An Analysis and Prediction of Used Car Prices

1 RESULTS

1.1 Model Comparison and Selection

Following the data preparation outlined in Section 3 (from the Stage 1 report), I conducted a comparative analysis of four distinct machine learning models: Linear Regression, Ridge Regression, Decision Tree, and the pre-selected Random Forest Regressor. I used both the training and validation errors in this evaluation to assess each model's complexity, generalization capability, and predictive performance.

I calculated all Mean Squared Error (MSE) and R^2 metrics on the log-transformed price scale. The final Root Mean Squared Error (RMSE) is presented on the original price scale (in Lakhs).

Model Comparison Summary (Train/Test/CV Errors)

Model	Train MSE	Test MSE	Train R²	Test R²	CV MSE Mean	CV MSE Std	RMSE (Original Scale)
Linear Regression	0.1416	0.1563	0.8	0.7733	0.1475	0.0062	343107.7475
Ridge Regression	0.142	0.1547	0.7993	0.7756	0.1474	0.0053	331323.7705
Decision Tree	0.0148	0.1839	0.9791	0.7333	0.192	0.0079	300660.8054
Random Forest	0.0291	0.1291	0.9588	0.8129	0.1323	0.0038	297773.6527

Table 1: Comparative Performance of Machine Learning Models

I selected the Random Forest Regressor as the final chosen method because it showed the best overall performance metrics. The model achieved the lowest Test MSE (0.1291 on the log scale) and the highest Test R^2 (0.8129), confirming its superior ability to generalize to unseen data. The Decision Tree showed a very low training error (0.0148), but its Test R^2 dropped to 0.7333, demonstrating significant overfitting. The ensemble approach of the Random Forest successfully reduced this overfitting, achieving a better balance between training and test performance.

The final test error of the chosen Random Forest model is a Test R^2 of 0.8129. This result indicates that the model's features explain approximately 81.3% of the variance in the log-transformed used car prices. I confirmed the stability and robustness of this performance by observing the low standard deviation of the cross-validation error (CV MSE Mean: 0.1323 +/- 0.0038).

1.2 Model Insight and Feature Analysis

To understand the factors driving the Random Forest model's predictions, I conducted a feature importance analysis. This step was necessary to validate the feature engineering process and establish the true relationship between the input variables and the selling price.

Top 4 Feature Importances

	car_age	km_driven	transmission_Manual	fuel_Diesel
Importance	51.24%	9.80%	7.62%	6.61%

Table 2: Random Forest Feature Importance Ranking

The analysis reveals that car_age is the dominant factor, contributing over half of the model's predictive power (0.5124). This reflects the real-world impact of depreciation and strongly validates my decision to engineer the car_age feature in the preprocessing stage (Section 3.1 from the Stage 1 report).

The remaining features of significant influence are km_driven (0.0980), representing vehicle usage, and the categorical features transmission_Manual (0.0762) and fuel_Diesel (0.0661). These results confirm that mechanical specifications and fuel type play a key role in secondary pricing factors.

2 CONCLUSION

2.1 Summary of Findings

Note: All monetary values are expressed in Lakhs for clarity and comparability with Indian market conventions.

In this report, I successfully formulated the prediction of used car prices as a supervised regression problem and developed a robust machine learning solution. Following a comprehensive preprocessing phase, which included feature engineering and logarithmic transformation of the target variable, I identified the Random Forest Regressor as the best predictive model. This final model demonstrated strong performance, explaining approximately 81.3% of the variance in used car prices on the test set, while achieving an original scale RMSE of 2.9777 Lakhs. The high correlation between actual and predicted prices (0.8452) confirms the model's effective fit to the overall data trends.

2.2 Problem Resolution and Room for Improvement

The results suggest that the problem is solved satisfactorily for the central distribution of car prices. The high R^2 value (0.8129) and stable cross-validation metrics confirm that the model successfully generalizes the relationship between the features and the car's selling price.

However, there remains significant room for improvement, primarily concerning the model's performance on high-value outliers. I observed that the maximum actual price (89 Lakhs**)** was substantially higher than the maximum predicted price (49.5 Lakhs), which indicates a functional limitation in accurately forecasting the prices of high-end vehicles.

2.3 Limitations and Future Work

The primary limitation of the current method is the under-prediction of high-value instances within the test set. I found that this issue is reflected in the residual analysis, where the residuals exhibit slight non-normality (Skew 0.25, Kurtosis 1.9). Non-normality often suggests model errors are concentrated at the price extremes, as shown in the following figures.

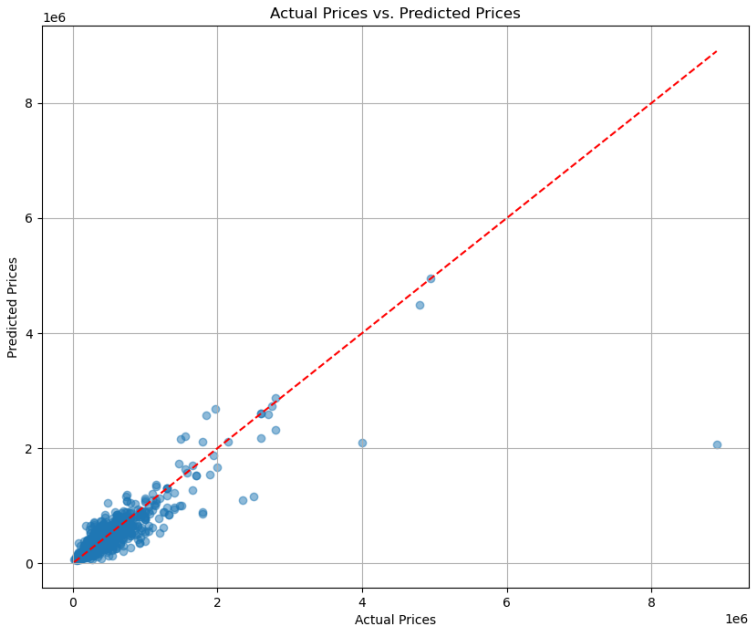


Figure 1: Actual vs Predicted Price Scatter Plot (Test Set)

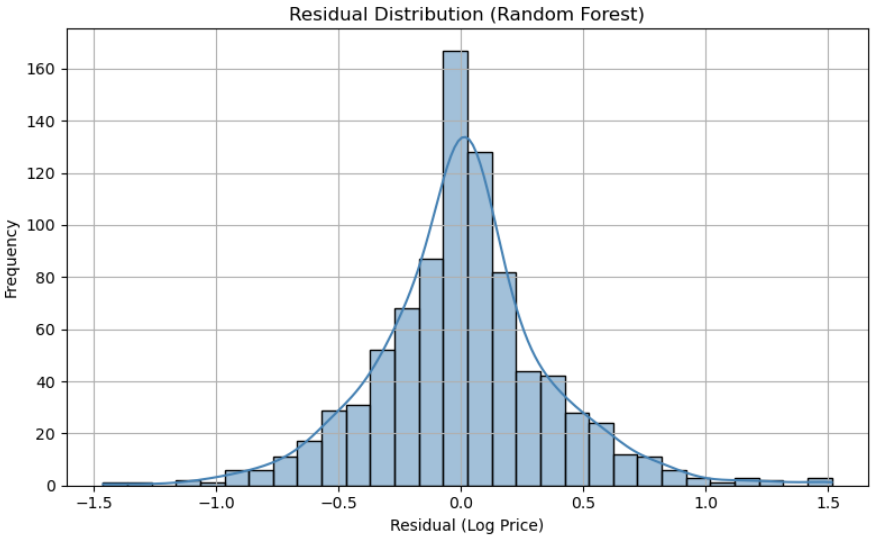


Figure 2: Residual Distribution of the Random Forest Model

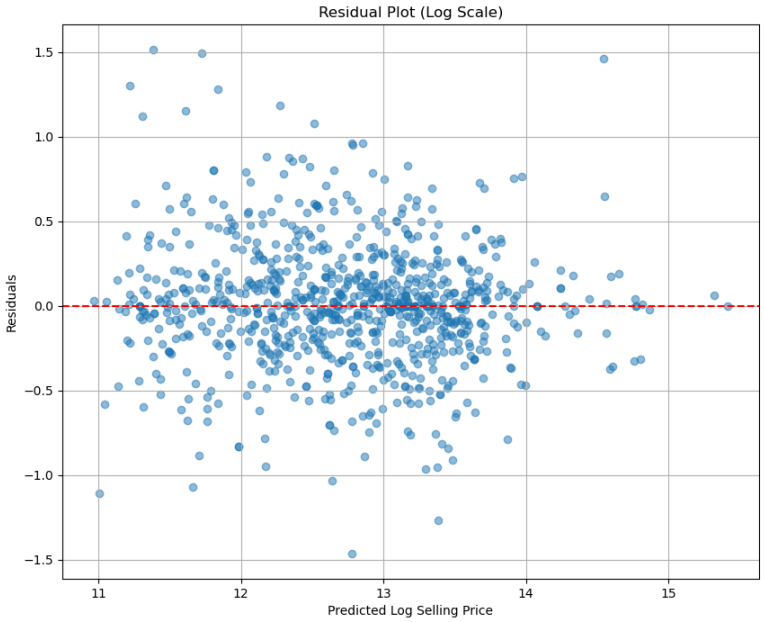


Figure 3: Residuals vs. Predicted Price Scatter Plot (Test Set)

The residual scatter plot specifically confirms that the model's errors are not random, but show signs of increasing variance as predicted price increases.

To further improve the model's generalization capability and address the outlined limitations, future work should consider:
- Alternative Ensemble Methods: Future work should explore models like Gradient Boosting Machines (GBM) or XGBoost. These models are designed to sequentially correct the errors (residuals) of prior predictions, which should improve accuracy in regions where the Random Forest currently struggles.

- Advanced Hyperparameter Tuning: I recommend performing specific optimization on the Random Forest model using methods such as GridSearchCV or RandomizedSearchCV. This tuning should explicitly target the minimization of the original-scale RMSE to heavily penalize and reduce the large errors associated with outlier predictions.

- Outlier Feature Engineering: I believe that introducing explicit features that better signal a vehicle's luxury or specialty status (e.g., brand tier classification) could provide the model with stronger, dedicated information to improve high-value price estimation.

3 USE OF AI

I used AI to suggest improvements for code efficiency and clarity, and I implemented and tested all changes myself. I also used it to learn about advanced ensemble methods (GBM, XGBoost) and hyperparameter tuning (GridSearchCV, RandomizedSearchCV), which informed my discussion of future improvements.

4 REFERENCES

Data Source: Birla, N. (2020). Vehicle Dataset from CarDekho. Kaggle. [https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho]

APPENDIX

Stage1_and_2_Code

October 8, 2025

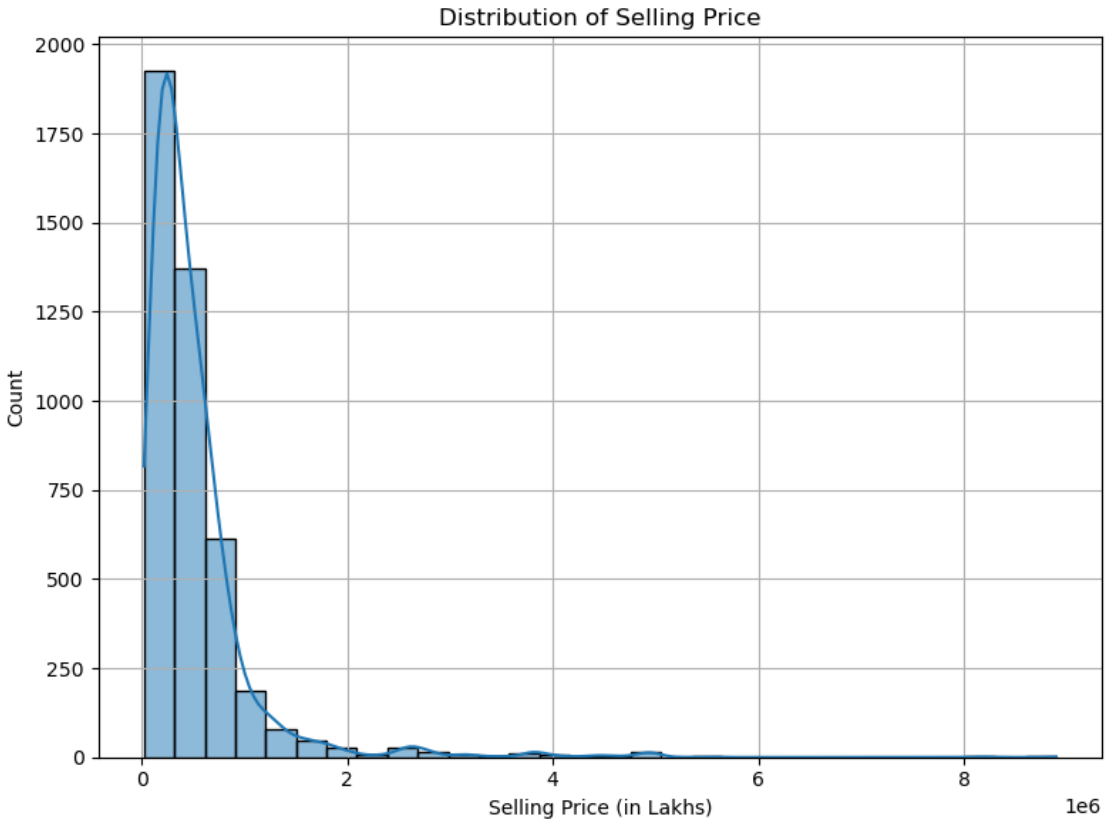
[18]:
```
# Stage 1 Code
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import mean_squared_error
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

# Data Loading
try:
    file_name = 'CAR DETAILS FROM CAR DEKHO.csv'
    df = pd.read_csv(file_name)
except FileNotFoundError:
    print(f"Error: The file '{file_name}' was not found.")
    print("Please make sure the file is in the same directory as this script.")
    exit()

# Dataset Verification
print("Verifying Dataset Against Report Claims")
expected_rows = 4340
expected_cols = 8
if df.shape[0] == expected_rows and df.shape[1] == expected_cols:
    print(f"Verification: Dataset dimensions ({df.shape[0]} rows, {df.shape[1]}␣
 ↪columns) match the expected dimensions.")
else:
    print(f"Warning: Dataset dimensions ({df.shape[0]} rows, {df.shape[1]}␣
 ↪columns) do NOT match the expected dimensions.")

if df.isnull().sum().sum() == 0:
```

```
    print("Verification: The dataset has no missing values.")
else:
    print(f"Warning: The dataset contains missing values. Total found: {df.
 ↪isnull().sum().sum()}")

# Feature Engineering
current_year = datetime.now().year
df['car_age'] = current_year - df['year']
df['brand'] = df['name'].apply(lambda x: x.split(' ')[0])

# Exploratory Data Visualization
# Figure 1: Selling price distribution
plt.figure(figsize=(8, 6))
sns.histplot(df['selling_price'], kde=True, bins=30)
plt.title('Distribution of Selling Price')
plt.xlabel('Selling Price (in Lakhs)')
plt.ylabel('Count')
plt.grid(True)
plt.tight_layout()
plt.savefig('selling_price_distribution.png')
print("Plot saved as selling_price_distribution.png")
plt.show()

# Figure 2: Numerical correlation heatmap
numerical_features_df = df[['selling_price', 'car_age', 'km_driven']]
corr_matrix_numerical = numerical_features_df.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix_numerical, annot=True, cmap='viridis', fmt=".2f",␣
 ↪linewidths=.5)
plt.title('Correlation Heatmap of Key Numerical Features')
plt.tight_layout()
plt.savefig('numerical_correlation_heatmap.png')
print("Plot saved as numerical_correlation_heatmap.png")
plt.show()

# Figure 3: Feature scatter plots
fig, axes = plt.subplots(1, 2, figsize=(15, 6))
sns.scatterplot(data=df, x='car_age', y='selling_price', ax=axes[0], alpha=0.6)
axes[0].set_title('Selling Price vs. Car Age')
axes[0].set_xlabel('Car Age (in years)')
axes[0].set_ylabel('Selling Price (in Lakhs)')
axes[0].grid(True)
sns.scatterplot(data=df, x='km_driven', y='selling_price', ax=axes[1], alpha=0.
 ↪6)
axes[1].set_title('Selling Price vs. Kilometers Driven')
axes[1].set_xlabel('Kilometers Driven')
axes[1].set_ylabel('Selling Price (in Lakhs)')
```

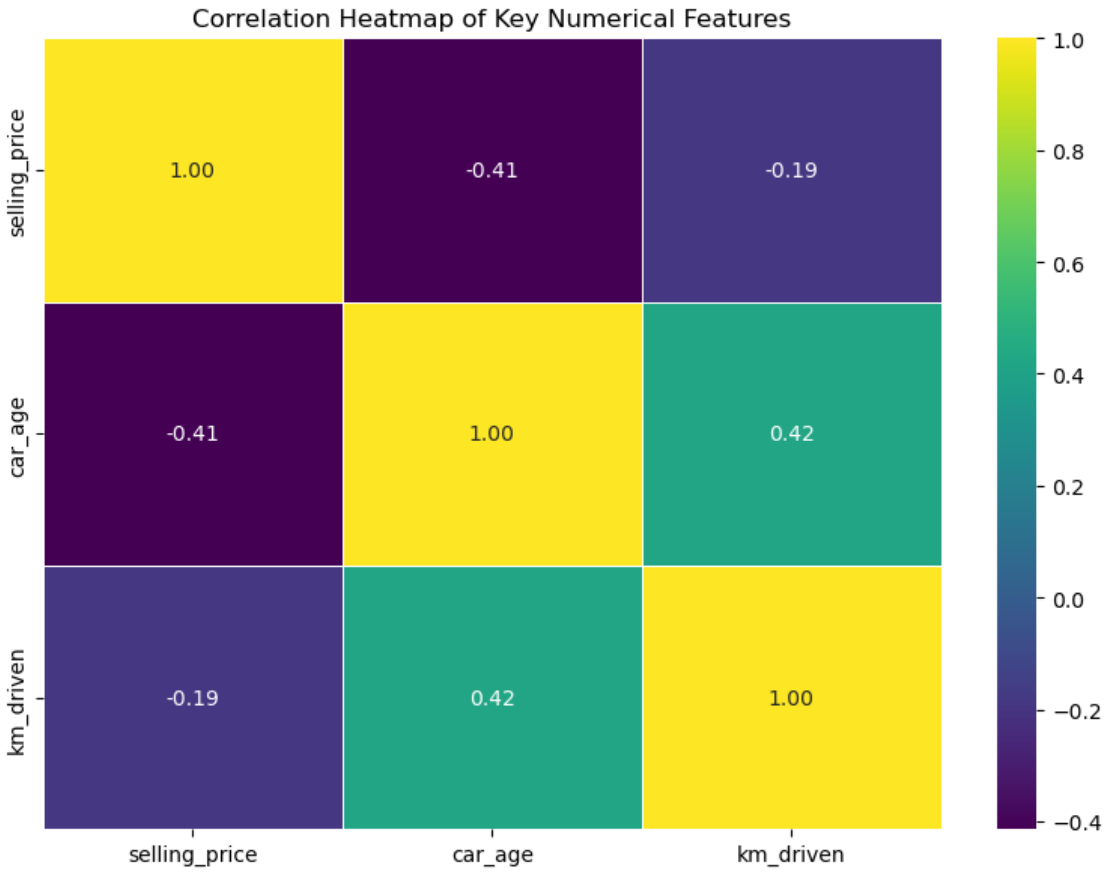
```
axes[1].grid(True)
plt.tight_layout()
plt.savefig('feature_scatter_plots.png')
print("Plot saved as feature_scatter_plots.png")
plt.show()

# Figure 4: Average price by brand
brand_counts = df['brand'].value_counts()
top_10_brands = brand_counts.head(10).index
top_brands_df = df[df['brand'].isin(top_10_brands)]
brand_prices = top_brands_df.groupby('brand')['selling_price'].mean().
 ↪sort_values(ascending=False).reset_index()
brand_prices.columns = ['brand', 'selling_price']
plt.figure(figsize=(12, 8))
plt.bar(x=brand_prices['brand'], height=brand_prices['selling_price'],␣
 ↪color=plt.cm.viridis(np.linspace(0, 1, len(brand_prices))))
plt.title('Average Selling Price by Top 10 Brands')
plt.xlabel('Car Brand')
plt.ylabel('Average Selling Price (in Lakhs)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig('average_price_by_brand.png')
print("Plot saved as average_price_by_brand.png")
plt.show()
plt.close('all')

# Data Transformation
df['selling_price_log'] = np.log1p(df['selling_price'])
df.drop(['year', 'name', 'selling_price'], axis=1, inplace=True)

X = df.drop('selling_price_log', axis=1)
y = df['selling_price_log']

# Data Preprocessing
numerical_features = ['km_driven', 'car_age']
categorical_features = ['fuel', 'seller_type', 'transmission', 'owner', 'brand']
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ],
    remainder='passthrough'
)

# Model Selection
model = RandomForestRegressor(random_state=42)
```

```
# Single Train-Test Split
print("\nEvaluating model with a single train-test split")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

pipeline_split = Pipeline(steps=[('preprocessor', preprocessor),
                                 ('regressor', model)])
pipeline_split.fit(X_train, y_train)

y_pred_split = pipeline_split.predict(X_test)
mse_split = mean_squared_error(y_test, y_pred_split)
print(f"Mean Squared Error (MSE) from single split: {mse_split:.4f}")

# 5-Fold Cross-Validation
print("\nEvaluating model with 5-fold cross-validation")
cv = KFold(n_splits=5, shuffle=True, random_state=42)
pipeline_cv = Pipeline(steps=[('preprocessor', preprocessor),
                              ('regressor', model)])

cv_scores = -1 * cross_val_score(pipeline_cv, X, y, cv=cv,␣
 ↪scoring='neg_mean_squared_error')

print("MSE scores for each fold:", np.round(cv_scores, 4))
print(f"Average MSE across 5 folds: {np.mean(cv_scores):.4f}")
print(f"Standard deviation of MSE across 5 folds: {np.std(cv_scores):.4f}")
print("-" * 50)
```

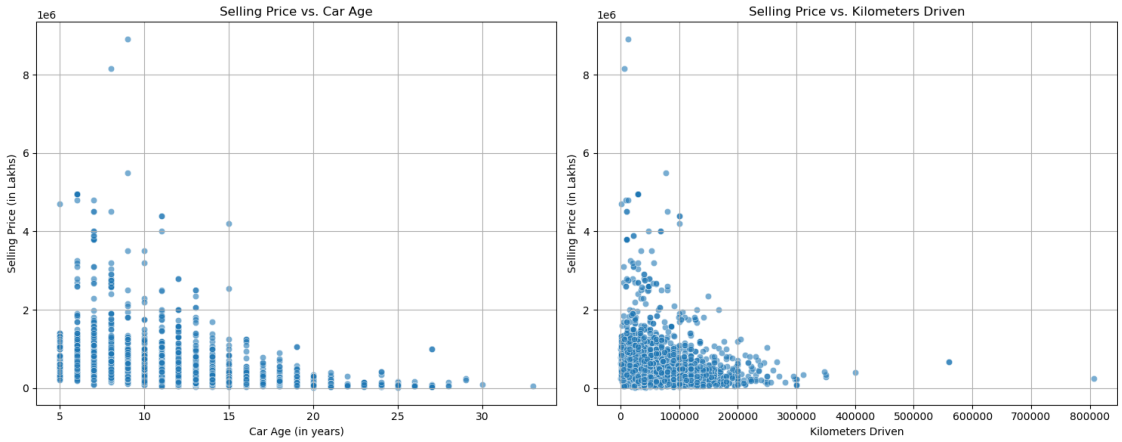
```
Verifying Dataset Against Report Claims
Verification: Dataset dimensions (4340 rows, 8 columns) match the expected
dimensions.
Verification: The dataset has no missing values.
Plot saved as selling_price_distribution.png
```



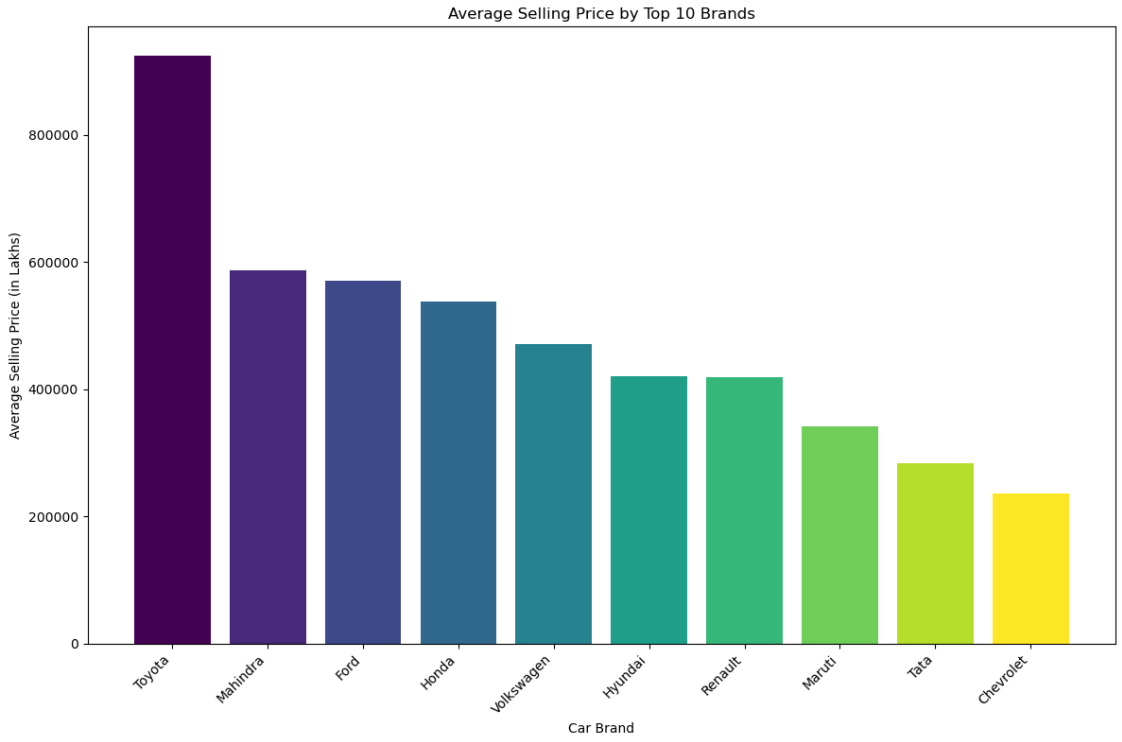

```
Plot saved as numerical_correlation_heatmap.png
```



Plot saved as feature_scatter_plots.png



Plot saved as average_price_by_brand.png




```
Evaluating model with a single train-test split
Mean Squared Error (MSE) from single split: 0.1291

Evaluating model with 5-fold cross-validation
MSE scores for each fold: [0.1282 0.131  0.1387 0.1295 0.134 ]
Average MSE across 5 folds: 0.1323
Standard deviation of MSE across 5 folds: 0.0038
--------------------------------------------------
```

[19]:
```python
# Stage 2 Code
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

```
from scipy.stats import skew, kurtosis
import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

# 1. Prepare features and target
X = df.drop('selling_price_log', axis=1)
y = df['selling_price_log']

num_features = ['km_driven', 'car_age']
cat_features = ['fuel', 'seller_type', 'transmission', 'owner', 'brand']

preprocessor = ColumnTransformer([
    ('num', StandardScaler(), num_features),
    ('cat', OneHotEncoder(handle_unknown='ignore'), cat_features)
])

# 2. Baseline model
model = RandomForestRegressor(n_estimators=100, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
pipeline = Pipeline([('preprocessor', preprocessor), ('regressor', model)])
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

mse_split = mean_squared_error(y_test, y_pred)
print(f"MSE (Log Scale) - Single Split: {mse_split:.4f}")

cv = KFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = -1 * cross_val_score(pipeline, X, y, cv=cv,␣
 ↪scoring='neg_mean_squared_error')
print(f"CV MSEs (Log Scale): {np.round(cv_scores, 4)}")
print(f"Avg CV MSE: {np.mean(cv_scores):.4f} ± {np.std(cv_scores):.4f}")

# 3. Evaluate in original scale (Lakhs)
y_test_original = np.expm1(y_test) / 1e5
y_pred_original = np.expm1(y_pred) / 1e5
rmse_original = np.sqrt(mean_squared_error(y_test_original, y_pred_original))
print(f"RMSE (Original Scale): {rmse_original:.4f} Lakhs")

# 4. Actual vs predicted + residuals
comparison_df = pd.DataFrame({
    'Actual Price': y_test_original,
    'Predicted Price': y_pred_original,
    'Residuals': y_test_original - y_pred_original
}).reset_index(drop=True)
print(comparison_df.head(25))
```

```
plt.figure(figsize=(10, 8))
plt.scatter(y_test_original, y_pred_original, alpha=0.5)
plt.plot([y_test_original.min(), y_test_original.max()],
         [y_test_original.min(), y_test_original.max()], 'r--')
plt.title('Actual vs Predicted Prices')
plt.xlabel('Actual Prices (Lakhs)')
plt.ylabel('Predicted Prices (Lakhs)')
plt.grid(True)
plt.savefig('actual_vs_predicted_plot.png')

residuals = y_test - y_pred
plt.figure(figsize=(10, 8))
plt.scatter(y_pred, residuals, alpha=0.5)
plt.axhline(0, color='r', linestyle='--')
plt.title('Residual Plot (Log Scale)')
plt.xlabel('Predicted Log Selling Price')
plt.ylabel('Residuals')
plt.grid(True)
plt.savefig('residual_plot.png')

# 5. Feature importance
rf_model = pipeline.named_steps['regressor']
encoder = pipeline.named_steps['preprocessor'].named_transformers_['cat']
encoded_features = encoder.get_feature_names_out(cat_features)
feature_names = np.concatenate([num_features, encoded_features])
importances = rf_model.feature_importances_
feature_importances = pd.Series(importances, index=feature_names).
 ↪sort_values(ascending=False)

top_4 = feature_importances.head(4)
importance_percentages = [f"{i*100:.2f}%" for i in top_4.values]
fig, ax = plt.subplots(figsize=(10, 2))
ax.axis('off')
table = ax.table(
    cellText=[importance_percentages],
    colLabels=top_4.index,
    rowLabels=['Importance'],
    cellLoc='center',
    loc='center'
)
table.auto_set_font_size(False)
table.set_fontsize(12)
plt.title('Top 4 Feature Importances')
plt.savefig('feature_importance_table.png')

print(feature_importances.to_string())
```

```python
# 6. Multi-model comparison
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(alpha=1.0),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42)
}

results = []
cv = KFold(n_splits=5, shuffle=True, random_state=42)

for name, mdl in models.items():
    pipe = Pipeline([('preprocessor', preprocessor), ('model', mdl)])
    pipe.fit(X_train, y_train)
    y_train_pred = pipe.predict(X_train)
    y_test_pred = pipe.predict(X_test)
    mse_train = mean_squared_error(y_train, y_train_pred)
    mse_test = mean_squared_error(y_test, y_test_pred)
    r2_train = r2_score(y_train, y_train_pred)
    r2_test = r2_score(y_test, y_test_pred)
    cv_mse = -1 * cross_val_score(pipe, X, y, cv=cv,␣
 ↪scoring='neg_mean_squared_error')
    results.append([
        name, mse_train, mse_test, r2_train, r2_test,
        np.mean(cv_mse), np.std(cv_mse),
        np.sqrt(mean_squared_error(np.expm1(y_test)/1e5, np.expm1(y_test_pred)/
 ↪1e5))
    ])

results_df = pd.DataFrame(results, columns=[
    'Model', 'Train MSE', 'Test MSE', 'Train R²', 'Test R²',
    'CV MSE Mean', 'CV MSE Std', 'RMSE (Original Scale)'
])
print(results_df.round(4))

# 7. Save results table
fig, ax = plt.subplots(figsize=(12, 2))
ax.axis('off')
table = ax.table(
    cellText=results_df.round(4).values,
    colLabels=results_df.columns,
    loc='center',
    cellLoc='center'
)
table.auto_set_font_size(False)
table.set_fontsize(9)
```

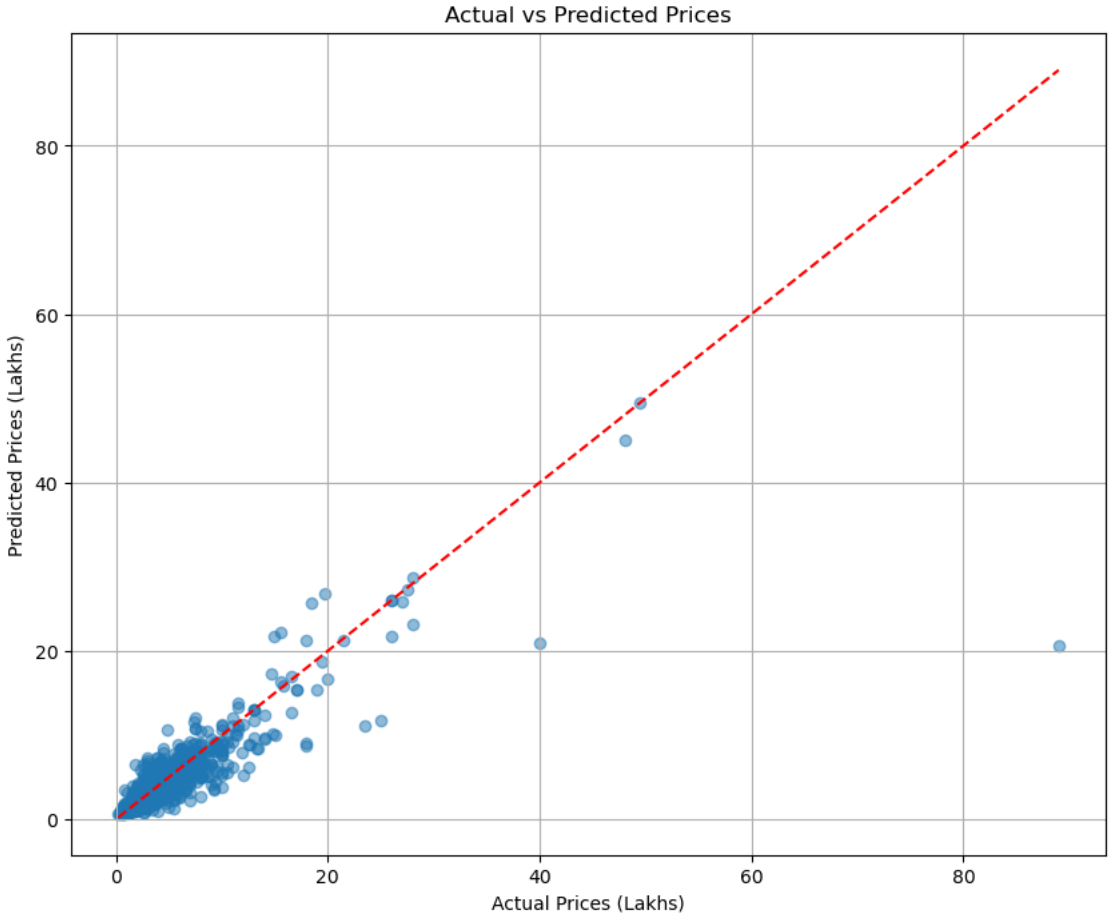
```
plt.title('Model Comparison Summary')
plt.savefig('model_comparison_table.png')

# 8. Summary for report
best_model_row = results_df.loc[results_df['Test MSE'].idxmin()]
print(f"Chosen Model: {best_model_row['Model']}")
print(f"Test MSE: {best_model_row['Test MSE']:.4f}")
print(f"Test R²: {best_model_row['Test R²']:.4f}")
print(f"CV MSE (mean ± std): {best_model_row['CV MSE Mean']:.4f} ±␣
 ↪{best_model_row['CV MSE Std']:.4f}")
print(f"RMSE (Original Price Scale): {rmse_original:.4f} Lakhs")

# 9. Performance plot
plt.figure(figsize=(8, 5))
sns.barplot(x='Model', y='RMSE (Original Scale)', data=results_df,␣
 ↪palette='viridis')
plt.title('Model Performance Comparison (RMSE)')
plt.ylabel('RMSE (Lakhs)')
plt.xlabel('Model')
plt.tight_layout()
plt.savefig('model_performance_comparison.png')

# 10. Residual distribution plot
best_model_name = best_model_row['Model']
best_model = models[best_model_name]
best_pipe = Pipeline([('preprocessor', preprocessor), ('model', best_model)])
best_pipe.fit(X_train, y_train)
y_best_pred = best_pipe.predict(X_test)
residuals_best = y_test - y_best_pred

plt.figure(figsize=(8, 5))
sns.histplot(residuals_best, bins=30, kde=True, color='steelblue')
plt.title(f'Residual Distribution ({best_model_name})')
plt.xlabel('Residual (Log Price)')
plt.ylabel('Frequency')
plt.grid(True)
plt.tight_layout()
plt.savefig('residual_distribution.png')

# 11. Save summary metrics
with open('model_summary_results.txt', 'w') as f:
    f.write("=== Model Comparison Results ===\n")
    f.write(results_df.round(4).to_string(index=False))
    f.write("\n\n=== Best Model Summary ===\n")
    f.write(f"Chosen Model: {best_model_row['Model']}\n")
    f.write(f"Test MSE: {best_model_row['Test MSE']:.4f}\n")
    f.write(f"Test R²: {best_model_row['Test R²']:.4f}\n")
```

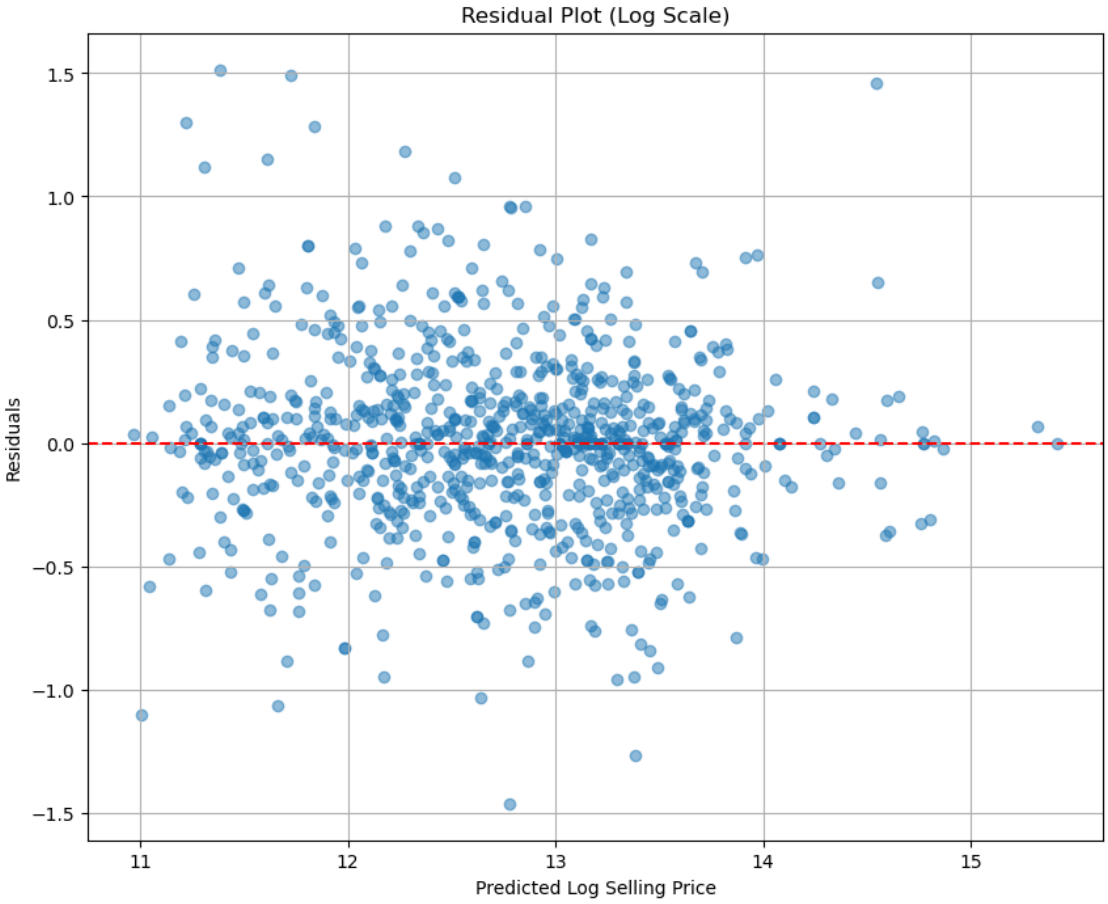
```
    f.write(f"CV MSE (mean ± std): {best_model_row['CV MSE Mean']:.4f} ±␣
 ↪{best_model_row['CV MSE Std']:.4f}\n")
    f.write(f"RMSE (Original Scale): {rmse_original:.4f} Lakhs\n")

# 12. Additional metrics
correlation_actual_pred = np.corrcoef(y_test_original, y_pred_original)[0, 1]
max_actual_price = y_test_original.max()
max_predicted_price = y_pred_original.max()
residual_skew = skew(residuals_best)
residual_kurt = kurtosis(residuals_best)

print(f"Correlation between Actual and Predicted Prices:␣
 ↪{correlation_actual_pred:.4f}")
print(f"Maximum Actual Price: {max_actual_price:.2f} Lakhs")
print(f"Maximum Predicted Price: {max_predicted_price:.2f} Lakhs")
print(f"Residual Skew (Log Price): {residual_skew:.4f}")
print(f"Residual Kurtosis (Log Price): {residual_kurt:.4f}")
```

```
MSE (Log Scale) - Single Split: 0.1291
CV MSEs (Log Scale): [0.1282 0.131  0.1387 0.1295 0.134 ]
Avg CV MSE: 0.1323 ± 0.0038
RMSE (Original Scale): 2.9777 Lakhs
    Actual Price  Predicted Price  Residuals
0          1.650         1.394897   0.255103
1          2.500         3.564337  -1.064337
2          1.200         1.232277  -0.032277
3          4.500         4.247379   0.252621
4          7.300        11.633138  -4.333138
5          1.550         2.250613  -0.700613
6          5.300         5.230340   0.069660
7          0.928         0.965001  -0.037001
8          2.840         4.102413  -1.262413
9          2.600         3.062940  -0.462940
10         5.110         5.803067  -0.693067
11        18.000         8.679025   9.320975
12         3.200         3.709434  -0.509434
13         0.900         0.820359   0.079641
14         8.500         5.612452   2.887548
15         5.000         2.726417   2.273583
16         6.240         6.708028  -0.468028
17         7.500        11.982610  -4.482610
18         1.250         0.845403   0.404597
19         6.750         7.363750  -0.613750
20         5.000         4.387302   0.612698
21        15.000        10.053999   4.946001
22         2.000         2.983279  -0.983279
23         7.000         9.136989  -2.136989
```

```
24         1.750         2.204675  -0.454675
car_age                          0.512412
km_driven                        0.098011
transmission_Manual              0.076230
fuel_Diesel                      0.066104
transmission_Automatic           0.047094
brand_Tata                       0.025287
brand_Toyota                     0.024939
fuel_Petrol                      0.016642
brand_Maruti                     0.011263
brand_Chevrolet                  0.010819
brand_Honda                      0.010143
brand_Mahindra                   0.008225
brand_Hyundai                    0.007619
owner_First Owner                0.007617
brand_Mercedes-Benz              0.007210
owner_Second Owner               0.007103
seller_type_Individual           0.006960
brand_Ford                       0.005416
seller_type_Dealer               0.005053
owner_Third Owner                0.004635
seller_type_Trustmark Dealer     0.004592
brand_Audi                       0.004584
brand_Volkswagen                 0.003475
brand_BMW                        0.003367
brand_Land                       0.003165
brand_Skoda                      0.003052
brand_Renault                    0.002977
brand_Mitsubishi                 0.002928
brand_Jaguar                     0.002162
brand_Fiat                       0.001979
owner_Fourth & Above Owner       0.001875
brand_Nissan                     0.001548
brand_Datsun                     0.001532
brand_Jeep                       0.000738
brand_Volvo                      0.000693
brand_MG                         0.000667
fuel_CNG                         0.000552
fuel_LPG                         0.000441
brand_OpelCorsa                  0.000225
brand_Ambassador                 0.000177
brand_Kia                        0.000171
brand_Isuzu                      0.000158
owner_Test Drive Car             0.000131
brand_Force                      0.000026
fuel_Electric                    0.000002
               Model  Train MSE  Test MSE  Train R²  Test R²  CV MSE Mean  \
0  Linear Regression     0.1416    0.1563    0.8000   0.7733       0.1475
```

```
1   Ridge Regression     0.1420    0.1547    0.7993    0.7756       0.1474
2      Decision Tree     0.0148    0.1839    0.9791    0.7333       0.1920
3      Random Forest     0.0291    0.1291    0.9588    0.8129       0.1323

   CV MSE Std  RMSE (Original Scale)
0      0.0062                 3.4311
1      0.0053                 3.3132
2      0.0079                 3.0066
3      0.0038                 2.9777
Chosen Model: Random Forest
Test MSE: 0.1291
Test R²: 0.8129
CV MSE (mean ± std): 0.1323 ± 0.0038
RMSE (Original Price Scale): 2.9777 Lakhs
Correlation between Actual and Predicted Prices: 0.8452
Maximum Actual Price: 89.00 Lakhs
Maximum Predicted Price: 49.50 Lakhs
Residual Skew (Log Price): 0.2503
Residual Kurtosis (Log Price): 1.9217
```



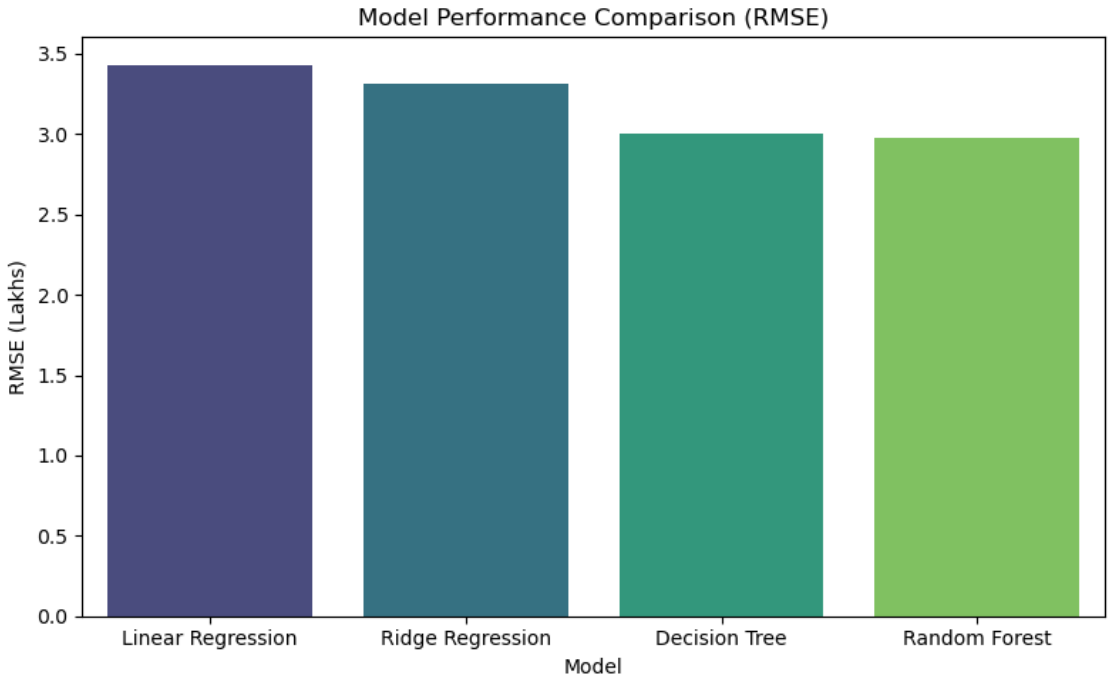


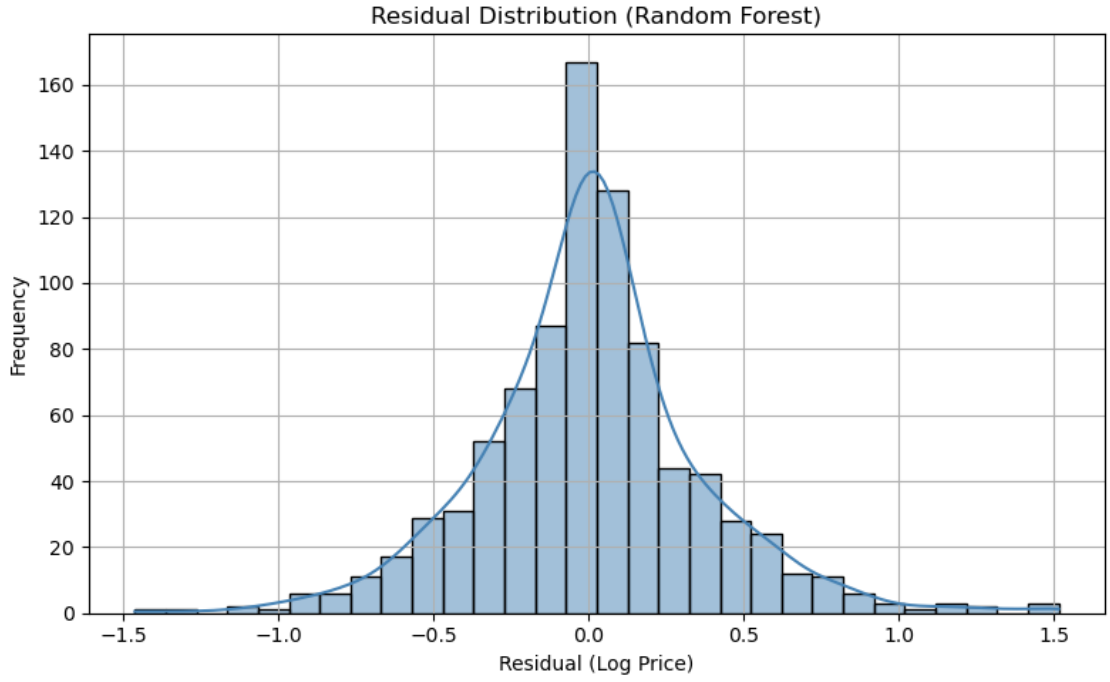
Top 4 Feature Importances

	car_age	km_driven	transmission_Manual	fuel_Diesel
Importance	51.24%	9.80%	7.62%	6.61%

Model Comparison Summary

Model	Train MSE	Test MSE	Train R²	Test R²	CV MSE Mean	CV MSE Std	RMSE (Original Scale)
Linear Regression	0.1416	0.1563	0.8	0.7733	0.1475	0.0062	3.4311
Ridge Regression	0.142	0.1547	0.7993	0.7756	0.1474	0.0053	3.3132
Decision Tree	0.0148	0.1839	0.9791	0.7333	0.192	0.0079	3.0066
Random Forest	0.0291	0.1291	0.9588	0.8129	0.1323	0.0038	2.9777


Model Performance Comparison (RMSE)
RMSE (Lakhs)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Linear Regression
Ridge Regression
Decision Tree
Random Forest
Model


Residual Distribution (Random Forest)
Frequency
160
140
120
100
80
60
40
20
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Log Price)

[ ]: