Medical Equipment Cost Prediction Challenge
(Checkpoint 1)

Team Members

1. Kavya Gupta - IMT2023016
2. Nainika Agrawal - IMT2023034
3. Pragya Rai - IMT2023529

Github Link: https://github.com/nainika0305/MedicalEquipmentCostPrediction.git

Task

Predict the **transport cost** required to deliver medical equipment to hospitals using the features provided in the dataset. This is a **regression problem**, where the model aims to estimate a continuous numerical output.

Dataset and Features Description

Transporting sensitive medical equipment comes with unique challenges:

- Ensuring fragile devices arrive safely.
- Handling urgent shipments for critical hospital needs.
- Managing cross-border deliveries and rural hospital locations.
- Coordinating installation and setup services when required.

Our company specializes in delivering medical equipment sourced from multiple suppliers worldwide. The goal is to build a model that predicts the transport cost of each order, enabling better logistics planning and fair pricing. The dataset folder contains the following files:

- **train.csv**: 5000 x 20
- **test.csv:** 500 x 19
- **sample_submission.csv:** 5 x 2

Column(feature) name	Description
Hospital_Id	Unique identification number of the hospital
Supplier_Name	Name of the medical equipment supplier
Supplier_Reliability	Supplier's reputation score in the market (higher means more reliable)
Equipment_Height	Height of the equipment
Equipment_Width	Width of the equipment
Equipment_Weight	Weight of the equipment
Equipment_Type	The type of equipment (e.g., diagnostic device, surgical tool)
Equipment_Value	Monetary value of the equipment
Base_Transport_Fee	Base fee for transporting the equipment
CrossBorder_Shipping	Whether the delivery is international
Urgent_Shipping	Whether the delivery was in express/urgent mode
Installation_Service	Whether installation/setup service was included
Transport_Method	Mode of transport used (e.g., air, road, sea)
Fragile_Equipment	Whether the equipment is fragile
Hospital_Info	Additional information about the hospital
Rural_Hospital	Whether the hospital is located in a remote/rural area
Order_Placed_Date	Date when the order was placed
Delivery_Date	Date when the delivery occurred
Hospital_Location	Location of the hospital
Transport_Cost	Target variable – cost of transporting the order

The following section outlines the necessary steps for conducting essential **Exploratory Data Analysis (EDA)** to gain insights, detect potential issues, and prepare data for model training.

1. Import Libraries and Load Dataset

The initial step in the EDA process involves importing essential libraries such as pandas, numpy, matplotlib, seaborn, and relevant modules from scikit-learn for data preprocessing. The dataset is loaded into a pandas DataFrame for further analysis and manipulation.

2. Data Overview

The dataset is first explored by viewing the initial rows, inspecting data types, checking for duplicates and invalid data, and detecting missing values to understand its structure and quality. A statistical summary of the numerical columns is then generated to examine their central tendencies, variability, and possible outliers.

3. Handling duplicates

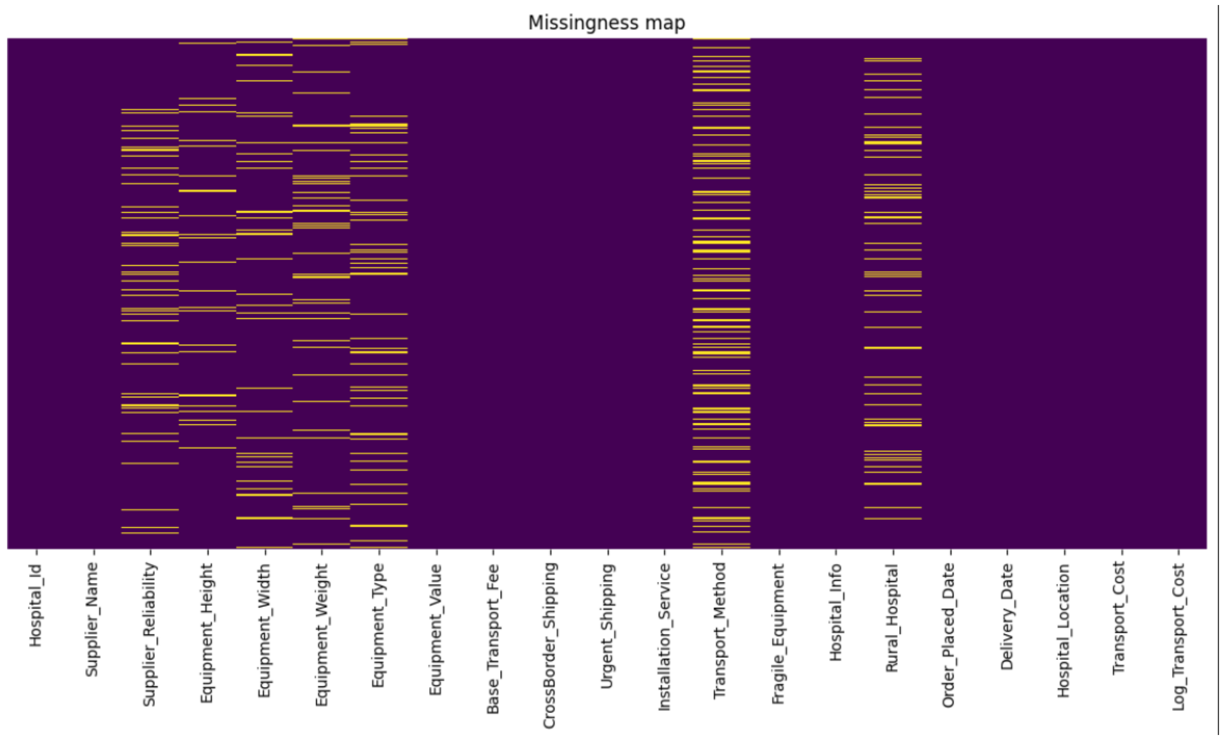
We found no duplicate rows in this dataset.

4. Handling target variable

On analyzing the negative values, we saw 493 rows have negative transport cost. There are many ways to fix this (mentioned in the file) but what worked best is either clipping to zero, or dropping the negative rows. We noticed that it is highly skewed, so we decided to create a log-transformed cost variable using np.log1p() to handle this. There are no missing values in the target.

5. Handling Missing Values

To address missing values, the dataset was carefully inspected for null entries and visualized using a missingness plot. The visualization confirmed that 7 columns had missing values. As the number of rows was a lot, we decided not to delete the rows. We made binary features 'column_name_Is_Missing' to help the model understand where we have imputed values. We visualized the missing columns using histograms and decided that for numerical columns, we impute using median and for categorical we impute using mode (most_frequent). This thorough verification ensures the dataset is fully complete and ready for subsequent preprocessing steps, without any risk of missing data affecting model performance.



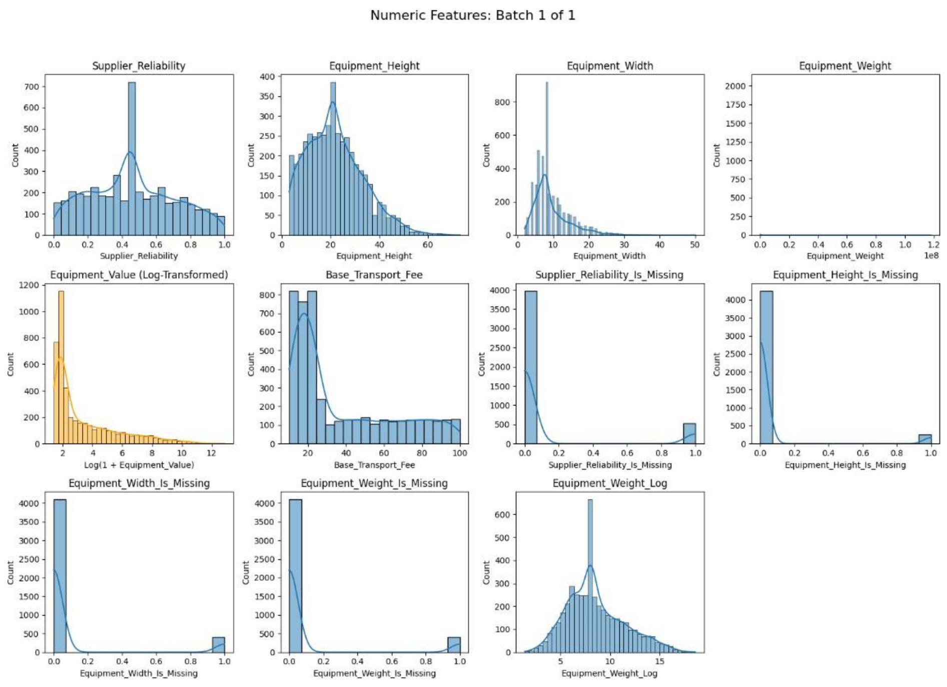
6. Exploratory Data Analysis (EDA)

Several plots were created to examine data distributions, identify outliers, and explore relationships between features. These visualizations offer valuable insights that guide the next steps in preprocessing and model development. The key visualizations and their interpretations are outlined below:

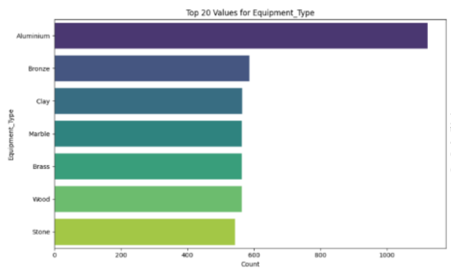
- **Histograms for each feature:**

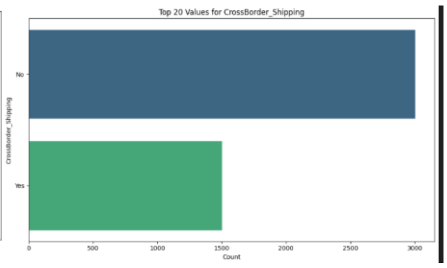
  For each numerical feature, histograms were plotted to observe the data's spread, skewness, and distribution shape. Conclusions we can draw from these histograms are:

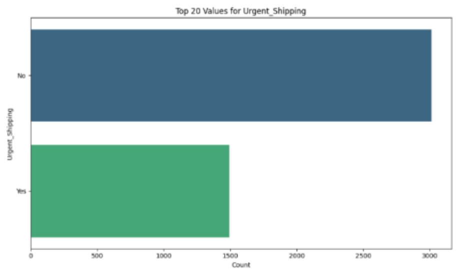
  - **Numeric and Binary Feature Plots:**
    We found 11 numeric features, and added some more numeric features, Equipment_Height_Is_Missing, Equipment_Width_Is_Missing, Equipment_Weight_Is_Missing, Equipment_Weight_Log.
    The 'Base_Transport_Fee' shows there is a standard low-cost fees and a special higher, variable fee. Supplier_Reliability is almost uniform with a peak at the median (imputed missing with median). Equipment_Weight and Equipment_Value are highly skewed, log-transform them for better results.

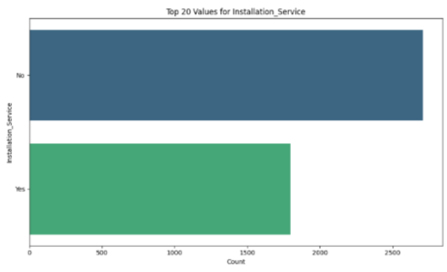


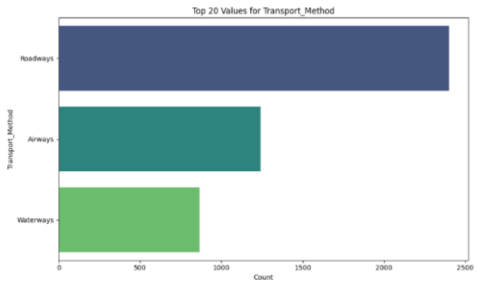
- **Categorical Feature Plots:**

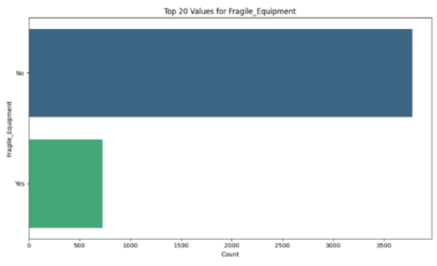


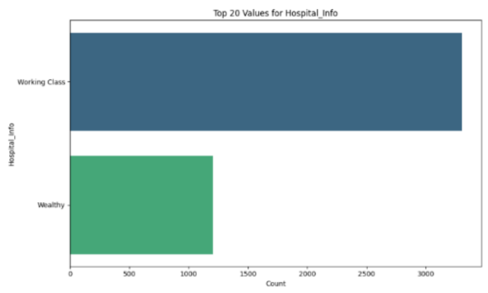


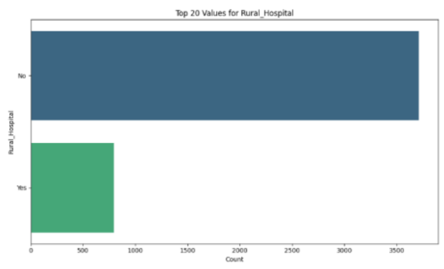












- **Scatter plots:**

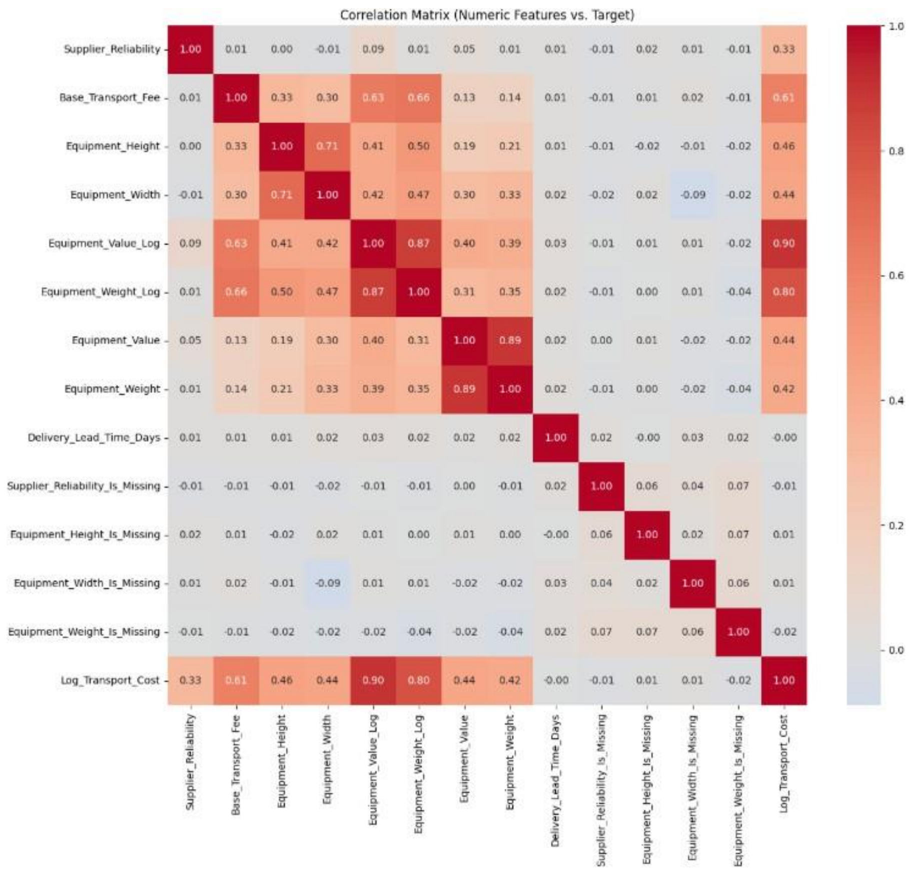
  These tell us the relationship between the features and the target. Urgent_Shipping, Installation_Service and Cross_Border_Shipping do not seem to be very predictive features.

- **Boxplots for Outlier detection:**

  Boxplots were used to examine each numerical feature for potential outliers, which appear as points beyond the plot's whiskers.

- **Correlation Matrix (Heatmap):**

  This correlation matrix visualizes the pairwise correlations between numeric features and the target variable (Log_Transport_Cost) in the dataset. Here are a few key takeaways:
  - We see that **'Equipment_Value_Log'** (0.90), '**Equipment_Weight_Log'**(0.80) and **'Base_Transport_Fee'**(0.61) have very high correlation with the target.
  - **Equipment Height, Width, Weight, and Value** are moderately correlated with the target.
  - **Supplier_Reliability** has a mild positive correlation (0.33), indicating some relationship but not a dominant factor for transport cost.
  - Rest of the features have near-zero correlation with the target.
  - High correlation between **Equipment_Value_Log** and **Equipment_Weight_Log** as well as between **Equipment_Weight** and **Equipment_Value**, may indicate multicollinearity among these features.



- **VIF (Variance Inflation Factor)**

  This is used to detect multicollinearity in regression models. It quantifies how much the variance of a regression coefficient is inflated due to correlations with other predictors. Some observations are:

  - **Equipment_Weight_Log** (VIF ≈ 24.9) and **Equipment_Value_Log** (VIF ≈ 12.5) have extremely high VIF values, indicating very strong multicollinearity with other variables in the dataset.
  - **Equipment_Height** (VIF ≈ 10.0) and **Equipment_Width** (VIF ≈ 9.4) are also near or above the common caution threshold, hinting at strong multicollinearity. For better representation we created a 'Equipment_Volume' feature.
  - **Base_Transport_Fee, Equipment_Value**, and **Equipment_Weight** all show VIFs between 5 and 6, which may cause moderate concern especially in combination.
  - **Supplier_Reliability** has a lower VIF (3.5), not indicating problematic multicollinearity on its own
  - **Delivery_Lead_Time_Days** and the **"_Is_Missing"** features have VIFs close to 1, indicating little or no collinearity with other variables in the model
  - We have experimented to reduce the VIF scores of features in the file.

7. Handling Date features

A significant data quality issue was identified: We have found 1774 rows where the order is placed after delivery date. These records represent either data entry errors (e.g., swapped dates) or issues in the order fulfillment tracking system. These negative values are visible on the far left of the scatter plot. Most deliveries occur on the same day. We created a Delivery_Lead_Time_Days feature which is the number of days between Order_Placed_Date and Delivery_Date. We can either impute the non-valid entries in this with the median, or clip them to 0. We have also tried to plot and see if any new derived features would be helpful.

8. Data Preprocessing

Now the data is preprocessed for modeling through the following steps:

- **Encoding categorical variables** using One Hot Encoding (or other encodings like label, target, etc, based on models) to convert categories into numerical values, enabling algorithms to interpret them effectively.
- **Standardizing numerical features** with a StandardScaler (or other Scalers like Robust, etc, based on models) to bring all features onto a similar scale and reduce model sensitivity to differences in magnitude.
- **Feature engineering**, when necessary, to create new features that capture nonlinear relationships or additional patterns in the data.

9. Splitting the Dataset

The train.csv dataset is divided into training and validation sets to allow model performance assessment on unseen data, typically using an 80/20 split. This step ensures the data is ready for both model training and evaluation.

Summary

The EDA and preprocessing steps we did include:

- Conducted a thorough inspection of data types, missing values, duplicates, and basic statistical properties.
- Addressed missing and invalid values, applied appropriate imputations, and created new features such as Log_Transport_Cost, Equipment_Volume, and Delivery_Lead_Time_Days
- Performed exploratory data analysis to visualize distributions, identify outliers, and examine relationships between features and the target variable.
- Encoded categorical variables, standardized numeric features, and can implement feature engineering to capture additional patterns
- Split the dataset into training and validation sets (80/20) to prepare it for model training and performance evaluation.

Find all the preprocessing done here: EDA_And_Preprocessing

Models Used For Training

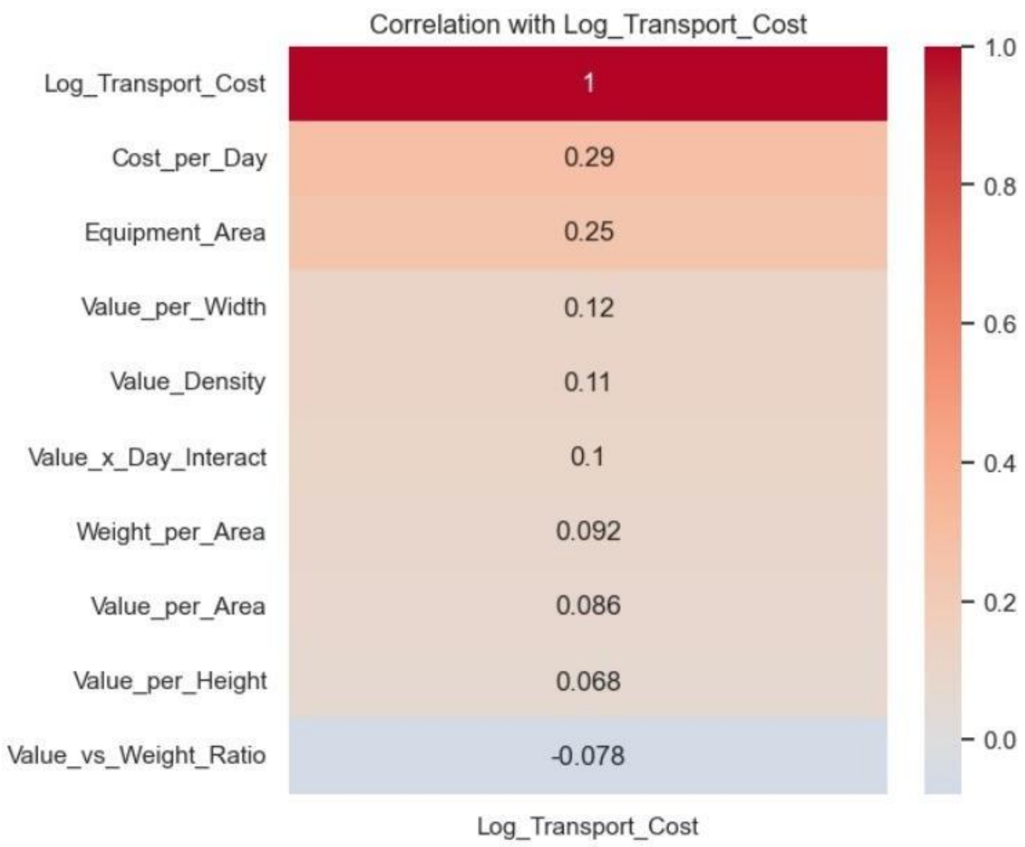
1. **Decision Tree**
    a. The preprocessing steps were consistent with those defined in the preprocessing section. We implemented the Decision Tree model in two approaches — a baseline version and an improved version – both using identical data cleaning and transformation pipelines.
    b. For both versions, we used RandomizedSearchCV to find the optimal hyperparameters for the DecisionTreeRegressor.
    The parameter grid included variations of max_depth, min_samples_split, min_samples_leaf, and max_features:
    c. In the baseline version of the Decision Tree model, all preprocessing and feature engineering steps were identical to the initial pipeline, and no additional features were introduced. This version achieved a Kaggle score of 5,213,105,184.494. In the improved version, a new derived feature called Delivery_Efficiency was added to capture the relationship between equipment value and delivery time, along with the application of a RobustScaler to handle skewed numerical features and reduce the influence of outliers. These enhancements resulted in a noticeable performance improvement, with the validation RMSE decreasing and the Kaggle score improving to 5,170,179,777.963.
    d. We saved the trained model in the finalised_model and we are using that model while testing, and then the results are saved in the submission_DECISION_TREE_ONLY_model.csv file.
    e. The MSE we obtained on the validation data of the optimized model was 2050625614.77.

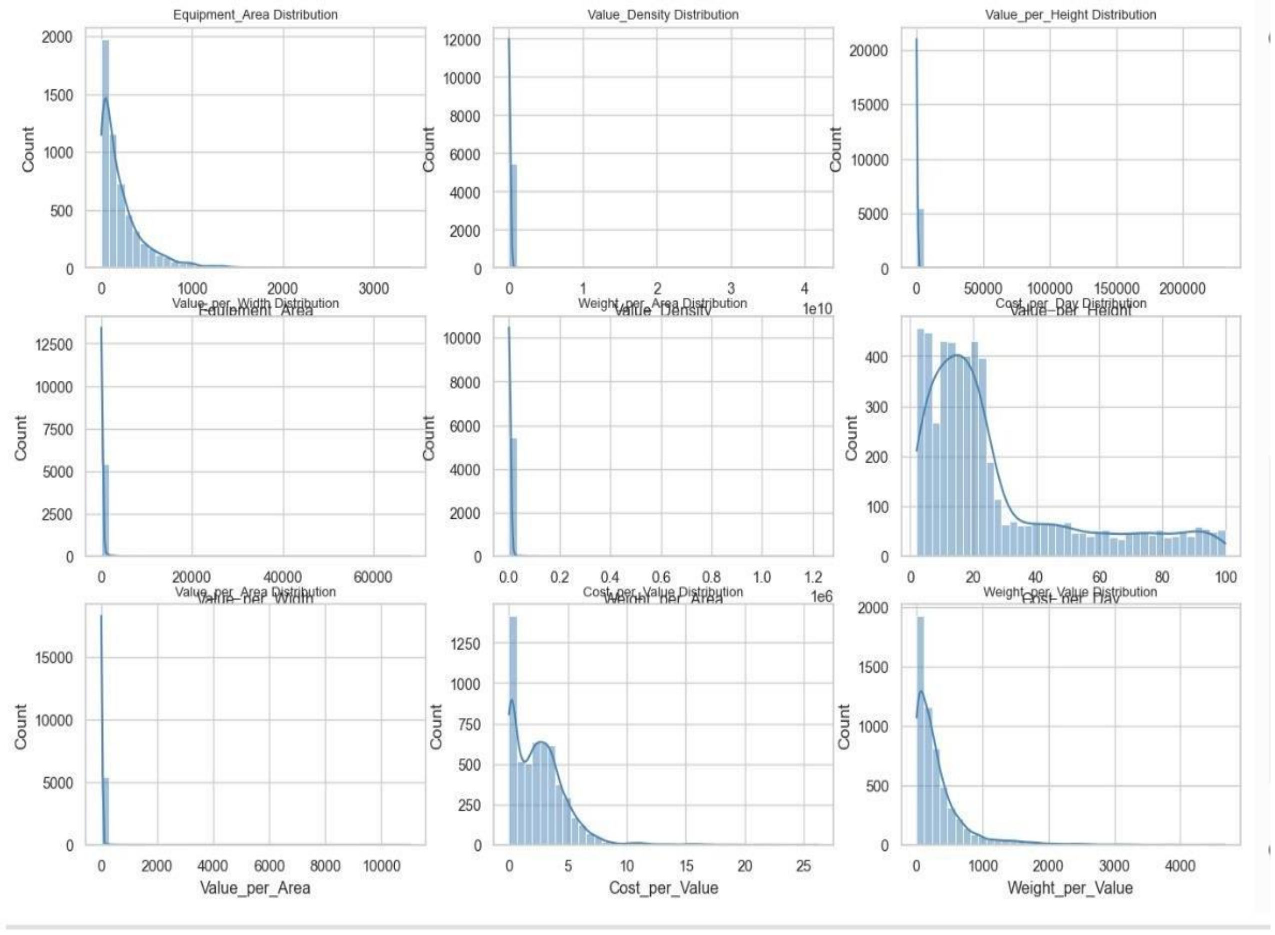
2. **K nearest neighbours**
    a. All engineered features were included, followed by Principal Component Analysis (PCA) to reduce dimensionality to 20 components. The intention was to reduce noise and improve model efficiency. However, KNN performance deteriorated significantly, leading to a high RMSE and poor score of 7,930,349,904.119.
    b. We Removed PCA and focused on feature engineering:
    We added new feature: Cost_per_Day
    Performed feature selection and drop manually.
    Used RobustScaler instead of StandardScaler to handle outliers.

c. Performed RandomizedSearchCV for hyperparameter tuning.
d. Kaggle score improved to 5,796,142,160.900.
e. We saved the trained model in the finalised_model and we are using that model while testing, and then the results are saved in the submission_KNN_ONLY_model.csv file.
f. The MSE we obtained on the validation data of the optimized model was 1960545274.93.

3. **Random Forest**
    a. The same preprocessing and feature engineering pipeline used in previous models (like LightGBM) was retained to ensure consistency and comparability across algorithms.
    b. Hyperparameter tuning was performed using RandomizedSearchCV to find optimal combinations
    c. The accuracy we got on the submission.csv file on kaggle was 5,060,625,934.717 from previous score of 6,107,167,412.799.
    d. ~17% improvement in predictive performance.
    e. The MSE we obtained on the validation data of the optimized model was 1740130982.78.

4. **XGBoost**
    a. Preprocessing is the same for initial model and the optimized model.
    b. For the optimized model, we used RandomizedSearchCV to find best parameters like learning rate, call sample by tree, reg alpha, lambda, max depth.
    c. We improved our model by adding new features like value_density, value_per_height, cost_per_day, value_per_area. We also analysed their correlation with the target log transport cost.
    d. We further optimized it by analyzing the existing and added features' distribution. We categorized them into skewed and normal features. We did log transform for the skewed features.
    e. This drastically improved out model performance from an initial Kaggle score of 58962398214.314 to 4370745455.994.
    f. The MSE we obtained on the validation data of the optimized model was 1679508513.61.


Correlation with Log_Transport_Cost
Log_Transport_Cost 1
Cost_per_Day 0.29
Equipment_Area 0.25
Value_per_Width 0.12
Value_Density 0.11
Value_x_Day_Interact 0.1
Weight_per_Area 0.092
Value_per_Area 0.086
Value_per_Height 0.068
Value_vs_Weight_Ratio -0.078
Log_Transport_Cost


Equipment_Area Distribution
Value_Density Distribution
Value_per_Height Distribution
Value_per_Width Distribution
Weight_per_Area Distribution
Cost_per_Day Distribution
Value_per_Area Distribution
Cost_per_Value Distribution
Weight_per_Value Distribution
Count
Equipment_Area
Value_Density
Value_per_Height
Value_per_Width
Weight_per_Area
Cost_per_Day
Value_per_Area
Cost_per_Value
Weight_per_Value

5. **GradientBoosting**
    a. We did same preprocessing as we did for xgboost and used RandomisedSearchCV to find best parameters.
    b. Our final optimized lgbm model on test dataset gave a score of 4812707793.280
    c. The score on validation data is 1834674359.79
6. **AdaBoost Regressor**
    a. After similar preprocessing, we applied GridSearchCV to compute the best parameters for AdaBoost.
    b. Excluded Hospital_State (dropped earlier), used only three categorical categories for One hot encoding. We clipped delivery times below zero to 0 instead of taking absolute.Cost per day was dropped and new features like Value per Area,Value_Density etc…
    c. This improved our score from 6,157,782,566.15 to 4,065,760,976.40
    d. Our validation score on validation dataset is 1382993115.3691

7. **Linear & Polynomial Regression:**
   Models trained: Linear Regression without regularization, LASSO, Ridge, Elastic Net, Polynomial Regression
    a. Preprocessing
       The preprocessing pipeline was consistent across all these models. It involved handling missing values using SimpleImputer, clipping outliers with a custom PercentileClipper (1st–99th percentile), scaling numeric features using StandardScaler, and encoding categorical variables with OneHotEncoder. Feature engineering added derived attributes such as Equipment_Volume = Equipment_Height × Equipment_Width. The data was split into training (80%) and validation (20%) sets, with the target variable log-transformed (Log_Transport_Cost) to stabilize variance. The pre-processing remains the same, which we have mentioned in the preprocessing section, involving checking for null values, removing duplicates, removing outlier points, and encoding the categorical columns.
    b. Model Implementation
       Linear Regression served as a baseline model with no regularization. Ridge, Lasso, and Elastic Net used regularization to control overfitting, with hyperparameters tuned automatically using cross-validation (RidgeCV, LassoCV, and ElasticNetCV). Polynomial Regression tested non-linearity using polynomial feature expansion to capture higher-order relationships among predictors.
    c. Evaluation Metrics
       Each model was evaluated on the validation set using RMSE, MAE, and R^2 scores in both log-transformed and actual value scales. Predictions were inverse-transformed using np.expm1 to compare with actual costs. Submission files were generated with non-negative predicted transport costs.

d. SelectKBest was tried but models worked better by doing our own feature engineering.

e. Model Observations and Performance

1. **Linear Regression** (without regularization)
   Despite its simplicity, it achieved robust performance. The model successfully captured key linear relationships between transport cost and engineered predictors.
2. **Ridge & Lasso Regression**
   Both regularized models improved generalization slightly but did not surpass the plain Linear Regression in terms of RMSE. Lasso automatically performed feature selection, dropping less important coefficients, while Ridge penalized all features uniformly.
   Kaggle score for ridge is 4823087907.942 and for lasso it is 4818595827.102.
3. **Elastic Net Regression**
   This model outperformed others by combining both L1 (Lasso) and L2 (Ridge) penalties. The elastic combination stabilized coefficient estimation, effectively balancing sparsity and robustness. It achieved the best overall RMSE on the validation data, showing lower overfitting compared to unregularized models. This is the best linear model, giving us a score of 4818631041.303 on Kaggle
4. **Polynomial Regression**
   Polynomial features (degree 2 and 3) were explored to capture non-linear interactions. However, while it improved slightly, it did not significantly lower the RMSE due to higher model complexity and risk of overfitting. Degree 3 had too many features and would lead to overfitting as well.

8. Ensemble Models

a) We used XGBoost and LightGBM models and then combined (ensembled) their predictions to improve accuracy and reduce overfitting.

b) We applied K-Fold Cross-Validation (10 folds) to train and validate the models, ensuring that every data point was used for both training and validation, improving generalization.

c) After training, we performed model ensembling by blending the predictions from both models:
oof_preds_xgb → predictions from XGBoost on validation data
oof_preds_lgbm → predictions from LightGBM on validation data
y_pred_actual_blend = 0.5 * y_pred_actual_xgb + 0.5 * y_pred_actual_lgbm

d) We then evaluated the performance using Mean Squared Error (MSE) for:
XGBoost alone

LightGBM alone
The blended ensemble

Finally, we generated test predictions, converted them back from log scale, applied the same blending (50% XGBoost + 50% LightGBM), clipped any negative values to 0, and saved the results as the final submission file (submission_ensemble.csv).

e) We got a score of 4967373224.435 on test set and score of 58243011266.16 on validation data.

Discussion on the Performance of Different Approaches:

- Linear Regression served as a simple baseline, it assumed a linear relationship between features and the target.
- Regularized models like Ridge and Lasso improved over plain Linear Regression by penalizing large coefficients, reducing overfitting, but they still couldn’t capture the strong non-linear patterns.
- On increasing the degree of polynomial, the model started overfitting, leading to more error.

- Gradient Boosting performed better than Random Forest by building trees sequentially, where each new tree corrected the residual errors of the previous one, achieving strong accuracy with well-tuned parameters.
- Decision Tree performed moderately well but tended to overfit due to its sensitivity to training data, capturing noise rather than general trends.
- KNN with PCA reduced dimensionality and improved generalization slightly, but distance-based learning struggled with high feature variance and categorical one-hot encoded data.
- Random Forest outperformed the previous two by averaging multiple trees, reducing variance, and capturing non-linear relationships, though it still faced limitations in handling skewed target distribution.

- AdaBoost showed the best overall performance because it iteratively focused on difficult samples that earlier weak learners mispredicted, improving accuracy through weighted ensemble learning.
- The combination of shallow decision trees as weak learners and the adaptive weighting mechanism allowed AdaBoost to model complex relationships without heavily overfitting.
- The log1p transformation of the target further helped stabilize training by reducing the effect of large outliers, complementing AdaBoost’s iterative error correction.
- Proper preprocessing—handling missing values, encoding categorical features, clipping negatives, and standardizing numerical variables—ensured AdaBoost received well-conditioned input, maximizing its performance.
- XGBoost enhanced Gradient Boosting using regularization (both L1 and L2) and tree pruning, resulting in a high-performing and stable model.

- LightGBM provided similar to XGBoost by using histogram-based splits and leaf-wise growth, but it required careful tuning to prevent overfitting on small datasets.

- Overall, ensemble boosting methods like AdaBoost, Gradient Boosting, XGBoost, and LightGBM consistently outperformed individual or linear models by adaptively focusing on difficult samples and effectively capturing complex, non-linear feature interactions.

Our interpretation of why AdaBoost Gave the Lowest MSE Score:

AdaBoost excelled due to the following factors:

- Linear regression assumes linear relationships.AdaBoost, with its adaptive weighted decision trees, can model non-linearities, interactions, and complex feature effects far better.
- A single decision tree can easily overfit or underfit depending on its depth. AdaBoost reduces variance and improves generalization by combining many weak learners, effectively smoothing the decision boundary.
- XGBoost and Gradient Boosting are powerful but complex. As our data is small and noisy, XGBoost and Gradient Boosting might be overfitting and hence are performing slightly lesser than Adaboost Regressor. AdaBoost, using very shallow stumps and exponential weighting, sometimes generalizes better by being simpler and less flexible. AdaBoost can act like a “gentle” booster — less likely to overfit when data volume or quality isn’t great.

Reference:

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html

Interesting Observations:

- The dataset contained several challenges typical of medical equipment transport, including skewed target distribution, missing values, categorical variables, and data entry issues like negative delivery times.
- Some models, like tree-based models, benefited from derived features capturing non-linear relationships, whereas simpler linear models did not gain as much from these additions.
- Linear Regression performed decently but was limited by its assumption of linearity; Elastic Net and outperformed other linear models by combining L1 and L2 penalties, balancing sparsity and robustness.
- KNN, after proper scaling and feature engineering, showed reasonable performance despite being a simple distance-based method, suggesting that engineered features helped reduce variance and improve neighborhood predictions.
- Adaboost outperformed XGBoost and Gradient Boosting likely because its adaptive weighing focused on hard to predict samples, and shallow decision trees as weak learners reduced overfitting while capturing complex patterns, making it more robust on noisy data.

References:

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html

https://scikit-learn.org/stable/modules/tree.html

https://scikit-learn.org/stable/modules/linear_model.html