

Understanding Used Car Sales Using Mathematics

Alberto, Benji, Mithun, Jackson

June 6, 2026

Abstract

Used car prices on cars.com are set by individual sellers and dealerships in a market where pricing information is asymmetric. We ask whether the listing price of a used car can be predicted from the features a buyer can observe before negotiating. This question motivates a careful study of four commonly used methods. The least squares are derived via calculus and orthogonal projection, which both result in $X^T X w = X^T y$. We then derive Ridge regression as the L2-regularized extension of OLS, with closed form $w = (X^T X + \lambda I)^{-1} X^T y$, and Lasso as its L1-regularized counterpart, showing why the L1 penalty admits no closed form and yields exact feature selection. Finally, we treat PCA as the eigendecomposition of the sample covariance matrix. On 641,406 listings with 606 features, our best linear model achieves test $R^2 \approx 0.785$ and RMSE $\approx \$8,920$. With a large number of observations per parameter, OLS does not overfit, the optimal Ridge penalty is effectively zero, and regularization neither helps nor hurts. PCA confirms that price-relevant variance is spread across many independent directions, so no small subset of components captures enough signal to justify dimensionality reduction.

1 Introduction and Setup

Our goal is to predict a single number for each car (its listing price in U.S. dollars) from a description of key features of the car. This regression problem is a natural way to study least squares, regularization, and dimensionality reduction. The dataset is a public collection of used-car listings. After cleaning it, it contains 641,406 rows. Data-cleaning logistics are covered in the encoding section and the limitations of listing-price data are covered in the limitations section; the bulk of this paper concerns the mathematics of the models.

We were curious whether the listing price of a used car could be predicted from features a buyer can observe before negotiating. How does its year, mileage, and manufacturer affect its price? This question is important because pricing information is asymmetric because dealerships see more transactions than individuals and try to use this to their advantage. A transparent model that shows observable features to predicted price gives both sides of the deal a clear understanding of how to price a car.

We describe the dataset and the encoding that converts categorical features into a numeric design matrix. We then derived the OLS estimator through both calculus and orthogonal projection. Our project then introduces Ridge and Lasso as regularized extensions of OLS, and PCA is treated as the eigendecomposition of the sample covariance matrix. We then compare performance across models, and address the limitations of listing-price data.

2 Data Representation and Encoding

2.1 Source and Scope

The data come from Andrei Novikov’s ”Used Cars Dataset” on Kaggle, which is a public scrape of cars.com. The raw dataset contains roughly 760,000 listings, each described by more than 20 fields covering vehicle specifications (manufacturer, model, year, drivetrain, fuel type, transmission), condition (mileage, accident and ownership history, exterior and interior color), and listing metadata (location, seller type, listing date). The data is from 2023.

2.2 Cleaning

A few listings carried impossible prices. Three were exactly \$1 billion (presumably data entry typos or non-serious listings). There were also \$1 vehicles (likely non-serious listing), and a scatter above \$500,000. We restricted the target to $\$500 \leq \text{price} \leq \$200,000$, removing 1,039 rows (0.16%); since the 99.9th price percentile was \$230,000, the cap stays close to the real distribution. This numerical cleaning was done through Stata, while an LLM was used to sort the numerous colors into one of twelve.

2.3 Features and one-hot encoding

As mentioned, the cars.com data set had over 20 different car features for each listing. To reduce the computational cost of the machine learning model, we used our judgement to narrow the 20+ features into the 12 features that, from our experience, had the greatest impact on car prices. The features we kept are outlined in Table 1 and Table 2, separated into numeric features and categorical features respectively. Before any model can be fit, the categorical descriptions of a car (its manufacturer, color, and so on) must be turned into numbers. A common practice when dealing with categorical variables in ML models is to use one-hot encoding, which transforms the categorical data into numerical data that the model can interpret. Take a categorical feature. One-hot encoding creates a new column for every ”level”. For example, for ”manufacturer”, a new column is created for each car manufacturer. This is done for all categorical features, and all one-hot columns are added to the design matrix. A specific entry on a one-hot column resolves to either 0 or 1, depending on whether the car has that specific feature. For example, the one-hot column for BMW resolves to 1 if the car is a BMW, and 0 otherwise.

Table 1: Numeric features (summary statistics).

Feature	Range	Mean
Year	1961–2023	2018
Mileage	0–1,119,067	55,490
Price	1,100–500,000	32,310
Accident Status	0–1	0.23
One-owner status	0–1	0.56

Table 2: Categorical features and number of one-hot columns each generates.

Feature	# levels
Manufacturer	30
Transmission	2
Drivetrain	2
Fuel type	5
Exterior color	13
Interior color	13
Car model	543
Total	608

3 Mathematical Setup and Notation

We predict one number per car, and we call the vector of all predicted prices $\hat{\mathbf{y}}$. The column vector $\hat{\mathbf{y}}$ is of length n , which in our case is equal to 641,406.

In our design matrix $\mathbf{X}$, every car in the data set is a row, so there are n rows, while every feature is a column. We also prepend a column of 1s to address the intercept. However, this column of 1s introduces a new problem. For a particular categorical feature, encoding all k levels makes $\mathbf{X}$ rank-deficient. Let $\mathbf{c}_1, \dots, \mathbf{c}_k$ be the k indicator columns for one categorical variable, and let $\mathbf{1}$ be the intercept column. Every car has exactly one level, so each row of the block $[\mathbf{c}_1 \cdots \mathbf{c}_k]$ contains exactly one 1. Summing the columns therefore reproduces the all-ones vector:

$$\mathbf{c}_1 + \mathbf{c}_2 + \cdots + \mathbf{c}_k = \mathbf{1}. \quad (1)$$

This means the intercept column is a linear combination of the indicator columns: the columns of $\mathbf{X}$ are linearly dependent, so $\mathbf{X}$ is not full column rank and $\mathbf{X}^T \mathbf{X}$ is singular. Essentially $\mathbf{X}^T \mathbf{X}$ is non-invertible, which will not work with the following section.

The way we fixed this is to drop one indicator per categorical variable. The dropped level is absorbed into the intercept, and each remaining coefficient is then read relative to that reference. Dropping a reference level for each category, we end up with $p = 606$ features. As a result, our design matrix is $\mathbf{X} \in \mathbb{R}^{n \times (p+1)}$. Diagrammatically, the matrix looks as follows

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,p} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{bmatrix}.$$

Finally, we define a weight vector $\mathbf{w}$. The specific weights are learned through training, which varies depending on the loss function used. Nonetheless, the weight vector is a column vector of length $p + 1$, and we interpret each w_j as the dollars added to the predicted price per one-unit increase in feature j . The resulting linear model is

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} = w_0 \cdot \mathbf{1} + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p. \quad (2)$$

4 Methods

4.1 Ordinary Least Squares (OLS)

OLS is the primary part of our project, with Ridge and Lasso being variations of it. In this section, we present the objective of OLS, justify it, derive its solution two independent ways, and then analyze when that solution exists and how it behaves.

Error on car i is the *residual* $y_i - (\mathbf{X}\mathbf{w})_i$. We need a single scalar scoring how wrong $\mathbf{w}$ is across all cars. Summing signed residuals is not the best solution, as a +\$10,000 miss and a -\$10,000 miss cancel, so we make each error positive by squaring it. Squaring beats the absolute value $|y_i - \hat{y}_i|$ in two ways: it is everywhere differentiable (the absolute value is not, at 0, which would make the calculus below impossible), and it penalizes large misses disproportionately. Summing gives the *sum of squared residuals*,

$$\text{SSR}(\mathbf{w}) = \sum_{i=1}^n (y_i - (\mathbf{X}\mathbf{w})_i)^2 = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2. \quad (3)$$

Two derivations for the solution of OLS:

4.2 Derivation 1: calculus

Expand using $\|\mathbf{v}\|^2 = \mathbf{v}^T \mathbf{v}$:

$$\text{SSR}(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) = \underbrace{\mathbf{y}^T \mathbf{y}}_{\text{constant}} - \underbrace{2\mathbf{w}^T \mathbf{X}^T \mathbf{y}}_{\text{linear in } \mathbf{w}} + \underbrace{\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w}}_{\text{quadratic in } \mathbf{w}}. \quad (4)$$

(The two cross terms $\mathbf{y}^T \mathbf{X}\mathbf{w}$ and $\mathbf{w}^T \mathbf{X}^T \mathbf{y}$ are equal because each is a 1×1 scalar and one is the transpose of the other.) $\mathbf{X}^T \mathbf{X}$ will always be positive, so the function, which is a parabola with respect to $\mathbf{w}$, opens upward. Taking the gradient with respect to $\mathbf{w}$,

$$\nabla_{\mathbf{w}} \text{SSR}(\mathbf{w}) = -2\mathbf{X}^T \mathbf{y} + 2\mathbf{X}^T \mathbf{X} \mathbf{w}. \quad (5)$$

Setting the gradient to zero yields the **normal equation** and its solution:

$$\boxed{\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}} \quad \implies \quad \boxed{\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}} \quad (6)$$

This inverse exists if and only if $\mathbf{X}^T \mathbf{X}$ is nonsingular, which holds exactly when $\mathbf{X}$ has full column rank—its columns are linearly independent. We previously showed how one-hot encoding can violate this, as the intercept column is an exact linear combination of others. We also showed how we dropping a level fixes this and makes $\mathbf{X}^T \mathbf{X}$ invertible.

In code we solve the linear system $\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$ directly rather than forming $(\mathbf{X}^T \mathbf{X})^{-1}$ explicitly, which is faster.

4.3 Derivation 2: orthogonal projection

The same equation falls out of geometry with no calculus. We seek the point in $\text{col}(\mathbf{X})$ closest to $\mathbf{y}$; by the projection theorem that point is the orthogonal projection of $\mathbf{y}$ onto $\text{col}(\mathbf{X})$, characterized by the residual $\mathbf{r} = \mathbf{y} - \mathbf{X}\hat{\mathbf{w}}$ being orthogonal to every column of $\mathbf{X}$:

$$\mathbf{X}^T (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}) = \mathbf{0} \quad \iff \quad \mathbf{X}^T \mathbf{X} \hat{\mathbf{w}} = \mathbf{X}^T \mathbf{y}. \quad (7)$$

This is (6) again. The fitted values can then be written through the *hat matrix*

$$H = \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T, \quad \hat{\mathbf{y}} = H\mathbf{y}, \quad (8)$$

which is symmetric ($H^T = H$). One immediate consequence is that because the intercept column $\mathbf{1}$ lies in $\text{col}(\mathbf{X})$, orthogonality (7) forces $\mathbf{1}^T \mathbf{r} = \sum_i r_i = 0$, i.e. the residuals of a model with an intercept always sum to zero. This is another reason we have an intercept column and a weight w_0 in our design matrix and weight matrix.

4.4 Ridge Regression (L2 Penalty)

The second type of regression we tested was Ridge. Ridge is an extension of OLS designed to tackle overfitting. It does so through the addition of an L2 penalty, weighted by a parameter λ . The overall goal of ridge is to find the weights $\mathbf{w}$, that satisfy,

$$\hat{\mathbf{w}}_{\text{ridge}} = \underset{\mathbf{w}}{\text{argmin}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|_2^2, \quad \|\mathbf{w}\|_2^2 = \sum_j w_j^2. \quad (9)$$

The penalty is smooth, so the calculus of Section 4.1 applies unchanged. Differentiating and setting to zero,

$$\nabla_{\mathbf{w}} \left[\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \mathbf{w}^T \mathbf{w} \right] = -2\mathbf{X}^T \mathbf{y} + 2\mathbf{X}^T \mathbf{X} \mathbf{w} + 2\lambda \mathbf{w} = \mathbf{0}, \quad (10)$$

which rearranges to the Ridge **closed form**

$$\boxed{\hat{\mathbf{w}}_{\text{ridge}} = (\mathbf{X}^T \mathbf{X} + \lambda I)^{-1} \mathbf{X}^T \mathbf{y}} \quad (11)$$

Fundamentally, Ridge trades off between variance and bias. By shrinking coefficients (through its penalty term), it decreases variance while introducing bias. In cases where OLS overfits training data, Ridge creates a model, with comparatively smaller weights, that performs better on the test data. However, in our case, OLS did not overfit. This is evident from our OLS results, where Train R^2 and Test R^2 are equal to two decimal places. Since Ridge is designed to address overfitting but there is no overfitting to address, we find that our results from OLS are almost identical to our results from Ridge. In fact, if we could have tested for every possible λ , we would have found that the optimal λ would have been marginally close to 0, nullifying the penalty term and reducing the Ridge regression to a standard OLS.

Unfortunately, in our code, we did not test for every possible λ . Instead, we used a single-hold-out split over 19 different values of λ . The values of λ we tested were $10^{-3}, 10^{-2.5}, 10^{-2}, 10^{-1.5}, \dots, 10^6$. For each value of λ , we trained the model to find $\mathbf{w}$ (equation 9), and used the λ that minimized the objective function ($\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2$). It is no surprise that the optimal $\lambda = 10^{-3}$, which was the smallest value we tested. The results of our ridge regression can be seen in section 6.

4.5 Lasso Regression (L1 Penalty)

Lasso replaces Ridge's L2 penalty with an L1 penalty. This small change has two large consequences. Firstly, the problem no longer admits a closed form, and coefficients are reduced to zero, rather than merely shrinking coefficients like in Ridge. As a result, Lasso doubles as a regularization tool and an automatic feature selection, since the least important coefficients in the dataset are reduced to zero. The objective function is as follows

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|, \quad (12)$$

and we train our model to find the $\mathbf{w}$ that satisfies,

$$\hat{\mathbf{w}}_{\text{lasso}} = \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|, \quad \|\mathbf{w}\| = \sum_j |w_j|. \quad (13)$$

OLS and Ridge have closed forms because their objectives are differentiable everywhere, so setting the gradient to zero and solving produces a linear system. The Lasso objective contains $|w_j|$, which is not differentiable at 0 since its right-hand slope is $+1$, its left-hand slope is -1 , and its slope is undefined at the kink.

In this case, we found optimal λ using 3-fold cross-validation on a subsample of 100,000 cars. The way this works is that we split our sub-sample into 3 equal "chunks", two for training and one for testing. Then, for each λ on the grid (our grid had 10 values), we trained on 2 chunks, and measured the error on the remaining chunk. We do this two more times so that every chunk serves as the testing set once. We then average the three errors to get one measurement error for that λ . We then repeated this across all 10 values of λ in the grid, and picked the λ that minimized the error as our optimal λ . In our case, the optimal $\lambda = 9.837$.

Our results for our Lasso regression can be seen in section 6. Specifically, we provide the Test R^2 as well as the pruned features through Lasso's feature selection mechanism.

5 Principal Component Analysis (PCA)

The regression models above ask which features predict price. PCA asks along which directions do the cars themselves vary most? The eigendecomposition of the covariance matrix is the answer to this question. We ran it partly to see whether the 606-dimensional feature space could be compressed to a handful of directions.

5.1 Setup

We first standardize every feature by subtracting its mean and dividing by its standard deviation, so that PCA's variance maximization is not dominated by features on large numeric scales. If **price** has variance in the tens of thousands and encoded categorical variables have a much smaller variance, PCA will essentially ignore the categoricals and the first principal component will just be **price**. Scaling puts everything on equal footing so PCA can identify the directions of greatest variance between car listings.

On the centered, standardized data the sample covariance matrix is

$$S = \frac{1}{n-1} \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \in \mathbb{R}^{p \times p}, \quad (14)$$

symmetric and positive semidefinite. PCA seeks orthonormal directions $\mathbf{v}$ that successively capture the most variance of the projected data $\tilde{\mathbf{X}}\mathbf{v}$, whose variance is $\operatorname{Var}(\tilde{\mathbf{X}}\mathbf{v}) = \mathbf{v}^T S \mathbf{v}$.

5.2 The first principal component

We maximize the projected variance subject to a unit-length constraint:

$$\mathbf{v}_1 = \underset{\|\mathbf{v}\|=1}{\operatorname{argmax}} \mathbf{v}^T S \mathbf{v}. \quad (15)$$

Introduce a Lagrange multiplier ℓ for the constraint $\mathbf{v}^T \mathbf{v} = 1$:

$$\mathcal{L}(\mathbf{v}, \ell) = \mathbf{v}^T S \mathbf{v} - \ell (\mathbf{v}^T \mathbf{v} - 1). \quad (16)$$

Setting the gradient to zero,

$$\nabla_{\mathbf{v}} \mathcal{L} = 2S\mathbf{v} - 2\ell\mathbf{v} = \mathbf{0} \implies S\mathbf{v} = \ell\mathbf{v}. \quad (17)$$

So any stationary $\mathbf{v}$ is an eigenvector of S with eigenvalue ℓ . At such a point the objective becomes $\mathbf{v}^T S \mathbf{v} = \ell \mathbf{v}^T \mathbf{v} = \ell$, meaning the variance captured along an eigenvector is exactly its eigenvalue. To maximize variance we therefore take the eigenvector corresponding to the largest eigenvalue ℓ_1 . This eigenvector is the first principal component $\mathbf{v}_1$.

5.3 Subsequent components and the spectral theorem

The second component maximizes the same objective $\mathbf{v}^T S \mathbf{v}$ under the added constraint of orthogonality to the first, $\mathbf{v}_1^T \mathbf{v} = 0$:

$$\mathbf{v}_2 = \arg \max_{\|\mathbf{v}\|=1, \mathbf{v} \perp \mathbf{v}_1} \mathbf{v}^T S \mathbf{v}, \quad (18)$$

whose Lagrangian carries a second multiplier and yields the eigenvector of the second-largest eigenvalue. Continuing adds one orthogonality constraint per component. Because S is symmetric, its eigenvectors are already orthogonal to one another, and the principal components are simply the eigenvectors of S taken in order of decreasing eigenvalue. Since standardization makes each feature’s variance 1, $\text{tr}(S) = p$, and the proportion of variance explained by component i is

$$\frac{\ell_i}{\sum_j \ell_j}. \quad (19)$$

5.4 Results and why PCA did not compress our data

The `car_model` variable had 543 unique values — the most of any feature by a wide margin. Because one-hot encoding such a high-cardinality variable would inflate dimensionality and dilute variance across hundreds of near-empty dummies, we ran two parallel analyses: **Version A** excluding car model (69 total components) and **Version B** including it (700+ components). The results summarized in Table 3.

Table 3: PCA results for Version A (without car model) and Version B (with car model).

Metric	Version A	Version B
Total components	69	700+
PC1 variance explained	4.03%	0.48%
PC1 + PC2 variance explained	7.30%	—
Variance at 50 components	—	~18%
Components needed for 90%	53	>700

In Version A, PC1 is dominated by drivetrain, year, mileage, and one-owner status, while PC2 is dominated by fuel type and transmission. All subsequent components account for negligible levels of variance. In Version B, the curve flattens even further: 50 components yield only about 18% of the variance, confirming that one-hot encoding the 543-level car-model variable sprays variance across hundreds of near-empty dummies.

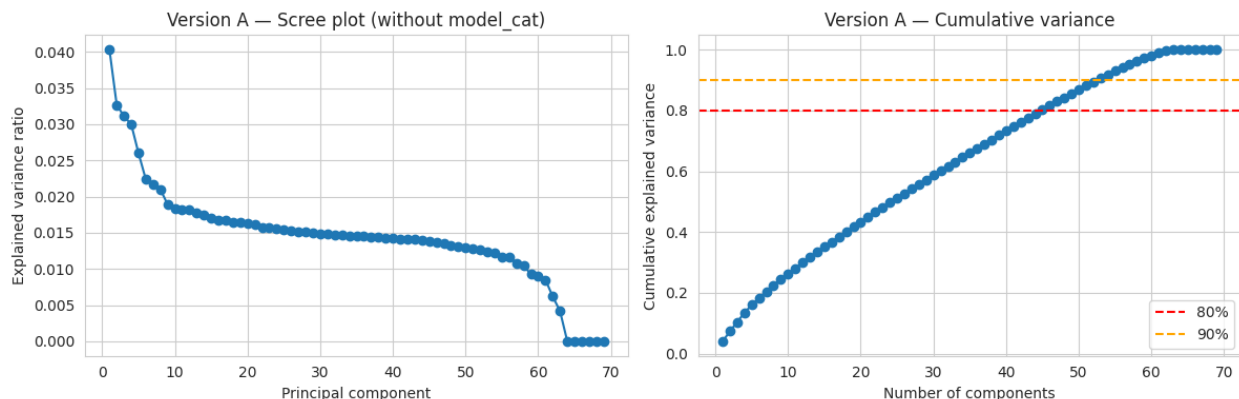


Figure 1: Version A — without car model. Left: the scree plot shows the variance explained by each individual principal component. The first component captures only 4.03% of the total variance, and the curve descends gradually rather than dropping sharply after a few components — there is no clear elbow. Right: the cumulative explained variance climbs nearly linearly, crossing 80% around the 45th component and 90% around the 53rd. With 69 total directions and no early concentration of variance, PCA finds no low-dimensional subspace to compress into.

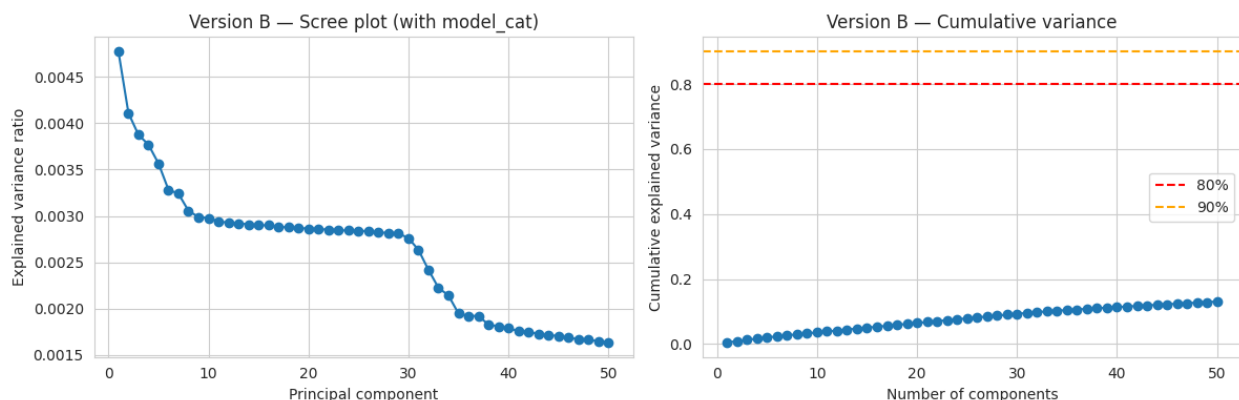


Figure 2: Version B — with car model. Adding the 543-level `car_model` variable inflates the feature space to 700+ dimensions and dilutes variance even further. Left: the leading component now captures only about 0.48% of the variance, an order of magnitude less than Version A. Right: the cumulative curve barely rises — after 50 components it has accumulated only $\sim 18\%$ of total variance, far below both the 80% and 90% thresholds. High-cardinality one-hot encoding sprays variance across hundreds of near-independent, near-equal-weight directions, which is the worst-case input for PCA.

The mathematics explains the failure. PCA compresses well only when a few eigenvalues of S dominate, with correlated features and low-dimensional subspace data. Our cars vary along many weakly-correlated directions. Worse, one-hot encoding is actively counterproductive for PCA because each indicator column is sparse and nearly orthogonal to the others, and standardizing a rare dummy inflates its unit-variance contribution, so high-cardinality categoricals spray variance across hundreds of near-independent low-variance axes. The takeaway is that PCA is the wrong tool for a predominantly one-hot feature space, as we could not reduce dimensionality without discarding real signal.

6 Results

6.1 OLS

Our results from OLS are as follows.

Table 4: OLS performance as features are added.

Model	Test R^2	Test RMSE
4 numeric only	0.2956	\$16,147.86
40 features (no car model)	0.5441	\$12,991.07
606 features (full)	0.7850	\$8,920.61

What we can observe is that adding more features drastically improves the performance of our model. Specifically, it decreases the Test RMSE, which measures how far the model's predictions are from the real price on average, from over \$16,000 to below \$9000. Furthermore, although not represented in the table, we find that our Train R^2 for the "full" model is equal to 0.7835. This is almost equivalent to the observed Test R^2 , implying that standard OLS leads to negligible levels of overfitting. This fact becomes important when explaining why the Ridge and Lasso are so similar to the OLS results. Finally, the lack of overfitting can be explained by the abundance of observations relative to features (641,406 cars vs 606 features).

6.2 Ridge

As mentioned, we tested the regularization strength λ across 19 values spanning 10^{-3} to 10^6 , retraining the closed-form Ridge estimator $\mathbf{w} = (X^\top X + \lambda I)^{-1} X^\top \mathbf{y}$ at each. The table below reports performance at each value of λ .

Table 5: Ridge regression on the full 606-feature design: test R^2 and RMSE across values of λ . The optimum is the smallest λ , where Ridge reproduces OLS; performance decays monotonically as λ grows.

λ	Test R^2	Test RMSE
0.001	0.7850	\$8,920.67
0.003162	0.7850	\$8,920.78
0.01	0.7850	\$8,921.08
0.03162	0.7850	\$8,921.68
0.1	0.7850	\$8,922.46
0.3162	0.7849	\$8,923.48
1	0.7848	\$8,925.71
3.162	0.7845	\$8,932.74
10	0.7832	\$8,959.50
31.62	0.7792	\$9,040.24
100	0.7698	\$9,230.46
316.2	0.7496	\$9,628.17
1,000	0.7116	\$10,333.09
3,162	0.6506	\$11,373.91
10,000	0.5706	\$12,607.36
31,620	0.4884	\$13,761.37
100,000	0.4158	\$14,706.67
316,200	0.3553	\$15,449.34
1,000,000	0.3162	\$15,910.86

From Table 5, it is clear that the optimal λ , which minimized Test RMSE and maximized Test R^2 , was $\lambda = 0.001$, a value so small that it essentially reduces Ridge to an OLS regression. It is no coincidence that the optimal λ coincided with the smallest λ we tested, which stems from the lack of overfitting in the standard OLS regression mentioned in section 4.4.

Table 6: Lasso regression at the optimal penalty $\lambda \approx 9.84$ (selected by 3-fold cross-validation; precise value 9.837).

Optimal λ	Test R^2	Test RMSE
≈ 9.84	0.7836	\$8,949.72

6.3 Lasso

Table 6 (above) reports the results from the Lasso regression. Table 7 exhibits Lasso’s role as a feature selector. In section 4.5, we mentioned how Lasso reduces ”unimportant” coefficients to zero, unlike Ridge which merely shrinks coefficients. From Table 7, we observe that Lasso reduced a total of 54 coefficients to zero, leaving behind a simpler model than that of standard OLS and Ridge. Most of the ”pruned” coefficients were colors, car models, and manufacturers.

Table 7: Lasso feature selection by category at optimal λ .

Category	Total	Kept	Pruned	% kept
Colors	24	21	3	87.5%
Models	543	500	43	92.1%
Manufacturers	29	22	7	75.9%
All non-numeric	602	548	54	91.0%

6.4 Model Comparison

Having derived and fit all three regression models, we place them side by side on the held-out test set. Table 8 reports test R^2 , test RMSE, and the number of features each model actually uses (a coefficient counts as “used” if it is nonzero).

Table 8: The three regression models on the held-out test set, all fit on the full extended feature set. “Features used” counts nonzero coefficients. Accuracy is essentially tied; only Lasso reduces the feature count.

Model	Test R^2	Test RMSE	Features used
OLS (extended)	0.7850	\$8,920.61	606
Ridge ($\lambda = 0.001$)	0.7850	\$8,920.67	606
Lasso ($\lambda \approx 9.84$)	0.7836	\$8,949.72	552

The three models are statistically indistinguishable on accuracy. Their test R^2 values agree to three decimals and their RMSEs fall within \$30 of one another, on prices ranging into six figures. Regularization neither helped nor hurt. This is a sign of the data-abundant, no-overfitting OLS model. At the end of the day, Ridge and Lasso are there to prevent overfitting, but there is no overfitting to prevent. Therefore, it makes sense why the outputs across the three models are so similar to each other.

Second, OLS and Ridge use all 606 features, whereas Lasso reaches virtually the same accuracy with 552, driving 54 coefficients exactly to zero. When predictive performance is tied, the model that attains it with fewer features is preferable on grounds of interpretability, simplicity, and cost-efficiency. This singles out Lasso as the most useful of the three.

7 Issues, Challenges, and Limitations

Several caveats stem from the dataset rather than the mathematics. The prices are *listing* (asking) prices, not closing sale prices, so the model cannot see bargaining, discounts, or auction dynamics. The data is roughly three years old, and used-car prices have shifted with inflation since. We attempt to ameliorate this challenge on the front end. First, we simply shifted the year value by three years, as a 2023 model in 2023 is the same age as a 2026 model in 2026. However, this does not account for inflation and the shifting rates in used car values. Therefore, we further adjusted the values in line with the Bureau of Labor Statistics Used Cars and Trucks Consumer Price Index.

The dataset is national and carries no geographic adjustment, though the same car lists for systematically different amounts in different regions. The feature set omits trim and condition, which are precisely the attributes that explain the largest residuals.

Methodological limitations. Our data-abundant regime, compared to the number of features, meant that OLS never overfit, so Ridge and Lasso had no accuracy advantage to demonstrate. That being said, although Ridge and Lasso look useless at face value (since the Test R^2 and RMSE are virtually the same as for OLS) both regressions were still very informative in different ways.

8 Conclusion

We set out to test whether the listing price of a used car could be predicted from features a buyer can observe before negotiating. On 641,406 listings with 606 features, our full OLS model reached a test R^2 of 0.785 and a test RMSE of \$8,920.

Of the three loss functions, Lasso produced the most valuable result, even though it was not the most accurate in terms of Test RMSE. All three were statistically indistinguishable, with R^2 values agreeing to three decimal places and RMSEs within \$30 of one another. What set Lasso apart was reaching that accuracy with only 552 features, compared to 606. In class, we have often talked about the trade-off between accuracy and complexity. If we can reduce computational cost (by using fewer features) without sacrificing performance, that is always a trade we are willing to make. Furthermore, the reason Ridge and Lasso offered no accuracy gain is that the OLS regression never overfit. This is why the optimal Ridge penalty settled at the smallest value we tested and why Lasso matched OLS while merely trimming the weakest columns.

Our PCA analysis yielded two key learnings. Firstly, Version A shows that cars in the data set vary along many independent dimensions, which is why we need many PCs to capture most of the variance. Secondly, Version B exposed the weakness of pairing PCA with one-hot encoding. One-hot encoding a high-cardinality variable like car model dilutes variance across hundreds of near-empty dummies, undermining the usefulness and interpretability of PCA.

Overall, although our model was not particularly useful as a tool to be used in our day-to-day lives, it was instrumental in helping us understand what goes on behind the scenes in an ML model.

9 Live Demo and Frontend

To make our models accessible beyond static figures, we built a web-based frontend that exposes all three regression models through a simple form interface. The user enters a car's features — year, mileage, manufacturer, model, drivetrain, fuel type, transmission, colors, accident history, and ownership history — and the frontend returns each model's predicted price side by side, along with our headline estimate. The interactive demo is hosted at:

<https://webapp-murex-iota.vercel.app>

The frontend turns the mathematics derived in this paper into something a buyer or seller can actually use: type in the car you're looking at, and see whether the listing price falls in line with what our model would predict.