising over 1,000 features—including prior utilization, diagnoses, and social characteristics—the authors demonstrated that their most comprehensive model improved prediction performance by over 30% compared to a baseline. The best model achieved an AUC of 0.786 and captured significantly more of the future cost burden. This work highlighted the potential of incorporating non-traditional features and leveraging high-dimensional data, though interpretability and overfitting remained concerns.

Orji et al. (2023) applied ensemble tree-based models (XGBoost, GBM, and random forest) to a smaller public dataset of 986 individuals to predict annual insurance charges. Their findings showed that XGBoost achieved the best performance ($R^2 \approx 0.86$, RMSE ≈ 0.34 in normalized units), and they complemented their analysis with explainable AI techniques such as SHAP values and ICE plots. While the study effectively illustrated ML workflows and model

interpretability, its utility was limited by the simplicity of the dataset, which excluded clinical histories or real claims data.

In a larger-scale application, Drewe-Boss et al. (2022) trained a deep neural network on a claims dataset covering 1.4 million individuals. The model—a multilayer perceptron with four hidden layers—outperformed traditional regression methods in predicting future costs, particularly for patients whose expenses fluctuated dramatically. The model's ability to detect complex interactions between diagnoses and demographics demonstrated the potential of deep learning in this context, albeit at the cost of reduced interpretability and higher computational demands.

Finally, Hautala et al. (2023) conducted a small-scale study predicting healthcare costs in patients recovering from acute coronary syndrome. By ranking features via variance decomposition and incrementally building linear models, they found that depression score alone explained 16% of cost variance—more than conventional cardiac metrics like LDL cholesterol or ejection fraction. Though the study's statistical power was limited by sample size (n=65), it offered valuable insight into the relevance of psychosocial factors in cost modeling, advocating for a more holistic understanding of patient care needs.

Taken together, these studies reveal several key trends. First, prior-year cost and utilization are consistently the most powerful predictors of future expense. Second, ensemble methods and deep learning generally achieve the highest accuracy, although simpler models may perform better in certain high-cost subgroups. Third, the field is increasingly attentive to interpretability, fairness, and ethical use. Tools like SHAP values and simplified models are being used to unpack "black-box" predictions and ensure equitable treatment across patient subgroups. While ML can meaningfully enhance cost prediction models, it is not a panacea: healthcare expenses are inherently noisy and influenced by unpredictable events. As such, ML-based tools are best viewed as aids for risk stratification and cost planning, rather than definitive forecasts.

4. Exploratory Data Analysis

In this section, we conduct exploratory data analysis (EDA) to better understand the structure, patterns, and potential issues within our datasets. We are working with three sources: hospital information, medical procedure pricing, and CPT/HCPCS code descriptions. Given the size and complexity of these datasets, our first step involves cleaning and filtering the data to isolate a meaningful subset that allows for more focused and interpretable analysis.

4.1 Data Cleaning & Preparation

To ensure the data was suitable for analysis, we began by examining the structure and statistical properties of each dataset. This initial inspection informed how we would clean, reduce, and integrate the data for effective exploratory analysis.

4.1.1 Initial Dataset Inspection

The prices dataset contains over 7 million records across all four four columns and here is the summary statistics for the prices: mean = 4226, standard deviation = 316960, min = -39585, 25% percentile = 131, 50% percentile = 542, 75% percentile = 2197, and max = 999007000.

The hospitals dataset contained information for 1400 hospitals but several columns had missing or highly variable entries:

- publish_date was available for only 610 entries.
- zip_code and street_address had 1357 and 1351 respectively.
- Some hospital names also appeared multiple times due to different locations or pricing structures and that's why the unique name counts 1176.

1. State Selection Based on Frequency

To focus the analysis on a regionally balanced and sufficiently large subset of hospitals, we examined the frequency of hospital entries by state. The table below summarizes the top 10 states with the highest number of hospitals in the dataset.

Rank	State	Number of Hospitals
1	MI	188
2	FL	180
3	TX	163
4	CA	107
5	WI	56
6	IN	51
7	AL	37
8	SD	36
9	MN	36
10	TN	29

Table 1: Number of hospitals in the 10 most frequent states

Based on this distribution, we retained only hospitals located in **Michigan (MI)**, **Florida (FL)**, and **Texas (TX)** for subsequent analysis. These three states offered the highest representation and provided a solid foundation for regional comparison while maintaining a manageable dataset size.

2. Dropping Irrelevant and Redundant Columns

After filtering the hospital dataset to include only records from Michigan, Florida, and Texas, we examined its structure and found several columns that were either redundant, high in cardinality, or unlikely to add predictive value to our analysis. The following columns were dropped from the hospital dataset:

- **Name**: high-cardinality identifier; merged hospitals often repeated names across multiple NPIs.
- **url, street_address, and city**: too granular and inconsistent for meaningful aggregation.
- **Publish_date**: missing in over half the records (only ~44% complete), making it unreliable for modeling.

In the **price dataset**, we later removed:

- **Zip_code**: had over 1,000 missing values after merging, and offered no strong advantage over the state column.
- **npi_number**: retained temporarily for joining purposes but dropped after merging with hospital metadata.

These steps helped simplify the dataset, reduce noise, and ensure a cleaner structure for feature engineering and visualization.

3. Merging Hospital and Price Data

To contextualize pricing data with hospital-level metadata, we merged the cleaned price dataset with the filtered hospital dataset using the shared column npi_number. This allowed us to associate each procedure's price with the corresponding hospital's state. Then we proceeded to drop NaN values and performed an additional cleaning step to address the high cardinality in the payer field.

The next step was grouping low - frequency Payers because many Payers appeared infrequently and introduced unnecessary noise. Thus, to simplify the feature space:

- We counted the frequency of each unique payer.
- The 10 most common payers were retained as they are as shown in Table2
- All other payer names were grouped into a single category: OTHER.

Payer	Count
OTHER	5.322.239
CASH	2.074.158
COFINITY	84.491
PRIORITY HIX	76.625
BCBS	76.248
AETNA PPO	75.013
PRIORITY HEALTH OP RATE	67.746
HUMANA	62.518
Aetna	45.908
COFINITY - AETNA OP RATE	45.354

Table 2: Frequency of the top payer groups in the dataset after consolidation

4.1.2 Univariate Analysis

To prepare the dataset for modeling, we conducted a series of exploratory analyses focused on the distribution of individual variables, the structure of categorical data, and the integration of additional contextual information. This process involved trimming outliers in the price variable, and merging with CPT/HCPCS metadata for richer feature representation. We also used visualizations to better understand the distribution of prices and the prevalence of key procedures and payer types.

1. **Price Distribution Analysis**

To examine the variability and structure of the target variable (price), we began by plotting a box plot of the merged dataset. As shown in Figure 1, the initial box plot was overwhelmed by extreme outliers, with prices reaching several million dollars. This made it difficult to interpret the central tendency and distribution of the majority of data points.

To address this issue, we applied a trimming filter, restricting the dataset to prices less than or equal to $10,000. This threshold allowed us to retain the majority of realistic billing data while removing extreme outliers that distorted the visualization. The updated box plot in Figure 2 provides a much clearer view of the price distribution.

In Figure 4, the box spans from approximately $0 to $1700. The median price falls near $700, indicating that half of the prices are below this value. The whiskers extend to prices within 1.5 times the interquartile range, while prices beyond $4000 are mostly outliers.

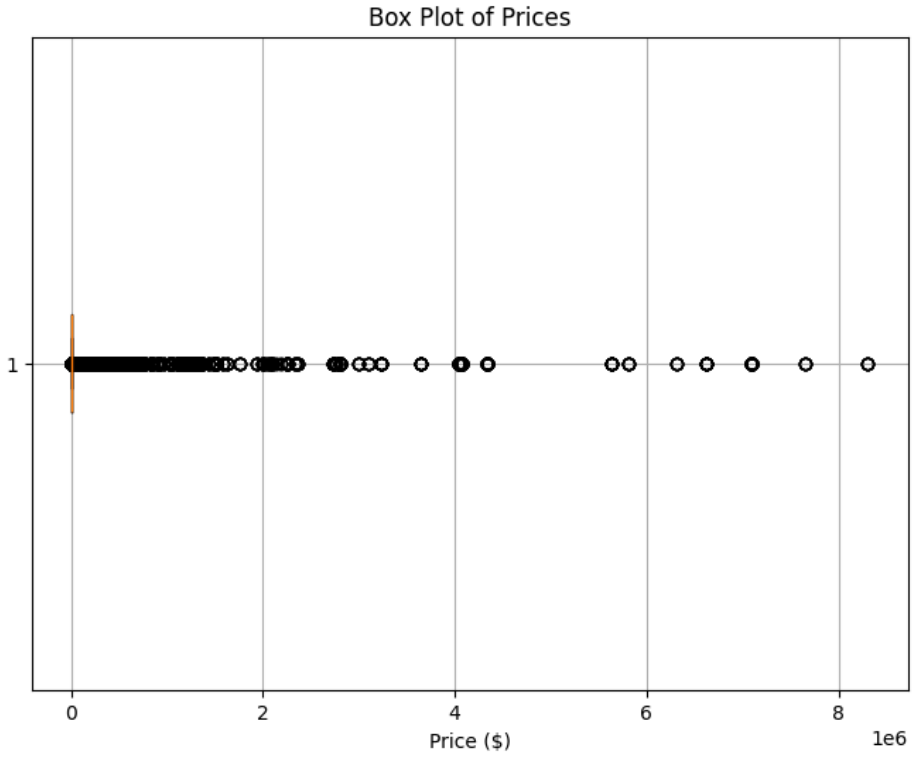


Figure 1: Box Plot of Prices before trimming

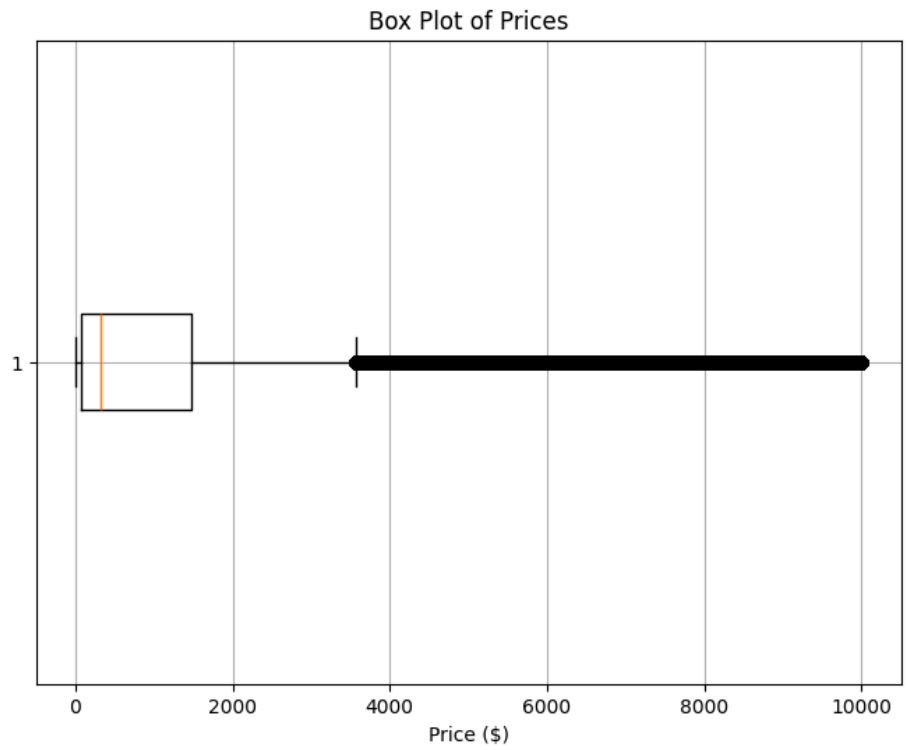


Figure 2: Box Plot of Prices after trimming

To further improve interpretability, we applied an additional filter to include only prices below $6,500, narrowing the range to where the majority of values were concentrated. This threshold helped reduce the influence of the long right tail while preserving most of the relevant data. A histogram of the trimmed prices was then plotted to guide the next steps in feature transformation and modeling. The resulting distribution showed a strong right skew, with the highest concentration of prices between $0 and $1000, peaking near the median. This confirmed that while some variation remained, the majority of billing values clustered tightly in a practical and interpretable range.

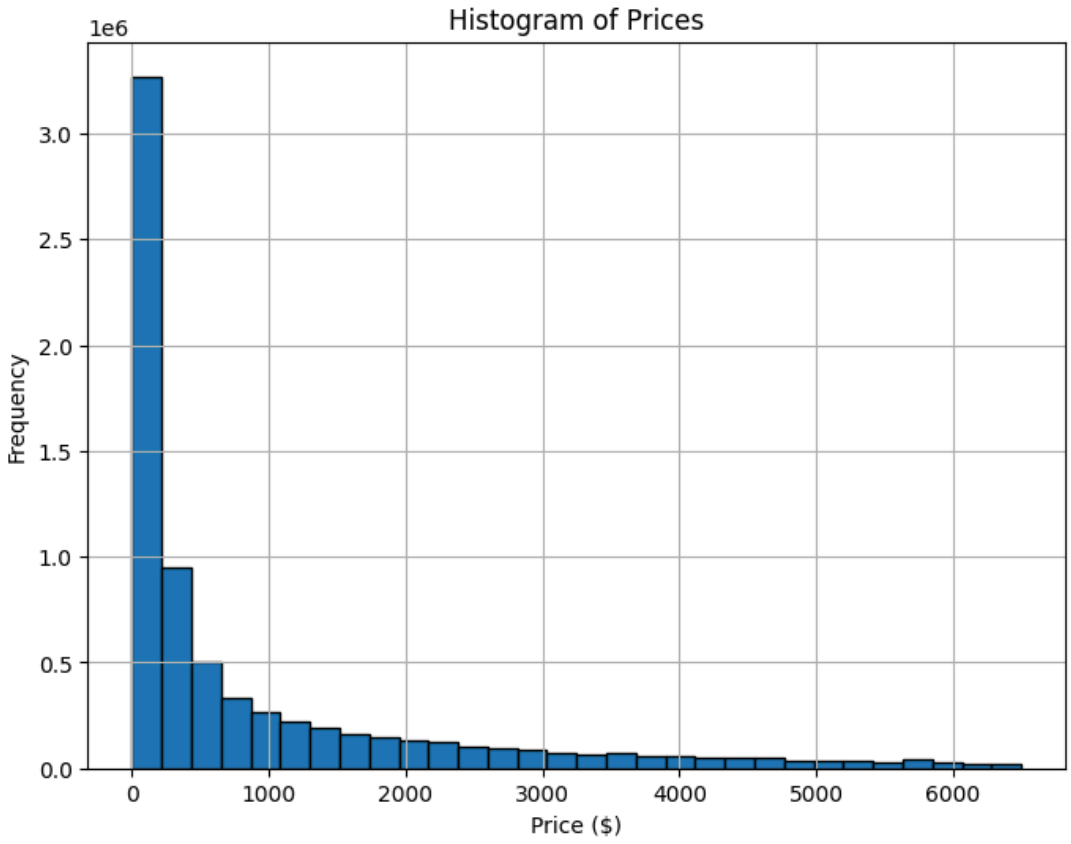


Figure 3: Histogram of prices trimmed to below $6,500.

5. Methodology

The shape of our merged dataset was (728607, 5).Before preprocessing, we analyzed the merged dataset's structure and composition (e.g., cardinality, null values, and data distribution) to identify redundant or high-variance columns that could negatively impact model training. The merged dataset had a shape of (728,607, 5), representing trimmed price entries from MI, FL, and TX hospitals enriched with CPT/HCPCS procedure descriptions.

5.1 Data Preparation & Preprocessing

This section outlines the systematic steps taken to clean and transform the merged dataset(prices + hospital + cpt_hcpcs), ensuring it is structured and suitable for machine learning models. With the dataset now narrowed down to hospitals from MI, FL, and TX and price values trimmed to ≤

$10,000, the sections below outline the steps taken to prepare the data for modeling. This phase starts from the merged dataset of trimmed prices and CPT/HCPCS descriptions.

5.1.1 Filtering by Most Frequent Procedure Code

To reduce variability and control for procedural differences in pricing, we examined the frequency of medical procedures using **short_description**. From our analysis we decided to focus on "**HC REMOVAL OF DAMAGED SKIN AND UNDERLYING TISSUE**" not only due to its frequency (2,512 entries) but also because it reflects a common, standardized medical intervention, ideal for comparative cost analysis across states and insurers.

After that, we filtered the merged dataset to only include rows corresponding to this code. This ensured a focused analysis of pricing differences across payer types and states for a single procedure.

5.1.2 Re-encoding Payer Categories

Even within the filtered dataset, **payer_grouped** included low-frequency categories. To prevent high dimensionality and model bias, we retained only the five most common payer categories: **CASH, UHC, HUMANA, AETNA, and PRIORITY HEALTH OP RATE**. Although the grouped **OTHER** category appeared frequent, it was excluded since it aggregated many rare payers, offering little interpretive value and potentially distorting model learning.

This decision was supported by a cumulative distribution plot of payer frequencies, showing diminishing returns beyond the top five. We encoded payer_grouped as an ordered categorical variable with **CASH** as the baseline and applied one-hot encoding, dropping **payer_CASH** to prevent multicollinearity

payer_grouped	count
OTHER	0.874204
CASH	0.968153
Aetna	0.980892
HUMANA	0.984873
AETNA PPO	0.988455
BCBS	0.991640
COFINITY	0.994825
PRIORITY HIX	0.997611
AETNA	0.999602
PRIORITY HEALTH OP RATE	1.000000

Table 3: Cumulative distribution scores for the top payers.

This was obtained by calculating the cumulative distribution scores for the top most common payers. All remaining categories were grouped under 'OTHER. Converted **payer_grouped** into an ordered categorical variable, with **CASH** as the baseline. Applied one-hot encoding and dropped the **payer_CASH** column to prevent multicollinearity.

5.1.3 One-Hot Encoding of State

The state variable, already filtered to three values (MI, FL, TX), was then one-hot encoded. This generated state_MI, state_FL, and state_TX columns, representing each hospital's location. All encoded dummy variables were then cast to integers for model compatibility.

5.1.4 Dropping Redundant Fields

The **short_description** column was dropped, as it was identical across all entries after filtering. Keeping constant features can add noise and mislead some models (e.g., tree-based regressors)

by falsely suggesting useful splits. This left us with: code, price, and one-hot encoded payer and state variables.

5.1.5 Target Transformation

Even after trimming, **price** remained right-skewed. To reduce heteroscedasticity and improve model performance, we applied a log transformation. This transformed the price variable into a more normally distributed target (**log_price**) for regression models. We used **np.log1p(price)** to log-transform the price, which handles zero and small values more gracefully than **np.log.**

5.2 Feature Selection

At the end of preprocessing, the dataset was structured as follows:

- Features:
    - 1 numerical column (**code**)
    - 5 one-hot encoded payer columns
    - 3 one-hot encoded state columns

      Total: 9 features
- Target:
    - log_price (log-transformed cost of the procedure)

These features formed the input matrix X, while the target variable y was **log_price**. This final structure enabled effective scaling, training, and evaluation using regression models. The decision to introduce log transformation of prices helps remove the skewness of the dataset as

seen in the figure below:

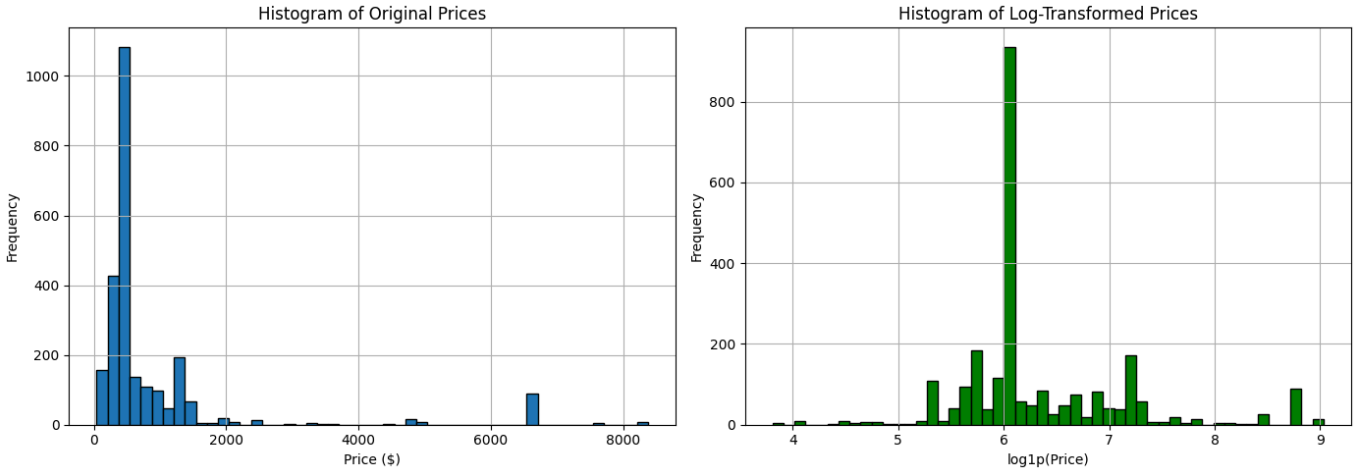


Figure 4: Price distribution before (left) and after (right) log transformation, showing reduced skewness.

5.3 Model Training

We applied three distinct regression models to predict hospital procedure prices: Linear Regression, Decision Trees, and a Neural Network with dense layers. Each model was trained on the same preprocessed dataset and evaluated using the following metrics:

Let:

- y_i: true value of the ii-th observation
- y^i: predicted value
- $\overline{y}$: mean of the true values
- n: number of observations

Mean Absolute Error (MAE)

- $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \widehat{y_i}\right|$

Mean Squared Error (MSE)

- $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2$

Coefficient of Determination (R^2)

- $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \widehat{y_i})2}{\sum_{i=1}^{n}(y_i - \bar{y})2}$

5.3.1 Linear Regression

We began with a Linear Regression model as a baseline due to its simplicity and interpretability. This model assumes a linear relationship between the input features (e.g., payer, state, procedure code) and the log-transformed target variable (**log_price**). Despite its limitations in capturing complex nonlinear interactions, it provides a strong foundation for evaluating feature contributions and quickly identifying underfit behavior.

5.3.1.1 Results and Observations.

Results:

The following results are based on both log-transformed prices and real prices:

R^2 (unitless)	**0.7415692134754555**
MSE (dollars^2)	**0.14751336840865417**

Table 4: Linear regression evaluation metrics on log-transformed prices.

R^2	**0.7274073244342987**
MSE (dollars^2)	**370782.86202567996**

RMSE (dollars)	608.9194216197083
MAE (dollars)	263.6246342305005

Table 5: Linear regression evaluation metrics on real prices.

Observations:

- The model captured basic trends but underfit high-price outliers. It's also evident that the model performed better on log prices because real prices contain skewed distribution.
- Prediction error increased with higher procedure prices.
- As a baseline, it provided useful insights into feature impacts via regression coefficients.

	Actual Price	Predicted Price	Error (Pred − Actual)	Absolute Error
0	1333.34	678.190689	−655.149311	655.149311
1	410.08	413.556703	3.476703	3.476703
2	932.00	758.390191	−173.609809	173.609809
3	410.08	413.556703	3.476703	3.476703
4	285.00	321.420090	36.420090	36.420090
5	410.08	413.556703	3.476703	3.476703
6	971.17	687.042554	−284.127446	284.127446
7	410.08	413.556703	3.476703	3.476703
8	410.08	413.556703	3.476703	3.476703
9	410.08	413.556703	3.476703	3.476703
10	6659.90	9745.731199	3085.831199	3085.831199
11	195.00	368.518444	173.518444	173.518444
12	6659.90	9745.731199	3085.831199	3085.831199
13	971.17	687.042554	−284.127446	284.127446
14	1407.75	867.964136	−539.785864	539.785864
15	410.08	413.556703	3.476703	3.476703
16	410.08	413.556703	3.476703	3.476703
17	212.00	515.725716	303.725716	303.725716
18	1108.69	614.795905	−493.894095	493.894095
19	410.08	413.556703	3.476703	3.476703

Figure 5 : Actual vs Predicted Prices after training with a linear regression model

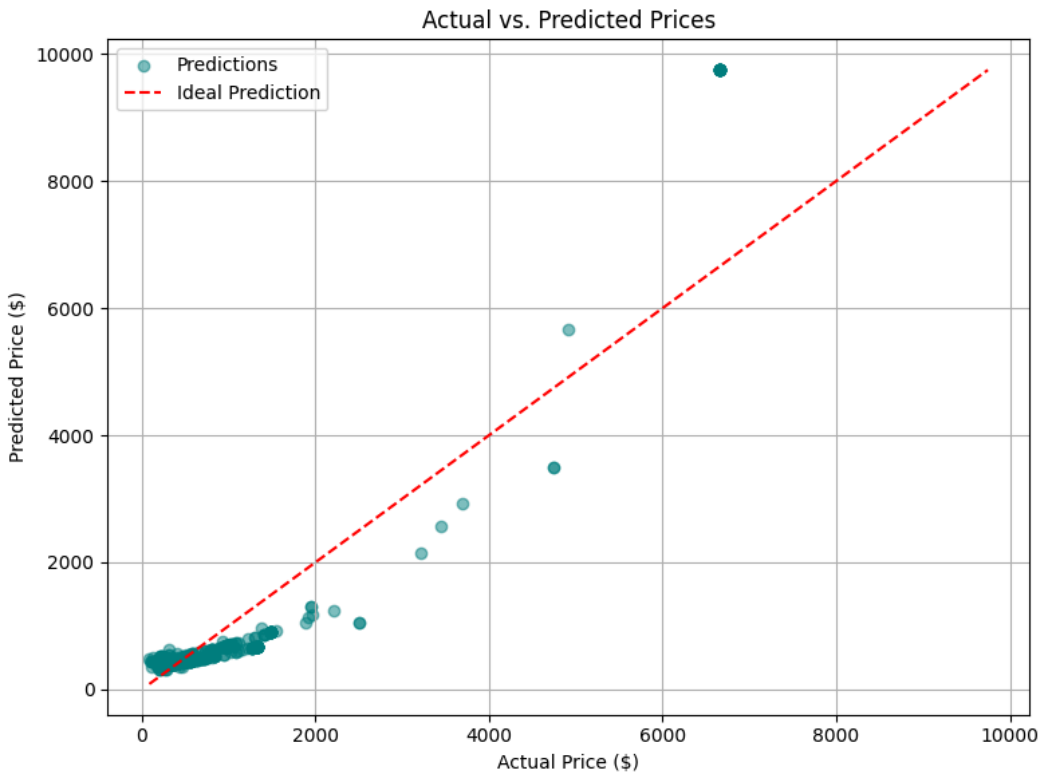


Figure 6: Scatter plot showing actual vs predicted prices using a linear regression model.

However, from the results above, it seemed that a quadratic regression model would best explain our model. Hence we proceeded to train using a quadratic regression model as described by the equation below:

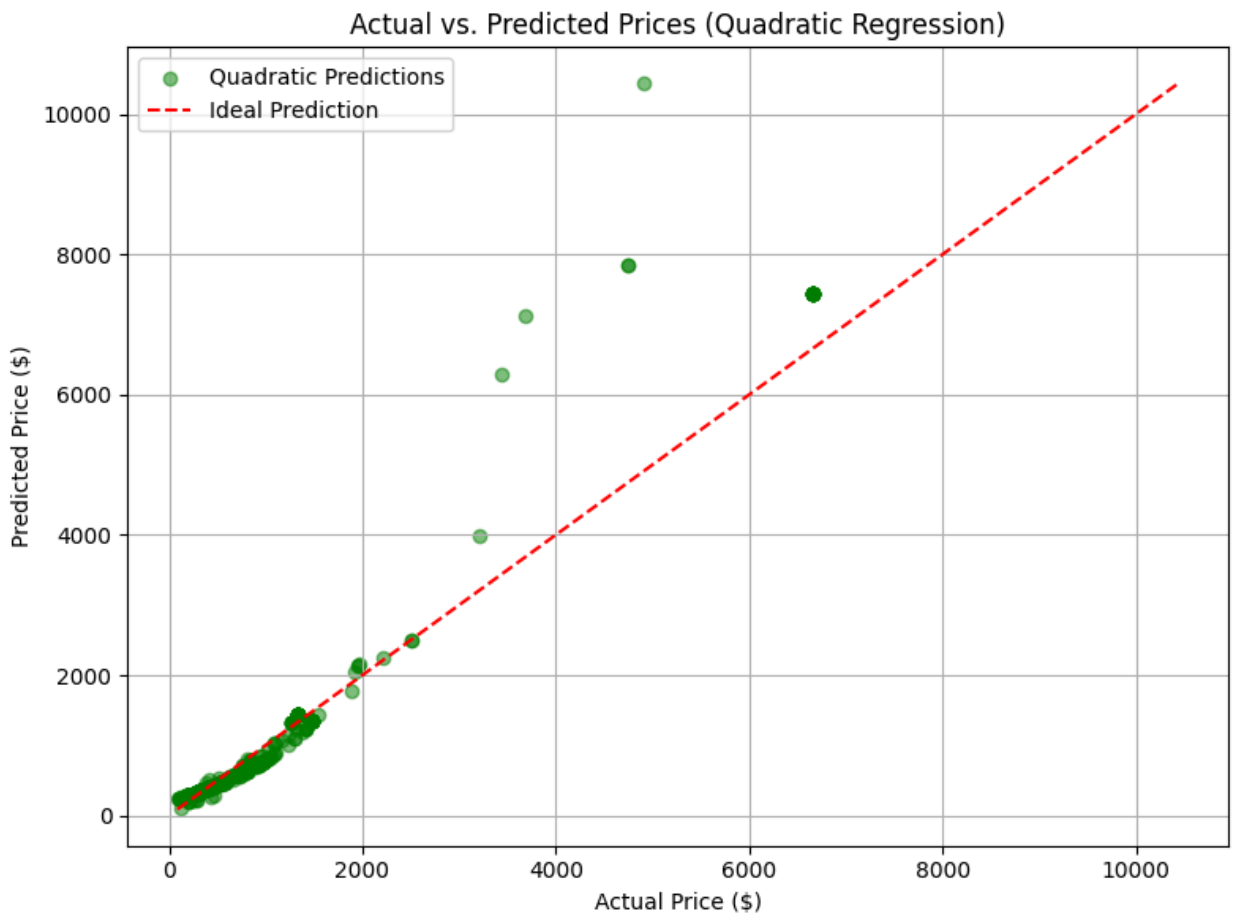


Figure 7: Scatter plot showing actual vs predicted prices using a quadratic regression model.

The dense cluster of green points at the bottom-left (prices under ~$2,000) shows that the model is accurate for the majority of procedures. For higher actual prices (right side of the plot, e.g. $4,000+), the green dots increasingly fall below the red line meaning the model is underpredicting high-price procedures. These results match with our previous analysis where we focused on a subset of the prices dataset due to outliers detected on higher prices. This can be referred to the box plot generated earlier. The quadratic regression model was trained as follows:

- Linear terms (e.g., code, price, payer_AETNA)
- Squared terms (e.g., $price^2$, $payer\_OTHER^2$)
- Interaction terms (e.g., price * state_MI)

The equation:

- $\log(\text{price}) = \beta_0 + \sum_i \beta_i x_i + \sum_j \beta_{jj} x_j^2 + \sum_{k,l} \beta_{kl} x_k x_l$
- After computing log(price), you transform it back to actual price:
- $\hat{y} = \exp(\log(\text{price})) - 1$

From the model we can also observe that the categorical variables such as payer_AETNA and state_MI showed weak individual influence, but their interactions with price revealed more. For example, the price*AETNA term (+0.1119) increased price predictions for AETNA patients, while price*state_FL (–0.0260) suppressed them in Florida. This means that being AETNA *amplifies* the effect of price whereas being in Florida meant less prices.

5.3.2 Decision Trees

Next, we trained a Decision Tree Regressor, which partitions the feature space into regions based on learned splits. Decision trees are capable of modeling nonlinear relationships and interactions between features without the need for explicit feature engineering. This makes them especially valuable for datasets with categorical variables, such as one-hot encoded payers and states. Furthermore, they provide intuitive interpretability through their tree structure.

5.3.2.1 Results and Observations.

Results:

The following results are based on real prices:

R^2 (unitless)	**0.9998419506331769**
MSE (dollars^2)	**214.98008503127645**
MAE (dollars)	**5.0072435714323955**

Table 6: Decision tree evaluation metrics on real prices.

Observations:

- The model achieved nearly perfect accuracy on real prices because they naturally handle non-linear relationships well and are less sensitive to skewed distributions. This is why we didn't train on log prices.
- However, the exceptionally low error suggests overfitting.
- While interpretability is high at shallow depth, the model becomes harder to explain as complexity increases.

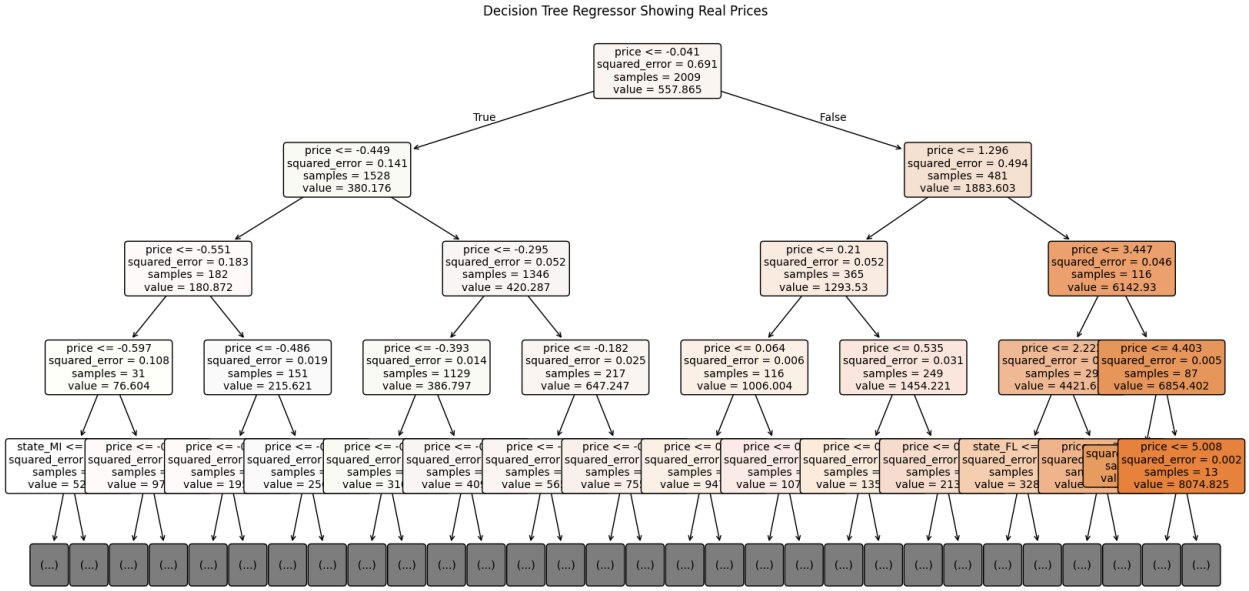


Figure 8: Decision tree regressor showing real prices.

5.3.3 Feedforward Neural Networks(FNNs)

To model complex and nonlinear relationships between hospital pricing and the input features (e.g., payer, state, procedure code), we implemented a feedforward neural network, also known as a Multilayer Perceptron (MLP).

This architecture consists of:

- **Input Layer:** Matching the number of features (one-hot encoded payers, states, and procedure codes)
- **Hidden Layers:** One or more dense (fully connected) layers with ReLU activation
- **Dropout Layers:** Included between dense layers to prevent overfitting
- **Output Layer:** A single neuron outputting the predicted **log_price**

The model was trained using:

- **Loss function:** Mean Squared Error (on log-transformed prices)
- **Optimizer:** Adam
- **Regularization:** Dropout and early stopping based on validation loss

5.3.3.1 Results and Observations.

Results:

The following results are based on log-transformed prices:

R^2 (unitless)	**0.691**
MSE (dollars^2)	**0.2304**
MAE (dollars)	**0.4166**

Table 7: Neural networks evaluation metrics on log-transformed prices.

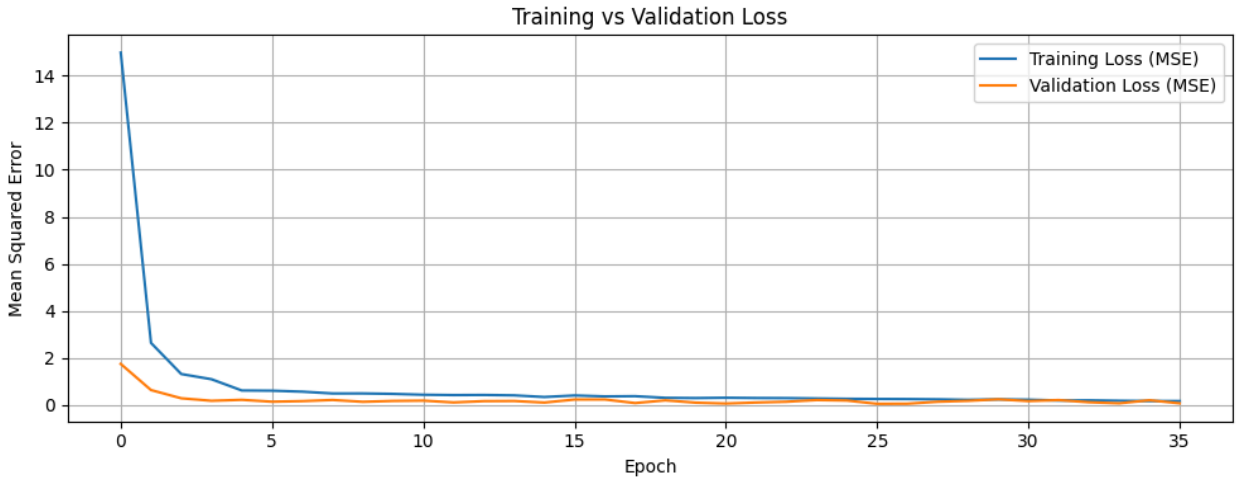


Figure 9: Training vs validation loss curve for MSE based on log-transformed prices.

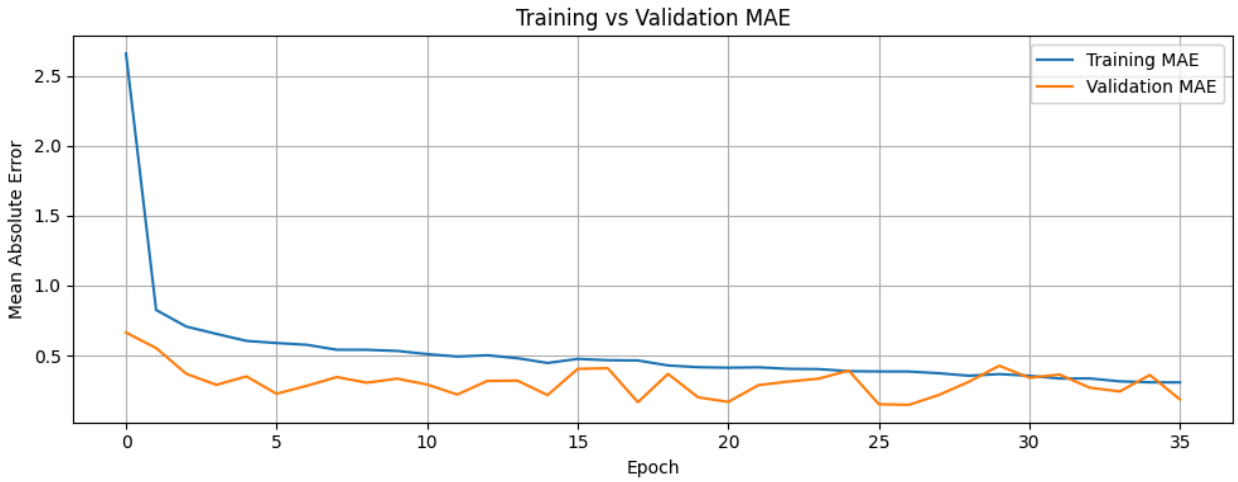


Figure 10: Training vs validation loss curve for MAE based on log-transformed prices.

Observations:

- The neural network captures nonlinear patterns and features interactions more effectively than linear regression.
- Neural networks performed best on log-transformed prices, where stabilized gradients and reduced outlier influence led to improved learning and generalization.
- Unlike the decision tree, it did not overfit the training data, and maintained robust generalization across a wide range of prices.
- The model slightly underpredicted very high prices, but performed well across the most common price range.
- Validation and training loss curves showed steady convergence, indicating a well-regularized model.

After evaluating the overall performance of the neural network, we further stratified our predictions by state and payer to extract interpretable pricing patterns.

A summary table of the test data grouped by state and payer shows the following:

- CASH payers in Michigan had the highest mean price ($4,322), nearly three times higher than in Texas, highlighting significant geographic variation in healthcare charges for the same procedure.
- Florida's CASH payers exhibited the highest price variance, with a standard deviation of $944, suggesting inconsistent hospital billing practices in that state. Despite this, the model performed reasonably well, approximating the mean closely.
- "OTHER" payers were the most frequent in both Michigan and Texas and had the lowest mean prices, which may reflect self-pay discounts, low-cost plan structures, or lack of strong bargaining power.

The figure below illustrates average prices for each payer group across MI, FL, and TX:

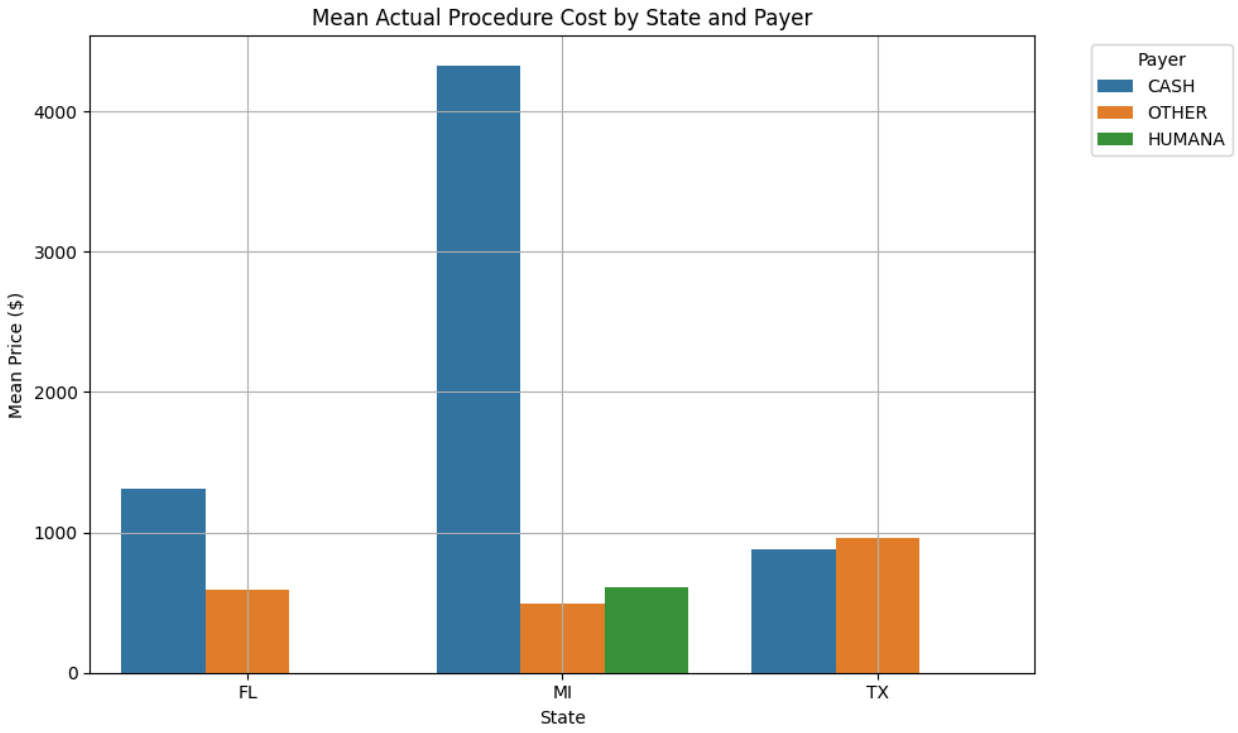


Figure 11 : Mean Actual Procedure Cost analysis by State and Payer from FNN.

This analysis provides real-world interpretability, revealing how hospital pricing behavior varies across states and insurers, and underscores the model's potential to support equitable cost forecasting.

6. Hyperparameter Usage

In this section, we detail the hyperparameters used for each of the machine learning models and explain the rationale behind their selection.

6.1 Linear Regression

- Hyperparameters: None (ordinary least squares)
- Tuning method: Not applicable
- Linear regression served as a baseline model and does not require hyperparameter tuning.

6.2 Decision Tree Regressor

- max_depth = 6
  Chosen to balance model interpretability and accuracy. Deeper trees tended to overfit, while shallower ones underfit.
- random_state = 42
  Ensured reproducibility of results.
- criterion = 'squared_error' (default)
  Measures the quality of a split using variance reduction.

This configuration resulted in extremely high accuracy ($R^2 \approx 0.999$), but with signs of overfitting due to the model's ability to memorize price segments precisely.

6.3 Quadratic Regression

- degree = 2 in PolynomialFeatures
- include_bias = False
  Prevents adding a constant term twice (already handled by the linear regression model).

This approach allowed the model to capture nonlinear relationships, particularly those involving diminishing effects of price and interaction terms between payer and state variables. It significantly reduced the mean absolute error from 263.6 to 118.7.

6.4 Feedforward Neural Network (MLP)

- Architecture:
    - Dense(256, ReLU)
    - Dropout(0.3)
    - Dense(128, ReLU)
    - Dropout(0.3)
    - Dense(64, ReLU)
    - Dense(1) (output node for regression)
- Optimizer: Adam
- Loss Function: Mean Squared Error (MSE)
- Metrics: Mean Absolute Error (MAE)
- Batch Size: 32
- Epochs: 100 (with early stopping)
- Early Stopping: patience=10, used to prevent overfitting by restoring the best weights based on validation loss.

These settings were chosen through manual experimentation. The dropout layers helped reduce overfitting, and early stopping ensured the model didn't train longer than necessary. The neural network performed well with an R^2 of 0.69 and MAE of ~$203, although it underperformed compared to the quadratic model.

6.5 Scaling and Log Transformation

- Scaler: StandardScaler

  Applied to all features before feeding into models sensitive to feature scale (e.g., neural networks, regression).
- Target Transformation:

- Applied np.log1p(price) to stabilize variance and normalize price distribution.
- Inverse-transformed predictions using np.expm1() for interpretability and evaluation in dollar terms.

7. Conclusion

In this project, we set out to predict the cost of a specific outpatient medical procedure—subcutaneous tissue debridement—across hospitals in the United States using a machine learning approach. Our methodology began with integrating three large datasets covering hospital metadata, procedure pricing, and CPT/HCPCS codes. We focused our analysis on a single, high-frequency procedure (CPT code 11042) to reduce variability and ensure consistency across examples. Careful data cleaning, feature selection, and preprocessing including price outlier trimming, one-hot encoding of categorical features, and log transformation of the target variable—prepared the dataset for model training.

We evaluated three regression models: a linear regression baseline, a quadratic regression to account for feature interactions, and two nonlinear models—decision trees and a feedforward neural network (FNN). The linear model achieved moderate performance ($R^2 \approx 0.74$ on log-transformed prices), capturing general trends but underperforming on higher-cost procedures. The quadratic model improved upon this by incorporating interaction and squared terms, allowing for more nuanced predictions. The decision tree regressor yielded near-perfect performance ($R^2 \approx 0.999$), but this came with clear signs of overfitting. In contrast, the neural network offered the best trade-off between accuracy and generalization on log-transformed prices by achieving an R^2 of approximately 0.691 and outperforming the linear model in both mean squared and absolute error. These results suggest that while simpler models provide useful interpretability, more flexible architectures are better equipped to capture the complexity of real-world healthcare pricing data.

To better understand how the model performs across different payer and regional conditions, we stratified our predictions by state and payer. This revealed substantial insights into how pricing varies geographically:

- Michigan exhibited the highest CASH prices, with an average cost of $4,322, nearly three times higher than in Texas. This reflects significant geographic variability in baseline charges for the same procedure.
- Florida's CASH prices showed the widest standard deviation, suggesting inconsistent hospital billing practices across facilities in the state. Although the neural network accurately captured the average, this high variance indicates lower prediction reliability at the individual hospital level.
- "OTHER" payers were the most common in both Michigan and Texas, and consistently associated with lower prices, possibly representing self-pay or less-negotiated rates.

These findings highlight that while predictive performance is important, model interpretability and stratified analysis are crucial for drawing real-world insights. In particular, the ability to surface payer- and region-specific pricing behavior could inform future transparency efforts and policy design aimed at equitable healthcare access.

Future work could involve expanding the pipeline to include more procedures, incorporating hospital-level quality and capacity data to better explain prediction drivers at both global and individual levels. These insights can inform future transparency initiatives and pricing policy decisions.

8. Future Directions

While our current modeling approach produced strong results, there are several opportunities for improvement and extension.

First, our decision to focus on a single procedure was intentional, allowing us to reduce noise and explore pricing variation in a controlled setting. However, expanding the scope to include multiple procedures would increase the practical relevance of the model. This could involve adding a classification layer to distinguish between procedure types or using multitask learning to predict both procedure category and price simultaneously.

Second, although grouping rare payers into an "OTHER" category simplified the feature space, more sophisticated approaches, such as clustering similar payers or learning dense vector

representations (embeddings), could preserve additional structure without increasing dimensionality. This would be especially helpful for capturing nuanced relationships in payer behavior.

Third, future iterations of this work could benefit from incorporating cross-validation and hyperparameter tuning to optimize model performance and prevent overfitting, particularly in more flexible models like decision trees and neural networks. Ensemble methods such as Random Forests or Gradient Boosted Trees (e.g., XGBoost) are also promising alternatives that balance predictive power with interpretability.

Lastly, as healthcare models begin to influence decision-making at scale, transparency and fairness become essential. Such steps would not only strengthen the reliability of the models but also ensure that their deployment aligns with ethical standards in healthcare analytics.

References

[1] Health System Tracker. (2024, December 20). *How has U.S. spending on healthcare changed over time?* Retrieved May 8, 2025, from https://www.healthsystemtracker.org/chart-collection/u-s-spending-healthcare-changed-time/#Total%20national%20health%20expenditures,%201970-2023

[2] Office of the Assistant Secretary for Planning and Evaluation. (2021). *Evidence on surprise billing: Protecting consumers with the No Surprises Act* (Issue Brief HP-2021-24). U.S. Department of Health and Human Services. Retrieved May 8, 2025, from https://aspe.hhs.gov/sites/default/files/documents/acfa063998d25b3b4eb82ae159163575/no-surprises-act-brief.pdf

[3] Lopes, L., Montero, A., Presiado, M., & Hamel, L. (2024, March 1). *Americans' challenges with health care costs*. KFF. Retrieved May 8, 2025, from https://www.kff.org/health-costs/issue-brief/americans-challenges-with-health-care-costs/

[4] Malik, S. A. (2025). Enhancing healthcare cost transparency: Assessing implementation challenges, criticisms, and alternative solutions. *Frontiers in Health Services, 4*. https://doi.org/10.3389/frhs.2024.1379416

```
# # Run this if running for the first time
# !wget -O cpt_hcpcs.csv "https://www.dropbox.com/scl/fi/v0tbs80ei3jrio3bwtnkv/cp
# !wget -O hospitals.csv "https://www.dropbox.com/scl/fi/x8lb76j3yjb6954ckq2yr/ho
# !wget -O prices.csv "https://www.dropbox.com/scl/fi/njx9t1dg71vqk45z6frlz/price
```

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
prices = pd.read_csv("/content/prices.csv")
cpt_hcpcs = pd.read_csv("/content/cpt_hcpcs.csv")
hospitals = pd.read_csv("/content/hospitals.csv")
```

```
<ipython-input-3-8c9258fdea1a>:1: DtypeWarning: Columns (1) have mixed types.
  prices = pd.read_csv("/content/prices.csv")
<ipython-input-3-8c9258fdea1a>:2: DtypeWarning: Columns (2) have mixed types.
  cpt_hcpcs = pd.read_csv("/content/cpt_hcpcs.csv")
```

New Section

```
print("Prices columns: ", prices.columns)
print("CPT HCPCS columns: ", cpt_hcpcs.columns)
print("Hospitals columns: ", hospitals.columns)
```

```
Prices columns:  Index(['code', 'npi_number', 'payer', 'price'], dtype='objec
CPT HCPCS columns:  Index(['code', 'short_description', 'long_description'], 
Hospitals columns:  Index(['npi_number', 'name', 'url', 'street_address', 'ci
       'zip_code', 'publish_date'],
      dtype='object')
```

```
prices.tail()
```

	code	npi_number	payer	price
72724847	nan,99	1003858408	CASH	83.34
72724848	package	1063969160	CASH	22.00
72724849	package	1063969160	KAISER FOUNDATION HEALTH PLAN OF THE NORTHWEST	15.40
72724850	package	1598917866	CASH	22.00
72724851	package	1598917866	KAISER FOUNDATION HEALTH PLAN OF	15.40

```
hospitals.head()
```

	npi_number	name	url	street_addres
0	1003139775.0	HCA Virginia	https://hcavirginia.com/about/legal/pricing-tr...	901 E. Cary S Suite 21
1	1003260480	Brookwood Baptist Medical Center	https://www.brookwoodbaptisthealth.com/docs/gl...	2010 Brookwoo Medical Center D
2	1003281452	Henderson Hospital	https://uhsfilecdn.eskycity.net/ac/henderson-h...	1050 We Galleria Driv
		CHI Health		

Next steps: Generate code with hospitals | View recommended plots | New interactive sheet

```
cpt_hcpcs.head()
```

	code	short_description	long_description
0	00000A	DVC REVASC 6X20MM 200CM	NaN
1	00001U	RBC DNA HEA 35 AG PLA	NaN
2	00001U,1	RBC DNA HEA 35 AG PLA	NaN
3	00013	PT INDIVIDUAL GYM	NaN
4	0001A	HC ADM PFIZER SARSCOV2 30MCG/0.3ML 1ST	NaN

```
prices.info()
prices.describe()
# Display as a table
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72724852 entries, 0 to 72724851
Data columns (total 4 columns):
 #   Column      Dtype  
---  ------      -----  
 0   code        object 
 1   npi_number  object 
 2   payer       object 
 3   price       float64
dtypes: float64(1), object(3)
memory usage: 2.2+ GB
```

	price
count	7.242092e+07
mean	4.226021e+03
std	3.169609e+05
min	-3.958500e+04
25%	1.310000e+02
50%	5.420900e+02
75%	2.197130e+03
max	9.990070e+08

```
hospitals.info()
hospitals.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1400 entries, 0 to 1399
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   npi_number      1400 non-null   object
 1   name            1400 non-null   object
 2   url             1400 non-null   object
 3   street_address  1351 non-null   object
 4   city            1379 non-null   object
 5   state           1377 non-null   object
 6   zip_code        1357 non-null   object
 7   publish_date    610 non-null    object
dtypes: object(8)
memory usage: 87.6+ KB
```

	npi_number	name	url	street_address	ci
count	1400	1400	1400	1351	13
unique	1400	1176	1070	1057	8
top	1992931109	WILLIAM BEAUMONT HOSPITAL	https://www.beaumont.org/patients-families/bil...	3601 W 13 MILE RD	ROY O/

```
cpt_hcpcs.info()
cpt_hcpcs.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3287818 entries, 0 to 3287817
Data columns (total 3 columns):
 #   Column             Dtype 
---  ------             ----- 
 0   code               object
 1   short_description  object
 2   long_description   object
dtypes: object(3)
memory usage: 75.3+ MB
```

	code	short_description	long_description
count	3287817	3177836	181429
unique	3287817	308652	99668
top	package	NJX INTERLAM	Measurement of antibody (IgE) to allergic subs...
freq	1	1367	121

```
# What states occur more frequently?
# Assuming your hospitals dataframe is called `hospitals`
state_counts = hospitals['state'].value_counts()

# Display the top states
print(state_counts.head(10))  # or use .to_string() for prettier output
```

```
state
MI    188
FL    180
TX    163
CA    107
WI     56
IN     51
AL     37
SD     36
MN     36
TN     29
Name: count, dtype: int64
```

```
# Decided to pick the top 3 states, MI, FL and TX to ensure our dataset is balanc
# Step 1: Keep only hospitals in MI, FL, and TX
hospitals = hospitals[hospitals['state'].isin(['MI', 'FL', 'TX'])]


# Step 2: Drop irrelevant or high-cardinality columns
hospitals = hospitals.drop(['publish_date', 'url', 'street_address', 'city', 'nam


# Step 3: Filter prices to only include those hospitals (by NPI)
# Because we filtered the hospital dataset to only include 3 states
target_npis = hospitals['npi_number'].unique()
prices = prices[prices['npi_number'].isin(target_npis)]


#  Merge prices with hospital info (state, type, ownership, etc.)
final = prices.merge(hospitals, on='npi_number', how='left')


final.tail()
```

	code	npi_number	payer	price	state	zip_code
7988858	j1745	1477643690	United Healthcare - Star-Kids MCD Outpatient	7108.28	TX	77030
7988859	j1745	1477643690	United Healthcare - Star-Plus MCD Inpatient	7680.63	TX	77030
7988860	j1745	1477643690	United Healthcare - Star-Plus MCD Outpatient	7680.63	TX	77030
7988861	j1745	1477643690	United Healthcare Inpatient	11816.36	TX	77030

```
# Drop rows with missing target (price)
final = final.dropna(subset=['price'])
```

```
# check for NaN values
print(final.isna().sum())
```

```
code            0
npi_number      0
payer           0
price           0
state           0
zip_code      1001
dtype: int64
```

```
# Drop zip code
final.drop('zip_code', axis=1, inplace=True)
```

```
# Drop npi_number
final.drop('npi_number', axis=1, inplace=True)
```

```
payer_counts = final['payer'].value_counts()
print(payer_counts)
```

```
payer
CASH                                                2074158
COFINITY                                              84491
PRIORITY HIX                                          76625
BCBS                                                  76248
AETNA PPO                                             75013
                                                      ...
BCBS OF TX PPO (BCTP)                                     1
FIRSTCARE EXCHANGE (FSTZ)                                 1
SOUTHWEST MEDICAL PRO NET (SMPN)                          1
FIRSTCARE (FST)                                           1
BCBSTX MEDICAID STAR/CHIP/STAR KIDS - (Medicaid)          1
Name: count, Length: 2028, dtype: int64
```

```
# A lot of unique payers, use the top 10 most common payers, and categorize the r
# as OTHER
top_payers = final['payer'].value_counts().nlargest(10).index
final['payer_grouped'] = final['payer'].apply(lambda x: x if x in top_payers else
final = final.drop('payer', axis=1)
```

```
payer_counts = final['payer_grouped'].value_counts()
print(payer_counts)
```

```
payer_grouped
OTHER                          5322039
CASH                           2074158
COFINITY                         84491
PRIORITY HIX                     76625
BCBS                             76248
AETNA PPO                        75013
PRIORITY HEALTH OP RATE          67746
HUMANA                           62518
AETNA                            55163
Aetna                            49508
COFINITY - AETNA OP RATE         45354
Name: count, dtype: int64
```

```
final.columns
```

```
Index(['code', 'price', 'state', 'payer_grouped'], dtype='object')
```

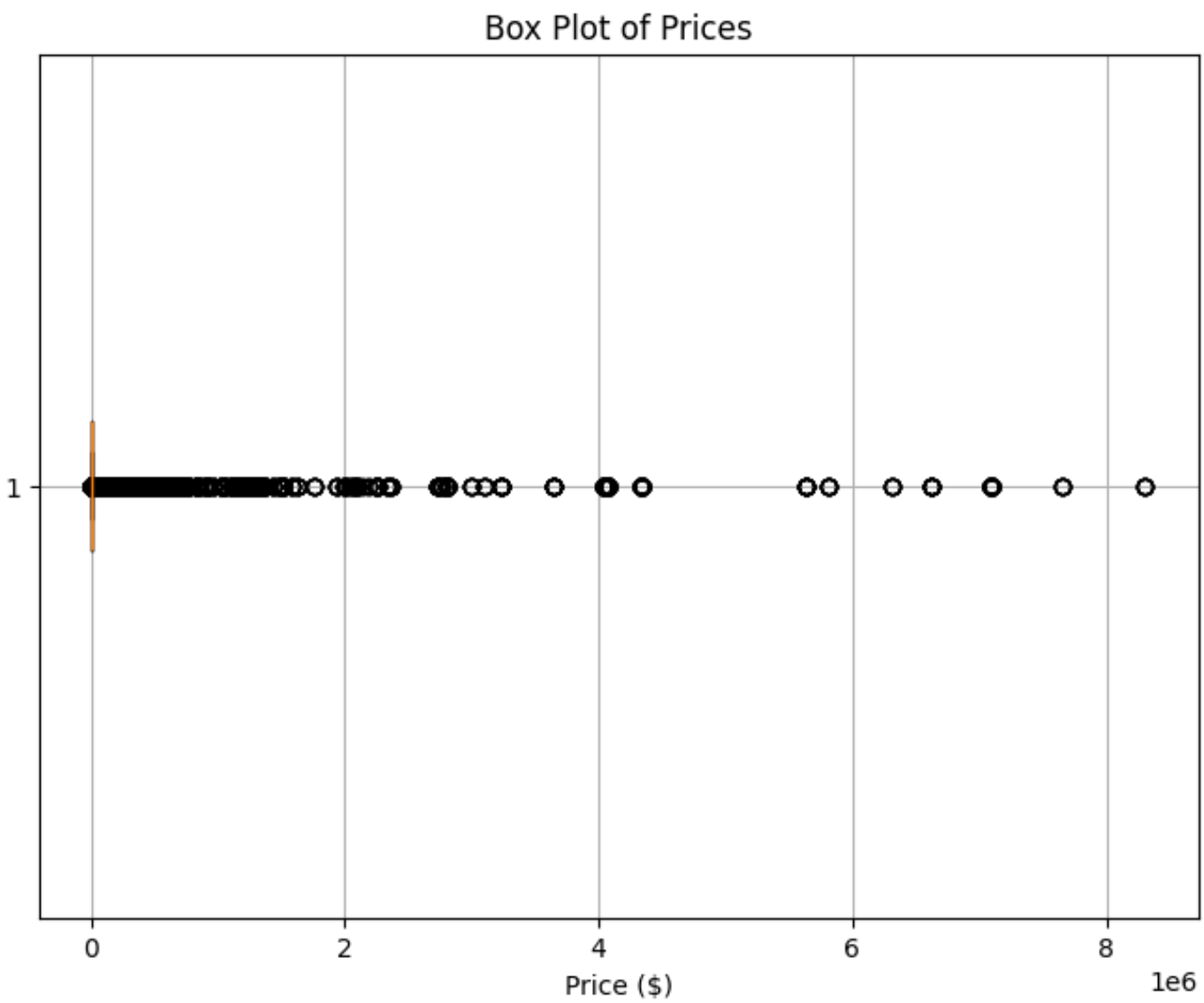
```
final.head()
```

	code	price	state	payer_grouped
0	00001U	296.00	FL	CASH
1	00001U	380.00	FL	CASH
2	00001U	380.00	FL	CASH
3	00001U	296.00	FL	CASH
4	0001M	228.44	MI	OTHER

```
price_counts = final['price'].value_counts()
print(price_counts)
```

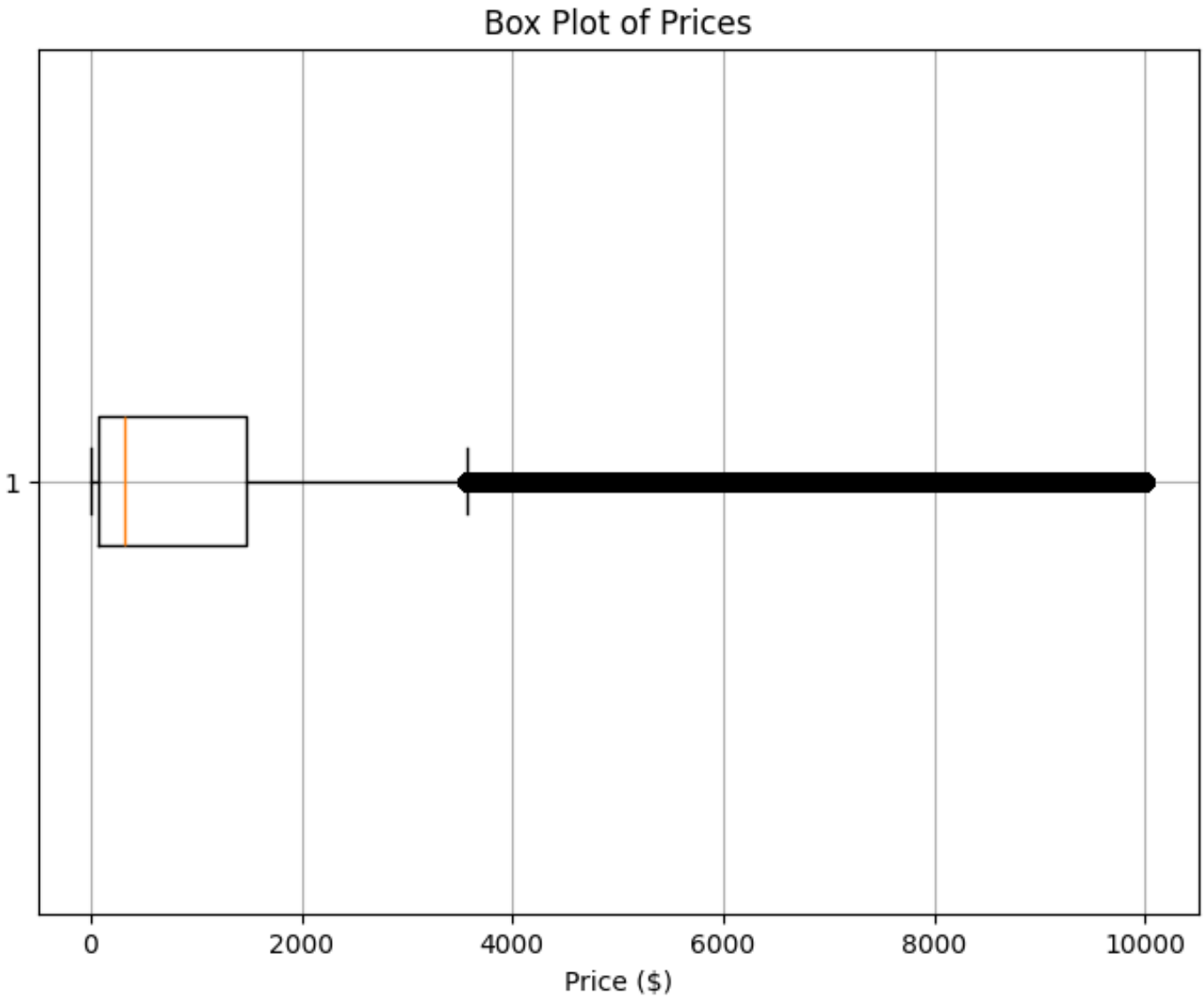
```
price
7.00       30754
8.00       26926
4.00       25400
5.00       25365
11.00      25339
           ...
3932.40        1
1109.65        1
713.63         1
770.35         1
776.37         1
Name: count, Length: 320909, dtype: int64
```

```
# Create a box plot
plt.figure(figsize=(8, 6))
plt.boxplot(final['price'], vert=False)
plt.title('Box Plot of Prices')
plt.xlabel('Price ($)')
plt.grid(True)
plt.show()
```




```
# Let's trim first and see the box plot
trimmed = final[(final['price'] <= 10000)]
```

```
# Create a box plot
plt.figure(figsize=(8, 6))
plt.boxplot(trimmed['price'], vert=False)
plt.title('Box Plot of Prices')
plt.xlabel('Price ($)')
plt.grid(True)
plt.show()
```

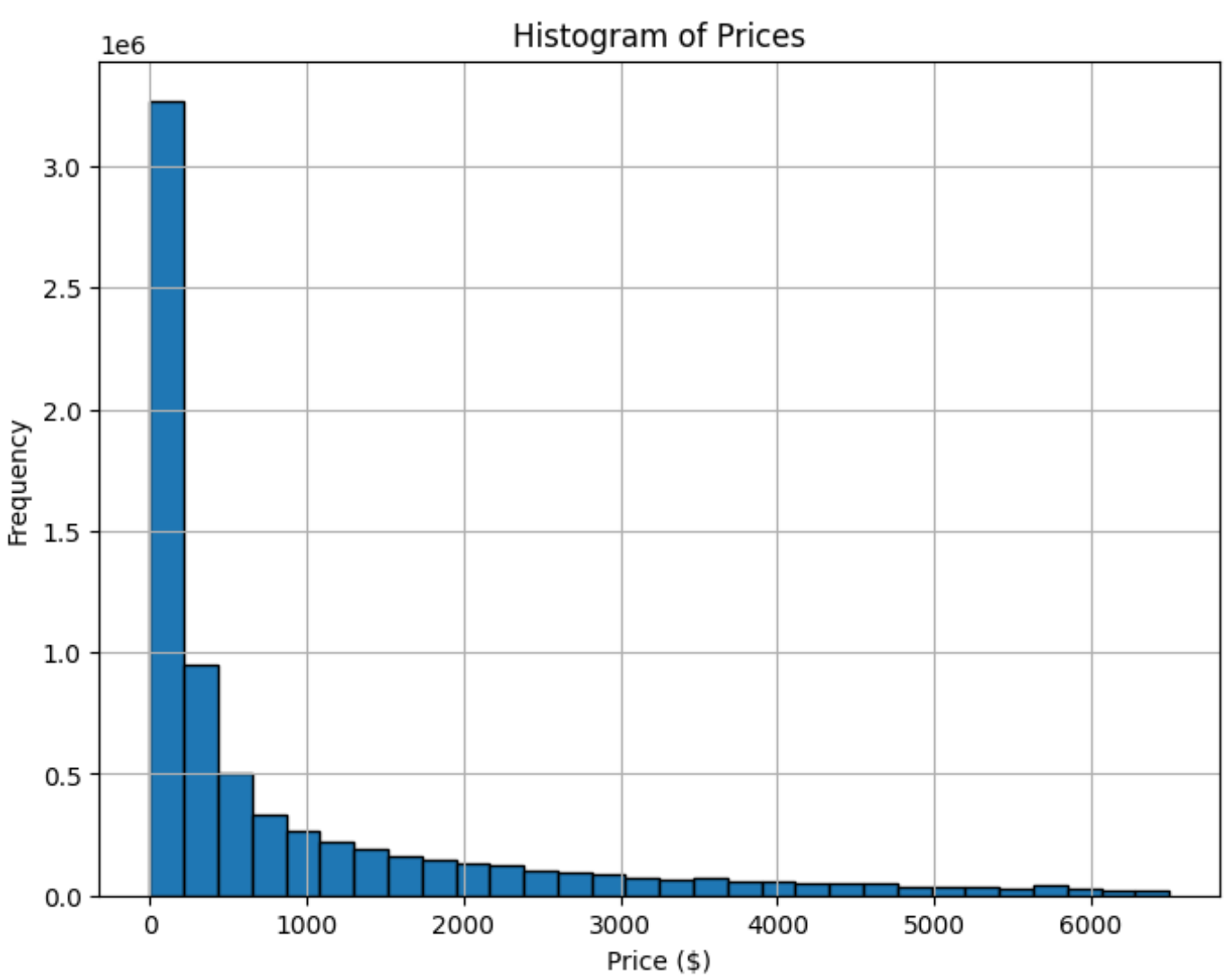


```
trimmed.head()
trimmed.info()
trimmed.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7601117 entries, 0 to 7988860
Data columns (total 4 columns):
 #   Column          Dtype  
---  ------          -----  
 0   code            object 
 1   price           float64
 2   state           object 
 3   payer_grouped   object 
dtypes: float64(1), object(3)
memory usage: 290.0+ MB
```

	price
count	7.601117e+06
mean	1.199952e+03
std	1.895321e+03
min	0.000000e+00
25%	6.940000e+01
50%	3.200000e+02
75%	1.471000e+03
max	1.000000e+04

```
plt.figure(figsize=(8, 6))
plt.hist(trimmed[trimmed['price'] <= 6500]['price'], bins=30, edgecolor='k')
plt.title('Histogram of Prices')
plt.xlabel('Price ($)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```




```
import numpy as np
np.shape(trimmed)
```

```
(7601117, 4)
```

```
# long description and short description are redundant, drop long description
# in addition most rows in long description columns in cpt_hcpcs are empty
cpt_hcpcs = cpt_hcpcs.drop('long_description', axis=1)


# DATA PREPARATION & PREPROCESSING
# Merge trimmed with CPT/HCPCS on 'code'
merged = trimmed.merge(cpt_hcpcs, on='code', how='left')


merged.head()
merged.info()
merged.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7601117 entries, 0 to 7601116
Data columns (total 5 columns):
 #   Column             Dtype  
---  ------             -----  
 0   code               object 
 1   price              float64
 2   state              object 
 3   payer_grouped      object 
 4   short_description  object 
dtypes: float64(1), object(4)
memory usage: 290.0+ MB
```

	price
count	7.601117e+06
mean	1.199952e+03
std	1.895321e+03
min	0.000000e+00
25%	6.940000e+01
50%	3.200000e+02
75%	1.471000e+03
max	1.000000e+04

```
np.shape(merged)
```

```
(7601117, 5)
```

```
shortdescription_counts = merged['short_description'].value_counts()

# Display the top short descriptions
print(shortdescription_counts.head(10))  # or use .to_string() for prettier outpu
```

```
short_description
 null                                              7103
Repair blood vessel lesion                         3292
COVID19                                            3135
HC IV INFUSION HYDRATION EACH ADDITIONAL HOUR      2961
HC CT THRC SPI C-MATRL                             2958
TC IMP IOL ALCON ACRYSOF SA/SN60AT ANY             2926
HC BILIRUBIN, TOTAL, FLD                           2896
ACETAMINOPHEN 160 MG/5 ML (5 ML) ORAL SOLUTION     2885
HC CRYOGLOBULIN ID                                 2877
HEPATITIS PANEL ACUTE                              2824
Name: count, dtype: int64
```

```
# Chose the description that occurs most frequently
# Filter rows where the short_description contains 'HC REMOVAL OF DAMAGED SKIN AN
merged = merged[merged['short_description' ] == 'HC REMOVAL OF DAMAGED SKIN AND UI

code_counts = merged['code'].value_counts()

# Display the top code counts
print(code_counts.head(10))  # or use .to_string() for prettier output
```

```
code
11042    2512
Name: count, dtype: int64
```

Figure out what columns to keep before merging the datasets! target variable = price descriptive variable = other variables

prices + hospital using npi_number cpt_hcpcs + (prices + hospital) using code

```
merged.head()
```

	code	price	state	payer_grouped	short_description
172237	11042	6659.9	MI	CASH	HC REMOVAL OF DAMAGED SKIN AND UNDERLYING TISSUE
172238	11042	900.9	TX	CASH	HC REMOVAL OF DAMAGED SKIN AND UNDERLYING TISSUE
172239	11042	1309.0	TX	Aetna	HC REMOVAL OF DAMAGED SKIN AND UNDERLYING TISSUE
					HC REMOVAL OF DAMAGED SKIN

Next steps: Generate code with merged | View recommended plots | New interactive sheet

```
# check for NaN values
print(merged.isna().sum())
```

```
code                0
price               0
state               0
payer_grouped       0
short_description   0
dtype: int64
```

```
payer_counts = merged['payer_grouped'].value_counts()

# Display the top code counts
print(payer_counts.head(10))  # or use .to_string() for prettier output
```

```
payer_grouped
OTHER                       2196
CASH                         236
Aetna                         32
HUMANA                        10
AETNA PPO                      9
BCBS                           8
COFINITY                       8
PRIORITY HIX                   7
AETNA                          5
PRIORITY HEALTH OP RATE        1
Name: count, dtype: int64
```

```
payer_counts = merged['payer_grouped'].value_counts()
payer_counts.cumsum() / payer_counts.sum()
```

payer_grouped	count
OTHER	0.874204
CASH	0.968153
Aetna	0.980892
HUMANA	0.984873
AETNA PPO	0.988455
BCBS	0.991640
COFINITY	0.994825
PRIORITY HIX	0.997611
AETNA	0.999602
PRIORITY HEALTH OP RATE	1.000000

dtype: float64

```
# group the top most 5 to avoid higher dimensionality later on
# and overfitting don't group OTHER because previously this was also a lumpsum of
# Hardcode your top payers in order, with CASH as the baseline
top_payers = ['CASH', 'UHC', 'HUMANA', 'AETNA', 'PRIORITY HEALTH OP RATE']

# Step 1: Categorize and lock order
merged['payer_grouped'] = pd.Categorical(
    merged['payer_grouped'],
    categories=top_payers + ['OTHER'],
    ordered=True
)

# Step 2: One-hot encode and drop CASH as baseline
payer_dummies = pd.get_dummies(merged['payer_grouped'], prefix='payer')
payer_dummies.drop('payer_CASH', axis=1, inplace=True)

# Step 3: Merge back
merged = pd.concat([merged.drop('payer_grouped', axis=1), payer_dummies], axis=1)

# Ensure all dummies are integers (for models)
merged[payer_dummies.columns] = merged[payer_dummies.columns].astype(int)
```

```
<ipython-input-43-e6f386305b87>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  merged['payer_grouped'] = pd.Categorical(
```

```
# also drop short description since they are now a one-to-one match with code
merged.drop('short_description', axis=1, inplace=True)
```

```
merged.head()
```

	code	price	state	payer_UHC	payer_HUMANA	payer_AETNA	payer_PRIORIT HEALTH OP RAT
172237	11042	6659.9	MI	0	0	0	
172238	11042	900.9	TX	0	0	0	
172239	11042	1309.0	TX	0	0	0	
172240	11042	455.0	TX	0	0	0	
172241	11042	305.0	TX	0	0	0	

Next steps: Generate code with merged View recommended plots New interactive sheet

```
# one-hot encode states
merged = pd.get_dummies(merged, columns=['state'])
```

```
merged.head()
```

	code	price	payer_UHC	payer_HUMANA	payer_AETNA	payer_PRIORITY HEALTH OP RATE	paye
172237	11042	6659.9	0	0	0	0	
172238	11042	900.9	0	0	0	0	
172239	11042	1309.0	0	0	0	0	
172240	11042	455.0	0	0	0	0	
172241	11042	305.0	0	0	0	0	

Next steps: Generate code with merged View recommended plots New interactive sheet

```
merged['state_FL'] = merged['state_FL'].astype(int)
merged['state_MI'] = merged['state_MI'].astype(int)
merged['state_TX'] = merged['state_TX'].astype(int)
```

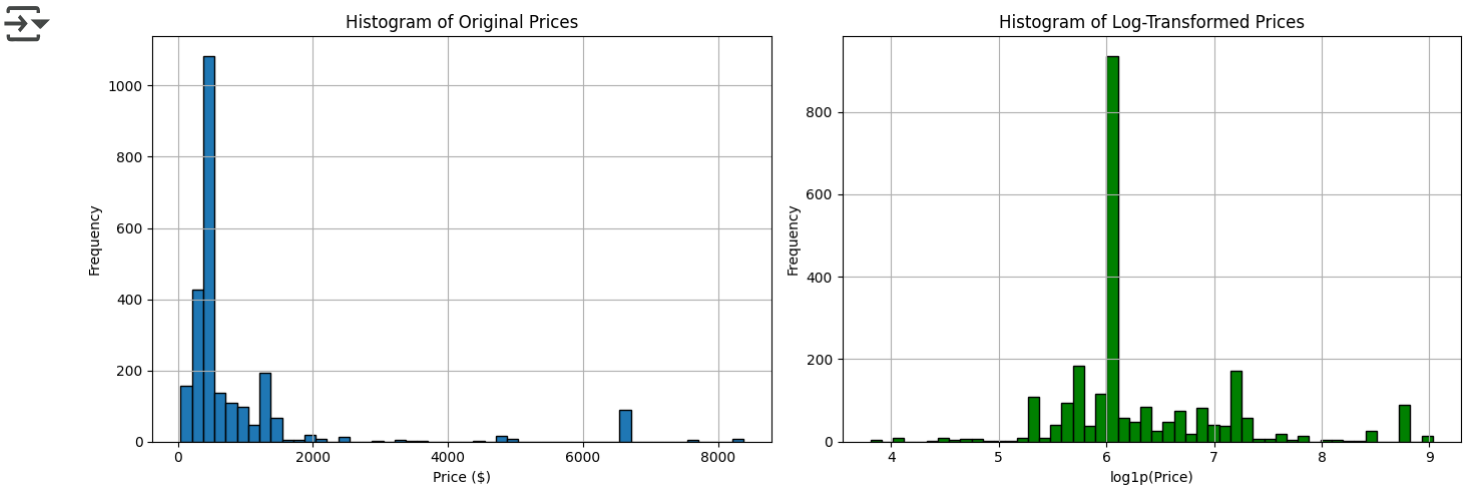
```
merged.head()
```

	code	price	payer_UHC	payer_HUMANA	payer_AETNA	payer_PRIORITY HEALTH OP RATE	paye
172237	11042	6659.9	0	0	0	0	
172238	11042	900.9	0	0	0	0	
172239	11042	1309.0	0	0	0	0	
172240	11042	455.0	0	0	0	0	
172241	11042	305.0	0	0	0	0	

Next steps: Generate code with merged | View recommended plots | New interactive sheet

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.tree import DecisionTreeRegressor


# X = all features, y = target (price)
# Replace 'price' with its log-transformed version
merged['log_price'] = np.log1p(merged['price'])


# Histogram for normal real vs log-transformed prices
# Create subplots
plt.figure(figsize=(14, 5))

# Histogram of original prices
plt.subplot(1, 2, 1)
plt.hist(merged['price'], bins=50, edgecolor='k')
plt.title('Histogram of Original Prices')
plt.xlabel('Price ($)')
plt.ylabel('Frequency')
plt.grid(True)

# Histogram of log-transformed prices
plt.subplot(1, 2, 2)
```

```
plt.hist(merged['log_price'], bins=50, edgecolor='k', color='green')
plt.title('Histogram of Log-Transformed Prices')
plt.xlabel('log1p(Price)')
plt.ylabel('Frequency')
plt.grid(True)

plt.tight_layout()
plt.show()
```




```
X = merged.drop('log_price', axis=1)
X.columns = X.columns.astype(str)
y = merged['log_price']
```

```
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

# Scale for all models
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# Linear Regression
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)

print("Linear Regression R²:", r2_score(y_test, y_pred_lr))
print("Linear Regression MSE:", mean_squared_error(y_test, y_pred_lr))
print("Linear Regression MAE:", mean_absolute_error(y_test, y_pred_lr))
```

```
Linear Regression R²: 0.7415692134754555
Linear Regression MSE: 0.14751336840865417
```

```
from sklearn.metrics import mean_absolute_error

# Convert log prices back to real prices
y_test_real = np.expm1(y_test)
y_pred_real = np.expm1(y_pred_lr)

# Real-world metrics
print("Real Price R²:", r2_score(y_test_real, y_pred_real))
print("Real Price MSE:", mean_squared_error(y_test_real, y_pred_real))
print("Real Price RMSE:", np.sqrt(mean_squared_error(y_test_real, y_pred_real)))
print("Real Price MAE:", mean_absolute_error(y_test_real, y_pred_real))
```

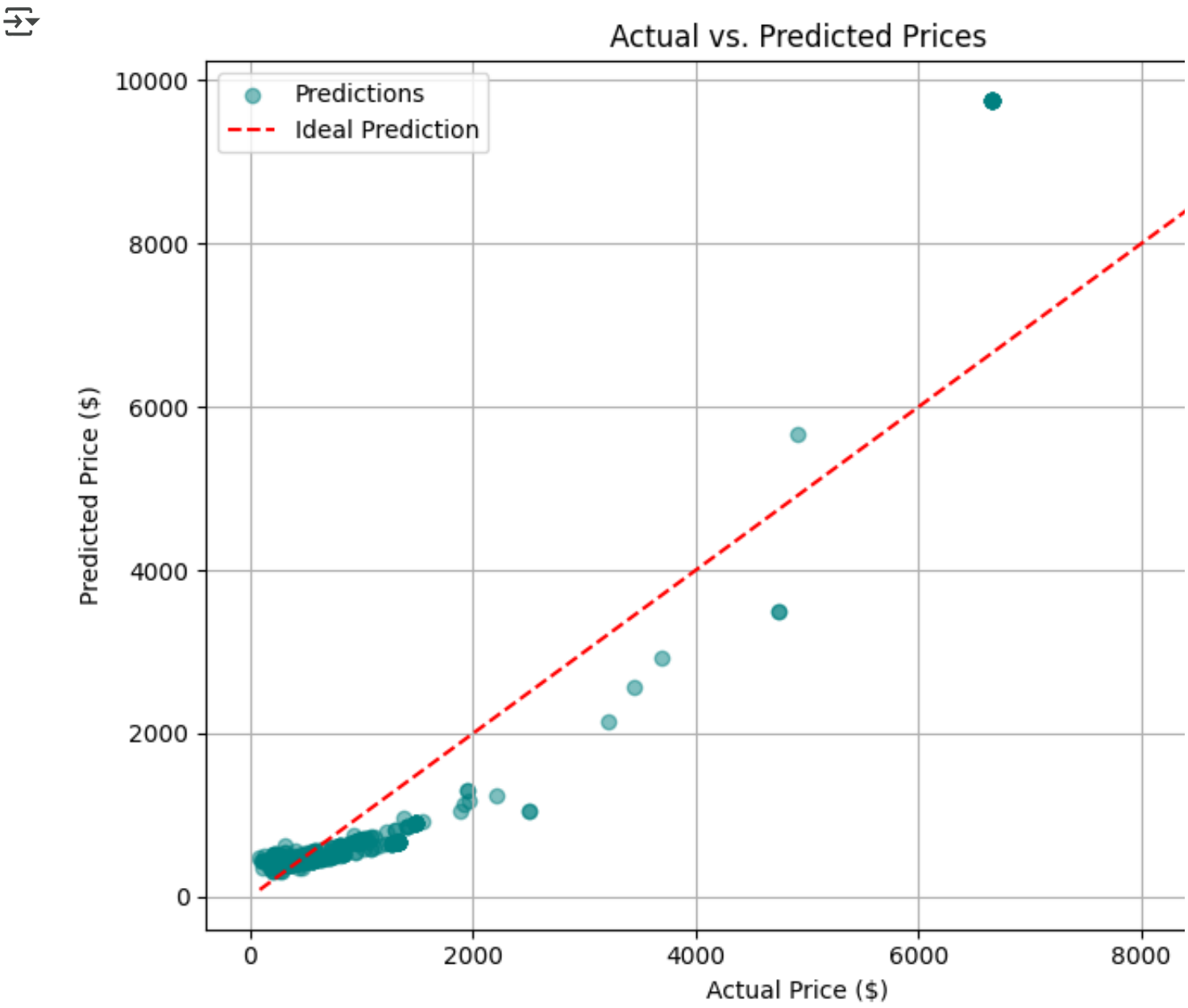
```
Real Price R²: 0.7274073244342987
Real Price MSE: 370782.86202567996
Real Price RMSE: 608.9194216197083
Real Price MAE: 263.6246342305005
```

```
# Step 1: Create a DataFrame to compare predictions
comparison_df = pd.DataFrame({
    'Actual Price': y_test_real.values,
    'Predicted Price': y_pred_real,
    'Error (Pred - Actual)': y_pred_real - y_test_real.values,
    'Absolute Error': np.abs(y_pred_real - y_test_real.values)
})

# Step 2: Print first few rows
print(comparison_df.head(20))

# Step 3: Calculate manual MAE
manual_mae = comparison_df['Absolute Error'].mean()
print(f"\nManual Mean Absolute Error: {manual_mae:.2f}")
```

```
    Actual Price  Predicted Price  Error (Pred - Actual)  Absolute Error
0        1333.34       678.190689            -655.149311      655.149311
1         410.08       413.556703               3.476703        3.476703
2         932.00       758.390191            -173.609809      173.609809
3         410.08       413.556703               3.476703        3.476703
4         285.00       321.420090              36.420090       36.420090
5         410.08       413.556703               3.476703        3.476703
6         971.17       687.042554            -284.127446      284.127446
7         410.08       413.556703               3.476703        3.476703
8         410.08       413.556703               3.476703        3.476703
9         410.08       413.556703               3.476703        3.476703
10       6659.90      9745.731199            3085.831199     3085.831199
11        195.00       368.518444             173.518444      173.518444
12       6659.90      9745.731199            3085.831199     3085.831199
13        971.17       687.042554            -284.127446      284.127446
14       1407.75       867.964136            -539.785864      539.785864
15        410.08       413.556703               3.476703        3.476703
16        410.08       413.556703               3.476703        3.476703
17        212.00       515.725716             303.725716      303.725716
18       1108.69       614.795905            -493.894095      493.894095
19        410.08       413.556703               3.476703        3.476703

Manual Mean Absolute Error: 263.62
```

```
# Scatter plot: actual vs predicted prices
plt.figure(figsize=(8, 6))
plt.scatter(y_test_real, y_pred_real, alpha=0.5, color='teal', label='Predictions

# Add reference line: y = x (perfect predictions)
min_val = min(min(y_test_real), min(y_pred_real))
```

```
max_val = max(max(y_test_real), max(y_pred_real))
plt.plot([min_val, max_val], [min_val, max_val], color='red', linestyle='--', lab

plt.xlabel('Actual Price ($)')
plt.ylabel('Predicted Price ($)')
plt.title('Actual vs. Predicted Prices')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Create a pipeline that transforms features to 2nd-degree polynomial and applies
quad_model = make_pipeline(PolynomialFeatures(degree=2, include_bias=False), Line

# Train the model
quad_model.fit(X_train_scaled, y_train)

# Predict log prices
y_pred_log_quad = quad_model.predict(X_test_scaled)

# Convert back to real prices
y_pred_real_quad = np.expm1(y_pred_log_quad)
y_test_real = np.expm1(y_test)  # already defined

# Evaluate
print("Quadratic Regression R²:", r2_score(y_test_real, y_pred_real_quad))
print("Quadratic Regression MSE:", mean_squared_error(y_test_real, y_pred_real_qu
print("Quadratic Regression MAE:", mean_absolute_error(y_test_real, y_pred_real_q
```

```
Quadratic Regression R²: 0.8788059866448413
Quadratic Regression MSE: 164849.1216389026
Quadratic Regression MAE: 118.78097402845736
```

```
# Step 1: Train a quadratic regression model
quad_model = make_pipeline(
    PolynomialFeatures(degree=2, include_bias=False),
    LinearRegression()
)

quad_model.fit(X_train_scaled, y_train)

# Step 2: Predict on test data (in log space)
y_pred_log_quad = quad_model.predict(X_test_scaled)

# Step 3: Convert back to real prices
y_pred_real_quad = np.expm1(y_pred_log_quad)
y_test_real = np.expm1(y_test)

# Step 4: Evaluate
print("Quadratic Regression R²:", r2_score(y_test_real, y_pred_real_quad))
print("Quadratic Regression MSE:", mean_squared_error(y_test_real, y_pred_real_qu
```

```
print("Quadratic Regression MAE:", mean_absolute_error(y_test_real, y_pred_real_q

# Step 5: Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test_real, y_pred_real_quad, alpha=0.5, color='green', label='Quadr

# Ideal reference line (y = x)
min_val = min(min(y_test_real), min(y_pred_real_quad))
max_val = max(max(y_test_real), max(y_pred_real_quad))
plt.plot([min_val, max_val], [min_val, max_val], color='red', linestyle='--', lab

plt.xlabel('Actual Price ($)')
plt.ylabel('Predicted Price ($)')
plt.title('Actual vs. Predicted Prices (Quadratic Regression)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()


# Get the model from the pipeline
linreg = quad_model.named_steps['linearregression']
poly = quad_model.named_steps['polynomialfeatures']

# Get the feature names (works only if X was a DataFrame with column names)
feature_names = poly.get_feature_names_out(input_features=X.columns)  # if X is a

# Get coefficients and intercept
coeffs = linreg.coef_
intercept = linreg.intercept_

# Print full equation
print(f"log(price) = {intercept:.4f}", end='')
for name, coef in zip(feature_names, coeffs):
    print(f" + ({coef:.4f})*{name}", end='')
```

```
Quadratic Regression R²: 0.8788059866448413
Quadratic Regression MSE: 164849.1216389026
Quadratic Regression MAE: 118.78097402845736
```


```
log(price) = 6.6832 + (-0.0000)*code + (1.8800)*price + (0.0000)*payer_UHC + (
```

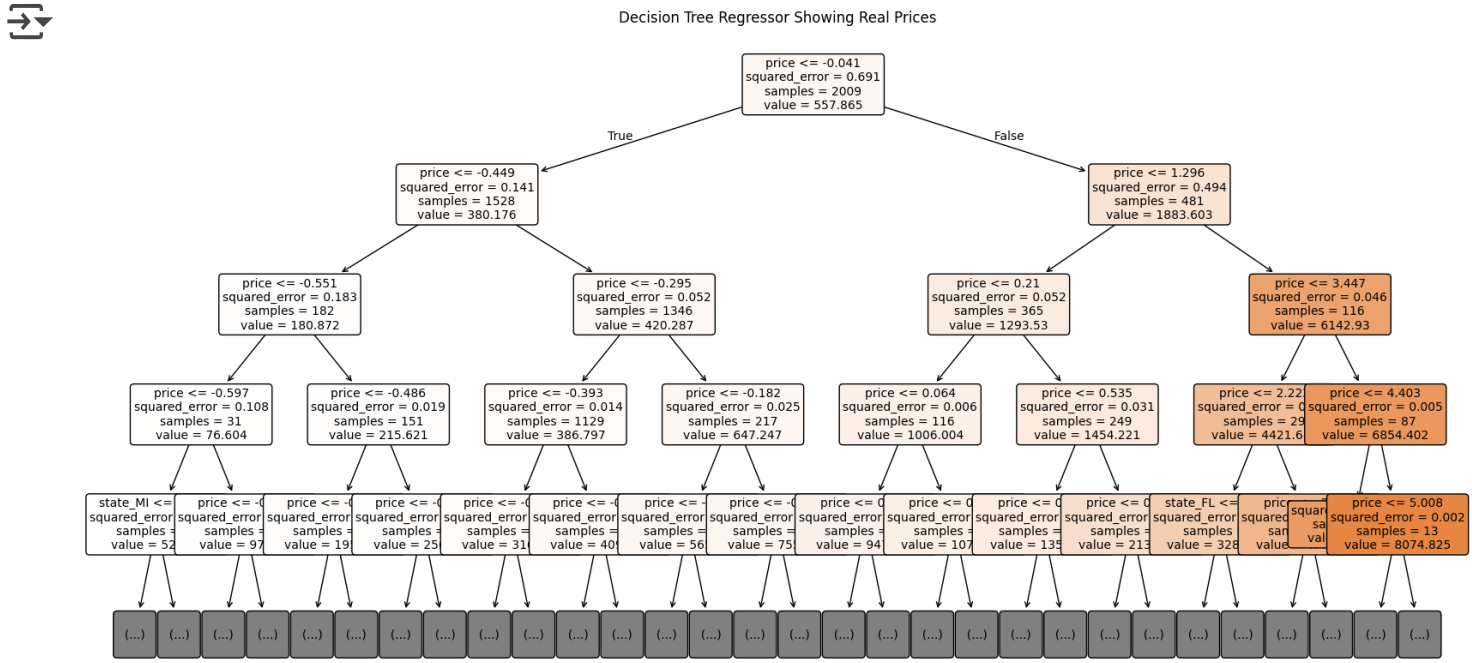
the error has reduced from 264 to 120

```
from sklearn.tree import DecisionTreeRegressor

# Train the model
dt = DecisionTreeRegressor(max_depth=6, random_state=42)
dt.fit(X_train_scaled, y_train)

# Predict on test set
y_pred_dt_log = dt.predict(X_test_scaled)

# Convert back from log to actual prices
y_test_real = np.expm1(y_test)
y_pred_dt_real = np.expm1(y_pred_dt_log)

# Evaluate performance
r2 = r2_score(y_test_real, y_pred_dt_real)
mse = mean_squared_error(y_test_real, y_pred_dt_real)
mae = mean_absolute_error(y_test_real, y_pred_dt_real)

print("Decision Tree R²:", r2)
print("Decision Tree MSE:", mse)
print("Decision Tree MAE:", mae)
```

```
Decision Tree R²: 0.9998419506331769
Decision Tree MSE: 214.98008503127645
Decision Tree MAE: 5.0072435714323955
```

```
# Plot the decision tree using the real prices not log(prices)
from sklearn.tree import plot_tree

# Copy the trained tree and convert its internal values from log to real prices
dt_real = DecisionTreeRegressor(max_depth=dt.get_depth(), random_state=42)
dt_real.tree_ = dt.tree_

# Exponentiate the predicted values inside the tree
dt_real.tree_.value[:, 0, 0] = np.expm1(dt_real.tree_.value[:, 0, 0])

# Plot the decision tree showing real prices
plt.figure(figsize=(20, 10))
plot_tree(
    dt_real,
    feature_names=X_train.columns,
    filled=True,
    rounded=True,
    fontsize=10,
```

```
    max_depth=4,  # Optional: limit depth for clarity
)
plt.title("Decision Tree Regressor Showing Real Prices")
plt.show()
```



```
# Let's try neural networks
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping


# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)



model = tf.keras.Sequential([
    Dense(256, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1)
])



model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: Use
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

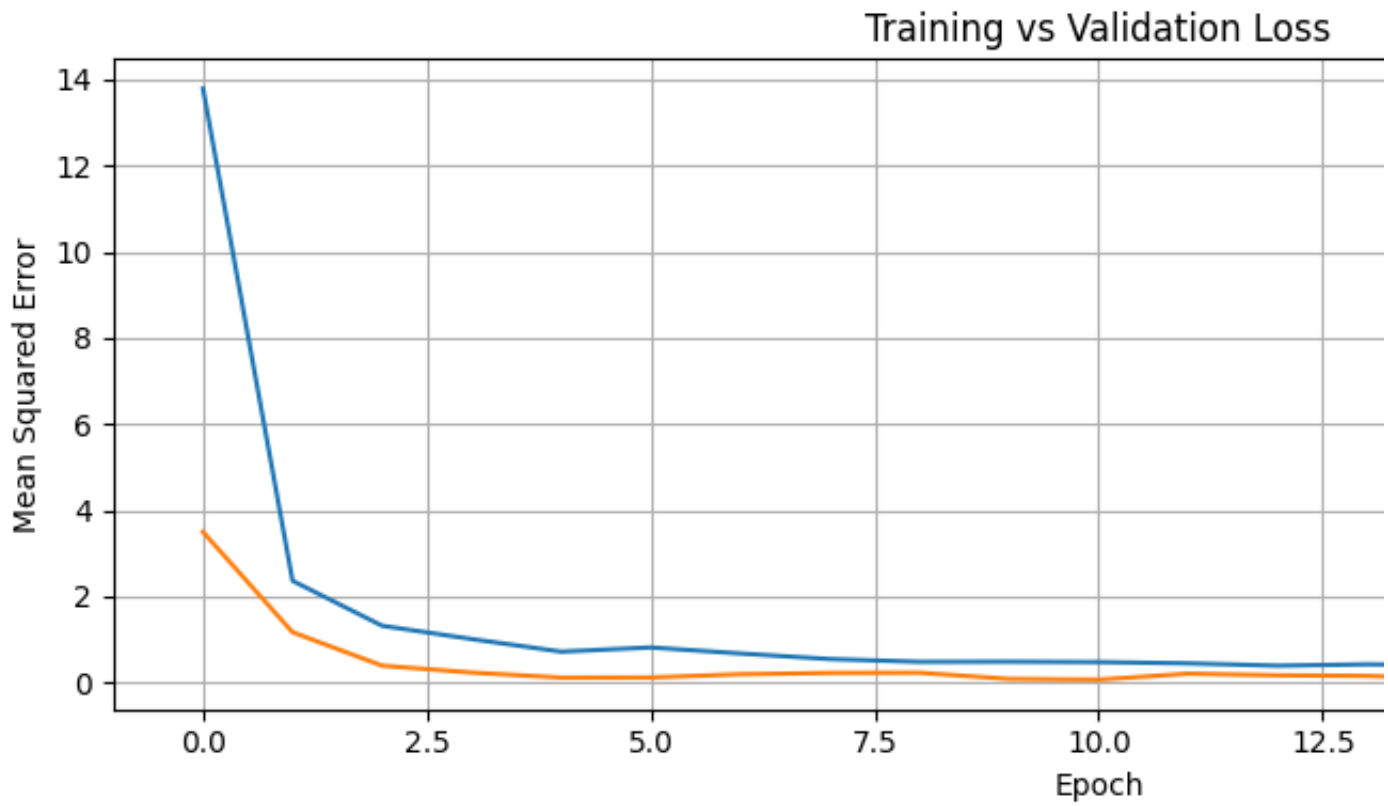
```
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=

history = model.fit(
    X_train_scaled, y_train,
    validation_split=0.2,
    epochs=100,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

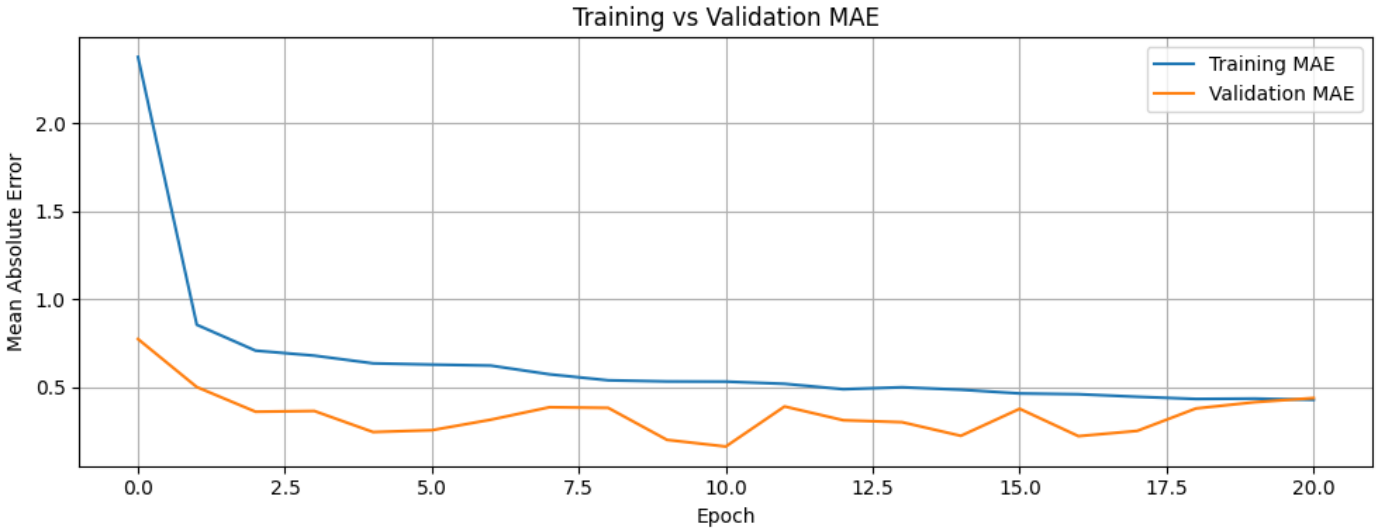
```
Epoch 1/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 23.1058 - mae: 3.8812 - val_lo
Epoch 2/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 2.6716 - mae: 0.8848 - val_los
Epoch 3/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 1.2431 - mae: 0.6977 - val_los
Epoch 4/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 1.1431 - mae: 0.6748 - val_loss
Epoch 5/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.8612 - mae: 0.6455 - val_los
Epoch 6/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.6820 - mae: 0.6128 - val_los
Epoch 7/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 0.7099 - mae: 0.6141 - val_los
Epoch 8/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.5254 - mae: 0.5713 - val_loss
Epoch 9/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.5003 - mae: 0.5566 - val_los
Epoch 10/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 0.4568 - mae: 0.5285 - val_los
Epoch 11/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 0.5273 - mae: 0.5536 - val_los
Epoch 12/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5034 - mae: 0.5473 - val_loss
Epoch 13/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.3801 - mae: 0.4828 - val_loss
Epoch 14/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.4270 - mae: 0.5034 - val_loss
Epoch 15/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.3823 - mae: 0.4820 - val_loss
Epoch 16/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.3472 - mae: 0.4681 - val_loss
Epoch 17/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.3660 - mae: 0.4624 - val_loss
Epoch 18/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.3529 - mae: 0.4612 - val_loss
Epoch 19/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.3063 - mae: 0.4349 - val_loss
Epoch 20/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.3110 - mae: 0.4333 - val_loss
Epoch 21/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.3120 - mae: 0.4212 - val_loss
```

```
# Evaluate on real prices
y_pred_log = model.predict(X_test_scaled).flatten()
y_pred_real = np.expm1(y_pred_log)
y_test_real = np.expm1(y_test)

print("NN R²:", r2_score(y_test_real, y_pred_real))
print("NN MSE:", mean_squared_error(y_test_real, y_pred_real))
print("NN MAE:", mean_absolute_error(y_test_real, y_pred_real))
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
NN R²: 0.6905524522389965
NN MSE: 420913.17078693345
NN MAE: 202.79608899459663
```

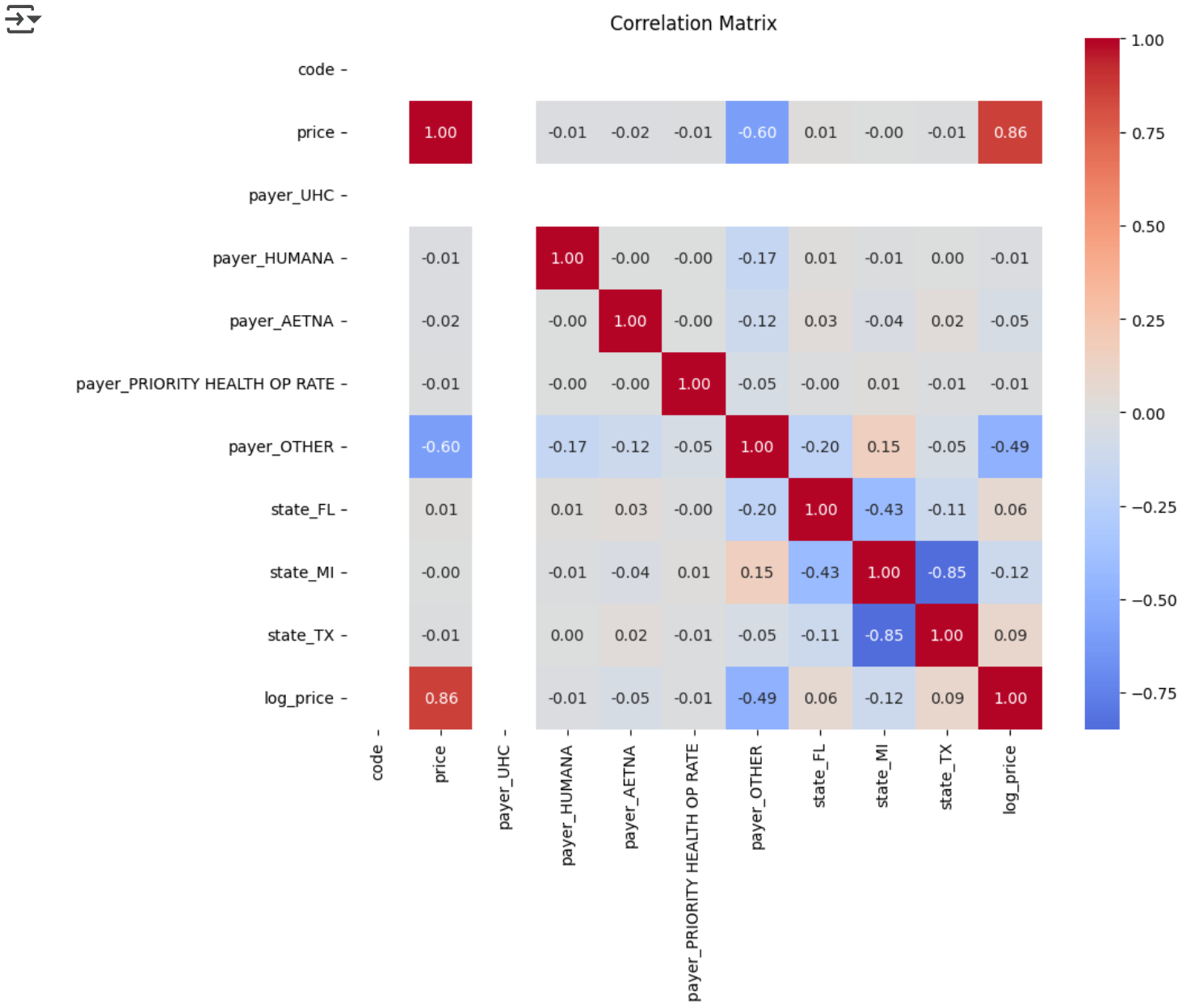
```
# Plot Loss (MSE)
plt.figure(figsize=(10, 4))
plt.plot(history.history['loss'], label='Training Loss (MSE)')
plt.plot(history.history['val_loss'], label='Validation Loss (MSE)')
plt.title('Training vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Plot MAE
plt.figure(figsize=(10, 4))
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Training vs Validation MAE')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```






```
# Any correlation between log prices and numerical features?
```

```
# Assuming 'merged' contains numerical/log-transformed features and target
corr_matrix = merged.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', center=0)
plt.title("Correlation Matrix")
plt.show()
```



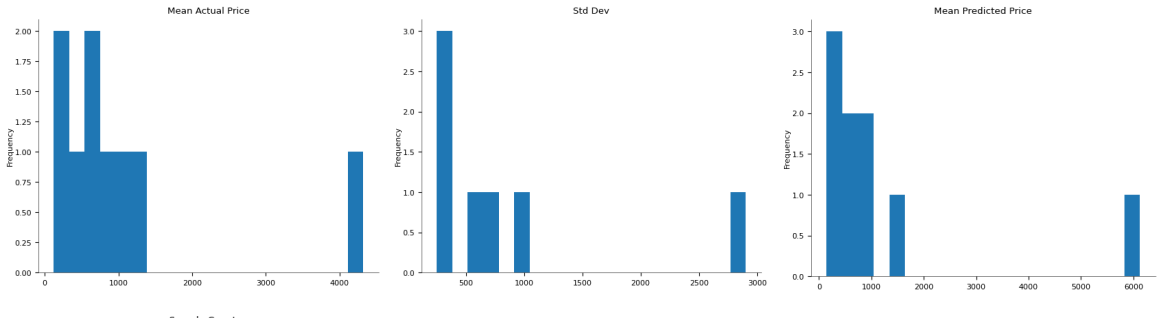
```
# Ensure predictions and actuals are mapped
y_test_real = np.expm1(y_test)
y_pred_real = np.expm1(model.predict(X_test_scaled).flatten())

# Copy test set features
test_df = X_test.copy()
test_df['Actual Price'] = y_test_real
test_df['Predicted Price'] = y_pred_real

# Get payer category
for payer in ['payer_UHC', 'payer_HUMANA', 'payer_AETNA',
              'payer_PRIORITY HEALTH OP RATE', 'payer_OTHER']:
    test_df[payer] = X_test[payer]

def get_payer(row):
    for col in ['payer_UHC', 'payer_HUMANA', 'payer_AETNA',
                'payer_PRIORITY HEALTH OP RATE', 'payer_OTHER']:
        if row[col] == 1:
            return col.replace("payer_", "")
    return "CASH"

def get_state(row):
    if row['state_MI'] == 1:
        return 'MI'
    elif row['state_FL'] == 1:
        return 'FL'
    elif row['state_TX'] == 1:
        return 'TX'
    else:
        return 'Unknown'

test_df['State'] = test_df.apply(get_state, axis=1)
```

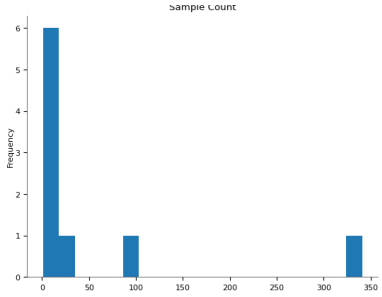
```
test_df['Payer'] = test_df.apply(get_payer, axis=1)

# Group by state and payer
summary_df = test_df.groupby(['State', 'Payer']).agg({
    'Actual Price': ['mean', 'std'],
    'Predicted Price': 'mean',
    'Payer': 'count'
}).rename(columns={'count': 'Sample Count'})

# Clean up column names
summary_df.columns = ['Mean Actual Price', 'Std Dev', 'Mean Predicted Price', 'Sa
summary_df = summary_df.reset_index()

# Display
from IPython.display import display
display(summary_df)
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

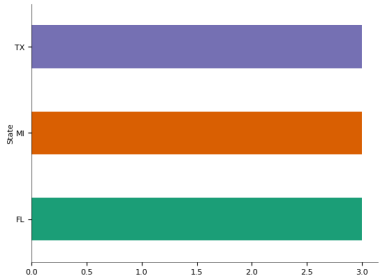
	State	Payer	Mean Actual Price	Std Dev	Mean Predicted Price	Sample Count
0	FL	CASH	1309.673333	944.659598	1487.739502	9
1	FL	HUMANA	306.560000	NaN	145.734818	1
2	FL	OTHER	591.747333	644.344832	646.135559	15
3	MI	CASH	4322.355926	2903.358005	6126.587402	27
4	MI	HUMANA	609.540000	282.079037	338.404663	2
5	MI	OTHER	492.576862	282.383578	461.588379	341
6	TX	CASH	882.086000	252.107861	764.896240	15
7	TX	HUMANA	122.830000	NaN	176.108673	1
8	TX	OTHER	964.547283	744.942119	824.724426	92

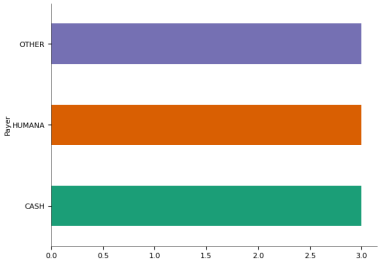
Distributions



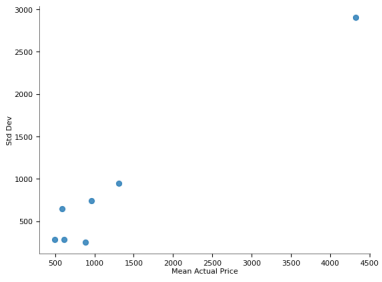


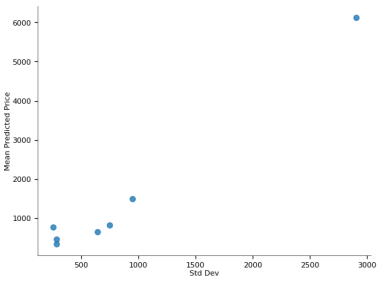
Categorical distributions

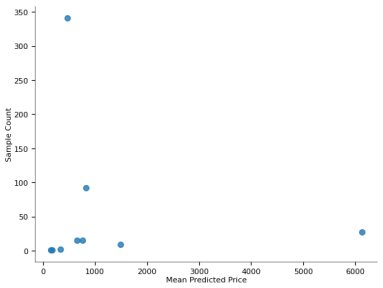




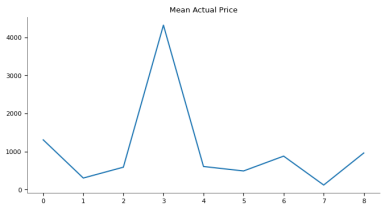
2-d distributions

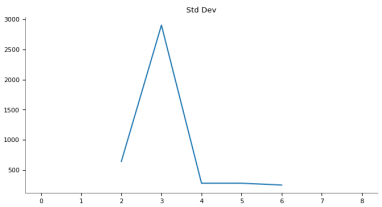


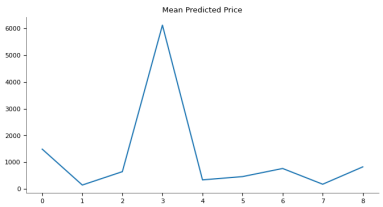


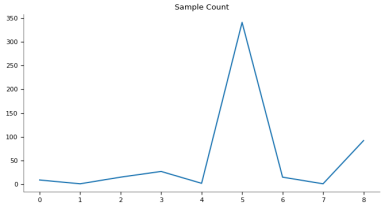


Values





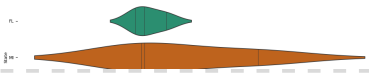




2-d categorical distributions

Faceted distributions

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed ir
```



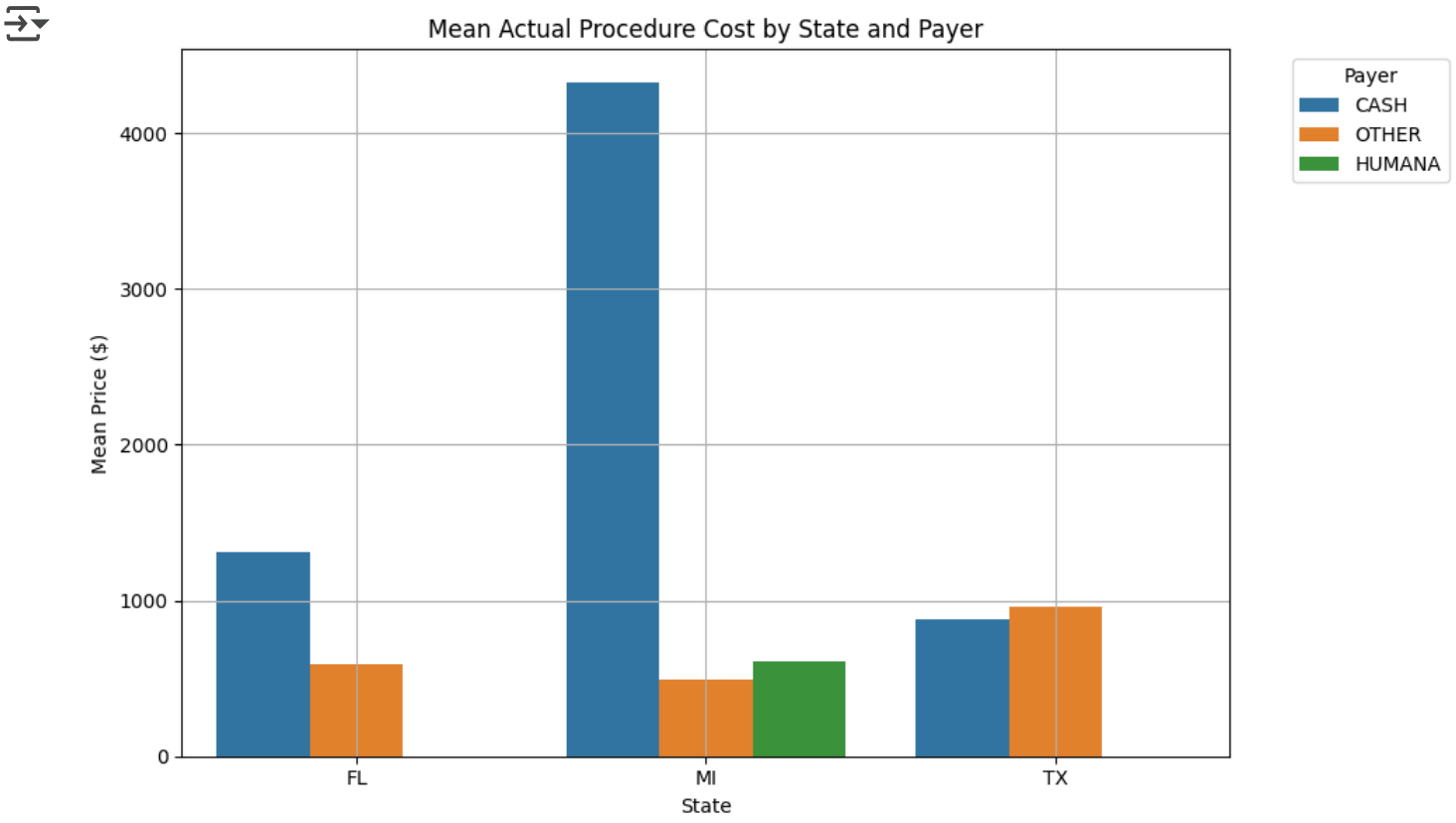
```
<string>:5: FutureWarning:
```

Next steps: Generate code with summary_df | View recommended plots | New interactive sheet

From the table above, we can infer the following results: 1) Michigan exhibits the highest average prices for CASH payers, with an average procedure cost of $4,322 — nearly three times higher than the average in Texas. This suggests significant geographic variation in baseline charges, potentially driven by higher hospital costs or payer-negotiated rates in Michigan. 2) Florida's CASH payers show a high standard deviation, indicating wide variability in procedure pricing across hospitals. Despite this, the neural network was able to approximate the mean price reasonably well. However, the high variance implies reduced reliability for individual predictions. Additional data from Florida hospitals may help improve the model's accuracy and generalizability in that region 3) OTHER" payers are more common in both Michigan and Texas, and are associated with lower average prices. This could suggest that these plans reflect lower bargaining power, alternative insurance structures, or self-pay/discounted arrangements.

```
import seaborn as sns

# Use the already-prepared summary_df
summary_plot_df = summary_df.copy()
summary_plot_df = summary_plot_df.reset_index()  # Ensure State and Payer are col
summary_plot_df = summary_plot_df.dropna()  # Drop rows with NaN for plotting

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(
    data=summary_plot_df,
    x='State',
    y='Mean Actual Price',
    hue='Payer'
)
```

```
plt.title('Mean Actual Procedure Cost by State and Payer')
plt.ylabel('Mean Price ($)')
plt.xlabel('State')
plt.legend(title='Payer', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.