ML Project Report: Medical Equipment Cost Prediction

TEAM: Hers

- Mannat Kaur Bagga - IMT2023071
- Divya Siddavatam – IMT2023059

Github link: Divyareddy8/ML_Project

Task Overview

The main goal of this project is to accurately predict the transport cost for medical equipment orders. By training machine learning models on historical delivery data, we aim to uncover how factors like equipment size, supplier reliability, and shipping urgency influence the final cost.

Having accurate predictions will help the company:

1. Plan logistics more efficiently.
2. Set fair pricing for shipping services.
3. Ensure timely delivery, especially for urgent or fragile items.

Dataset Description

The dataset contains detailed information about medical equipment deliveries to various hospitals. Each record represents a single delivery instance along with its corresponding transport cost. The dataset includes information about the hospital, supplier, equipment characteristics, shipping method, and special services.

- **Training set (train.csv):** 6,000 records with 20 columns, including the target variable Transport_Cost.
- **Test set (test.csv):** 500 records with 19 columns (target excluded).

Features Overview

- **Numeric Features:** Supplier Reliability, Equipment_Height, Equipment_Width, Equipment_Weight, Equipment_Value, Base_Transport_Fee
- **Binary Features:** CrossBorder_Shipping, Urgent_Shipping, Installation_Service, Fragile_Equipment, Rural_Hospital
- **Categorical Features:** Equipment_Type, Transport_Method, Hospital_Info, Supplier_Name, Hospital_Location
- **Date Features:** Order Placed Date, Delivery Date
- **Target Variable:** Transport Cost — represents the total cost of transporting equipment from supplier to hospital

EDA and Pre-processing

This section covers key steps for Exploratory Data Analysis (EDA) to extract insights, identify issues, and prepare the data for modeling.

1. Import Libraries and Load Dataset

- Imported essential libraries: pandas, numpy, matplotlib, seaborn, scikit-learn
- Loaded dataset into a pandas DataFrame for analysis

2. Duplicate and Missing Value Analysis

- Checked for and removed duplicate rows from the training set.
  - Duplicates Found: 0
  - Duplicates Removed: None (data already clean)
- Missing Value Analysis in Train Dataset:
  - Used train.isnull().sum() to identify columns with missing data
  - A summary of missing columns was shown to decide which imputations were required

3. Exploratory Data Analysis (EDA)

A. Feature Distributions

- Histograms were used to check how values are spread.
- **Highly Skewed Features:** Transport_Cost, Equipment_Value, Equipment_Weight, Equipment_Height, Equipment_Width showed strong right-skew
  - **Action Required** : Apply log transformation for balanced distribution, improving regression performance
- **Well-Distributed Features:** Supplier_Reliability, Base_Transport_Fee required less adjustment

B. Outlier Detection (Boxplots)

- Boxplots confirmed extreme outliers in Equipment_Height, Equipment_Width, Equipment_Weight, Equipment_Value, and Transport_Cost
- **Action Required** : Apply outlier clipping

C. Correlation Matrix

- Strong Positive Correlation: Equipment_Weight & Equipment_Value (~0.9)
- Moderate Positive Correlation: Equipment_Height & Equipment_Width (~0.77)
- **Action Required** : Feature engineering to handle co-related features

All the graphs plotted (correlation matrix, distribution plots, boxplots) are shown in *ML_Project/EDA_&_preprocessing.ipynb* on GitHub (Divyareddy8/ML_Project).

Data Cleanup

1. Date-based Features

- Converted dates to datetime format
- Derived features: delivery lag, day of the week, month of order
- Purpose: Capture temporal patterns influencing shipping delays or costs

2. Equipment and Cost Features (Handling co-related features)

- Created features capturing size, weight, density, cost efficiency, and special handling
- Added interaction terms
- Purpose: Encode structural relationships and operational factors affecting transport cost

3. Binary Features

- Standardized yes/no columns to 0/1
- Purpose: Ensure consistent interpretation by the model

4. Rare Category Grouping

- Low-frequency categories grouped into "Other"
- Purpose: Reduce noise and prevent overfitting during encoding

5. Column Drop & Data Split

- Dropped irrelevant/redundant columns, such as identifiers and original date columns
- Purpose: Simplify dataset, avoid leakage or redundancy

6. Target Variable Handling

- Applied log transformation to the target variable after clipping zeros
- Purpose: Reduce skewness, stabilize variance, improve regression performance

7. Outlier Handling

- Clipped extreme numeric feature values at the 1st and 99th percentiles
- Purpose: Limit influence of outliers without removing data

8. Feature Groups

- Organized features into numeric, categorical, and binary groups
- Purpose: Allow targeted preprocessing for each type

9. Preprocessing Pipelines

- **Numeric pipeline:** Missing value imputation, outlier clipping, standardization
- **Categorical pipeline:** Missing value filling, one-hot encoding
- **Column transformer:** Applies the appropriate pipeline to each feature group
- Purpose: Ensure consistent, reproducible transformations for training and test data

Find all preprocessing steps: EDA_&_preprocessing.ipynb (provided in the git repo)

Models Used For Training

1. XGBoost Regressor

- Handles non-linearities, feature interactions, and outliers efficiently
- Robust to overfitting with regularization (reg_alpha, reg_lambda) and subsampling
- **Pipeline:** Preprocessing + XGBoost
- **Hyperparameter Tuning:** GridSearchCV optimized n_estimators, max_depth, learning_rate, subsample, colsample_bytree, and regularization

Best Parameters:

{'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 4,

'n_estimators': 300, 'reg_alpha': 0.1, 'reg_lambda': 1.5, 'subsample': 1.0}

- Balanced complexity, stable learning, moderate trees, slight regularization → prevents overfitting
- **Performance (Test Dataset):** MSE ≈ 45x10^8

2. Random Forest Regressor

- Ensemble of decision trees; handles non-linearities, feature interactions, and outliers
- **Pipeline:** Preprocessing + Random Forest
- **Hyperparameter Tuning:** GridSearchCV optimized n_estimators, max_depth, min_samples_split, min_samples_leaf

Best Parameters:

{'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200}

- Balanced depth and leaf/split → stable, less overfitting
- **Performance:** MSE ≈ 48x10^8

3. AdaBoost Regressor (Decision Tree Base)

- Sequentially combines weak learners to reduce errors

- **Pipeline:** Preprocessing + AdaBoost
- **Hyperparameter Tuning:** GridSearchCV optimized n_estimators, learning_rate, tree max_depth

Best Parameters:

{'regressor__learning_rate': 0.05, 'regressor__n_estimators': 200,

'regressor__estimator__max_depth': 5}

- Balanced trees, learning rate, tree depth → stable, less overfitting
- **Performance:** MSE ≈ 41x10^8

4. Gradient Boosting Regressor

- Sequentially builds weak learners to reduce residual errors
- Handles non-linearities and feature interactions effectively
- **Pipeline:** Preprocessing + Gradient Boosting
- **Hyperparameter Tuning:** GridSearchCV optimized n_estimators, learning_rate, max_depth, subsample, max_features

Best Parameters:

{'learning_rate': 0.05, 'max_depth': 4, 'max_features': 'sqrt',

'n_estimators': 300, 'subsample': 0.8}

- Moderate depth, subsampling, learning rate → prevents overfitting, stabilizes learning
- **Performance:** MSE ≈ 59x10^8

5. Linear, Lasso, Ridge, ElasticNet, Polynomial (Degree 2)

Model	Notes	Performance	Pros & Cons
Linear Regression	Baseline, captures linear relationships, log-transformed target	MSE ≈ 43x10^8	Quick, interpretable; cannot handle multicollinearity/regularization
Ridge Regression	L2 regularization, reduces overfitting, handles multicollinearity	MSE ≈ 42x10^8	Stabilizes coefficients, better generalization

Model	Notes	Performance	Pros & Cons
Lasso Regression	L1 regularization, feature selection	MSE ≈ 42x10^8	Shrinks irrelevant features to zero, implicit feature selection
ElasticNet Regression	L1 + L2 regularization	MSE ≈ 43x10^8	Best trade-off bias/variance, robust to correlated features
Polynomial Regression (Degree=2)	Expands features with interactions and squares	MSE ≈ 50x10^8	Severe overfitting due to high dimensionality; not suitable without regularization

Hyperparameter Tuning for Regularized Linear Models
To optimize model performance, hyperparameter tuning was applied on the regularization parameter **α (alpha)** for **Lasso**, **Ridge**, and **ElasticNet** using **GridSearchCV with K-Fold cross-validation (K=5)**.
This ensured that the chosen alpha values generalized well across data splits instead of relying on a fixed alpha.
Regularization strength directly impacts bias–variance trade-off, and the tuned alphas yielded **more stable coefficients** and **lower mean squared errors**, confirming the effectiveness of cross-validated hyperparameter selection.

6. KNN Regressor

- Non-parametric, instance-based regression, predicts as weighted average of nearest neighbors
- **Tuned Parameters (GridSearchCV):** n_neighbors, weights ('uniform'/'distance'), p (distance metric 1=Manhattan, 2=Euclidean)
- **Strength:** Flexible, interpretable, captures non-linear patterns
- **Limitation:** Slow on large datasets, sensitive to feature scaling (handled via pipeline)
- **Performance:** MSE ≈ 57x10^8

7. Decision Tree Regressor

- Non-linear, tree-based regression, splits features recursively
- **Tuned Parameters (GridSearchCV):** max_depth=5, min_samples_split=2, min_samples_leaf=10
- **Strength:** Handles complex patterns, interpretable, robust to outliers, no scaling needed
- **Performance:** MSE ≈ 51x10^8

8. PCA Application

- Applied Principal Component Analysis (PCA) to Linear, Ridge, Lasso, ElasticNet, KNN, Polynomial
- Retained 95% variance, reducing dimensionality
- **Benefits:** Removes noise, reduces redundancy, improves computational efficiency, can help models generalize better
- **Note:** Slightly reduces interpretability of individual features
- **Performance (best score):** MSE ≈ 41x10^8

Performance Highlights and Observations

• **Best Performance with Linear Models with Regularization:**
o Lasso and Ridge Regression achieved the best MSE (~42×10^8) even without applying PCA.
o **Reason:** These linear models benefit from regularization (*L1* for Lasso, *L2* for Ridge), which stabilizes coefficients, reduces overfitting, and handles correlated features efficiently. Therefore, reducing dimensionality via PCA was not strictly necessary for them.

• **PCA Applied on KNN:**
o PCA slightly improved KNN performance (~41×10^8), as KNN is sensitive to high-dimensional feature space and feature correlations.
o **Reason:** PCA reduced noise and redundant information while retaining most variance, helping distance-based methods like KNN make better neighbor comparisons and generalize more effectively.

• **AdaBoost Regressor:**
o AdaBoost achieved competitive performance (MSE ≈ 41×10^8), outperforming most ensemble and tree-based models.
o **Reason:** The combination of weak learners (Decision Trees) with an optimal learning rate and number of estimators allowed AdaBoost to correct residual errors iteratively. This resulted in stable learning, reduced bias, and improved generalization compared to single decision trees or untuned ensemble models.

Takeaways

• Regularized linear models (**Lasso**, **Ridge**) performed well on the original feature space due to effective handling of multicollinearity through regularization.

• **PCA** was most beneficial for non-parametric, distance-based models like **KNN**, helping reduce dimensionality and improve accuracy.

• **AdaBoost** proved highly effective, leveraging weak learners and sequential learning to deliver strong predictive performance and generalization.

Key Observations from the Project

1. **Effectiveness of Hyperparameter Tuning with GridSearchCV**
    - GridSearchCV significantly improved model performance by identifying optimal hyperparameters for each model.
    - For ensemble models like XGBoost, Random Forest, and AdaBoost, tuning parameters such as max_depth, n_estimators, learning_rate, and regularization helped prevent overfitting and stabilize learning.
2. **Importance of Preprocessing Pipelines for Data Leakage Prevention**
    - Using separate pipelines for numeric and categorical features ensured that preprocessing steps (imputation, scaling, outlier clipping, encoding) were applied consistently to both training and test data.
    - This prevented data leakage and ensured reproducible, reliable model performance.
3. **PCA Application**
    - PCA was most useful for models sensitive to high-dimensional and correlated features, like KNN and Polynomial regression.
    - It reduced noise and redundancy while retaining 95% of variance, improving computational efficiency and helping KNN generalize better.
    - Linear models with Lasso and Ridge performed well **without PCA** due to their inherent regularization handling multicollinearity.
4. **Importance of Feature Selection and Engineering**
    - Engineered features such as equipment size, density, cost efficiency, delivery lag, and interaction terms significantly enhanced predictive power.
    - Handling skewed features via log transformation and clipping outliers improved model stability and accuracy.
    - Rare category grouping and binary standardization reduced noise and improved model generalization.
5. **Expectations from Non-linear Models**

   - Models like KNN, Decision Trees, Random Forests, and ensemble methods were expected to perform well due to their ability to capture non-linearities and feature interactions.
   - These models indeed handled complex patterns effectively, though performance varied based on hyperparameter tuning and preprocessing.

6. **Surprising Performance of Linear Regression**
   - Surprisingly, linear models with regularization (Lasso, Ridge) performed nearly as well as non-linear and ensemble methods, achieving some of the best MSE scores (~42x10^8).
   - This highlights the effectiveness of feature engineering, handling correlated features, and log-transforming skewed targets.

7. **Key Takeaways for Model Selection**
   - **Regularized linear models (Lasso/Ridge):** Best for structured features with correlations and moderate dataset size.
   - **Ensemble models (XGBoost, Random Forest, AdaBoost, Gradient Boosting):** Robust for capturing non-linear interactions and handling outliers.
   - **PCA:** Beneficial mainly for high-dimensional, distance-based models like KNN.

"Overall, the project demonstrated the effectiveness of regularized linear and ensemble models in predicting transport costs, with Lasso, Ridge, and AdaBoost achieving the most balanced performance."

References used:

Feature Importance in PCA: Analyzing Loadings and Biplots - GeeksforGeeks

Regularization in Machine Learning - GeeksforGeeks

https://scikit-learn.org/stable/