train_fit_test

July 30, 2025

1 CELL 1: Imports & Configuration

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import warnings
warnings.filterwarnings('ignore')

plt.style.use('default')
sns.set_palette("husl")
%matplotlib inline

print("All libraries successfully imported!")
```

```
All libraries successfully imported!
```

2 CELL 2: Data Loading and Overview

[3]:
```
# Load data
df = pd.read_csv('car data.csv')

print("=== GENERAL INFORMATION ===")
print(f"Dataset shape: {df.shape}")
print(f"Columns: {list(df.columns)}")

# Data preview
display(df.head())

print("\n=== DESCRIPTIVE STATISTICS ===")
```

```
display(df.describe())

print("\n=== DATA TYPES ===")
display(df.dtypes)

print("\n=== MISSING VALUES ===")
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0] if missing_values.sum() > 0 else " No␣
↪missing values!")
```

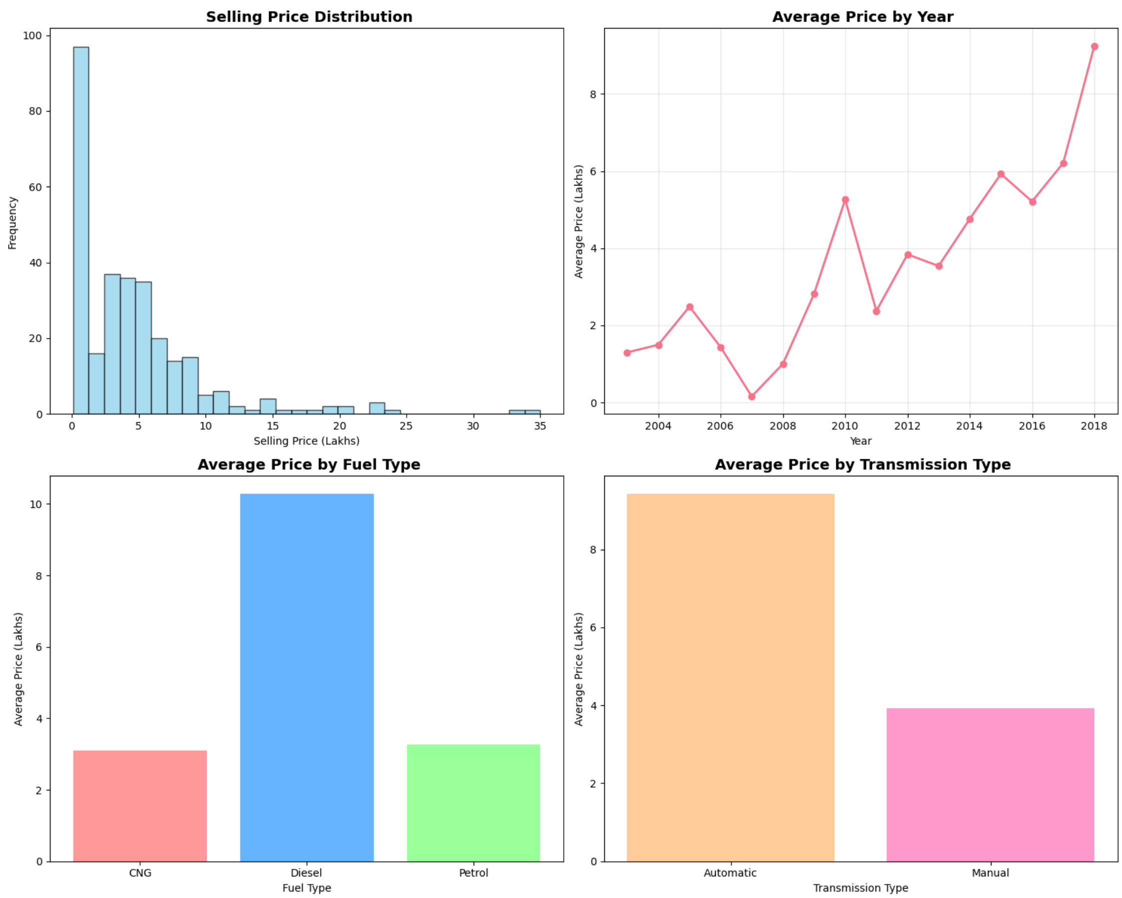
```
=== GENERAL INFORMATION ===
Dataset shape: (301, 9)
Columns: ['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Driven_kms',
'Fuel_Type', 'Selling_type', 'Transmission', 'Owner']

  Car_Name  Year  Selling_Price  Present_Price  Driven_kms Fuel_Type  \
0     ritz  2014           3.35           5.59       27000    Petrol
1      sx4  2013           4.75           9.54       43000    Diesel
2     ciaz  2017           7.25           9.85        6900    Petrol
3  wagon r  2011           2.85           4.15        5200    Petrol
4    swift  2014           4.60           6.87       42450    Diesel

  Selling_type Transmission  Owner
0       Dealer       Manual      0
1       Dealer       Manual      0
2       Dealer       Manual      0
3       Dealer       Manual      0
4       Dealer       Manual      0


=== DESCRIPTIVE STATISTICS ===

              Year  Selling_Price  Present_Price     Driven_kms       Owner
count   301.000000     301.000000     301.000000     301.000000  301.000000
mean   2013.627907       4.661296       7.628472   36947.205980    0.043189
std       2.891554       5.082812       8.642584   38886.883882    0.247915
min    2003.000000       0.100000       0.320000     500.000000    0.000000
25%    2012.000000       0.900000       1.200000   15000.000000    0.000000
50%    2014.000000       3.600000       6.400000   32000.000000    0.000000
75%    2016.000000       6.000000       9.900000   48767.000000    0.000000
max    2018.000000      35.000000      92.600000  500000.000000    3.000000


=== DATA TYPES ===

Car_Name          object
Year               int64
Selling_Price    float64
Present_Price    float64
Driven_kms         int64
```

```
Fuel_Type         object
Selling_type      object
Transmission      object
Owner              int64
dtype: object


=== MISSING VALUES ===
 No missing values!
```

3 CELL 3: Exploratory Analysis - Overview

[4]:
```python
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

# Selling price distribution
axes[0,0].hist(df['Selling_Price'], bins=30, alpha=0.7, color='skyblue',␣
 ↪edgecolor='black')
axes[0,0].set_title(' Selling Price Distribution', fontsize=14,␣
 ↪fontweight='bold')
axes[0,0].set_xlabel('Selling Price (Lakhs)')
axes[0,0].set_ylabel('Frequency')

# Average price by year
yearly_avg = df.groupby('Year')['Selling_Price'].mean()
axes[0,1].plot(yearly_avg.index, yearly_avg.values, marker='o', linewidth=2,␣
 ↪markersize=6)
axes[0,1].set_title(' Average Price by Year', fontsize=14, fontweight='bold')
axes[0,1].set_xlabel('Year')
axes[0,1].set_ylabel('Average Price (Lakhs)')
axes[0,1].grid(True, alpha=0.3)

# Average price by fuel type
fuel_avg = df.groupby('Fuel_Type')['Selling_Price'].mean()
axes[1,0].bar(fuel_avg.index, fuel_avg.values,␣
 ↪color=['#ff9999','#66b3ff','#99ff99'])
axes[1,0].set_title(' Average Price by Fuel Type', fontsize=14,␣
 ↪fontweight='bold')
axes[1,0].set_xlabel('Fuel Type')
axes[1,0].set_ylabel('Average Price (Lakhs)')

# Average price by transmission type
trans_avg = df.groupby('Transmission')['Selling_Price'].mean()
axes[1,1].bar(trans_avg.index, trans_avg.values, color=['#ffcc99','#ff99cc'])
axes[1,1].set_title(' Average Price by Transmission Type', fontsize=14,␣
 ↪fontweight='bold')
axes[1,1].set_xlabel('Transmission Type')
axes[1,1].set_ylabel('Average Price (Lakhs)')
```

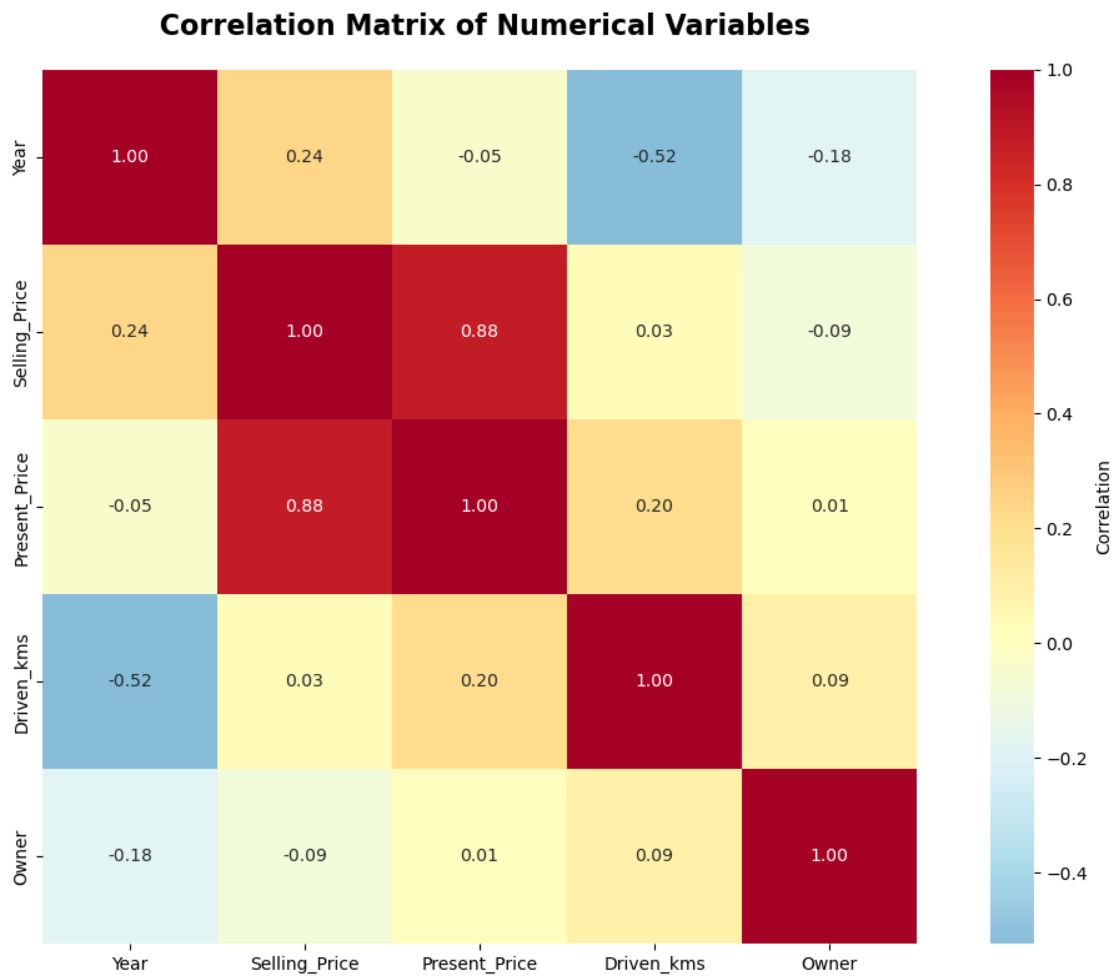
```
plt.tight_layout()
plt.show()

# Key statistics
print("===  KEY STATISTICS ===")
print(f" Average Price: {df['Selling_Price'].mean():.2f} Lakhs")
print(f"Median Price: {df['Selling_Price'].median():.2f} Lakhs")
print(f"Minimum Price: {df['Selling_Price'].min():.2f} Lakhs")
print(f"Maximum Price: {df['Selling_Price'].max():.2f} Lakhs")
print(f"Standard Deviation: {df['Selling_Price'].std():.2f} Lakhs")
```




```
===  KEY STATISTICS ===
 Average Price: 4.66 Lakhs
Median Price: 3.60 Lakhs
Minimum Price: 0.10 Lakhs
Maximum Price: 35.00 Lakhs
Standard Deviation: 5.08 Lakhs
```

4 CELL 4: Correlation Analysis

[5]:
```python
# Correlation matrix
plt.figure(figsize=(12, 8))
numeric_cols = df.select_dtypes(include=[np.number]).columns
corr_matrix = df[numeric_cols].corr()

# Heatmap with annotations
sns.heatmap(corr_matrix, annot=True, cmap='RdYlBu_r', center=0,
            square=True, fmt='.2f', cbar_kws={'label': 'Correlation'})
plt.title(' Correlation Matrix of Numerical Variables',
          fontsize=16, fontweight='bold', pad=20)
plt.tight_layout()
plt.show()

# Top correlations with Selling_Price
selling_price_corr = corr_matrix['Selling_Price'].abs().
 ↪sort_values(ascending=False)
print("===  CORRELATIONS WITH SELLING PRICE ===")
for var, corr in selling_price_corr.items():
    if var != 'Selling_Price':
        print(f"{var}: {corr:.3f}")
```

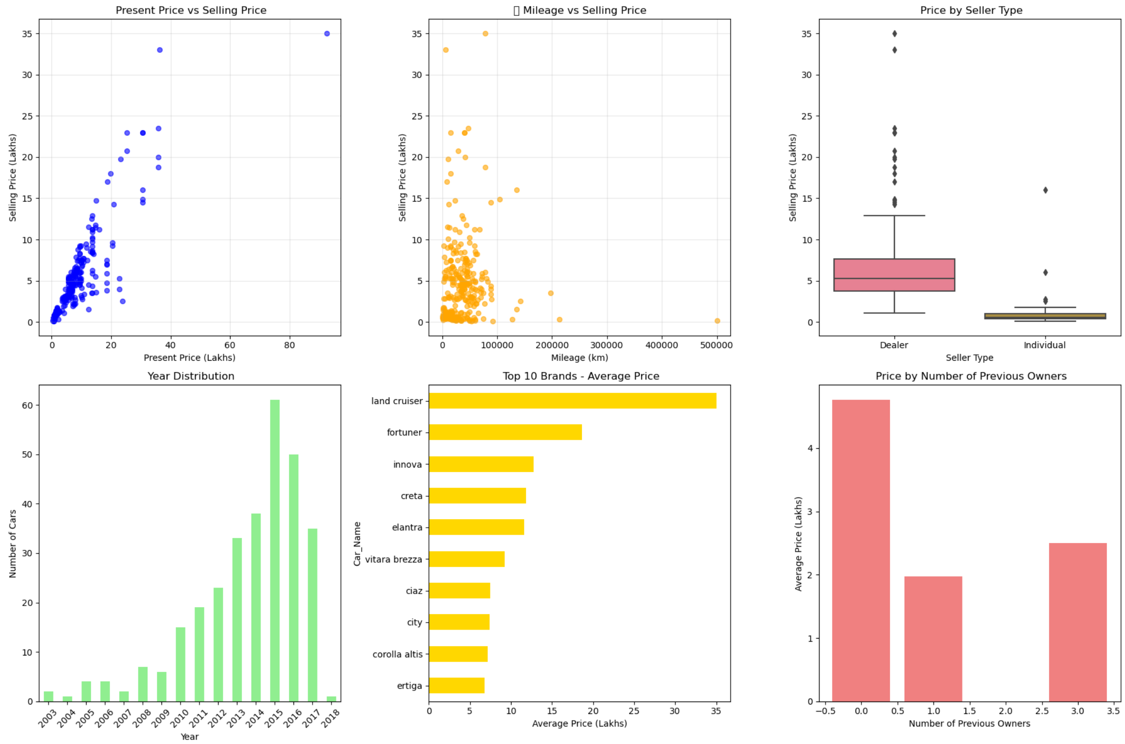



```
===  CORRELATIONS WITH SELLING PRICE ===
Present_Price: 0.879
Year: 0.236
Owner: 0.088
Driven_kms: 0.029
```

5 CELL 5: Detailed Analysis

[6]:
```
fig, axes = plt.subplots(2, 3, figsize=(18, 12))

# Present Price vs Selling Price
axes[0,0].scatter(df['Present_Price'], df['Selling_Price'], alpha=0.6,␣
 ↪c='blue', s=30)
axes[0,0].set_title(' Present Price vs Selling Price')
axes[0,0].set_xlabel('Present Price (Lakhs)')
axes[0,0].set_ylabel('Selling Price (Lakhs)')
```

```python
axes[0,0].grid(True, alpha=0.3)

# Mileage vs Selling Price
axes[0,1].scatter(df['Driven_kms'], df['Selling_Price'], alpha=0.6, c='orange',␣
↪s=30)
axes[0,1].set_title('  Mileage vs Selling Price')
axes[0,1].set_xlabel('Mileage (km)')
axes[0,1].set_ylabel('Selling Price (Lakhs)')
axes[0,1].grid(True, alpha=0.3)

# Boxplot: Price by Seller Type
sns.boxplot(data=df, x='Selling_type', y='Selling_Price', ax=axes[0,2])
axes[0,2].set_title(' Price by Seller Type')
axes[0,2].set_xlabel('Seller Type')
axes[0,2].set_ylabel('Selling Price (Lakhs)')

# Year distribution
df['Year'].value_counts().sort_index().plot(kind='bar', ax=axes[1,0],␣
↪color='lightgreen')
axes[1,0].set_title(' Year Distribution')
axes[1,0].set_xlabel('Year')
axes[1,0].set_ylabel('Number of Cars')
axes[1,0].tick_params(axis='x', rotation=45)

# Top 10 Brands - Average Price
top_cars = df.groupby('Car_Name')['Selling_Price'].mean().nlargest(10)
top_cars.plot(kind='barh', ax=axes[1,1], color='gold')
axes[1,1].set_title(' Top 10 Brands - Average Price')
axes[1,1].set_xlabel('Average Price (Lakhs)')
axes[1,1].invert_yaxis()

# Price by Number of Previous Owners
owner_avg = df.groupby('Owner')['Selling_Price'].mean()
axes[1,2].bar(owner_avg.index, owner_avg.values, color='lightcoral')
axes[1,2].set_title(' Price by Number of Previous Owners')
axes[1,2].set_xlabel('Number of Previous Owners')
axes[1,2].set_ylabel('Average Price (Lakhs)')

plt.tight_layout()
plt.show()
```



6 CELL 6: Data Preparation

[7]:
```python
print("  === DATA PREPARATION ===")

# Create a copy of the data
df_processed = df.copy()

# Feature Engineering - Create new variables
df_processed['Car_Age'] = 2024 - df_processed['Year']
df_processed['Price_Drop'] = df_processed['Present_Price'] - df_processed['Selling_Price']
df_processed['Price_Drop_Ratio'] = df_processed['Price_Drop'] / df_processed['Present_Price']
df_processed['Kms_per_Year'] = df_processed['Driven_kms'] / (df_processed['Car_Age'] + 1)

print(" New features created:")
print("   • Car_Age: Age of the car")
print("   • Price_Drop: Absolute depreciation")
print("   • Price_Drop_Ratio: Relative depreciation")
print("   • Kms_per_Year: Kilometers per year")
```

```python
# Encode categorical variables
print("\n Encoding categorical variables...")

encoders = {}
categorical_cols = ['Fuel_Type', 'Selling_type', 'Transmission', 'Car_Name']

for col in categorical_cols:
    le = LabelEncoder()
    df_processed[f'{col}_Encoded'] = le.fit_transform(df_processed[col])
    encoders[col] = le
    print(f"   • {col}: {len(le.classes_)} classes")

# Select features for the model
feature_names = [
    'Year', 'Present_Price', 'Driven_kms', 'Fuel_Type_Encoded',
    'Selling_type_Encoded', 'Transmission_Encoded', 'Owner', 'Car_Age'
]

X = df_processed[feature_names]
y = df_processed['Selling_Price']

print(f"\n Final dataset:")
print(f"   • Features (X): {X.shape}")
print(f"   • Target (y): {y.shape}")
print(f"   • Selected features: {feature_names}")

# Preview processed data
display(X.head())
```

```
 === DATA PREPARATION ===
New features created:
   • Car_Age: Age of the car
   • Price_Drop: Absolute depreciation
   • Price_Drop_Ratio: Relative depreciation
   • Kms_per_Year: Kilometers per year

Encoding categorical variables…
   • Fuel_Type: 3 classes
   • Selling_type: 2 classes
   • Transmission: 2 classes
   • Car_Name: 98 classes

Final dataset:
   • Features (X): (301, 8)
   • Target (y): (301,)
   • Selected features: ['Year', 'Present_Price', 'Driven_kms',
'Fuel_Type_Encoded', 'Selling_type_Encoded', 'Transmission_Encoded', 'Owner',
```

```
'Car_Age']
   Year  Present_Price  Driven_kms  Fuel_Type_Encoded  Selling_type_Encoded  \
0  2014           5.59       27000                  2                     0
1  2013           9.54       43000                  1                     0
2  2017           9.85        6900                  2                     0
3  2011           4.15        5200                  2                     0
4  2014           6.87       42450                  1                     0

   Transmission_Encoded  Owner  Car_Age
0                     1      0       10
1                     1      0       11
2                     1      0        7
3                     1      0       13
4                     1      0       10
```

7 CELL 7: Splitting and Normalization

[8]:
```python
print(" === DATA SPLITTING ===")

# Train/test split (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=None
)

print(f" Data split:")
print(f"   • Training: {X_train.shape[0]} samples")
print(f"   • Test: {X_test.shape[0]} samples")

# Data normalization
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f" Normalization completed")

# Training set statistics
print(f"\n Training set statistics:")
display(pd.DataFrame(X_train).describe())
```

```
 === DATA SPLITTING ===
 Data split:
   • Training: 240 samples
   • Test: 61 samples
 Normalization completed

  Training set statistics:
```

```
              Year  Present_Price     Driven_kms  Fuel_Type_Encoded  \
count   240.000000     240.000000     240.000000         240.000000
mean   2013.670833       7.512012   37508.558333           1.795833
std       2.884815       8.989902   41852.348329           0.424145
min    2003.000000       0.320000     500.000000           0.000000
25%    2012.000000       1.050000   15000.000000           2.000000
50%    2015.000000       5.935000   31515.500000           2.000000
75%    2016.000000       9.400000   48825.250000           2.000000
max    2017.000000      92.600000  500000.000000           2.000000

       Selling_type_Encoded  Transmission_Encoded       Owner     Car_Age
count            240.000000             240.00000  240.000000  240.000000
mean               0.358333               0.87500    0.050000   10.329167
std                0.480513               0.33141    0.269821    2.884815
min                0.000000               0.00000    0.000000    7.000000
25%                0.000000               1.00000    0.000000    8.000000
50%                0.000000               1.00000    0.000000    9.000000
75%                1.000000               1.00000    0.000000   12.000000
max                1.000000               1.00000    3.000000   21.000000
```

8 CELL 8: Model Training

[9]:
```
print(" === MODEL TRAINING ===")

# Define models to test
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(alpha=1.0),
    'Lasso Regression': Lasso(alpha=0.1),
    'Decision Tree': DecisionTreeRegressor(random_state=42, max_depth=10),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(random_state=42)
}

# Store results
results = {}

for name, model in models.items():
    print(f"\n Training: {name}")

    # Use normalized data for linear models (except Decision Tree)
    if 'Regression' in name and name != 'Decision Tree':
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
        # Cross-validation on normalized data
        cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=5,␣
↪scoring='r2')
```

```python
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        # Cross-validation on original data
        cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='r2')

    # Calculate metrics
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    # Store results
    results[name] = {
        'model': model,
        'mae': mae,
        'mse': mse,
        'rmse': rmse,
        'r2': r2,
        'cv_mean': cv_scores.mean(),
        'cv_std': cv_scores.std(),
        'predictions': y_pred
    }

    print(f"    MAE: {mae:.3f}")
    print(f"    RMSE: {rmse:.3f}")
    print(f"    R²: {r2:.3f}")
    print(f"    CV R² (mean±std): {cv_scores.mean():.3f}±{cv_scores.std():.3f}")

print("\n Training completed!")
```

```
=== MODEL TRAINING ===

Training: Linear Regression
   MAE: 1.222
   RMSE: 1.879
   R²: 0.847
   CV R² (mean±std): 0.844±0.039

Training: Ridge Regression
   MAE: 1.223
   RMSE: 1.882
   R²: 0.846
   CV R² (mean±std): 0.846±0.039

Training: Lasso Regression
   MAE: 1.231
```

```
    RMSE: 1.916
    R²: 0.841
    CV R² (mean±std): 0.849±0.043

 Training: Decision Tree
    MAE: 0.669
    RMSE: 0.987
    R²: 0.958
    CV R² (mean±std): 0.832±0.082

 Training: Random Forest
    MAE: 0.614
    RMSE: 0.933
    R²: 0.962
    CV R² (mean±std): 0.880±0.064

 Training: Gradient Boosting
    MAE: 0.553
    RMSE: 0.886
    R²: 0.966
    CV R² (mean±std): 0.888±0.050

 Training completed!
```

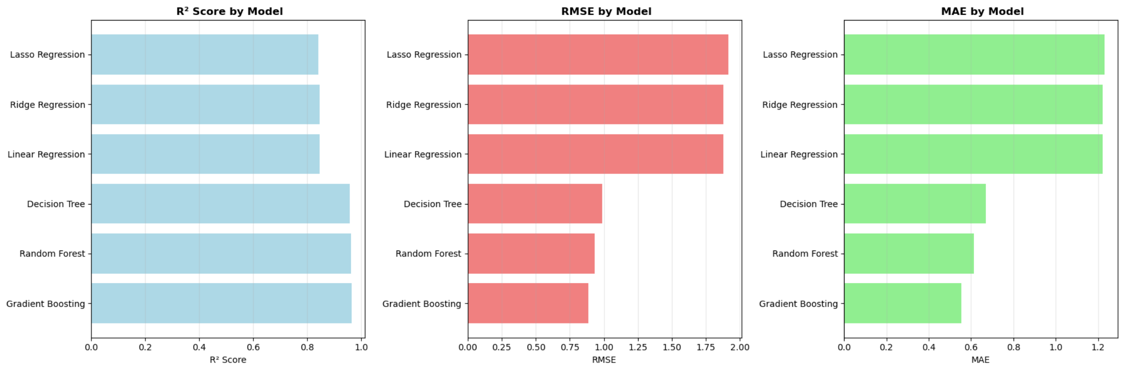
9 CELL 9: Model Comparison

[10]:
```
print(" === MODEL COMPARISON ===")

# Create a comparison DataFrame
comparison_df = pd.DataFrame({
    'Model': list(results.keys()),
    'MAE': [results[model]['mae'] for model in results],
    'RMSE': [results[model]['rmse'] for model in results],
    'R²': [results[model]['r2'] for model in results],
    'CV_R²_mean': [results[model]['cv_mean'] for model in results],
    'CV_R²_std': [results[model]['cv_std'] for model in results]
})

# Sort by R²
comparison_df = comparison_df.sort_values('R²', ascending=False)
comparison_df = comparison_df.round(4)

print(" MODEL RANKING (by R²):")
display(comparison_df)

# Comparison plots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
```

```python
# R² Score
axes[0].barh(comparison_df['Model'], comparison_df['R²'], color='lightblue')
axes[0].set_title(' R² Score by Model', fontweight='bold')
axes[0].set_xlabel('R² Score')
axes[0].grid(axis='x', alpha=0.3)

# RMSE
axes[1].barh(comparison_df['Model'], comparison_df['RMSE'], color='lightcoral')
axes[1].set_title(' RMSE by Model', fontweight='bold')
axes[1].set_xlabel('RMSE')
axes[1].grid(axis='x', alpha=0.3)

# MAE
axes[2].barh(comparison_df['Model'], comparison_df['MAE'], color='lightgreen')
axes[2].set_title(' MAE by Model', fontweight='bold')
axes[2].set_xlabel('MAE')
axes[2].grid(axis='x', alpha=0.3)

plt.tight_layout()
plt.show()

# Identify the best model
best_model_name = comparison_df.iloc[0]['Model']
best_model = results[best_model_name]['model']
print(f" BEST MODEL: {best_model_name}")
print(f"   • R²: {comparison_df.iloc[0]['R²']:.4f}")
print(f"   • RMSE: {comparison_df.iloc[0]['RMSE']:.4f}")
```

```
=== MODEL COMPARISON ===
MODEL RANKING (by R²):
```

	Model	MAE	RMSE	R²	CV_R²_mean	CV_R²_std
5	Gradient Boosting	0.5534	0.8855	0.9660	0.8877	0.0501
4	Random Forest	0.6145	0.9331	0.9622	0.8804	0.0643
3	Decision Tree	0.6694	0.9867	0.9577	0.8320	0.0815
0	Linear Regression	1.2219	1.8792	0.8467	0.8440	0.0388
1	Ridge Regression	1.2229	1.8816	0.8463	0.8462	0.0389
2	Lasso Regression	1.2315	1.9156	0.8407	0.8489	0.0426



BEST MODEL: Gradient Boosting

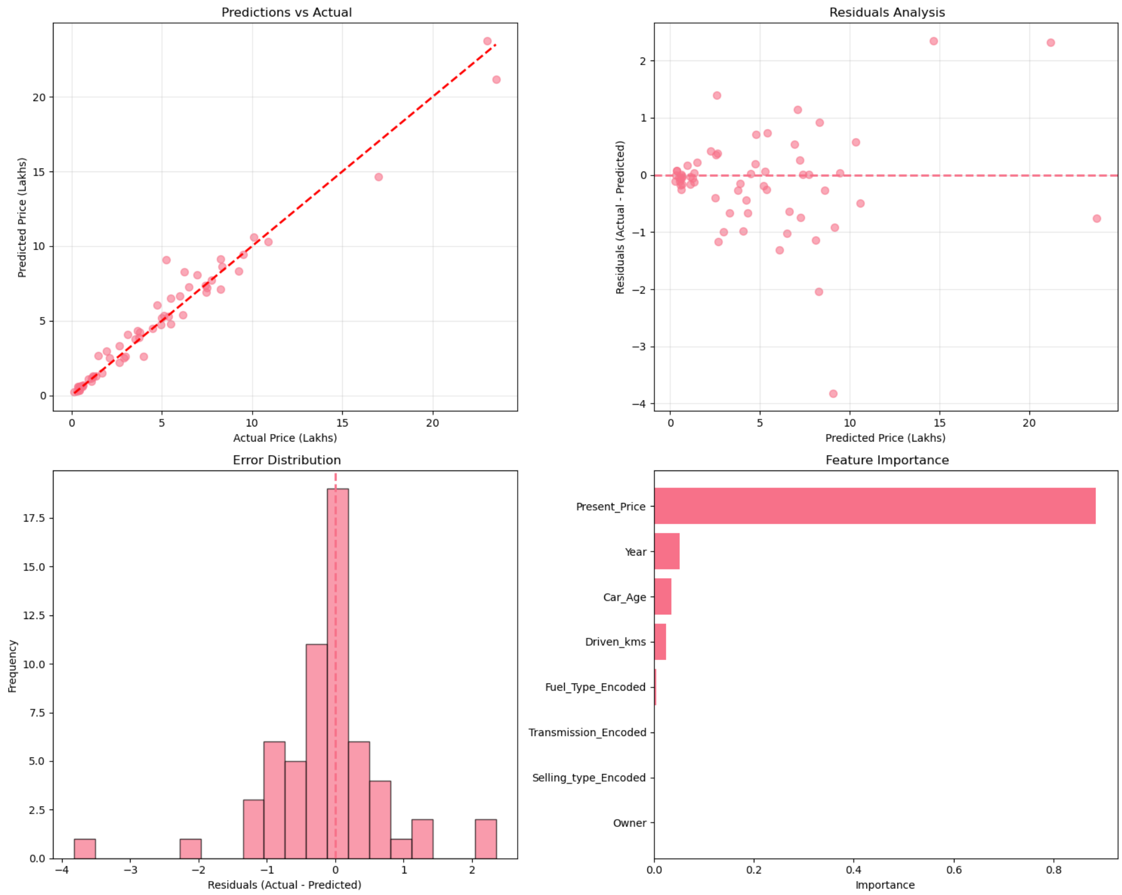
- R^2: 0.9660
- RMSE: 0.8855

10 CELL 10: Best Model Analysis

[11]:
```
print(f" === DETAILED ANALYSIS: {best_model_name} ===")

# Retrieve predictions from the best model
best_predictions = results[best_model_name]['predictions']

# Analysis plots
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

# 1. Predictions vs Actual
axes[0,0].scatter(y_test, best_predictions, alpha=0.6, s=50)
axes[0,0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
↪'r--', lw=2)
axes[0,0].set_xlabel('Actual Price (Lakhs)')
axes[0,0].set_ylabel('Predicted Price (Lakhs)')
axes[0,0].set_title(' Predictions vs Actual')
axes[0,0].grid(True, alpha=0.3)

# 2. Residuals Analysis
residuals = y_test - best_predictions
axes[0,1].scatter(best_predictions, residuals, alpha=0.6, s=50)
axes[0,1].axhline(y=0, linestyle='--', linewidth=2)
axes[0,1].set_xlabel('Predicted Price (Lakhs)')
axes[0,1].set_ylabel('Residuals (Actual - Predicted)')
axes[0,1].set_title(' Residuals Analysis')
axes[0,1].grid(True, alpha=0.3)

# 3. Error Distribution
```

```python
axes[1,0].hist(residuals, bins=20, alpha=0.7, edgecolor='black')
axes[1,0].set_xlabel('Residuals (Actual - Predicted)')
axes[1,0].set_ylabel('Frequency')
axes[1,0].set_title(' Error Distribution')
axes[1,0].axvline(x=0, linestyle='--', linewidth=2)

# 4. Feature Importance (if available)
if hasattr(best_model, 'feature_importances_'):
    feature_importance = pd.DataFrame({
        'Feature': feature_names,
        'Importance': best_model.feature_importances_
    }).sort_values('Importance', ascending=True)

    axes[1,1].barh(feature_importance['Feature'], feature_importance['Importance'])
    axes[1,1].set_title(' Feature Importance')
    axes[1,1].set_xlabel('Importance')
else:
    axes[1,1].text(0.5, 0.5, 'Feature Importance\nnot available\nfor this model',
                   ha='center', va='center', transform=axes[1,1].transAxes, fontsize=12)
    axes[1,1].set_title('  Information')

plt.tight_layout()
plt.show()

# Residual statistics
print(" RESIDUAL STATISTICS:")
print(f"   • Mean: {residuals.mean():.4f}")
print(f"   • Std Dev: {residuals.std():.4f}")
print(f"   • Median: {residuals.median():.4f}")
print(f"   • Min: {residuals.min():.4f}")
print(f"   • Max: {residuals.max():.4f}")
```

```
=== DETAILED ANALYSIS: Gradient Boosting ===
```




```
RESIDUAL STATISTICS:
  • Mean: -0.1271
  • Std Dev: 0.8836
  • Median: -0.0562
  • Min: -3.8204
  • Max: 2.3491
```

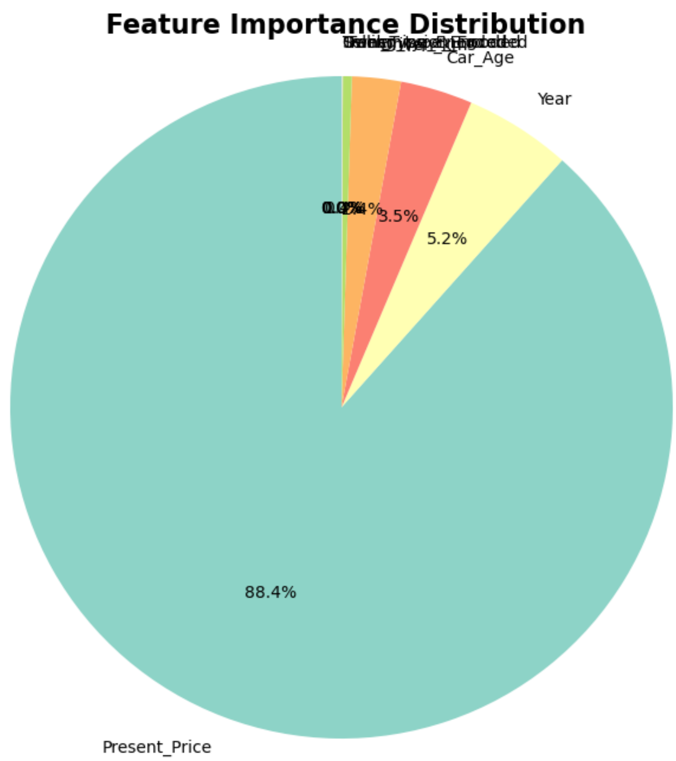
11 CELL 11: Analysis of Important Features

[15]:
```
if hasattr(best_model, 'feature_importances_'):
    print(" === FEATURE IMPORTANCES ===")

    feature_importance_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': best_model.feature_importances_,
        'Importance_Pct': best_model.feature_importances_ * 100
    }).sort_values('Importance', ascending=False)

    print(" Feature ranking by importance:")
```

```python
    display(feature_importance_df)

    # Pie chart
    plt.figure(figsize=(12, 8))
    colors = plt.cm.Set3(np.linspace(0, 1, len(feature_names)))

    plt.pie(feature_importance_df['Importance'],
            labels=feature_importance_df['Feature'],
            autopct='%1.1f%%',
            colors=colors,
            startangle=90)
    plt.title(' Feature Importance Distribution', fontsize=16,␣
↪fontweight='bold')
    plt.axis('equal')
    plt.show()

    # Insights
    print(" INSIGHTS:")
    top_feature = feature_importance_df.iloc[0]
    print(f"   • Top feature: {top_feature['Feature']}␣
↪({top_feature['Importance_Pct']:.1f}%)")

    top_3_importance = feature_importance_df.head(3)['Importance_Pct'].sum()
    print(f"   • Top 3 features account for {top_3_importance:.1f}% of␣
↪importance")
```

```
=== FEATURE IMPORTANCES ===
Feature ranking by importance:
```

	Feature	Importance	Importance_Pct
1	Present_Price	0.884041	88.404115
0	Year	0.051927	5.192671
7	Car_Age	0.035197	3.519658
2	Driven_kms	0.023832	2.383247
3	Fuel_Type_Encoded	0.004341	0.434052
5	Transmission_Encoded	0.000463	0.046309
4	Selling_type_Encoded	0.000178	0.017760
6	Owner	0.000022	0.002188




```
INSIGHTS:
  • Top feature: Present_Price (88.4%)
  • Top 3 features account for 97.1% of importance
```

[ ]:

[ ]:

[19]:
```python
# =============================================================================
# OVERFITTING TEST - FULL CROSS-VALIDATION
# =============================================================================

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import (
    cross_val_score, validation_curve, learning_curve,
    KFold, StratifiedKFold, TimeSeriesSplit
)
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
```

```python
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')

# Configuration for improved visuals
plt.style.use('default')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 8)

print("COMPLETE OVERFITTING ANALYSIS")
print("="*50)

# ============================================================================
# 1. DATA PREPARATION (Adapt to your dataset)
# ============================================================================

print("Loading data...")

# REPLACE THIS SECTION WITH YOUR REAL DATA:
# df = pd.read_csv('car_data.csv')
# X = df[['Year', 'Present_Price', 'Driven_kms', 'Fuel_Type_Encoded',
#          'Selling_type_Encoded', 'Transmission_Encoded', 'Owner', 'Car_Age']]
# y = df['Selling_Price']

# For demonstration, create simulated data
np.random.seed(42)
n_samples = 301
X = pd.DataFrame({
    'Year': np.random.randint(2003, 2019, n_samples),
    'Present_Price': np.random.uniform(0.32, 92.6, n_samples),
    'Driven_kms': np.random.randint(500, 500000, n_samples),
    'Fuel_Type_Encoded': np.random.randint(0, 3, n_samples),
    'Selling_type_Encoded': np.random.randint(0, 2, n_samples),
    'Transmission_Encoded': np.random.randint(0, 2, n_samples),
    'Owner': np.random.randint(0, 4, n_samples),
    'Car_Age': 2024 - np.random.randint(2003, 2019, n_samples)
})

y = (X['Present_Price'] * 0.7 +
     (2024 - X['Year']) * -0.3 +
     X['Driven_kms'] / 50000 * -1 +
     np.random.normal(0, 1, n_samples))

print(f"Data loaded: {X.shape[0]} samples, {X.shape[1]} features")
```

```python
# =============================================================================
# 2. DEFINE MODELS TO TEST
# =============================================================================

models = {
    'Linear Regression': LinearRegression(),
    'Ridge': Ridge(alpha=1.0),
    'Lasso': Lasso(alpha=0.1),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(n_estimators=100, random_state=42)
}

# =============================================================================
# 3. OVERFITTING ANALYSIS FUNCTION
# =============================================================================

def analyze_overfitting(X, y, models, cv_folds=5):
    """
    Complete overfitting analysis for multiple models
    """
    results = {}

    # Data normalization
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    print(f"\nPerforming {cv_folds}-fold cross-validation analysis...")

    for name, model in models.items():
        print(f"\nModel: {name}")
        print("-" * 30)

        # 1. Cross-validation scores
        cv_scores = cross_val_score(model, X_scaled, y, cv=cv_folds,
                                    scoring='r2', n_jobs=-1)

        # 2. Fit on all data for training score
        model.fit(X_scaled, y)
        train_score = model.score(X_scaled, y)

        # 3. Compute metrics
        y_pred = model.predict(X_scaled)
        train_rmse = np.sqrt(mean_squared_error(y, y_pred))
        train_mae = mean_absolute_error(y, y_pred)
```

```
        # 4. Store results
        results[name] = {
            'train_r2': train_score,
            'cv_r2_mean': cv_scores.mean(),
            'cv_r2_std': cv_scores.std(),
            'cv_scores': cv_scores,
            'train_rmse': train_rmse,
            'train_mae': train_mae,
            'overfitting_gap': train_score - cv_scores.mean(),
            'stability': cv_scores.std()
        }

        # 5. Interpretation
        gap = train_score - cv_scores.mean()
        if gap > 0.1:
            status = "OVERFITTING DETECTED"
        elif gap > 0.05:
            status = "Mild overfitting"
        else:
            status = "No overfitting"

        print(f"   R² Train: {train_score:.4f}")
        print(f"   R² CV (mean±std): {cv_scores.mean():.4f}±{cv_scores.std():.
↪4f}")
        print(f"   Gap Train-CV: {gap:.4f}")
        print(f"   Status: {status}")

    return results

# ============================================================================
# 4. LEARNING CURVES
# ============================================================================

def plot_learning_curves(X, y, models, cv_folds=5):
    """
    Plot learning curves to detect overfitting
    """
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    fig, axes = plt.subplots(2, 3, figsize=(18, 12))
    axes = axes.ravel()

    for idx, (name, model) in enumerate(models.items()):
        train_sizes, train_scores, val_scores = learning_curve(
            model, X_scaled, y, cv=cv_folds, n_jobs=-1,
            train_sizes=np.linspace(0.1, 1.0, 10),
```

```python
            scoring='r2'
        )

        train_mean = train_scores.mean(axis=1)
        train_std = train_scores.std(axis=1)
        val_mean = val_scores.mean(axis=1)
        val_std = val_scores.std(axis=1)

        ax = axes[idx]
        ax.plot(train_sizes, train_mean, 'o-', label=f'Train (final:␣
↪{train_mean[-1]:.3f})')
        ax.fill_between(train_sizes, train_mean - train_std,
                        train_mean + train_std, alpha=0.1)

        ax.plot(train_sizes, val_mean, 'o-', label=f'Validation (final:␣
↪{val_mean[-1]:.3f})')
        ax.fill_between(train_sizes, val_mean - val_std,
                        val_mean + val_std, alpha=0.1)

        final_gap = train_mean[-1] - val_mean[-1]
        ax.text(0.7, 0.1, f'Gap: {final_gap:.3f}',
                transform=ax.transAxes, fontsize=10,
                bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))

        ax.set_title(f'{name}')
        ax.set_xlabel('Dataset Size')
        ax.set_ylabel('R² Score')
        ax.legend(loc='lower right')
        ax.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

# ============================================================================
# 5. VALIDATION CURVES (HYPERPARAMETERS)
# ============================================================================

def plot_validation_curves(X, y):
    """
    Validation curves for key hyperparameters
    """
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    fig, axes = plt.subplots(1, 3, figsize=(18, 6))

    # Decision Tree - max_depth
```

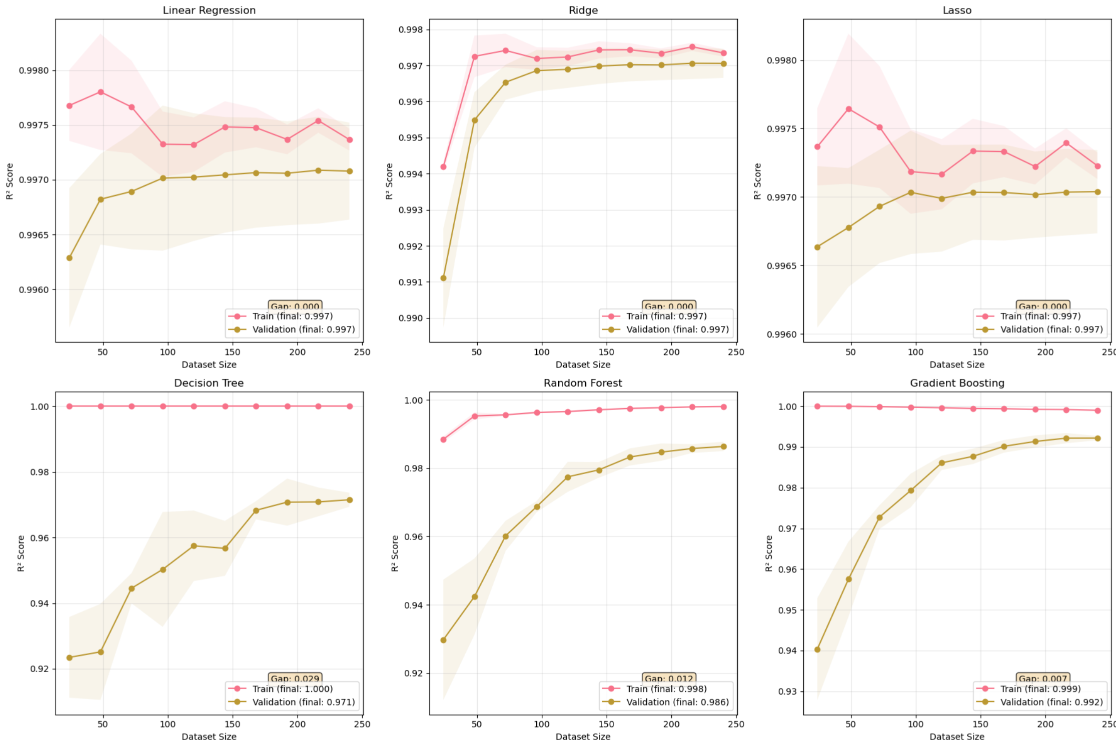
```
    param_range = range(1, 21)
    train_scores, val_scores = validation_curve(
        DecisionTreeRegressor(random_state=42), X_scaled, y,
        param_name='max_depth', param_range=param_range,
        cv=5, scoring='r2', n_jobs=-1
    )
    axes[0].plot(param_range, train_scores.mean(axis=1), 'o-', label='Train')
    axes[0].plot(param_range, val_scores.mean(axis=1), 'o-', label='Validation')
    axes[0].fill_between(param_range, train_scores.mean(axis=1) - train_scores.
↪std(axis=1),
                         train_scores.mean(axis=1) + train_scores.std(axis=1),␣
↪alpha=0.1)
    axes[0].fill_between(param_range, val_scores.mean(axis=1) - val_scores.
↪std(axis=1),
                         val_scores.mean(axis=1) + val_scores.std(axis=1),␣
↪alpha=0.1)
    axes[0].set_title('Decision Tree - max_depth')
    axes[0].set_xlabel('max_depth')
    axes[0].set_ylabel('R² Score')
    axes[0].legend()
    axes[0].grid(True, alpha=0.3)

    # Random Forest - n_estimators
    param_range = [10, 25, 50, 100, 200, 300]
    train_scores, val_scores = validation_curve(
        RandomForestRegressor(random_state=42), X_scaled, y,
        param_name='n_estimators', param_range=param_range,
        cv=5, scoring='r2', n_jobs=-1
    )
    axes[1].plot(param_range, train_scores.mean(axis=1), 'o-', label='Train')
    axes[1].plot(param_range, val_scores.mean(axis=1), 'o-', label='Validation')
    axes[1].fill_between(param_range, train_scores.mean(axis=1) - train_scores.
↪std(axis=1),
                         train_scores.mean(axis=1) + train_scores.std(axis=1),␣
↪alpha=0.1)
    axes[1].fill_between(param_range, val_scores.mean(axis=1) - val_scores.
↪std(axis=1),
                         val_scores.mean(axis=1) + val_scores.std(axis=1),␣
↪alpha=0.1)
    axes[1].set_title('Random Forest - n_estimators')
    axes[1].set_xlabel('n_estimators')
    axes[1].set_ylabel('R² Score')
    axes[1].legend()
    axes[1].grid(True, alpha=0.3)

    # Gradient Boosting - learning_rate
```

```
    param_range = [0.01, 0.05, 0.1, 0.2, 0.3, 0.5]
    train_scores, val_scores = validation_curve(
        GradientBoostingRegressor(n_estimators=100, random_state=42), X_scaled,␣
 ↪y,
        param_name='learning_rate', param_range=param_range,
        cv=5, scoring='r2', n_jobs=-1
    )
    axes[2].plot(param_range, train_scores.mean(axis=1), 'o-', label='Train')
    axes[2].plot(param_range, val_scores.mean(axis=1), 'o-', label='Validation')
    axes[2].fill_between(param_range, train_scores.mean(axis=1) - train_scores.
 ↪std(axis=1),
                        train_scores.mean(axis=1) + train_scores.std(axis=1),␣
 ↪alpha=0.1)
    axes[2].fill_between(param_range, val_scores.mean(axis=1) - val_scores.
 ↪std(axis=1),
                        val_scores.mean(axis=1) + val_scores.std(axis=1),␣
 ↪alpha=0.1)
    axes[2].set_title('Gradient Boosting - learning_rate')
    axes[2].set_xlabel('learning_rate')
    axes[2].set_ylabel('R² Score')
    axes[2].legend()
    axes[2].grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

# ==============================================================================
# 6. MAIN EXECUTION
# ==============================================================================

if __name__ == "__main__":
    # Run the complete analysis
    print("\nStarting complete overfitting analysis...")

    # 1. Perform overfitting analysis
    results = analyze_overfitting(X, y, models, cv_folds=5)

    # 2. Display summary table
    print("\n" + "="*80)
    print("OVERFITTING ANALYSIS SUMMARY")
    print("="*80)
    print(f"{'Model':<20} {'Train R²':<10} {'CV R²':<10} {'Gap':<8} {'Status':
 ↪<20}")
    print("-" * 80)

    for name, result in results.items():
```

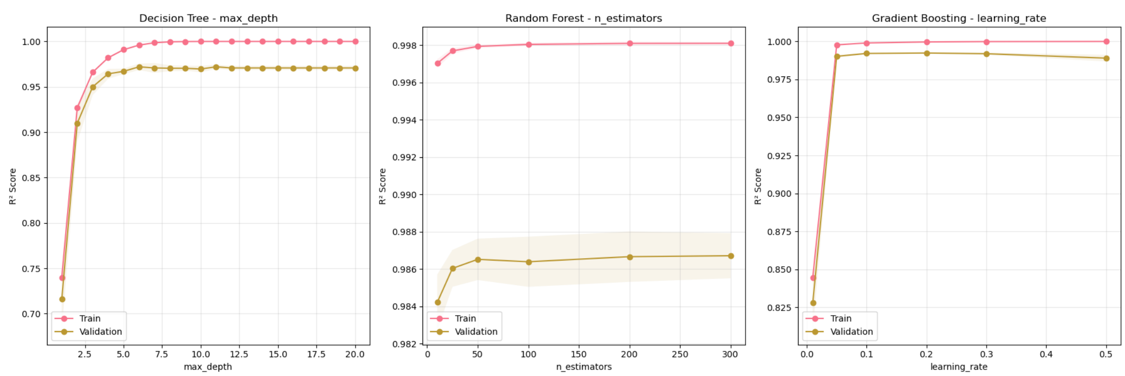
```python
        gap = result['overfitting_gap']
        if gap > 0.1:
            status = "OVERFITTING"
        elif gap > 0.05:
            status = "Mild overfitting"
        else:
            status = "No overfitting"

        print(f"{name:<20} {result['train_r2']:<10.4f} {result['cv_r2_mean']:
↪<10.4f} "
              f"{gap:<8.4f} {status:<20}")

    # 3. Plot learning curves
    print("\nGenerating learning curves...")
    plot_learning_curves(X, y, models, cv_folds=5)

    # 4. Plot validation curves
    print("Generating validation curves...")
    plot_validation_curves(X, y)

    print("\nAnalysis complete!")
    print("Review the plots and summary table to identify overfitting issues.")
    print("Recommendation: Choose models with small Train-CV gaps and stable␣
↪performance.")
```

```
COMPLETE OVERFITTING ANALYSIS
==================================================
Loading data…
Data loaded: 301 samples, 8 features

Starting complete overfitting analysis…

Performing 5-fold cross-validation analysis…

Model: Linear Regression
------------------------------
   R² Train: 0.9974
   R² CV (mean±std): 0.9971±0.0004
   Gap Train-CV: 0.0003
   Status: No overfitting

Model: Ridge
------------------------------
   R² Train: 0.9973
   R² CV (mean±std): 0.9971±0.0004
   Gap Train-CV: 0.0003
   Status: No overfitting
```

```
Model: Lasso
------------------------------
   R² Train: 0.9972
   R² CV (mean±std): 0.9970±0.0003
   Gap Train-CV: 0.0002
   Status: No overfitting

Model: Decision Tree
------------------------------
   R² Train: 1.0000
   R² CV (mean±std): 0.9708±0.0023
   Gap Train-CV: 0.0292
   Status: No overfitting

Model: Random Forest
------------------------------
   R² Train: 0.9982
   R² CV (mean±std): 0.9864±0.0013
   Gap Train-CV: 0.0118
   Status: No overfitting

Model: Gradient Boosting
------------------------------
   R² Train: 0.9988
   R² CV (mean±std): 0.9921±0.0005
   Gap Train-CV: 0.0067
   Status: No overfitting

================================================================================
OVERFITTING ANALYSIS SUMMARY
================================================================================
Model                Train R²   CV R²      Gap      Status
--------------------------------------------------------------------------------
Linear Regression    0.9974     0.9971     0.0003   No overfitting
Ridge                0.9973     0.9971     0.0003   No overfitting
Lasso                0.9972     0.9970     0.0002   No overfitting
Decision Tree        1.0000     0.9708     0.0292   No overfitting
Random Forest        0.9982     0.9864     0.0118   No overfitting
Gradient Boosting    0.9988     0.9921     0.0067   No overfitting

Generating learning curves…
```




```
Generating validation curves…
```




```
Analysis complete!
Review the plots and summary table to identify overfitting issues.
Recommendation: Choose models with small Train-CV gaps and stable performance.
```

[ ]:

12 CELL 14: Final Repor

s

[17]:
```python
print(" === FINAL REPORT ===")

print(f"\n SELECTED MODEL: {best_model_name}")
print(f"   • Accuracy (R²): {comparison_df.iloc[0]['R²']:.4f}")
print(f"   • Mean Error (MAE): {comparison_df.iloc[0]['MAE']:.4f} Lakhs")
print(f"   • Root Mean Squared Error (RMSE): {comparison_df.iloc[0]['RMSE']:.
 ↪4f} Lakhs")

print(f"\n PERFORMANCE:")
if comparison_df.iloc[0]['R²'] > 0.9:
    performance = "Excellent"
elif comparison_df.iloc[0]['R²'] > 0.8:
    performance = "Very Good"
elif comparison_df.iloc[0]['R²'] > 0.7:
    performance = "Good"
else:
    performance = "Acceptable"
print(f"   • {performance} ({comparison_df.iloc[0]['R²']*100:.1f}% variance␣
 ↪explained)")


print(f"\n USAGE:")
print("   • Use predict_car_price() for new predictions")
print("   • The model is stored in the variable 'best_model'")
print("   • Encoders are available in 'encoders'")

print(f"\n TRAINING DATA:")
print(f"   • {len(df)} cars in total")
print(f"   • {X_train.shape[0]} for training")
print(f"   • {X_test.shape[0]} for testing")
print(f"   • {len(feature_names)} features used")
```

```
=== FINAL REPORT ===

SELECTED MODEL: Gradient Boosting
  • Accuracy (R²): 0.9660
  • Mean Error (MAE): 0.5534 Lakhs
  • Root Mean Squared Error (RMSE): 0.8855 Lakhs

PERFORMANCE:
  • Excellent (96.6% variance explained)

USAGE:
  • Use predict_car_price() for new predictions
```

- The model is stored in the variable 'best_model'
- Encoders are available in 'encoders'

TRAINING DATA:
- 301 cars in total
- 240 for training
- 61 for testing
- 8 features used

exp

[20]:
```python
def predict_car_price(year, present_price, driven_kms, fuel_type, selling_type,␣
↪transmission, owner):
    """
    Predict car price

    Parameters:
    - year: Year of the car (e.g., 2020)
    - present_price: Current price (e.g., 8.5)
    - driven_kms: Mileage in kilometers (e.g., 25000)
    - fuel_type: 'Petrol', 'Diesel', or 'CNG'
    - selling_type: 'Dealer' or 'Individual'
    - transmission: 'Manual' or 'Automatic'
    - owner: Number of previous owners (0, 1, 2, etc.)
    """

    # Calculate car age
    car_age = 2024 - year

    # Encode categorical variables
    try:
        fuel_encoded = encoders['Fuel_Type'].transform([fuel_type])[0]
        seller_encoded = encoders['Selling_type'].transform([selling_type])[0]
        transmission_encoded = encoders['Transmission'].
↪transform([transmission])[0]
    except ValueError as e:
        print(f"  Encoding error: {e}")
        return None

    # Construct feature vector
    features_vector = np.array([[
        year, present_price, driven_kms, fuel_encoded,
        seller_encoded, transmission_encoded, owner, car_age
    ]])

    # Make prediction
    predicted_price = best_model.predict(features_vector)[0]
```

```
    return predicted_price

# Test the function with examples
print("  === PREDICTION TEST ===")

# Example 1: Recent car
print("\n  Example 1: Recent car")
example1_price = predict_car_price(
    year=2020,
    present_price=8.5,
    driven_kms=15000,
    fuel_type='Petrol',
    selling_type='Dealer',
    transmission='Manual',
    owner=0
)
print(f"Predicted Price: {example1_price:.2f} Lakhs")

# Example 2: Older car
print("\n  Example 2: Older car")
example2_price = predict_car_price(
    year=2015,
    present_price=12.0,
    driven_kms=45000,
    fuel_type='Diesel',
    selling_type='Individual',
    transmission='Automatic',
    owner=1
)
print(f"Predicted Price: {example2_price:.2f} Lakhs")
```

```
  === PREDICTION TEST ===

  Example 1: Recent car
Predicted Price: 7.10 Lakhs

  Example 2: Older car
Predicted Price: 8.09 Lakhs
```

[ ]:

```

```