Laptop Price Prediction with Machine Learning

J. Kalistan
Department of Electrical, Electronic and Telecommunication, Faculty of Engineering,
General Sir John Kotelawala Defence University,
Rathmalana, Sri Lanka.

Abstract- In a rapidly evolving consumer electronics market, predicting laptop prices based on specifications is critical for both informed purchasing and strategic pricing. This project presents an end-to-end machine learning workflow for laptop price prediction, incorporating robust data preprocessing, feature engineering, and exploratory data analysis. Multiple regression models were evaluated, with the Random Forest Regressor demonstrating superior performance due to its accuracy and ability to model nonlinear relationships. The final model was deployed via a locally hosted Flask web application, providing an intuitive interface for real-time predictions. This system highlights the practical integration of machine learning into consumer-focused solutions and offers promising avenues for scalability and enhancement.

I. Introduction

In today's tech-savvy world, consumers face a wide range of choices when buying laptops, with prices varying greatly depending on specifications. This project aims to predict laptop prices using machine learning techniques, helping consumers make informed decisions and businesses offer competitive pricing.

The end-to-end workflow encompasses data preprocessing, feature engineering, exploratory data analysis (EDA), model selection, and evaluation. A user-friendly, locally hosted web application was also developed using Flask to allow price prediction through a simple interface.

II. Data Preprocessing

Python has become one of the most widely used programming languages in the field of data science and machine learning due to its simplicity, readability, and extensive ecosystem of libraries. It provides a versatile platform for building machine learning models, from data preprocessing to deployment. Machine Learning (ML), a subfield of artificial intelligence, involves training algorithms to recognize patterns in data and make predictions or decisions without being explicitly programmed. In this project, ML techniques are applied to predict laptop prices based on various hardware specifications.

This project was developed using Visual Studio Code (VS Code), it made use of the Jupyter Notebook interface within the editor. Jupyter Notebook is an open-source web-based tool that allows users to combine live code, visualizations, and narrative text in a single document. It is particularly popular in data science and machine learning workflows because it supports iterative development, interactive data exploration, and immediate feedback. Within VS Code, Jupyter functionality is seamlessly integrated, allowing users to run code in individual cells, visualize outputs directly, and maintain an organized and readable workflow.

To build the laptop price prediction model, several essential Python libraries were utilized. Pandas and NumPy were used for efficient data manipulation and numerical operations, allowing structured handling of tabular data and arrays. For data visualization, Matplotlib provided low-level plotting capabilities, while Seaborn offered higher-level statistical plots and attractive default styles, making it easier to interpret trends and relationships in the dataset. Finally, Scikit-learn was used for implementing machine learning models, data preprocessing, and performance evaluation, as it offers a rich suite of tools for classification, regression, and model selection.

The dataset, which includes various laptop specifications and their corresponding prices, was loaded from a .csv file using the pandas.read_csv() function. After loading, initial data exploration was performed using functions such as .head(), .info(), and .shape to inspect the structure of the dataset, identify data types, and check for missing values. This preliminary analysis provided insights into the overall composition of the data and helped guide subsequent preprocessing steps.

During the data cleaning phase, several preprocessing steps were applied to make the dataset suitable for modeling. Units such as 'GB' in the RAM column and 'kg' in the Weight column were removed, and the values were converted to numeric types. Rare laptop brands in the Company column were grouped under the label 'Other' to reduce category sparsity. The Screen Resolution column was parsed to extract resolution details, with uncommon resolutions similarly grouped. Two binary features, Touchscreen and IPS Display, were created by identifying keywords within the screen resolution text. Additionally, the Memory column was split into separate features for HDD and SSD storage capacities, enabling more granular analysis. Unnecessary columns that did not contribute to model performance, such as identifiers and redundant fields, were also removed from the dataset.

III. Feature Engineering

In the feature engineering phase, both categorical and numerical attributes were carefully transformed to improve model performance. For categorical features, one-hot encoding was applied to variables such as Company, TypeName, Operating System, and Screen Resolution, after consolidating rare categories into a general 'Other' group to reduce sparsity. The CPU and GPU columns were parsed to extract the primary processor and graphics card brands, which were then grouped and encoded based on frequency and relevance. The Memory column was decomposed into four distinct storage types: SSD, HDD, Flash, and Hybrid. However, Flash and Hybrid categories were later dropped from the dataset, as exploratory data analysis showed they had minimal impact on the laptop's price and were relatively rare, contributing little to predictive accuracy. For numerical features, the screen size (Inches) was rounded to the nearest common values and one-hot encoded to prevent overfitting from outliers. Key numeric features such as RAM, Weight, SSD, and HDD were preserved in their original form to retain their quantitative influence on price prediction.

IV. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) played a crucial role in uncovering patterns and relationships between various features and laptop prices. A correlation heatmap was generated to visualize the strength of relationships among numerical variables, particularly identifying how features like RAM, SSD size, and weight correlate with the target variable. Bar plots were used to display the average laptop prices across different categorical features such as brand (Company), processor type (CPU), graphics card (GPU), operating system, and display characteristics (Touchscreen and IPS), revealing distinct price trends across categories. Additionally, scatter plots were created to observe how continuous variables like SSD capacity, RAM size, and weight influence laptop pricing. These visualizations collectively provided valuable insights into which features have the most significant impact on price and guided further steps in model development.

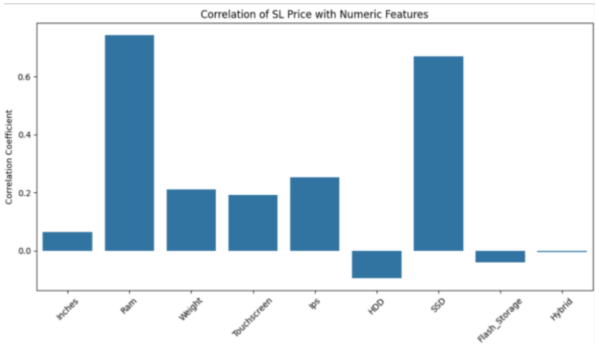


Figure 1: Correlation of Price with Numeric Features

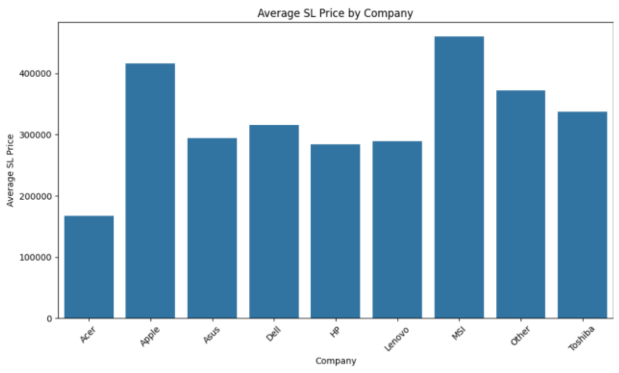


Figure 2: Average Price by Company

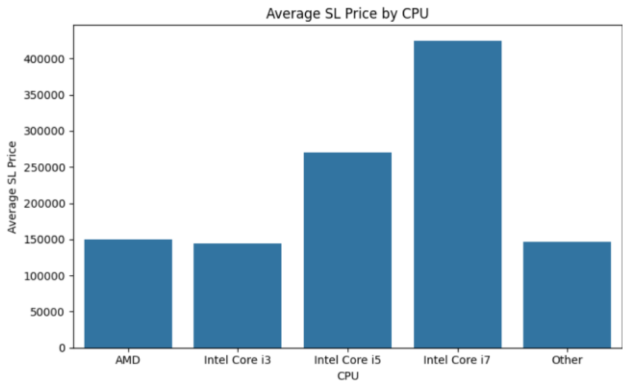


Figure 3: Average Price by CPU

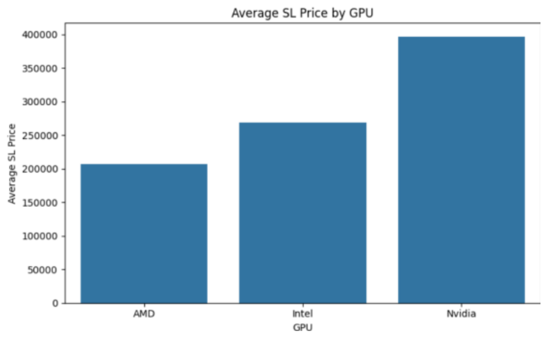


Figure 4: Average Price by GPU

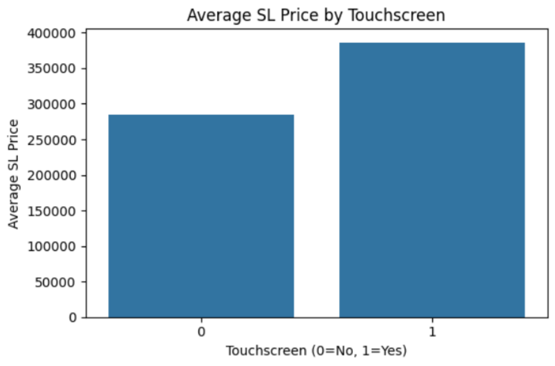


Figure 5: Average Price by Touchscreen

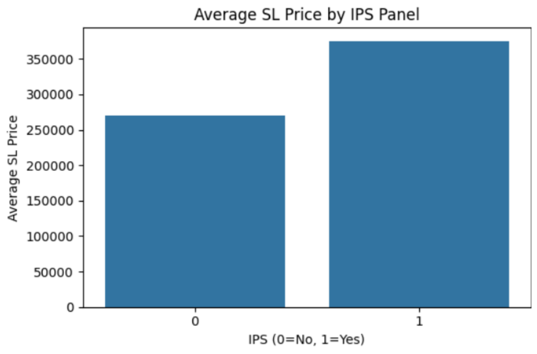


Figure 6: Average Price by IPS Panel

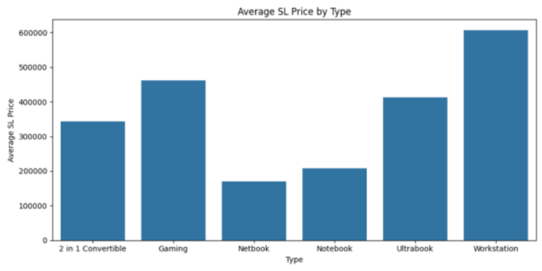


Figure 7: Average Price by Laptop Type

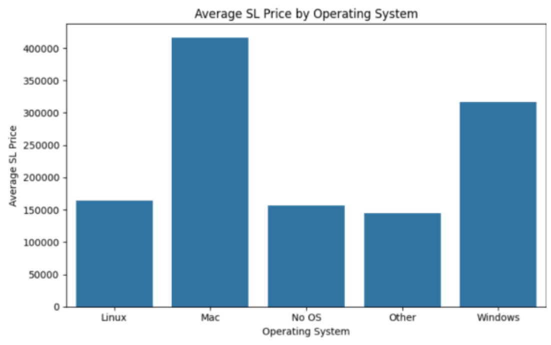


Figure 8: Average Price by Operating System

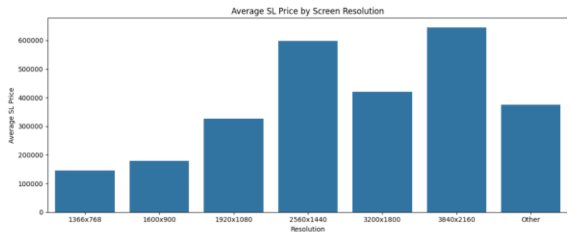


Figure 9: Average Price by Screen Resolution

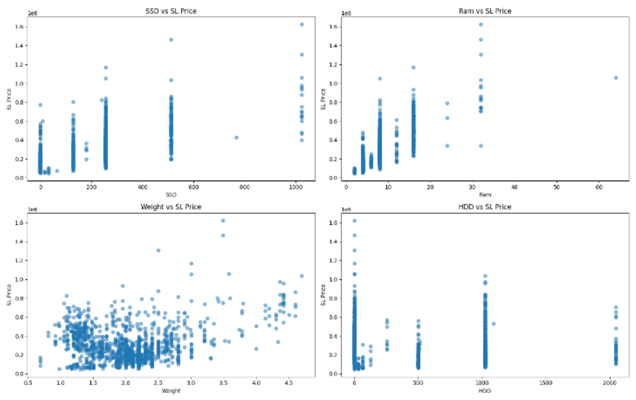


Figure 10: Scatter plots for SSD, HDD, Ram, Weight vs Price

V. Modeling

Before training the machine learning models, the dataset was prepared by splitting it into features and target variables. The target variable (y) was defined as the laptop price, while the remaining columns constituted the input features (X). To evaluate model performance on unseen data, the dataset was divided into training and testing subsets using the train_test_split() function from the scikit-learn library, with 75% of the data used for training and 25% reserved for testing. This approach ensures that the models are evaluated fairly and helps prevent overfitting by assessing their generalization ability on new, untrained data.

To identify the most effective algorithm for predicting laptop prices, four different regression models were implemented and evaluated: Linear Regression, Decision Tree Regressor, Random Forest Regressor, and Lasso Regression. Each model was trained using the prepared training set and evaluated on the test set based on the R² score, which measures the proportion of variance in the dependent variable explained by the model.

These models were chosen for their varying complexity, interpretability, and ability to capture linear or non-linear patterns, allowing a comprehensive comparison of performance on the dataset.

The performance and characteristics of each model were carefully analyzed. Linear Regression is one of the most fundamental algorithms in supervised learning, based on the assumption that there is a linear relationship between the input features and the target variable. It attempts to fit a straight line (or hyperplane in higher dimensions) through the data by minimizing the difference between the predicted and actual values using least squares minimization.

The mathematical representation of a multiple linear regression model is:

$$\hat{y}=\beta_0+\beta_1x_1+\beta_2x_2+\cdots+\beta_nx_n$$

Where:

- $\hat{y}$ is the predicted price,
- β_0 is the intercept,
- $\beta_1,\beta_2,\ldots,\beta_n$ are the model coefficients,
- $x_1,x_2,\ldots,x_n$ are the input features (e.g., RAM, SSD, Weight, etc.).

Although Linear Regression is fast, interpretable, and often a good baseline model, it assumes that the relationship between all features and the target variable is linear and independent, which is rarely the case in real-world data. It does not handle feature interactions or non-linear trends unless those are manually introduced as new features (e.g., polynomial terms). In this project, the model achieved an R^2 score of 0.797918, indicating that it explains about 79.8% of the variance in the laptop prices. However, this relatively modest score shows signs of underfitting, meaning the model was too simplistic to capture the complex relationships inherent in the data.

The Decision Tree Regressor is a non-linear model that uses a tree-like structure to split the dataset into subsets based on specific feature thresholds. At each internal node, the algorithm selects the feature and threshold that best reduce a chosen error metric (such as mean squared error) to divide the data. This process continues recursively until the tree reaches its maximum depth or another stopping criterion is met.

The prediction is made by averaging the target values within the leaf node where a data point lands. Unlike linear regression, decision trees can naturally model complex feature interactions and non-linear relationships, which makes them more flexible for real-world datasets.

However, this flexibility often comes at the cost of overfitting. Decision trees tend to memorize the training data when they grow too deep, resulting in poor generalization to unseen test data. In this project, while the model performed better than linear regression in capturing data patterns, it showed signs of overfitting. This was evident from its test set R^2 score of 0.731155, the lowest among the four models, indicating that the model was unable to maintain consistent performance on new data despite its higher complexity.

Lasso Regression is a linear regression model enhanced with L1 regularization, which adds a penalty term to the loss function proportional to the absolute values of the model coefficients. This encourages the model to reduce less important feature coefficients to zero, effectively performing automatic feature selection. The objective function minimized by Lasso is:

$$Loss=\sum(y_i - \hat{y}_i)^2 + \lambda\sum|\beta_j|$$

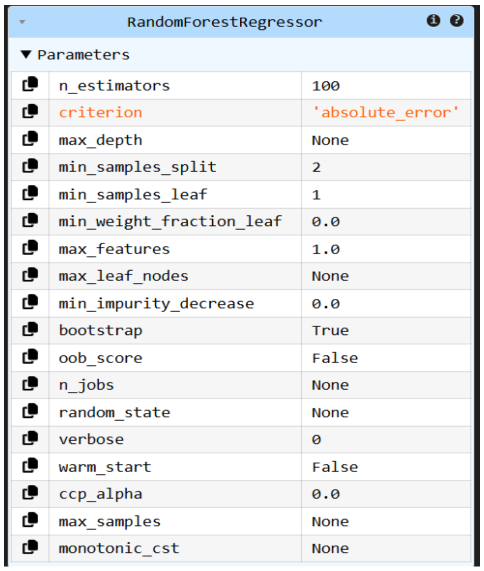
Where λ is the regularization parameter and β_j are the model coefficients. This regularization helps prevent overfitting by discouraging overly complex models.

While Lasso is effective when only a few features are significant, it still assumes a linear relationship between the inputs and the target variable. As a result, it was unable to capture the complex, non-linear interactions present in the laptop price dataset. Nevertheless, it performed slightly better than basic Linear Regression, achieving an R^2 score of 0.797978, which was the second-highest among the four models tested. Despite this, its performance was still inferior to the Random Forest Regressor, which better handled the data's complexity and variability.

The Random Forest Regressor is an ensemble learning method that combines the predictions of multiple decision trees to improve overall model accuracy and robustness. It operates by constructing a large number of decision trees during training, each on a random subset of the data and features (a technique known as bagging). The final prediction is then made by averaging the outputs of all individual trees, which helps reduce the variance typically seen in single decision trees and significantly mitigates the risk of overfitting.

This ensemble approach allows the model to capture complex, non-linear relationships in the data while maintaining strong generalization performance. Additionally, because each tree is exposed to a slightly different subset of the data, the forest collectively becomes more resilient to noise and anomalies.

In this project, the Random Forest Regressor achieved the highest R^2 score of 0.863563 on the test set, indicating that it explained approximately 86.36% of the variance in laptop prices. This superior performance, along with its robustness to overfitting and ability to model real-world data distributions, made it the best-performing model and the final choice for deployment.



RandomForestRegressor

▼ Parameters

n_estimators	100
criterion	'absolute_error'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	1.0
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
random_state	None
verbose	0
warm_start	False
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

Figure 11: GridSearchCV output

The hyperparameter tuning process was conducted using GridSearchCV, which systematically tested multiple combinations of parameters to identify the optimal configuration for the Random Forest Regressor. The best-performing model was found with n_estimators set to 100, meaning the ensemble was built from 100 individual decision trees. The criterion parameter was set to 'absolute_error', instructing the model to minimize the mean absolute error (MAE) rather than the default mean squared error (MSE). This choice makes the model more robust to outliers, as MAE gives equal weight to all errors. Other parameters such as max_depth, max_features, and random_state were left at their default values, suggesting that deeper customization did not yield better results during tuning. This configuration helped the model achieve better generalization and improved prediction accuracy to 0.867729, on the test dataset.

The Random Forest Regressor was ultimately chosen as the final model due to its:

- Superior accuracy on both training and unseen test data
- Robustness against noise and overfitting
- Ability to model complex, real-world relationships between features
- Consistent performance even before tuning, and significant improvement after optimization

Its balance between performance, flexibility, and robustness made it the most suitable model for this prediction task.

To enable deployment in the web application, the best-performing model—Random Forest Regressor—was serialized using Python's pickle library. This allowed the trained model to be saved in a binary format and later loaded directly into the Flask application for real-time predictions without the need for retraining.

VI. Flask Web Application

To make the machine learning model accessible to users, a web-based interface was developed using Flask, a lightweight and efficient Python web framework. Flask was chosen for its simplicity and speed, allowing for rapid development and deployment compared to heavier alternatives like Django. Its flexibility makes it particularly well-suited for small to medium machine learning applications where the developer needs full control over the application's structure. Additionally, Flask integrates seamlessly with Python-based ML models and supports libraries like pickle for loading pre-trained models. The application was designed for local hosting, making it ideal for prototyping and demonstrations without requiring cloud infrastructure. The interface presents a user-friendly form where users can input laptop specifications, and upon submission, the trained model instantly predicts and displays the estimated laptop price.

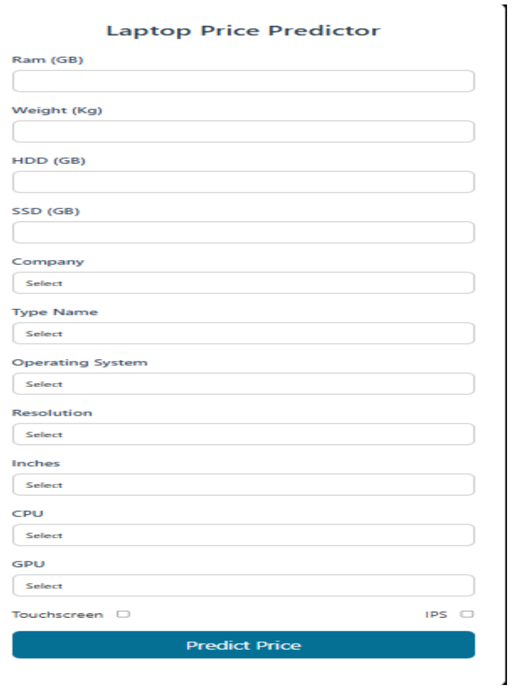


Figure 12: Laptop Price Predictor Web Interface

VII. Future Improvements

To further enhance the model's robustness and performance, several improvements can be considered. First, incorporating cross-validation would provide a more reliable estimate of model accuracy by reducing the impact of data split variability. Performing a residual error analysis, such as plotting predicted versus actual values, could help identify patterns of bias and model weaknesses. Analyzing feature importance may also reveal redundant or less significant variables, enabling simplification without sacrificing accuracy. Additionally, exploring more advanced algorithms like XGBoost, LightGBM, or CatBoost could lead to performance gains due to their superior handling of complex data patterns and built-in regularization. Finally, deploying the Flask web application on a public server such as Heroku, Render, or AWS would make the model accessible to a broader audience and pave the way for real-world usability.

VIII. Conclusion

This project showcases a complete machine learning pipeline designed to predict laptop prices based on hardware specifications. The process spans from thorough data cleaning and sophisticated feature engineering to model training, evaluation, and real-time deployment through a user-friendly web interface. The system delivers an accurate, interpretable, and practical solution for price prediction. Among the models tested, the Random Forest Regressor stood out as the best performer due to its strong generalization capability and ability to model non-linear relationships effectively. The integration of this model into a Flask web application further enhances accessibility, enabling seamless interaction between end-users and the predictive system.

References

[1] YouTube, “Machine Learning Project | Regression Problem | Laptop Price Predictor | Sinhala,” *YouTube*. http://www.youtube.com/playlist?list=PL495mke12zYBQjqBy-wUYh2LCAWSj4Ayn.

[2] “scikit-learn Tutorials,” *scikit-learn*, 2024. https://scikit-learn.org/1.4/tutorial/index.html

[3] “Tutorials — Matplotlib 3.6.0 documentation,” *matplotlib.org*. https://matplotlib.org/stable/tutorials/index.html

[4] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proc. 9th Python in Science Conf.*, 2010, pp. 51–56.

[5] Flask Documentation, "Flask: Web Development, One Drop at a Time," [Online]. Available: https://flask.palletsprojects.com/

[6] S. Raschka and V. Mirjalili, *Python Machine Learning*, 3rd ed. Birmingham, UK: Packt Publishing, 2019.

[7] B. Prajna, A. Sri, A. Sahiti, A. Teja, and A. Arzoo, “LAPTOP PRICE PREDICTION USING MACHINE LEARNING,” © *2023 IJNRD* |, vol. 8, p. 191, 2023, Available: https://www.ijnrd.org/papers/IJNRD2303124.pdf

[8] Saad Abd El-Ghaffar, “Laptop Prices,” *Kaggle.com*, 2023. https://www.kaggle.com/datasets/abdocan/laptop-prices (accessed Jul. 04, 2025).

[9] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York, NY, USA: Springer, 2009.

[10] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA, USA: Wadsworth, 1984.

[11] L. Breiman, “Random Forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. doi: 10.1023/A:1010933404324

[12] R. Tibshirani, “Regression Shrinkage and Selection via the Lasso,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.