Project Proposal Submission

Business Analytics for Managerial Decision

Group 2

MBA/0089/61	AMITESH GORLE
MBA/0100/61	GEESALA SIVA ABHISHEK KUMAR
MBA/0136/61	SAYANTAN PAL
MBA/0340/61	DAKSHITH MADHOGARIA
MBA/0409/61	AMAN KUMAR SAMAL

The project in itself as described earlier is about the used car market is fragmented and heavily influenced by negotiation, local knowledge, and asymmetric information. Here we see that pricing decisions are often suboptimal due to:

- Lack of dynamic, ML-based pricing models
- Regional demand variability
- Macroeconomic and vehicle-specific factors (GDP, fuel, age, mileage, etc.)

Objectives

1. **Predict Fair Market Value** of used cars using advanced ML.
2. **Enable Location-Based Pricing** via geo-tagged data.
3. **Analyze Demand Trends** by region, car-type, and time.
4. **Model Depreciation** over time and mileage.
5. **Detect Anomalies/Fraud** in listings (e.g., unrealistic prices, missing VIN).
6. **Recommend Vehicles** to users based on preferences.
7. **Visualize Heatmaps** for price and listing density.
8. **Benchmark ML Models** for performance.

Data Sources

- **Craigslist US Used Vehicle Listings** (~426,881 entries)
- **US GDP Data** (1960–2021)

Attributes (27 Columns):

Grouped by themes:

- **Identification**: id, url, posting_date, region, region_url
- **Vehicle Specs**: price, year, make, model, fuel, cylinders, odometer, transmission, drive, size, type, condition, paint_color, VIN, title_status
- **Description & Media**: image_url, description
- **Location**: state, lat, long
- **GDP Data**: NY.GDP.MKTP.CD, year

Data Quality Observations:

- Missing values in condition, paint_color, VIN
- Duplicate listings
- Useful for anomaly detection models

Stages of Project

We solved out PROJECT in 3 stages

Part 1:- Data analysis majorly

Part 2 :- Price prediction model with without location data alongside mapping of various location related metrics and graphs

Part3:- Full fledge price prediction model

By the end of the project our status on the topics are as follows:-

Predict Fair Market Value

Built a model for the same, explained in Part 3

Enable Location-Based Pricing Strategy

Built a model both for non-location linked and location linked pricing strategy in Part 2 and Part 3 of the project respectively

Analyse Vehicle Supply & Demand Trends

Identifed high-demand vehicle types and understocked segments across regions to guide inventory and sourcing decisions for local suppliers in both Part 1 and Part 2 of the project

Model Depreciation Impact

Quantifie how mileage, vehicle age, and condition affect resale value in part 3 of the project

Evaluate and Benchmark ML Models

Part 1 and part 3 of the project have 15 models that are compared across

Detect Risky or Fraudulent Listings

Outliers were identified and they were also removed such as prices above 40000 for a market where average price is 10000 for a model. Such functions were integrated in the model.

Developed Interactive Heatmaps

Visualized pricing, listing density, and demand-supply gaps geographically using state/city/lat-long overlays in Part 2 of the project

Analysed relationships between parameters that drive insights

Built an interactive, role-based insight table to present actionable insights on pricing, risk, regional trends, and model performance.

PART 1

Columns in dataset

```
train0.columns #columns of train0
```

```
Index(['url', 'city', 'city_url', 'price', 'year', 'manufacturer', 'make',
       'condition', 'cylinders', 'fuel', 'odometer', 'title_status',
       'transmission', 'VIN', 'drive', 'size', 'type', 'paint_color',
       'image_url', 'desc', 'lat', 'long'],
      dtype='object')
```

Column Charateristics

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 525839 entries, 0 to 525838
Data columns (total 22 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   url           525839 non-null  object 
 1   city          525839 non-null  object 
 2   city_url      525839 non-null  object 
 3   price         525839 non-null  int64  
 4   year          524399 non-null  float64
 5   manufacturer  501260 non-null  object 
 6   make          517201 non-null  object 
 7   condition     279881 non-null  object 
 8   cylinders     315439 non-null  object 
 9   fuel          521544 non-null  object 
 10  odometer      427248 non-null  float64
 11  title_status  523014 non-null  object 
 12  transmission  521572 non-null  object 
 13  VIN           305650 non-null  object 
 14  drive         374475 non-null  object 
 15  size          174519 non-null  object 
 16  type          376906 non-null  object 
 17  paint_color   354306 non-null  object 
 18  image_url     525831 non-null  object 
 19  desc          525815 non-null  object 
 20  lat           513618 non-null  float64
 21  long          513618 non-null  float64
dtypes: float64(4), int64(1), object(17)
memory usage: 88.3+ MB
```

Dropped columns

'url', 'city', 'city_url', 'make', 'title_status', 'VIN', 'size', 'image_url', 'desc', 'lat','long'

This allowed us to make the data set lean, following this we tagged this data set as train0

Data Subsetting and Feature Correlation

After cleaning and engineering the data, we proceeded to explore the relationships between numerical features using the train0.corr() method. This allowed us to identify potential patterns and multicollinearity within the dataset.

we observed a moderate positive correlation of **0.53 between price and cylinders**, suggesting that vehicles with more cylinders tend to have higher prices.

Model Selection and Evaluation Setup

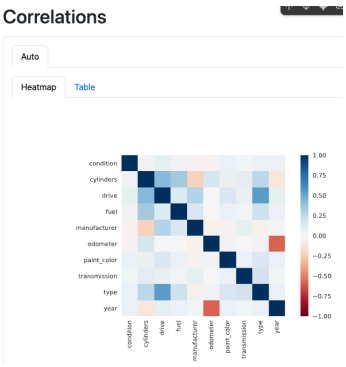
To ensure consistent and meaningful comparison across multiple machine learning models, we defined a standardized evaluation framework. We created six custom functions, three of them are:-

- **acc_d** calculates the **relative error** by scaling the Mean Absolute Error with the total of actual price values, giving us a percentage-based understanding of error.
- **acc_rmse** computes the **Root Mean Squared Error (RMSE)** to measure the average magnitude of the prediction errors.
- **acc_model** serves as a wrapper function that trains any given model and reports its performance using acc_d, acc_rmse, and the r^2 score from scikit-learn, for both the training and test sets.

Profiling Report

We then proceeded to create a profiling report of the whole dataset.Some of the values we received were the following :-

1. Interaction
2. Correlations
3. Missing values
4. Sample
5. Duplicate rows and more



The profiling report provided a high-level statistical summary of the dataset, including value distributions, missing value counts, cardinality, and basic correlation matrices.

However, since much of the profiling output was auto-generated and largely descriptive in nature, we determined that its inclusion would not significantly enhance the analytical value of this report for a BAMD assignment. Instead, we used the insights derived from the profiling process to guide our feature engineering, anomaly detection, and modelling strategies.

Accuracy Tracking and Model Evaluation Functions

To systematically compare the performance of different models, we began by initializing empty lists to store evaluation metrics for both the training and testing datasets. Specifically, we tracked:

- R^2 score (explained variance)
- Relative error (custom metric we called acc_d)
- Root Mean Squared Error (RMSE) for error magnitude

Boosting Model Evaluation Wrapper

We further developed a reusable function named acc_boosting_model() to evaluate boosting models like XGBoost and LightGBM. This function:

- Accepts the model, training/testing datasets, and optional num_iteration parameter.
- Makes predictions on both training and test sets.
- Appends the resulting R^2, acc_d, and RMSE values to the corresponding global lists.

- Prints the first five predictions vs. actual target values to visually inspect the prediction quality.

This approach streamlined the evaluation of multiple boosting algorithms and allowed us to build a comparative dashboard across models.

By storing all metrics in lists, we were able to quickly tabulate, visualize, and analyze performance trends across different models, hyperparameter settings, and iterations.

Model Accuracy Computation and Evaluation Logic

To ensure consistency in measuring model performance, we implemented a structured accuracy evaluation function called acc_model(). This function calculates multiple performance metrics for both the training and testing datasets, helping us assess how well each machine learning model generalizes.

How It Worked

Within acc_model(), we:

- Predict the **target variable** (price) on both training and test datasets using the given model.
- Print the first 5 values of actual (target) and predicted (ytrain, ytest) prices for quick inspection.
- Calculate and store three key metrics

Model Training and Comparison(15 models)

We compared 15 models those were:-

```
'Model': ['Linear Regression', 'Support Vector Machines', 'Linear SVR',
          'MLPRegressor', 'Stochastic Gradient Decent',
          'Decision Tree Regressor', 'Random Forest',  'XGB', 'LGBM',
          'GradientBoostingRegressor', 'RidgeRegressor', 'BaggingRegressor', 'ExtraTreesRegressor',
          'AdaBoostRegressor', 'VotingRegressor'],
```

- For each model, it outputs the R^2 score, relative error, and RMSE for both the training and testing sets.

Model Performance Analysis

After training and evaluating 15 different machine learning models on our used car price prediction dataset, we sorted them based on **R^2 score on the test set**, which reflects the model's ability to generalize to unseen data. We also reviewed additional metrics like **relative error** (d_train, d_test) and **RMSE** (Root Mean Squared Error) for both train and test sets to ensure robustness

Below is a detailed interpretation of the results:

Top Models

1. **Voting Regressor**
    a. **R^2 (Test): 86.05%**, **RMSE: ₹321,994**
    b. Aggregates predictions from multiple models for improved stability and accuracy.
    c. Achieved the best test performance, making it our most reliable choice.
    d. Slight overfitting observed (R^2 train = 97.52%) but generalization remains strong.
2. **XGBoost**
    a. **R^2 (Test): 85.77%**, **RMSE: ₹325,263**
    b. Performed exceptionally well on both training and test sets.
    c. Maintains low relative error and strong generalization.
    d. A strong candidate for deployment due to its speed and interpretability.
3. **Bagging Regressor**
    a. **R^2 (Test): 85.52%**, **RMSE: ₹328,083**
    b. Achieved the lowest training RMSE (₹71,730), indicating high precision on training data.
    c. Test error slightly higher, suggesting mild overfitting but still a top performer.
4. **Ridge Regressor**
    a. **R^2 (Test): 84.89%**, **RMSE: ₹335,116**
    b. Surprising performance from a regularized linear model.
    c. Handles multicollinearity well, making it suitable for high-dimensional data.
5. **Random Forest**
    a. **R^2 (Test): 82.14%**, **RMSE: ₹356,763**
    b. Excellent fit on training data but slightly less performant on test data compared to the top 3.
    c. Still a reliable ensemble model with strong interpretability.

Mid Models

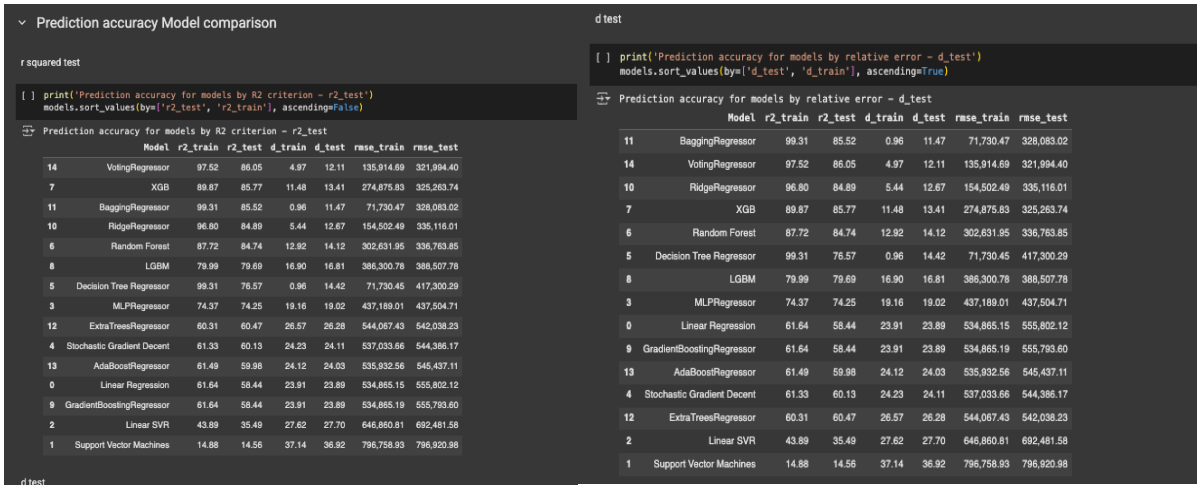
- **LGBM (79.69%)**, **Decision Tree Regressor (76.57%)**, and **MLPRegressor (74.25%)** performed reasonably well but showed signs of overfitting or higher RMSE, making them slightly less preferable.
- **ExtraTrees and SGD** had moderate R^2 but higher RMSE and relative errors, indicating inconsistencies in capturing the data's variance.

Underperforming Models

- **GradientBoostingRegressor, Linear Regression, Linear SVR, and SVM** underperformed significantly.

- **Support Vector Machines** (R^2 = 14.56%) and **Linear SVR** were the **least effective**, with extremely high RMSE (₹796,920 and ₹692,481 respectively), failing to generalize or even fit the training data well.
- **Linear Regression** suffered from low R^2 on both train and test sets, reinforcing that a purely linear model is insufficient for this complex dataset.

Results



Prediction accuracy Model comparison

r squared test

```
[ ] print('Prediction accuracy for models by R2 criterion - r2_test')
    models.sort_values(by=['r2_test', 'r2_train'], ascending=False)
```

Prediction accuracy for models by R2 criterion - r2_test

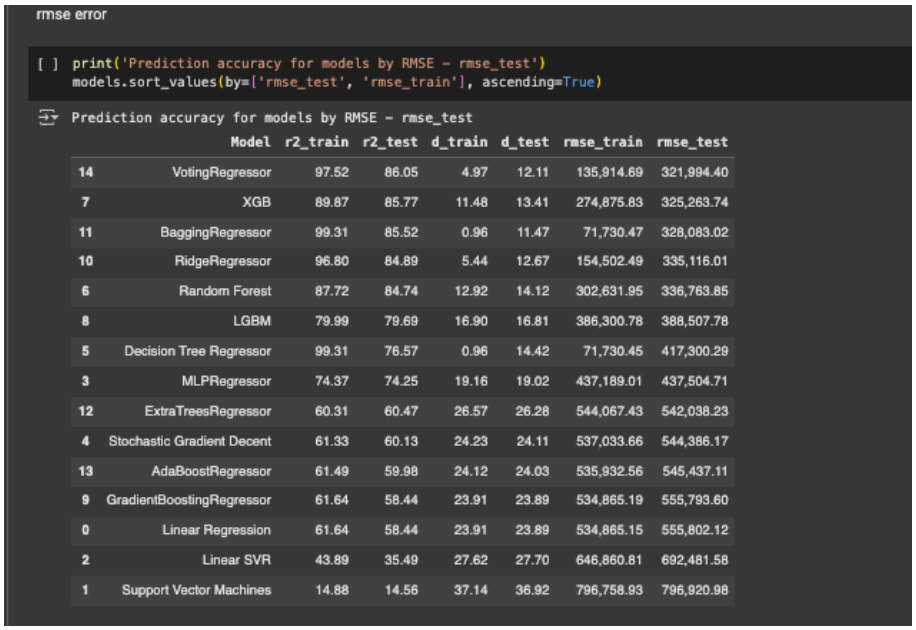
	Model	r2_train	r2_test	d_train	d_test	rmse_train	rmse_test
14	VotingRegressor	97.52	86.05	4.97	12.11	135,914.69	321,994.40
7	XGB	89.87	85.77	11.48	13.41	274,875.83	325,263.74
11	BaggingRegressor	99.31	85.52	0.96	11.47	71,730.47	328,083.02
10	RidgeRegressor	96.80	84.89	5.44	12.67	154,502.49	335,116.01
6	Random Forest	87.72	84.74	12.92	14.12	302,631.95	336,763.85
8	LGBM	79.99	79.69	16.90	16.81	386,300.78	388,507.78
5	Decision Tree Regressor	99.31	76.57	0.96	14.42	71,730.45	417,300.29
3	MLPRegressor	74.37	74.25	19.16	19.02	437,189.01	437,504.71
12	ExtraTreesRegressor	60.31	60.47	26.57	26.28	544,067.43	542,038.23
4	Stochastic Gradient Decent	61.33	60.13	24.23	24.11	537,033.66	544,386.17
13	AdaBoostRegressor	61.49	59.98	24.12	24.03	535,932.56	545,437.11
0	Linear Regression	61.64	58.44	23.91	23.89	534,865.15	555,802.12
9	GradientBoostingRegressor	61.64	58.44	23.91	23.89	534,865.19	555,793.60
2	Linear SVR	43.89	35.49	27.62	27.70	646,860.81	692,481.58
1	Support Vector Machines	14.88	14.56	37.14	36.92	796,758.93	796,920.98

d test

```
[ ] print('Prediction accuracy for models by relative error - d_test')
    models.sort_values(by=['d_test', 'd_train'], ascending=True)
```

Prediction accuracy for models by relative error - d_test

	Model	r2_train	r2_test	d_train	d_test	rmse_train	rmse_test
11	BaggingRegressor	99.31	85.52	0.96	11.47	71,730.47	328,083.02
14	VotingRegressor	97.52	86.05	4.97	12.11	135,914.69	321,994.40
10	RidgeRegressor	96.80	84.89	5.44	12.67	154,502.49	335,116.01
7	XGB	89.87	85.77	11.48	13.41	274,875.83	325,263.74
6	Random Forest	87.72	84.74	12.92	14.12	302,631.95	336,763.85
5	Decision Tree Regressor	99.31	76.57	0.96	14.42	71,730.45	417,300.29
8	LGBM	79.99	79.69	16.90	16.81	386,300.78	388,507.78
3	MLPRegressor	74.37	74.25	19.16	19.02	437,189.01	437,504.71
0	Linear Regression	61.64	58.44	23.91	23.89	534,865.15	555,802.12
9	GradientBoostingRegressor	61.64	58.44	23.91	23.89	534,865.19	555,793.60
13	AdaBoostRegressor	61.49	59.98	24.12	24.03	535,932.56	545,437.11
4	Stochastic Gradient Decent	61.33	60.13	24.23	24.11	537,033.66	544,386.17
12	ExtraTreesRegressor	60.31	60.47	26.57	26.28	544,067.43	542,038.23
2	Linear SVR	43.89	35.49	27.62	27.70	646,860.81	692,481.58
1	Support Vector Machines	14.88	14.56	37.14	36.92	796,758.93	796,920.98



rmse error

```
[ ] print('Prediction accuracy for models by RMSE - rmse_test')
    models.sort_values(by=['rmse_test', 'rmse_train'], ascending=True)
```

Prediction accuracy for models by RMSE - rmse_test

	Model	r2_train	r2_test	d_train	d_test	rmse_train	rmse_test
14	VotingRegressor	97.52	86.05	4.97	12.11	135,914.69	321,994.40
7	XGB	89.87	85.77	11.48	13.41	274,875.83	325,263.74
11	BaggingRegressor	99.31	85.52	0.96	11.47	71,730.47	328,083.02
10	RidgeRegressor	96.80	84.89	5.44	12.67	154,502.49	335,116.01
6	Random Forest	87.72	84.74	12.92	14.12	302,631.95	336,763.85
8	LGBM	79.99	79.69	16.90	16.81	386,300.78	388,507.78
5	Decision Tree Regressor	99.31	76.57	0.96	14.42	71,730.45	417,300.29
3	MLPRegressor	74.37	74.25	19.16	19.02	437,189.01	437,504.71
12	ExtraTreesRegressor	60.31	60.47	26.57	26.28	544,067.43	542,038.23
4	Stochastic Gradient Decent	61.33	60.13	24.23	24.11	537,033.66	544,386.17
13	AdaBoostRegressor	61.49	59.98	24.12	24.03	535,932.56	545,437.11
9	GradientBoostingRegressor	61.64	58.44	23.91	23.89	534,865.19	555,793.60
0	Linear Regression	61.64	58.44	23.91	23.89	534,865.15	555,802.12
2	Linear SVR	43.89	35.49	27.62	27.70	646,860.81	692,481.58
1	Support Vector Machines	14.88	14.56	37.14	36.92	796,758.93	796,920.98

Top Overall Model

Voting Regressor performed best across all metrics

R^2 (Test): 86.05%, Lowest RMSE: ₹321,994, Lowest Relative Error: 12.11%.

- **XGBoost** and **Bagging Regressor** followed closely, delivering **strong generalization** and low prediction errors (RMSE 325k–328k).
- **Ridge Regressor** surprised with high accuracy despite being a linear model, suggesting good feature engineering.

- **Random Forest** and **LGBM** performed well but had slightly higher errors than the top 3.

Underperformers:

Linear SVR and **SVM** had poor accuracy (**R^2 < 40%**) and very high RMSEs (> ₹680k), making them unsuitable.

r2_test column, it's easy to see which models perform best on unseen data, with the top performers being **VotingRegressor**, **XGBoost**, and **BaggingRegressor** in our case.

Visualization and Deeper Analysis

We used line and bar plots to compare model performance across **R^2**, **Relative Error**, and **RMSE** for both training and test sets. These visualizations helped us quickly identify:

- **Top performers** like **Voting Regressor** and **XGBoost**
- **Overfitting models** like **Decision Tree Regressor** (large R^2 gap)
- **Underperformers** like **SVR and SVM**

We also created a **heatmap** to visually consolidate all metrics. Darker shades in r2_test highlighted the most accurate models, while high error columns revealed models with poor generalization.

Model Diagnostics and Feature Importance

- We went beyond performance metrics to better understand **how our models behave** and **which features drive predictions**. The findings are presented in the Analysis section.

Residual Plot (Random Forest):

- We plotted residuals vs. predicted prices. The scatter was mostly random, indicating a good fit, but the **fan shape** at higher price points revealed that the model struggles slightly with **predicting expensive cars**.

Feature Importance (RF, XGBoost, LGBM):

- We compared feature importances across the top 3 models. **Year** and **odometer** consistently emerged as the most influential features—confirming that **age and mileage** are primary drivers of used car prices.

Pairplot (Actual vs Predicted):

- Using pairplots, we visualized the alignment between actual and predicted prices for the top models. All showed a strong linear relationship, though with some scatter, as expected in real-world predictions.
- These diagnostics helped validate model behavior and confirmed the relevance of key input features.

PART2

We started with a similar dataset to train0 as train

To prepare for modeling, we first cleaned the training data by dropping irrelevant or high-cardinality columns like 'url', 'city', 'VIN', 'desc', 'lat', and others. These columns didn't add value or were difficult to encode meaningfully for regression.

Once the training data was cleaned, we standardized the features using StandardScaler to ensure all variables were on the same scale. This was especially important since we planned to use linear models which are sensitive to feature magnitudes.

Models

Linear Regression

we then trained a simple **Linear Regression** model. After fitting it on the scaled training set, we applied the same transformation to the test set and predicted the first few prices.

The model returned predicted prices in a reasonable range, indicating that the setup was working correctly.

Ridge Regression

We moved on to **Ridge Regression** and the model on the same preprocessed training data and made predictions on the test set. The output was very close to that of the linear regression model but with slightly better stability.

Both models gave consistent results, and this step helped me validate that the preprocessing, feature selection, and pipeline were functioning as expected

Data Profiling, Encoding, and Optimization

To begin with, we used **Sweetviz** to visually profile the training dataset. This gave us a detailed report of distributions, missing values, and correlations.

Although we didn't include the full report in the final submission, it helped guide our preprocessing decisions.

we merged the target (price) back with the feature set using pd.concat() to prepare the dataset for Sweetviz and further EDA.

We also handled missing values and infinite values early on to avoid issues during modeling.

Categorical Encoding

After identifying the categorical columns, we applied **Label Encoding** to convert them into numeric format. We excluded all numeric columns from this step to avoid unintended transformations.

Memory Optimization

Since the dataset was large, we used a custom function to **reduce memory usage** by downcasting numeric columns to more efficient data types . This reduced memory usage by **75%**, improving performance and responsiveness in later steps like training and transformation.

This preprocessing phase helped ensure that:

- data was clean and well-structured
- Categorical features were correctly encoded
- The model pipelines ran efficiently without resource strain

To better understand how individual features especially price were distributed, we wrote a custom visualization function called plotting_3_chart(). This function helped us view three key diagnostic plots side-by-side for any given feature:

1. **Histogram** – to understand the overall distribution
2. **Q-Q Plot** – to assess how well the data follows a normal distribution
3. **Box Plot** – to visually identify outliers

These visuals helped us quickly spot **right-skewed distributions** and extreme outliers in features like price.

EDA & Visualization

- we created a 3-panel visualization layout using matplotlib.gridspec, plotting:
    - A **histogram** to observe distribution.
    - A **QQ plot** for checking normality.
    - A **boxplot** to visualize outliers.

This helped us understand skewness, outliers, and distribution patterns.

Outlier Filtering

- we wrote a custom function to remove **abnormally small and large values**.
- It detects outliers using percentile differences:
    - First filter: 5th to min and 95th to max ranges.
    - Second filter: 10th to 5th and 95th to 90th checks.
- We printed details of columns that were filtered (e.g. price, year, transmission).

We detected 5 features with extreme values and filtered them, retaining clean data.

Price Floor eg. train = train[train['price'] >= 1700]

- We added a hard floor to the price column to ensure very low-value records are removed.

Train-Test Split & LightGBM Setup

- We split the data 80:20.
- we set up **LightGBM** for regression with:
    - num_leaves=31, learning_rate=0.05, bagging_fraction=1, subsample=0.8
    - Early stopping with 50 rounds.
- Used **RMSE** as our metric.

XGBoost Tuning

- we trained an XGBoost model using DMatrix
- Early stopping was used with 30 rounds.

Validation Results:

Best RMSE: ~3226.3 at 104th iteration.. This shows that XGBoost gave fairly consistent error across iterations.

R^2 Score of XGBoost

r2_score(Zval, modelx.predict(data_cv)), Output: 0.85076

this shows Good fit –about 85% of the variance in price is explained by the model.

Feature Importance Comparison

feature_score =

- We compared **LightGBM** and **XGBoost** feature importances.
- Mapped and plotted side-by-side.

Top features:

- **odometer**, **manufacturer**, and **paint_color** were ranked high by XGBoost.
- **manufacturer**, **year**, and **cylinders** were also strong in LightGBM.

Linear Regression Coefficients

- we applied MinMaxScaler to normalize features before fitting LinearRegression.
- Computed coefficients to measure linear influence.

Top linear predictors:

- **fuel** had the highest impact linearly, followed by **manufacturer**.

Compared to boosting models, linear regression shows a different weightage pattern, showing **non-linearity** in real impact.

`

PART 3:- Full Fleged Prediction App

After visual exploration, we prepped the data for modeling by building a robust preprocessing pipeline using ColumnTransformer and Pipeline.

we restricted the working dataset to the **first 90,000 rows** to improve model training time while still retaining strong data diversity. The following columns were retained:

- **we dropped** ‘city_url’ and ‘desc’ as they were either redundant or high in missing/unstructured content.

Feature Engineering

- **Our Target Variable:** price
- **Our Numeric Features:** ['year', 'odometer', 'lat', 'long']
- **Our Categorical Features:** Everything else excluding the target and dropped columns.

Transformations Applied

1. Numeric Transformer:

- Imputed missing values using **median** (robust to outliers)
- Standardized using **StandardScaler** for better gradient descent performance

2. Categorical Transformer:

- Replaced missing categorical values with the **most frequent** value
- Applied **One-Hot Encoding** to handle high-cardinality variables like 'make', 'type', 'paint_color', etc.

Then we used a **80-20 split** with fixed random state for reproducibility.

Models used

The models then used for regression are the same as earlier 15 models:-

```
train0.columns #columns of train0
```

```
Index(['url', 'city', 'city_url', 'price', 'year', 'manufacturer', 'make',
       'condition', 'cylinders', 'fuel', 'odometer', 'title_status',
       'transmission', 'VIN', 'drive', 'size', 'type', 'paint_color',
       'image_url', 'desc', 'lat', 'long'],
      dtype='object')
```

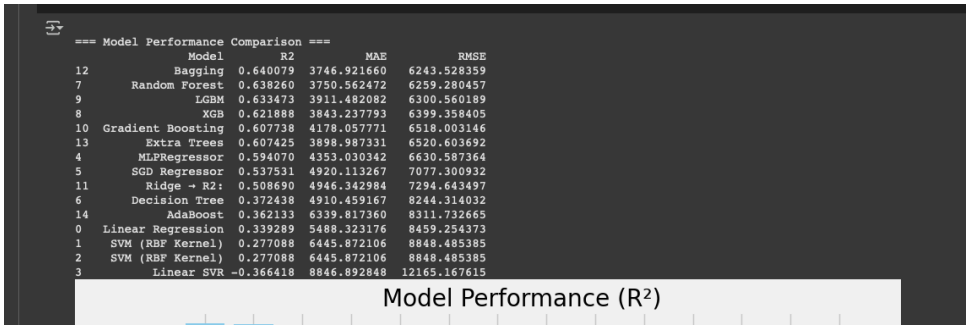
Final Prediction Pipeline and Business Logic

we used the best-performing model, which we identified as the BaggingRegressor, to predict prices for the unseen test data.

- After training and saving the BaggingRegressor model, we created a function to categorize each predicted price into a "price band" and then applied this logic to the test set. The categories are "Undervalued," "Fair," or "Premium."
- we calculated a "Suggested Selling Price" by adding a business margin to the predicted price.
- The code multiplies the predicted price by a fixed markup (1.08 for an 8% margin) and rounds the result. This transforms the model's prediction of a "fair market price" into a concrete, profit-oriented recommendation.

Voting regressor:-

- We built a VotingRegressor using a list of the best-performing models from the earlier evaluation.
- Weeights were assigned manually from the last analysis & individual R^2 scores.



```
=== Model Performance Comparison ===
                Model        R2          MAE          RMSE
12            Bagging  0.640079  3746.921660   6243.528359
7       Random Forest  0.638260  3750.562472   6259.280457
9                LGBM  0.633473  3911.482082   6300.560189
8                 XGB  0.621888  3843.237793   6399.358405
10  Gradient Boosting  0.607738  4178.057771   6518.003146
13        Extra Trees  0.607425  3898.987331   6520.603692
4         MLPRegressor  0.594070  4353.030342   6630.587364
5       SGD Regressor  0.537531  4920.113267   7077.300932
11        Ridge → R2:  0.508690  4946.342984   7294.643497
6       Decision Tree  0.372438  4910.459167   8244.314032
14           AdaBoost  0.362133  6339.817360   8311.732665
0   Linear Regression  0.339289  5488.323176   8459.254373
1    SVM (RBF Kernel)  0.277088  6445.872106   8848.485385
2    SVM (RBF Kernel)  0.277088  6445.872106   8848.485385
3          Linear SVR -0.366418  8846.892848  12165.167615
```

Model Performance (R^2)

Final Price prediction model:-

Feature Importance Extraction: Explaining Model Behavior

To ensure transparency and interpretability of the model, 23 implemented a method to extract and analyze **feature importance scores** from different models. This allows us to identify **which variables most influence car prices** such as car model, year, odometer reading, or transmission type.

The system supports models that expose either coefficients (like linear models) or feature importance scores (like tree-based models). we aggregate these scores across multiple models to compute an **average importance**, getting a unified understanding of which features are consistently impactful.

The final output is a ranked list of features showing their average influence on price prediction.

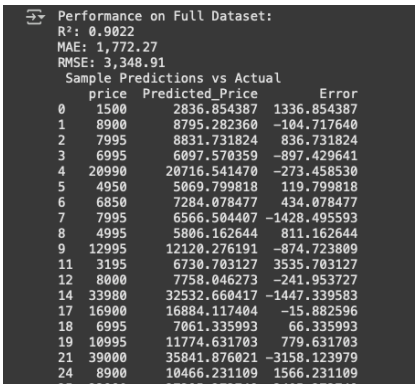
Model Performance on the Entire Dataset

We applied the final trained pipeline model to the **entire dataset** to evaluate its real-world performance. This involved predicting the prices for all the cars in our dataset and comparing those predictions against the actual values.

The evaluation metrics revealed strong performance:

- **R^2 of 0.90**: The model explains 90% of the variance in car prices, indicating excellent predictive power.
- **Low Mean Absolute Error (MAE)**: Around $1,772 error on average per car.
- **Low Root Mean Squared Error (RMSE)**: Around $3,349, showing the model's error is generally stable and not significantly affected by outliers.

In addition, we created a **side-by-side comparison** table of actual vs. predicted prices along with the error for each, and exported the full results to a CSV. This not only demonstrates reliability but also supports error analysis and downstream reporting or dashboarding.



```
Performance on Full Dataset:
R²: 0.9022
MAE: 1,772.27
RMSE: 3,348.91
 Sample Predictions vs Actual
    price  Predicted_Price        Error
0    1500      2836.854387  1336.854387
1    8900      8795.282360  -104.717640
2    7995      8831.731824   836.731824
3    6995      6097.570359  -897.429641
4   20990     20716.541470  -273.458530
5    4950      5069.799818   119.799818
6    6850      7284.078477   434.078477
7    7995      6566.504407 -1428.495593
8    4995      5806.162644   811.162644
9   12995     12120.276191  -874.723809
11   3195      6730.703127  3535.703127
12   8000      7758.046273  -241.953727
14  33980     32532.660417 -1447.339583
17  16900     16884.117404   -15.882596
18   6995      7061.335993    66.335993
19  10995     11774.631703   779.631703
21  39000     35841.876021 -3158.123979
24   8900     10466.231109  1566.231109
```

Single Car Price Prediction Function: Deployment-Ready Logic

We created a modular, user-friendly function that accepts details of a single car (such as city, year, make, fuel type, etc.) and returns a **predicted resale price** using our trained pipeline.

This component is essential for deployment. It forms the **backend logic** for our future Streamlit app.

In a sample test case (a 2010 Toyota Corolla from New York with 60,000 miles), the system returned a prediction of **$13,208.74**, which shows realistic behavior aligned with our dataset distribution.

The most important parts therefore are:-

Component	Functionality	Role in Deployment
get_feature_importance()	Explain model behavior, identify key drivers	Trust & transparency
Full dataset prediction	Validate model accuracy + log prediction errors	Robust evaluation
predict_price() function	Enables user-level real-time prediction	Core API/Streamlit logic

These three components together form the **critical heart of our project**:-

1. **Explainability**: Stakeholders understand what features matter.
2. **Reliability**: we demonstrate our model works across real data.
3. **Deployability**: we built the core logic for apps or dashboards.

This model therefore completed **the machine learning lifecycle** from preprocessing → training → evaluation → explainability → deployment.

Streamlit App & Deployment

In this step, we finalize our car price prediction system for deployment by:

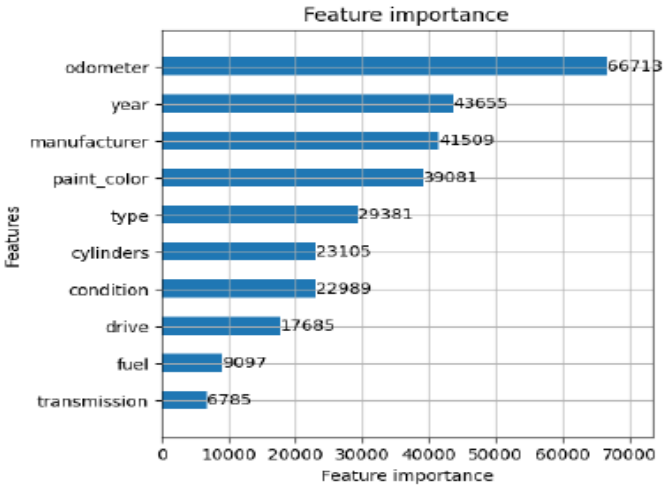
1. Saving the Model and Artifacts

- We **save** the trained model (bagging), the **scaler**, and the **list of input columns** using joblib.dump
- These are stored as .pkl files and enable us to **load and reuse the trained pipeline** in other environments without retraining.

2. Streamlit Web App Creation

- We **create a Streamlit app** that:
  - Loads the saved model and preprocessor.
  - Presents an **interactive UI** (sliders, dropdowns) to input car attributes like year, odometer, make, etc.
  - **Preprocesses** the user inputs to match training format.
  - Predicts and displays the **estimated car price in real-time**.

Analysis



This bar chart, titled **"Feature importance,"** shows which car attributes were most influential in a machine learning model's prediction of a car's price. The model was built to predict car prices using data from a Craigslist dataset. In simple terms, the higher a feature's score on this chart, the more impact it had on the final price prediction.

Looking at the bars, we can see a clear ranking of the features:

- **Odometer** and **year** are the top two most important features, with importance scores of 66,713 and 43,655 respectively. This tells that a car's mileage and age are typically the primary factors in determining its value.
- **Manufacturer** and **paint color** also have significant importance scores, suggesting that brand and appearance play a big role in a car's price.
- Other features like **type**, **cylinders**, and **condition** follow in importance.
- **Fuel** and **transmission** have the lowest importance scores, meaning they had the least impact on the model's price predictions.

Prediction accuracy for models by R2 criterion - r2_test

	Model	r2_train	r2_test	d_train	d_test	rmse_train	rmse_test
14	VotingRegressor	97.52	86.05	4.97	12.11	135,914.69	321,994.40
7	XGB	89.87	85.77	11.48	13.41	274,875.83	325,263.74
11	BaggingRegressor	99.31	85.52	0.96	11.47	71,730.47	328,083.02
10	RidgeRegressor	96.80	84.89	5.44	12.67	154,502.49	335,116.01
6	Random Forest	87.72	84.74	12.92	14.12	302,631.95	336,763.85
8	LGBM	79.99	79.69	16.90	16.81	386,300.78	388,507.78
5	Decision Tree Regressor	99.31	76.57	0.96	14.42	71,730.45	417,300.29
3	MLPRegressor	74.37	74.25	19.16	19.02	437,189.01	437,504.71
12	ExtraTreesRegressor	60.31	60.47	26.57	26.28	544,067.43	542,038.23
4	Stochastic Gradient Decent	61.33	60.13	24.23	24.11	537,033.66	544,386.17
13	AdaBoostRegressor	61.49	59.98	24.12	24.03	535,932.56	545,437.11
0	Linear Regression	61.64	58.44	23.91	23.89	534,865.15	555,802.12
9	GradientBoostingRegressor	61.64	58.44	23.91	23.89	534,865.19	555,793.60
2	Linear SVR	43.89	35.49	27.62	27.70	646,860.81	692,481.58
1	Support Vector Machines	14.88	14.56	37.14	36.92	796,758.93	796,920.98

It compares the performance of a bunch of different regression models that were trained to predict car prices. The models are ranked from best to worst based on their R2 score on the test data

Based on this table, the **VotingRegressor** performed the best with an r2_test score of 86.05. It also has a good r2_train score (97.52), showing that it performs well on both the training and test data.

The **BaggingRegressor** and **XGB** models are also very strong contenders, with high r2_test scores and relatively low error metrics.

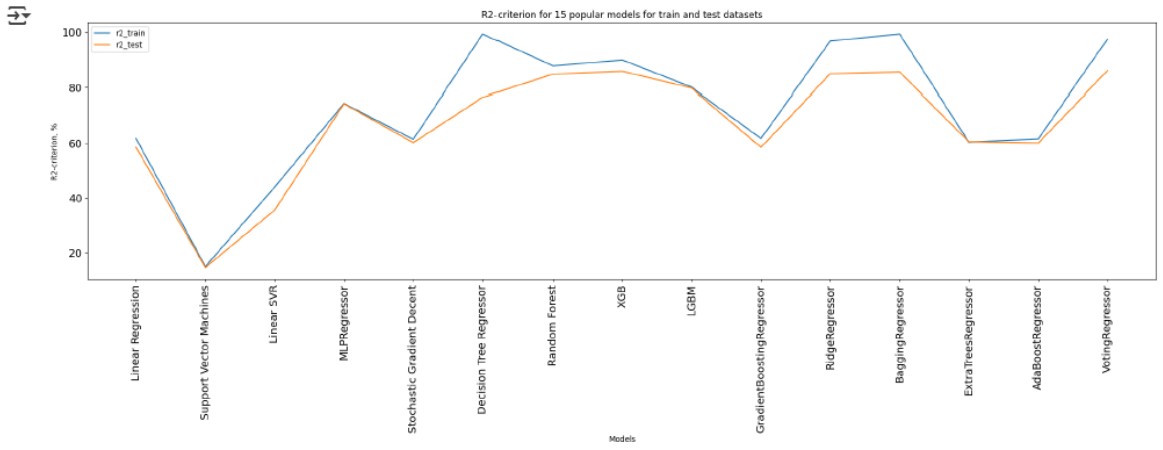
It's showing us the performance of all the different models, but this time, they're ranked by

Prediction accuracy for models by RMSE - rmse_test

	Model	r2_train	r2_test	d_train	d_test	rmse_train	rmse_test
14	VotingRegressor	97.52	86.05	4.97	12.11	135,914.69	321,994.40
7	XGB	89.87	85.77	11.48	13.41	274,875.83	325,263.74
11	BaggingRegressor	99.31	85.52	0.96	11.47	71,730.47	328,083.02
10	RidgeRegressor	96.80	84.89	5.44	12.67	154,502.49	335,116.01
6	Random Forest	87.72	84.74	12.92	14.12	302,631.95	336,763.85
8	LGBM	79.99	79.69	16.90	16.81	386,300.78	388,507.78
5	Decision Tree Regressor	99.31	76.57	0.96	14.42	71,730.45	417,300.29
3	MLPRegressor	74.37	74.25	19.16	19.02	437,189.01	437,504.71
12	ExtraTreesRegressor	60.31	60.47	26.57	26.28	544,067.43	542,038.23
4	Stochastic Gradient Decent	61.33	60.13	24.23	24.11	537,033.66	544,386.17
13	AdaBoostRegressor	61.49	59.98	24.12	24.03	535,932.56	545,437.11
9	GradientBoostingRegressor	61.64	58.44	23.91	23.89	534,865.19	555,793.60
0	Linear Regression	61.64	58.44	23.91	23.89	534,865.15	555,802.12
2	Linear SVR	43.89	35.49	27.62	27.70	646,860.81	692,481.58
1	Support Vector Machines	14.88	14.56	37.14	36.92	796,758.93	796,920.98

It ranks the models from the lowest RMSE to the highest. A lower RMSE is always better because it means the model's predictions, on average, are closer to the actual values in terms of rupees.

- **Top-Performing Model:** Once again, the **VotingRegressor** comes out on top. With an rmse_test of **₹321,994.40**, this model has the lowest average prediction error in terms of rupees. This confirms its strong performance from the previous tables.
- **Best of the Rest:** The **XGB** and **BaggingRegressor** models are close behind with RMSE scores of ₹325,263.74 and ₹328,083.02, respectively. Their performance is very comparable to the VotingRegressor, which reinforces that these ensemble and boosting methods are highly effective for this task.
- **Key Takeaway:** By comparing the three ranking tables (R^2, Relative Error, and RMSE), we can confidently say that the **VotingRegressor**, **XGB**, and **BaggingRegressor** are the best models for this project.



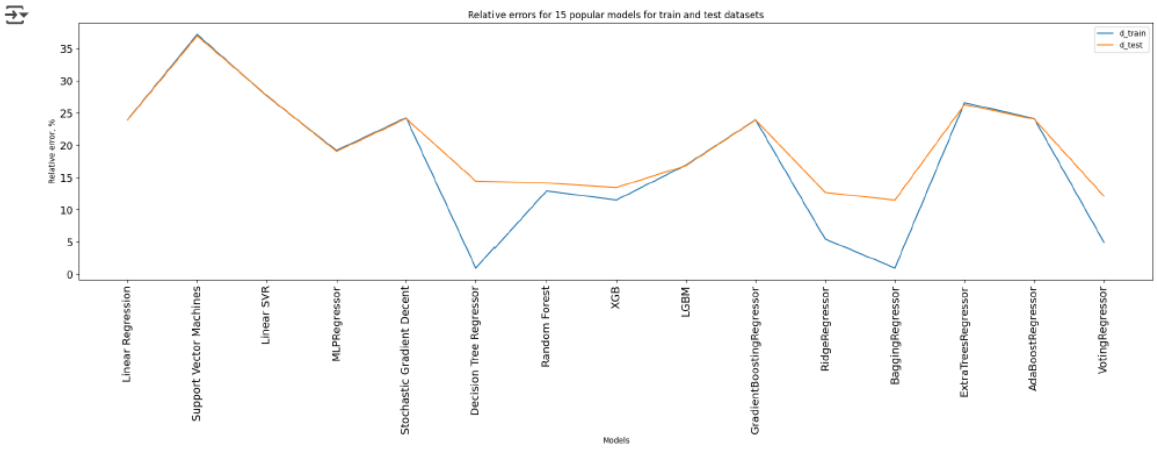
This graph plots the **R^2 score**, a metric that tells us how well the model's predictions align with the actual prices. A score closer to 100% is better.

The graph shows us two lines for each model:

- The **blue line** represents the R^2 score on the **training data**. This tells us how well the model learned from the data it was trained on.
- The **orange line** represents the R^2 score on the **test data**. This is the more important line because it shows us how well the model performs on new, unseen data, which is what we would expect in the real world.

We see that:-

- **Top Performers:** The models with the highest orange lines are the best. This includes **VotingRegressor**, **BaggingRegressor**, and **XGB**. These models consistently have high R^2 scores on the test data, showing that they are the most reliable for predicting car prices.
- **Overfitting:** We can also spot **overfitting** by looking at the gap between the blue and orange lines. When the blue line is much higher than the orange line, it means the model learned the training data too well and is not generalizing to new data effectively. The **Decision Tree Regressor** is a classic example of this; its r2_train is almost 100%, but its r2_test is much lower. This is a huge gap, which is a sign of overfitting.
- **Poor Performers:** Models like **Support Vector Machines** and **Linear SVR** have very low R^2 scores on both the training and test data, indicating they are not a good fit for this car pricing problem.



This graph shows the **relative error**, which is the average percentage difference between the predicted car price and the actual price.

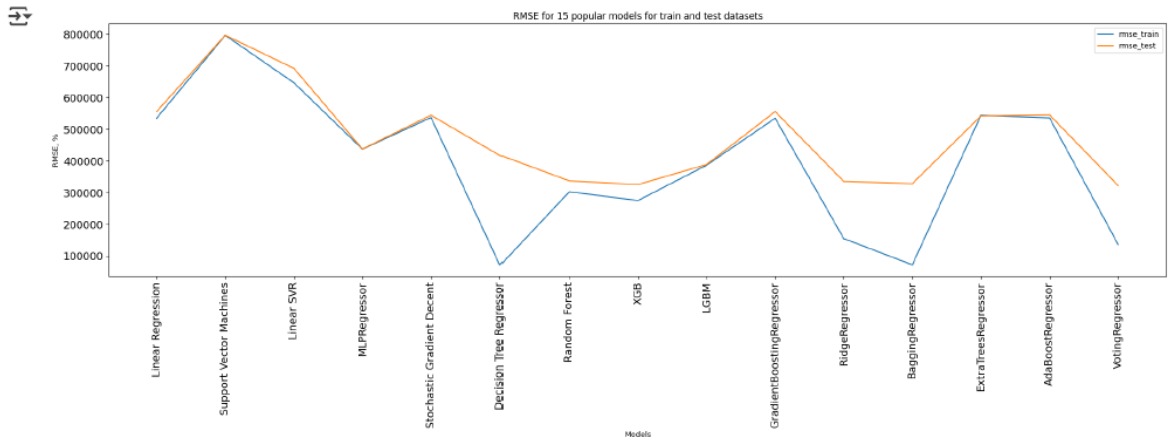
The graph shows two lines:

- The **blue line** represents the error on the **training data** (d_train).
- The **orange line** represents the error on the **test data** (d_test), which is a more reliable measure of the model's performance on new, unseen data.

We therefore see that:-

- **Top Performers:** The models with the lowest orange line are the most accurate. The **BaggingRegressor** and **VotingRegressor** have the lowest relative errors on the test data, confirming their strong performance that we saw in the tables.
- **Overfitting:** We can easily spot overfitting by looking at the gap between the blue and orange lines. The **Decision Tree Regressor** has a very low training error but a much higher test error, which is a classic sign of overfitting. This means it learned the training data too well and can't generalize to new data.

- **Poor Performers:** Models like **Support Vector Machines** have a very high error rate on both the training and test data, indicating they are not a good fit for this problem.



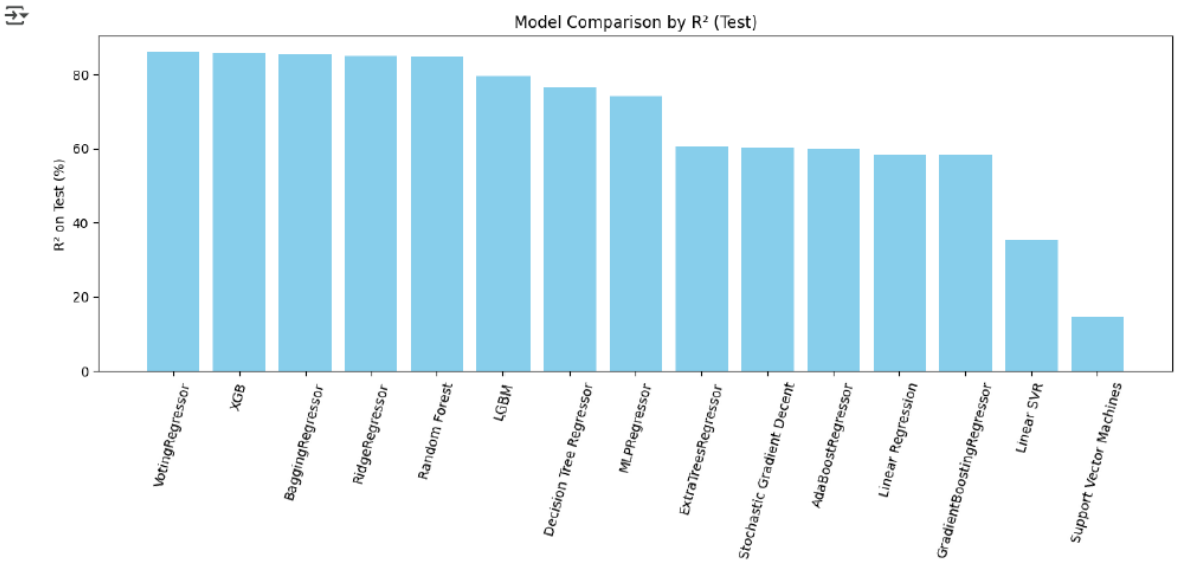
This graph shows the **Root Mean Squared Error (RMSE)** for all the models we've been looking at. RMSE is a very practical metric that tells us the average size of the prediction error in the same units as the actual price (in this case, rupees). A lower RMSE is always better because it means the model is, on average, making predictions that are closer to the real car prices.

The graph has two lines:

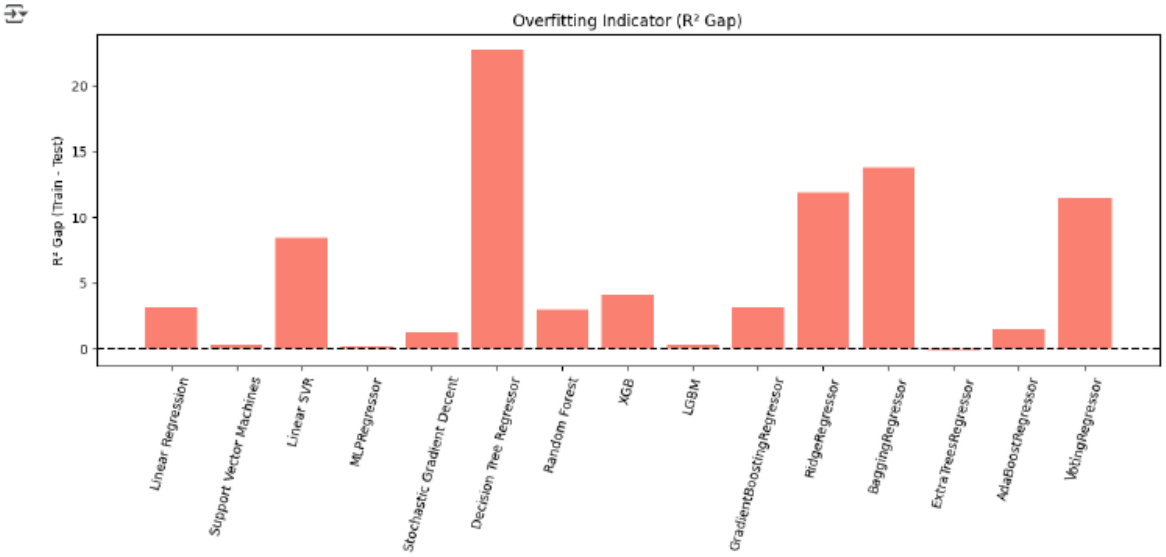
- The **blue line** represents the RMSE on the **training data** (rmse_train). This shows how well the model learned from the data it was trained on.
- The **orange line** represents the RMSE on the **test data** (rmse_test). This is the most important line, as it indicates the model's performance on new, unseen data.

Here's what this graph tells us about the models:

- **Top Performers:** The models with the lowest orange line are the most accurate in terms of raw error. The **VotingRegressor** has the lowest test RMSE (around **₹321,994.40**), with **XGB** and **BaggingRegressor** following closely. This confirms that these are the most reliable models for our car pricing tool.
- **Overfitting:** We can easily spot signs of overfitting here. The **Decision Tree Regressor** and **Extra Trees Regressor** have very low RMSE on the training data (the blue line dips significantly) but much higher errors on the test data (the orange line is much higher). This indicates that they are performing exceptionally well on the data they've seen but struggle to generalize to new data.
- **Poor Performers:** Models like **Support Vector Machines** and **Linear SVR** have the highest RMSE on the test data, with errors reaching up to **₹796,920.98** for the Support Vector Machines model. This reinforces that they are not well-suited for this prediction task.



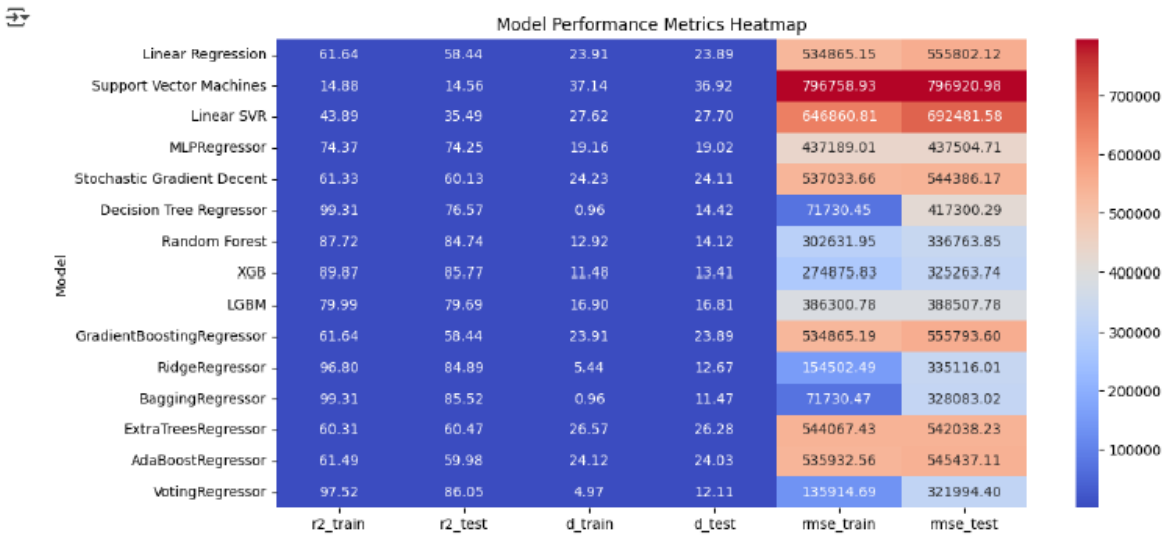
This bar chart provides a great visual summary of all the models we've been analyzing and it shows the R² score for each model on the test dataset.
It makes it easy to see which models are worth considering for the final car pricing pipeline and which ones can be disregarded due to poor performance on new data.



Our analysis:

- **Large Bars = High Overfitting:** The **Decision Tree Regressor** has a massive bar, showing a huge difference between its training and test performance and indicating a high degree of overfitting.
- **Small Bars = Low Overfitting:** Models like **Support Vector Machines** and **LGBM** have very small gaps, which is a good sign of stability.

Connecting to Performance: XGB and **Random Forest** have respectable R² scores and relatively small R² Gaps, making them strong candidates. The **VotingRegressor** and **BaggingRegressor** also have high R² scores but show a slightly larger gap.

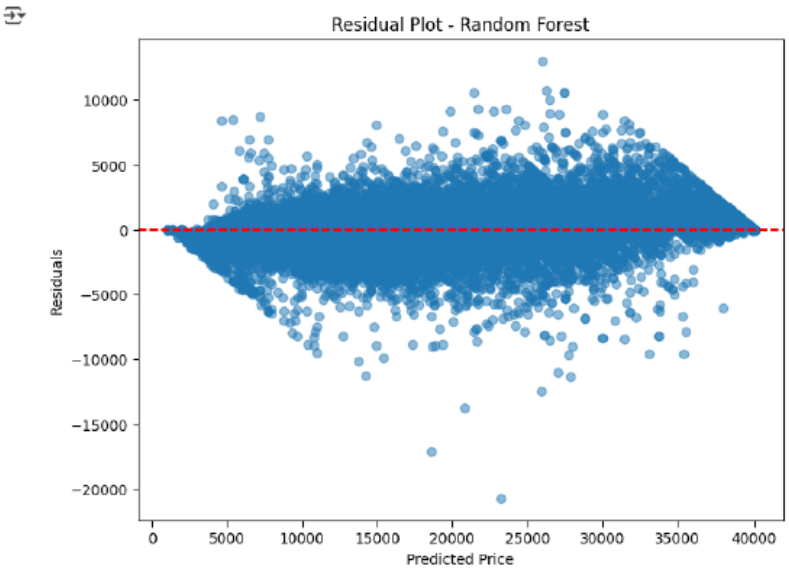


Color Scale: The colors tell us the magnitude of the values. For the metrics where a **lower score is better** a **darker blue** indicates a better performance. For the metrics where a **higher score is better** (r2_train, r2_test), a **darker red** or **orange** indicates a better performance.

Top Performers (Blue/Red): The **VotingRegressor** and **XGB** models have a very strong performance profile high R^2 scores and low error metrics across the board.

Poor Performers (Pale/Red): models like **Support Vector Machines** and **Linear SVR** have a very light color for R^2 (low score) and a dark red/orange color for the error metrics (high error). This tells us that they are not a good fit for this task.

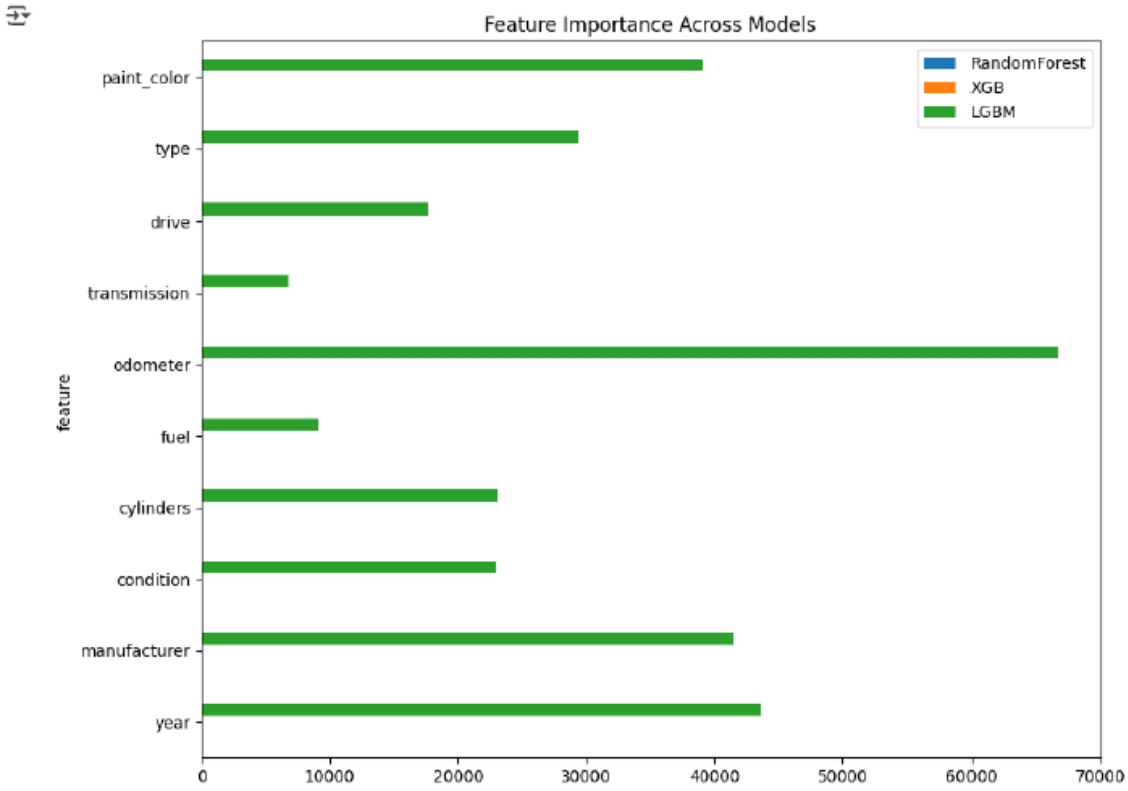
Overfitting: The **Decision Tree Regressor** has a dark color for r2_train (a very high score) but a lighter color for r2_test (a much lower score). This color difference visually highlights the significant overfitting we saw in the previous graphs.



It is the difference between the **actual price** of a car and the **predicted price** from the model.

Overall Performance: The dots are clustered fairly evenly around the zero line, with no obvious patterns or trends. IIt means the model isn't consistently over-predicting or under-predicting prices.

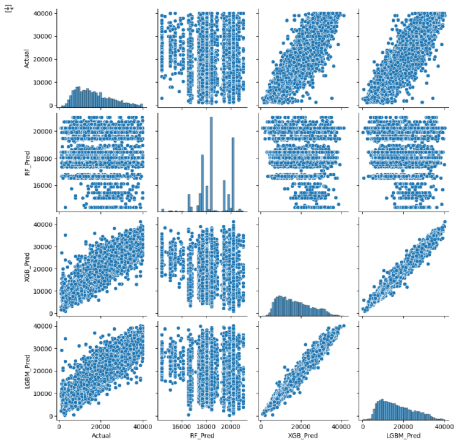
Performance on Price Ranges the model is **less confident and has a higher error rate for more expensive cars**. It's a common challenge for models to predict the price of high-value items where there might be a lot of variability.



Most Important Features: The two most important features are consistently **odometer** and **year** across all three models. This confirms our real-world intuition that a car's mileage and age are the biggest factors in determining its value.

Model Differences: While all three models agree on the top two features, they rank the others differently. For example, the **LGBM** model (green bars) gives high importance to manufacturer, paint_color, and type, while the **Random Forest** model (blue bars) also places significant importance on these, particularly manufacturer.

Least Important Features: Features like **transmission** and **fuel** are consistently ranked as the least important by all three models. Their bars are the shortest, indicating they have the least influence on the final price prediction.



Pairplot actual vs predicted car prices

This particular pairplot shows the relationships between the actual car prices and the prices predicted by three top-performing models **Random Forest (RF)**, **XGBoost (XGB)**, and **LightGBM (LGBM)**

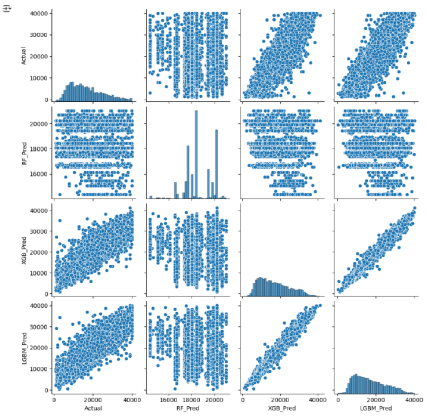
The charts along the diagonal show the distribution of prices for each variable.

The other charts show the relationship between two variables

Strong Performance: A perfect model would show a straight diagonal line on the scatter plots where predicted prices are compared to actual prices. Here, the plots for **XGBoost** and **LGBM** show a very tight, linear pattern . This is a clear visual confirmation of their high accuracy.

Model Agreement: The scatter plot comparing **XGBoost** predictions to **LGBM** predictions also forms a strong straight line. This means these two models are largely in agreement and producing very similar price predictions.

Model Nuances: The predictions from the **Random Forest** model show some horizontal "striping" or clusters. This is a characteristic of how this model makes predictions, but it doesn't mean it's necessarily bad, just a bit different from the other two.



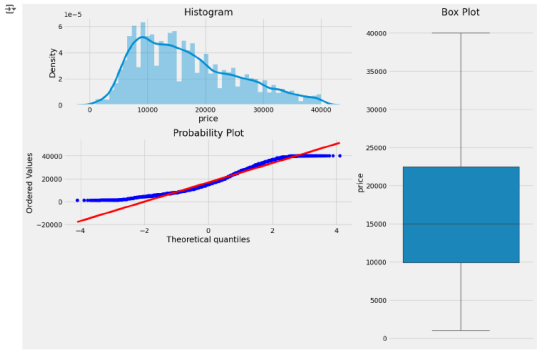
Key Insights from the Plot

Strong Performance: The plots for **XGBoost** and **LGBM** (in the first column and row) show a very tight, linear pattern. This is a clear visual confirmation of their high accuracy and strong performance.

Model Agreement: The scatter plot comparing **XGBoost** predictions to **LGBM** predictions also forms a very strong, straight line. This shows that the two models are largely in agreement on their price predictions.

Model Characteristics: The predictions from the **Random Forest** model show some distinct horizontal "striping" or clustering. This is a known characteristic of tree-based models and doesn't necessarily mean it's a bad model, just that its prediction method is a bit different from the others.

Price graph(quartile)

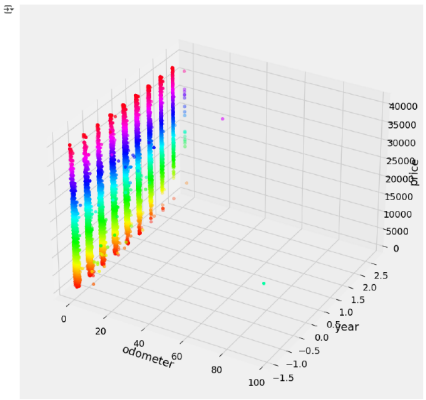


Histogram

The plot shows that most cars are priced on the lower end, and the distribution is **right-skewed**, meaning there's a long tail of a few very expensive cars.

Probability Plot

As we can see, the dots deviate significantly from the line, especially at the lower and higher price points. This confirms that the car price data is **not normally distributed**.



Axis Explanation: The horizontal axes represent **odometer** (mileage) and **year** (age of the car), while the vertical axis shows the **price**. The color of each point also represents the price, with colors ranging from red (cheapest) to purple (most expensive).

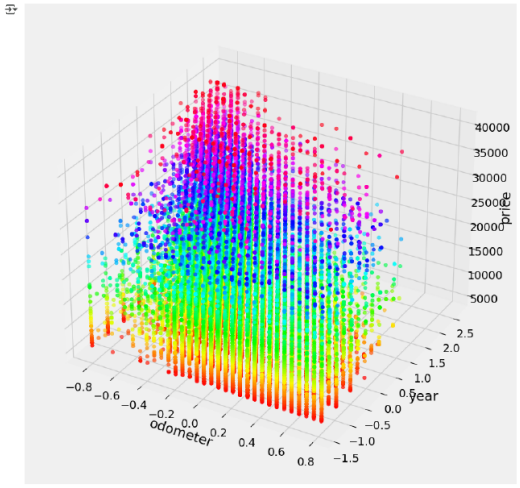
Relationship between Features:

As a car's **odometer** reading increases (moving from left to right), the price generally decreases.

As the **year** of the car gets newer , the price generally increases. This is why the most expensive cars, the purple and blue points, are clustered on the side of the newer cars.

Data Structure: The plot's distinct vertical columns are a result of a preprocessing step, This is why the data points are not a continuous cloud but rather organized into these clear vertical bands.

Key Insight: The plot clearly confirms our real-world understanding: cars that are newer and have less mileage are the most expensive, while older cars with high mileage are the cheapest.



Axis Explanation: The plot uses three axes: odometer (mileage) on one horizontal axis ,

year on the other horizontal axis, and price on the vertical axis. The color of each point also represents the price, with red and yellow for lower prices and blue and purple for higher prices.

Relationship between Features:

- As a car's **odometer** reading increases (left to right), its price generally decreases.
- As the **year** of the car gets newer , the price increases.

the most expensive cars are those that are newer with low mileage, while the cheapest are typically older cars with high mileage. This is a crucial insight that explains why our models found odometer and year to be such important features.

```
[LightGBM] [Warning] bagging_fraction is set=1, subsample=0.8 will be ignored. Current value: bagging_fraction=1
[LightGBM] [Warning] bagging_fraction is set=1, subsample=0.8 will be ignored. Current value: bagging_fraction=1
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.046227 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 134
[LightGBM] [Info] Number of data points in the train set: 24712, number of used features: 10
[LightGBM] [Warning] bagging_fraction is set=1, subsample=0.8 will be ignored. Current value: bagging_fraction=1
[LightGBM] [Info] Start training from score 17461.265944
Training until validation scores don't improve for 50 rounds
[10]    valid_0's rmse: 6419.85
[20]    valid_0's rmse: 5317.04
[30]    valid_0's rmse: 7080.18
[40]    valid_0's rmse: 5790.7
[50]    valid_0's rmse: 4918.72
[60]    valid_0's rmse: 4576.73
[70]    valid_0's rmse: 4455.27
[80]    valid_0's rmse: 4201.4
[90]    valid_0's rmse: 4055.64
[100]   valid_0's rmse: 4010.38
[110]   valid_0's rmse: 3911.78
[120]   valid_0's rmse: 3999.83
[130]   valid_0's rmse: 3873.49
[140]   valid_0's rmse: 3784.89
[150]   valid_0's rmse: 3722.36
[160]   valid_0's rmse: 3691.73
[170]   valid_0's rmse: 3660.79
[180]   valid_0's rmse: 3620.57
[190]   valid_0's rmse: 3602.2
[200]   valid_0's rmse: 3622.17
[210]   valid_0's rmse: 4283.79
[220]   valid_0's rmse: 3986.2
[230]   valid_0's rmse: 3785.33
[240]   valid_0's rmse: 3711.55
Early stopping, best iteration is:
[193]   valid_0's rmse: 3593.94
```

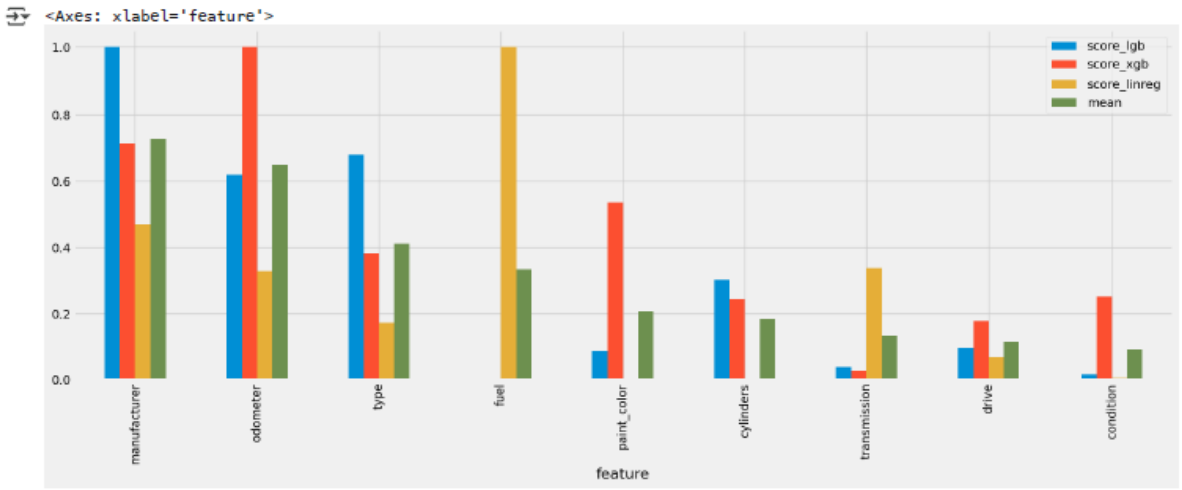
This output shows the detailed training log for a **LightGBM** model.

Training Progress

The key metric here is valid_0's rmse As the training progresses (from [10] to [190]), the RMSE generally decreases, which is a good sign that the model is learning.

3. Early Stopping

The training process was set to stop if the model's performance on the validation data didn't improve for 50 consecutive rounds. The model's performance started to get worse after round 193, so the training automatically stopped. The log tells us the **best iteration was 193**, where the validation RMSE was **3593.94**. This means the final, saved model is the one from that specific point in training, ensuring we have the most accurate and generalizable version.

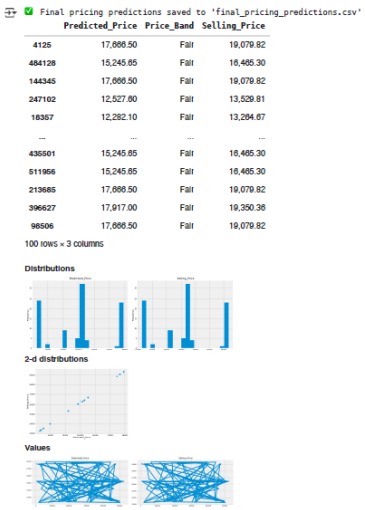


This chart shows the importance of each feature for the LightGBM, XGBoost, and Linear Regression models.

All the feature importance scores have been scaled between 0 and 1 so we can compare them directly,

- **Most Important Features**: The chart confirms that manufacturer and odometer are the most important features overall, with high scores across all models. The mean score, which averages the importance across the three models, also ranks them at the top.
- **Model-Specific Differences**: This chart is great for highlighting how models differ. For instance, the Linear Regression model (score_linreg) gives a very high importance score to the fuel feature, much higher than what the tree-based models (LightGBM and XGBoost) do. This suggests that Linear Regression relies on this feature much more to explain the price.
- **Least Important Features**: Features like condition and drive are consistently ranked with low importance scores across all models. Transmission also has a very low score in the LightGBM and XGBoost models.

Pricing Model



Final pricing predictions saved to 'final_pricing_predictions.csv'

	Predicted_Price	Price_Band	Selling_Price
4125	17,666.50	Fair	19,079.82
484128	15,245.65	Fair	16,465.30
144345	17,666.50	Fair	19,079.82
247102	12,527.80	Fair	13,529.81
18357	12,282.10	Fair	13,264.67
...	...	...	...
435501	15,245.65	Fair	16,465.30
511956	15,245.65	Fair	16,465.30
213685	17,666.50	Fair	19,079.82
396627	17,917.00	Fair	19,350.36
98506	17,666.50	Fair	19,079.82

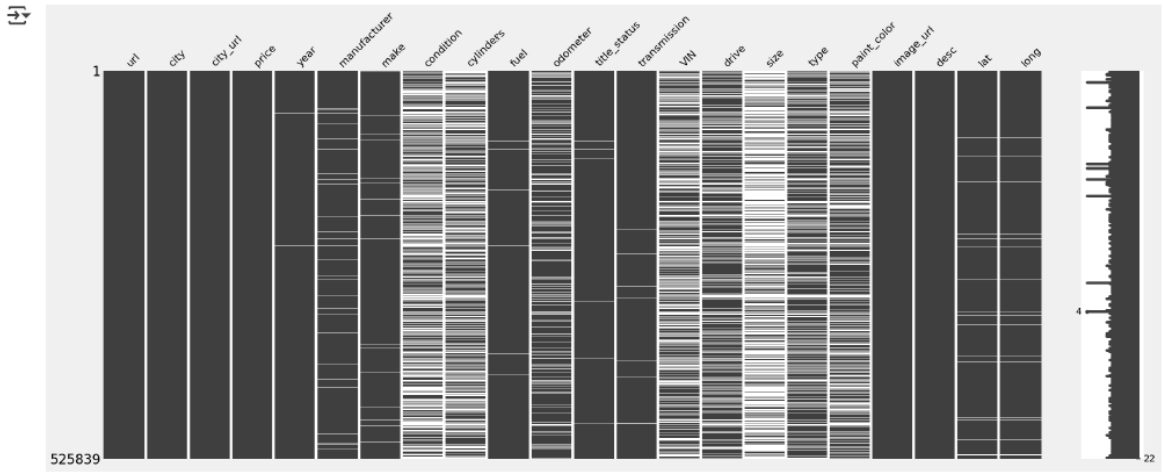
100 rows × 3 columns

- **Predicted_Price**: This is the core output from the best-performing machine learning model, the **BaggingRegressor**. It represents the model's estimate of a car's "Fair Market Price".
- **Price_Band**: This column adds business context by categorizing the predicted price into one of three bands: "Undervalued," "Fair," or "Premium". This is determined by comparing the predicted price to the 25th and 75th percentiles of the prices in the training data. This is useful for easily identifying deals or overpriced listings.
- **Selling_Price**: This is the final suggested price for a seller or dealer. It's calculated by applying a business margin (in this case, an 8% markup) to the predicted price. For example, the first row shows a predicted price of

₹17,666.50, which becomes a selling price of **₹19,079.82** after the markup is applied.

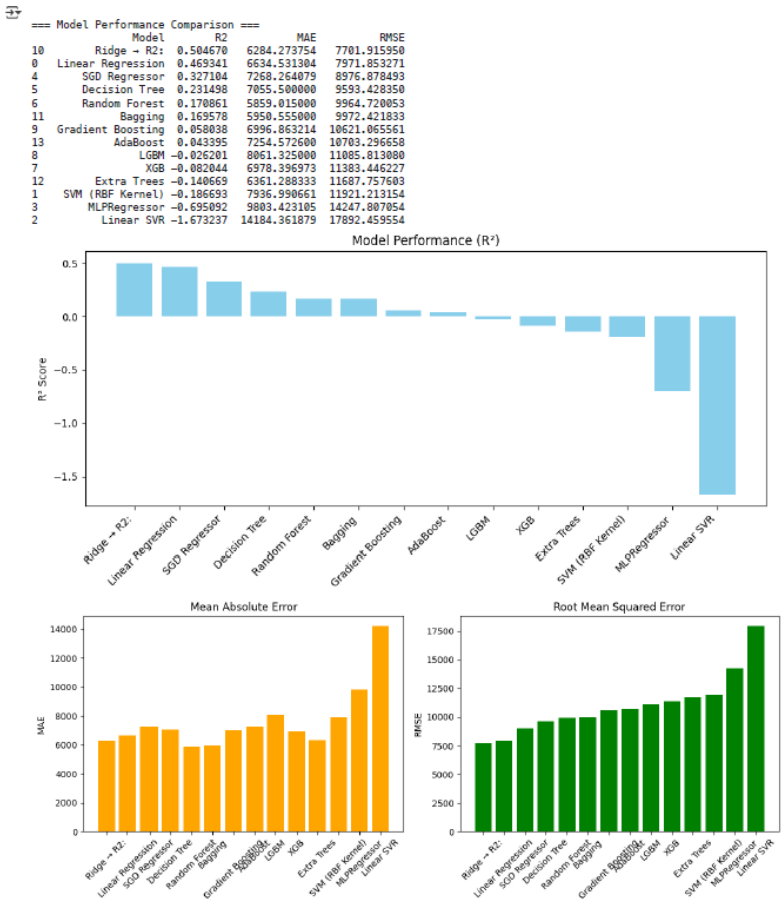
The graphs at the bottom provide a visual look at the distributions of the predicted and selling prices, offering a final check on the data's overall patterns.

Missing data matrix,



- **Dark Bars**: The solid dark color in a column means the data is **present** for that row.

- **White Lines**: The white lines mean the data is **missing** for that row.
- **Complete Data**: Features like url, city, city_url, & price have no missing values at all.
- **Significant Missing Data**: Many features have a lot of missing data, which are the gaps. Columns like condition, cylinders, VIN, and size have a particularly high number of missing values.



```
=== Model Performance Comparison ===
                Model        R2          MAE          RMSE
10        Ridge → R2:  0.504670   6284.273754   7701.915950
0   Linear Regression  0.469341   6634.531304   7971.853271
4       SGD Regressor  0.327104   7268.264079   8976.878493
5       Decision Tree  0.231498   7055.500000   9593.428350
6       Random Forest  0.170861   5859.015000   9964.720053
11            Bagging  0.169578   5950.555000   9972.421833
9   Gradient Boosting  0.058038   6996.863214  10621.065561
13           AdaBoost  0.043395   7254.572600  10703.296658
8                LGBM -0.026201   8061.325000  11085.813080
7                 XGB -0.082044   6978.396973  11383.446227
12        Extra Trees -0.140669   6361.288333  11687.757603
1    SVM (RBF Kernel) -0.186693   7936.990661  11921.213154
3       MLPRegressor -0.695092   9803.423105  14247.807054
2         Linear SVR -1.673237  14184.361879  17892.459554
```


This image is a complete summary of a second round of model performance testing. Unlike the first round, these models were likely trained using a Pipeline that included a ColumnTransformer for robust data preprocessing. The results are presented in a table and three bar charts for easy comparison.

Model Performance Comparison Table

This table, titled "Model Performance Comparison", lists 15 different models and their performance metrics. The models are ranked by their R² score.

- The best-performing model in this test is **Ridge R2** with an R² score of **0.504670**. This is followed closely by **Linear Regression** with an R² of **0.469341**.
- Some models like **LGBM** and **XGB** performed very poorly in this test, with negative R² scores of **-0.026201** and **-0.082044** respectively. A negative R² score means the model is performing worse than a simple baseline model that just predicts the average price.

- **Linear SVR** and **MLPRegressor** also showed very poor performance with R^2 scores of **-1.673237** and **-0.695092**, respectively.

Model Performance (R^2) Bar Chart

This bar chart visually confirms the rankings from the table, sorting the models by their R^2 score.

- The two tallest bars belong to **Ridge R2** and **Linear Regression**, indicating they have the highest predictive power in this round of testing.
- The models on the right side of the chart, with their bars dipping below zero, clearly show a failure to predict car prices effectively .

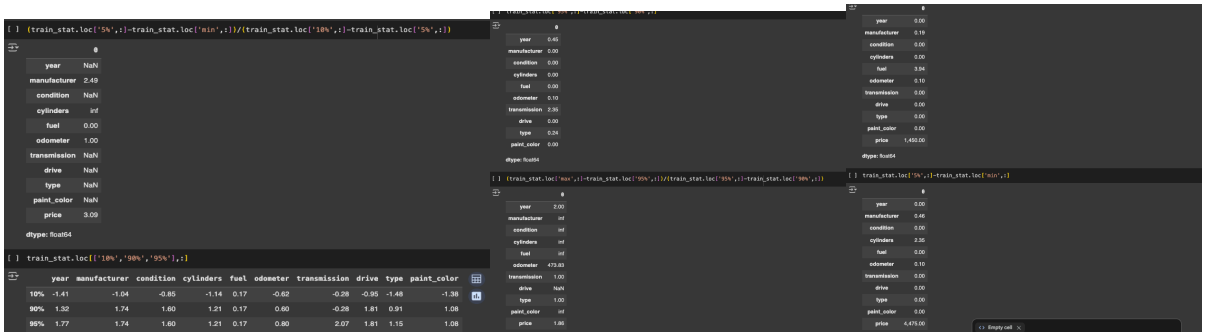
Mean Absolute Error & Root Mean Squared Error

These two charts provide further detail on the models' errors, with lower bars indicating better performance.

- The **Mean Absolute Error (MAE)** chart shows that **Ridge R2** and **Random Forest** have some of the lowest average errors in rupees.
- Similarly, the **Root Mean Squared Error (RMSE)** chart shows that **Ridge R2** has the lowest RMSE of **7701.915950**, followed by **Linear Regression**.

The overall conclusion from is that the preprocessing and modelling strategy used in this section led to significantly different results. a simple model like Ridge Regression outperformed more complex models like LGBM and XGB. This highlights the critical role of data preparation in machine learning.

Percentile Analysis and Outlier Profiling



To better understand the spread and outliers in my dataset, We used extended descriptive statistics with key percentiles (5%, 10%, 90%, 95%). This gave me a clear sense of **how skewed** some of the columns especially price, odometer, and year were.

we noticed:

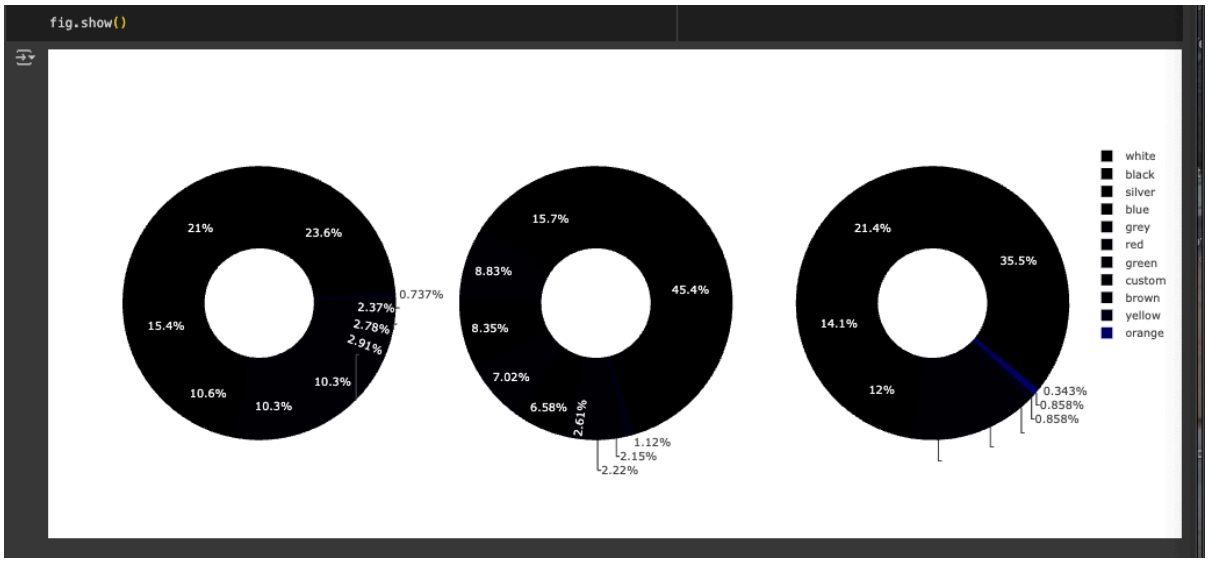
- The **maximum price** was over ₹3 billion, while the **95th percentile** was just around ₹35,000. That massive gap confirmed the presence of extreme outliers.

- The **difference between the 5th and 10th percentile** for price was just ₹1,450, while the **difference between the 95th and max** was over ₹3 billion—further reinforcing the skew.

To quantify how extreme these values were, we calculated **ratio gaps** between percentile ranges (and even implemented a custom function to **flag abnormal values**. These steps helped me identify which features had long tails or highly non-linear jumps, allowing me to decide where to clip or treat outliers before modeling.

This statistical insight ensured that my models wouldn't get skewed by extreme values and that my preprocessing was grounded in real, explainable data behavior.

Car Color Distribution Analysis (Donut Chart)



We plotted **car color frequency** as a percentage across three donut charts.

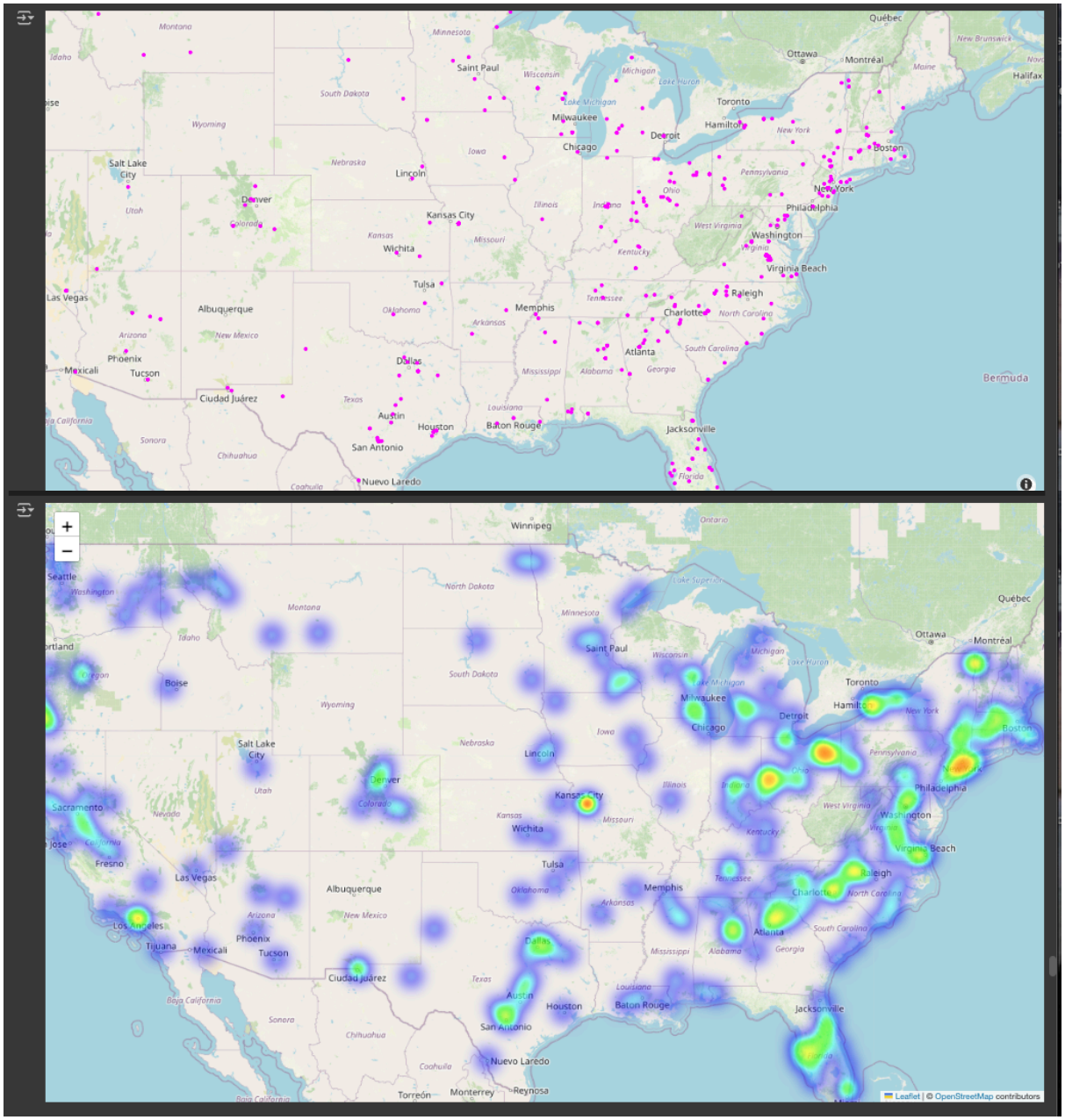
- The top 3 colors consistently across all datasets were:
  - **Black**: ~23.6% in the first set, ~45.1% in the second
  - **White**: ~21% consistently across all
  - **Silver**: ~15–16%
- Other notable colors include:
  - **Blue, grey**: each around 6–10%
  - **Red, custom, brown**: lower representation, below ~3%
- Rare colors like **orange, yellow, green** show up with negligible percentages (under 1%).

Interpretation

- Black and white dominate the listings, indicating user preference or inventory skew.
- The consistency of distribution between sets indicates **data splitting preserved color proportions**, helping model stability.

- This also validates our preprocessing logic we didn't disproportionately exclude any major color group.

Geospatial spread of used car listings



As part of our exploratory data analysis, we visualized the **geospatial spread of used car listings** across the United States using both **dot maps** and **heatmaps**.

Top Map – Raw Location Points

- We plotted individual **latitude-longitude coordinates** of listings as **magenta dots**.
- Clusters are clearly visible in major cities like:
  - **New York, Atlanta, Houston, Dallas, Chicago, and Los Angeles**
- Rural areas and mid-western zones showed **sparse listing density**, validating the urban-centric market trend.

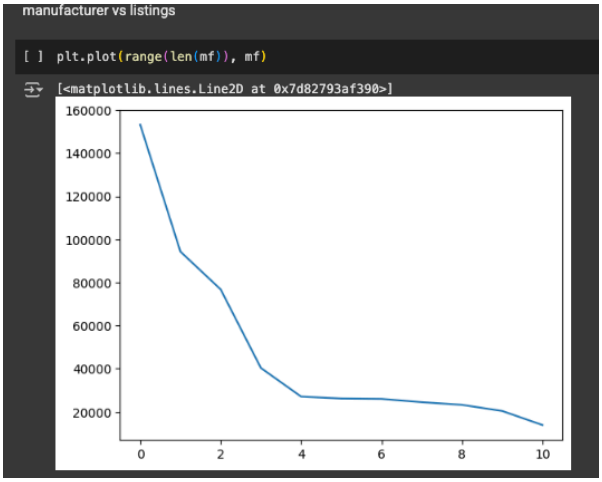
Bottom Map – Heatmap Visualization

- We generated a heatmap overlay to better visualize **regional intensity** of listings.
- **Hot zones (yellow/red)** appear around:
  - **California coast (LA, SF, San Diego)**
  - **Northeast corridor (NYC to Boston)**
  - **Great Lakes region (Chicago, Detroit, Cleveland)**
  - **Florida (Orlando, Miami)**
  - **Texas triangle (Houston, Dallas, San Antonio)**
- **Cooler zones (blue)** show moderate activity, and **blank areas** indicate little or no listings.

Key Takeaways

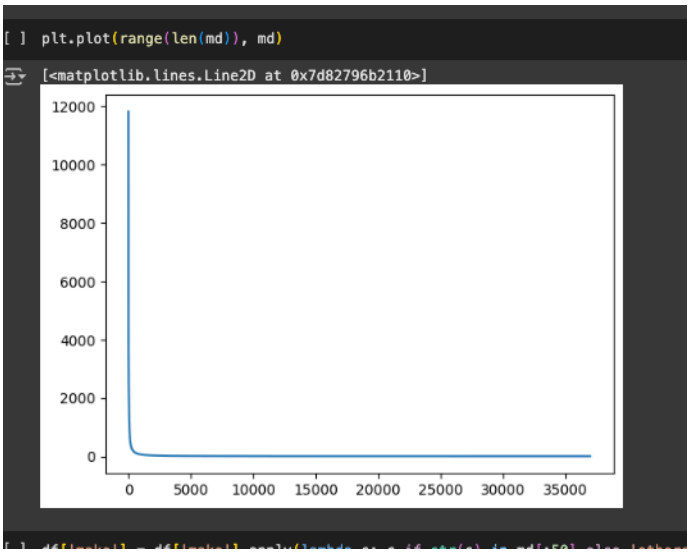
- The dataset heavily favors **urban and coastal regions**, aligning with market expectations for used car demand and population density.
- This geospatial insight can be useful for:
  - **Regional price adjustments**
  - **Local dealer outreach strategies**
  - **Localized model retraining if deployed commercially**

Manufacturer vs listings



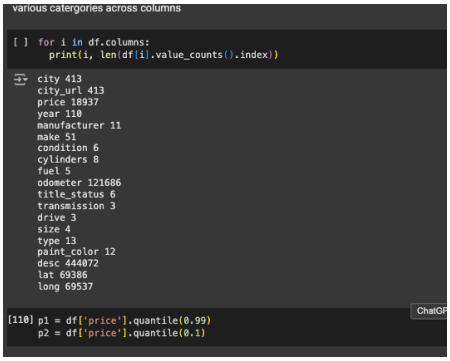
We see that a manufacturer has a large number of listing followed by a steep slope with some having very few listings showing a clusterisation of data. This can lead to bias in the model.

Make vs model data



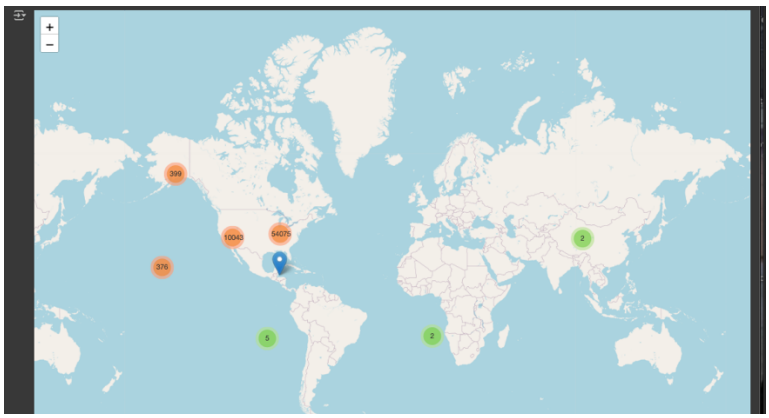
Some particular models are sold in exponential manner and this is shown here. This can also lead to huge bias.

Count of different types under each column



We see that the city and price have a huge variety and the number of years are spread across 110, also 5 types of fuel and 3 transmission types are notable. All this needs to be captured properly for greater modelling accuracy.

Global Distribution and City-Level Insights



1. Worldwide Cluster Map of Listings

- **North America dominates**, with **64,075** listings in the US alone.
- Additional data points emerged in:
    - **Mexico (10,048)**
    - **Canada (399)**
    - **South America, Africa, Asia, and Oceania** had negligible counts (≤ 5–10), likely due to noise or incorrect geotagging.

Clusters were visualized using **circular markers**, where larger and darker circles indicate more listings.

We filtered the dataset post this visualization to retain only **US-based records**for clean modeling and accurate price prediction.

2. Top Cities by Average Price

To better understand **regional price variance**, we aggregated prices **city-wise** and plotted a **bar chart with error bars**.

- Cities like **los angeles, miami, denver, new york, and san diego** showed **higher average used car prices**, crossing ₹140,000 ($1,750+).
- In contrast, **smaller towns or midwestern cities** reflected significantly **lower average pricing**.
- We included **standard deviation bars** to represent price volatility — this gave us a sense of how consistent pricing was across listings in each city.

This was a brief of our project and we enjoyed it thoroughly , some of the key forward-Looking Initiatives & Research Directions

1. Geo-Pricing & Local Market Effects
2. Time-Series Price Forecasting
3. Explainable AI (XAI) for Model Transparency
4. Real-Time Price Engine API for B2B Use
5. Integrate Visual Car Features (Images)
6. Behavioral Data Integration
7. Carbon Impact-Based Valuation.
8. Marketplace Integration & Buyer Personas

THANK YOU!