SUMMER TRAINING

PROJECT REPORT

(Term June-July 2025)

Car Price Prediction Model

Submitted by

Name	Registration Number
Prakhar Purwar	**12323770**
Prashant Kumar	**12315454**
Jubin Mazumdar	**12316322**
Asmit Kushagra	**12317766**
Kajal Kumari	**12312143**

Course Code: PETV79

Under the Guidance of

Mahipal Singh Papola (UID: 32137)

School of Computer Science and Engineering

Certificate

This is to certify that the project report titled "Car Price Prediction using Machine Learning" is a record of original work carried out by Prakhar Purwar, Prashant Kumar, Jubin Mazumdar, Asmit Kushagra and Kajal Kumari during their summer internship as a part of their B.Tech (Computer Science and Engineering) curriculum at Lovely Professional University. The work has been completed under my guidance and supervision and is a part of their partial fulfillment of the degree.

Date: 13-July-2025

Supervisor
Mahipal Sir
Department of CSE, LPU

Acknowledgement

We would like to express our heartfelt gratitude to our project guide Mahipal Sir for his invaluable support, continuous guidance, and encouragement throughout the duration of this project. We would also like to thank the faculty and staff of the School of Computer Science and Engineering, LPU, for providing the necessary academic environment and technical resources.

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION

1.1 COMPANY PROFILE

This project was undertaken as part of the academic curriculum at the School of Computer Science and Engineering, Lovely Professional University (LPU), Phagwara, Punjab. LPU is a premier educational institution in India, recognized for its focus on industry-oriented education and research. The School of Computer Science and Engineering is equipped with modern facilities and fosters innovation in fields such as machine learning, data science, artificial intelligence, and software engineering. The department emphasizes practical training through projects, enabling students to apply theoretical knowledge to real-world problems.

1.2 OVERVIEW OF TRAINING DOMAIN

The training domain for this project is Machine Learning (ML) and Data Science, with a focus on regression modeling, data preprocessing, exploratory data analysis (EDA), and web application deployment. The project involved developing a predictive model to estimate used car prices, a regression task requiring skills in data cleaning, feature engineering, model training, and user interface development. The training provided hands-on experience with Python-based ML tools and deployment frameworks, preparing us for real-world applications in the automotive industry.

1.3 OBJECTIVE OF THE PROJECT

The objectives of the Car Price Prediction project are:

- To preprocess a dataset of used car listings to ensure high data quality.
- To engineer features that capture key factors influencing car prices, such as brand, year, and mileage.
- To develop and train a Random Forest regression model to accurately predict car prices.
- To deploy the model as an interactive Streamlit web application for user accessibility.
- To evaluate the model's performance using regression metrics (e.g., R^2, MSE) and analyze its limitations to propose future improvements.

CHAPTER 2: TRAINING OVERVIEW

2.1 TOOLS & TECHNOLOGIES USED

The project utilized the following tools and technologies:

- PYTHON: Core programming language for data processing, modeling, and deployment.
- SCIKIT-LEARN: For implementing the Random Forest regression model and computing evaluation metrics.
- PANDAS: For data manipulation, cleaning, and preprocessing.
- NUMPY: For numerical computations and array operations.
- MATPLOTLIB AND SEABORN: For creating visualizations such as scatter plots, box plots, and regression plots.
- STREAMLIT: For developing and deploying the interactive web application.
- JUPYTER NOTEBOOK: For exploratory data analysis, prototyping, and code development.

2.2 AREAS COVERED DURING TRAINING

The training covered the following key areas:

- DATA PREPROCESSING: Handling missing values, removing duplicates, and encoding categorical variables.
- EXPLORATORY DATA ANALYSIS: Performing univariate and bivariate analyses to identify patterns and correlations.
- FEATURE ENGINEERING: Extracting relevant features (e.g., car brand) and cleaning numerical data (e.g., mileage, engine).
- MODEL DEVELOPMENT: Training a Random Forest regression model and evaluating its performance.
- DEPLOYMENT: Building a user-friendly web application using Streamlit for real-time predictions.
- VISUALIZATION: Creating plots to visualize feature distributions and relationships.

2.3 DAILY WORK SUMMARY

The project was executed over five days, with the following milestones:

- DAY 1: Collected the dataset from the CarDekho platform and performed initial exploration to identify issues (e.g., missing values, duplicates).
- DAY 2: Preprocessed the dataset by dropping unnecessary columns (e.g., torque), handling missing values (221 rows), removing duplicates (1,189 rows), and engineering features (e.g., brand extraction).
- DAY 3: Conducted EDA, including univariate analysis (e.g., price distribution), bivariate analysis (e.g., year vs. price), and correlation matrix computation.
- DAY 4: Trained the Random Forest model, evaluated it using R^2 (0.9128) and MSE (63,232,848,857.42 INR^2), and developed the initial Streamlit app.
- DAY 5: Refined the model, added input validation to the Streamlit app, and finalized the project report.

CHAPTER 3: PROJECT DETAILS

3.1 TITLE OF THE PROJECT

Car Price Prediction Using Machine Learning

3.2 PROBLEM DEFINITION

Pricing used cars is a challenging task due to the influence of multiple factors, including car brand, year of manufacture, kilometers driven, fuel type, seller type, transmission, ownership status, mileage, engine capacity, and maximum power. Manual price estimation is time-consuming, subjective, and prone to errors. This project addresses the need for an automated, data-driven solution to predict used car prices accurately, benefiting buyers, sellers, and dealerships in the automotive market.

3.3 SCOPE AND OBJECTIVES

SCOPE:

- Develop a machine learning model to predict used car prices based on a dataset of 8,148 car listings.
- Ensure data quality through preprocessing and feature engineering.
- Deploy the model as a user-friendly web application for real-time predictions.
- Achieve high predictive accuracy (target $R^2 > 0.85$) and provide insights into key price drivers.

OBJECTIVES:

- Clean and preprocess the dataset to remove inconsistencies and ensure model compatibility.
- Engineer features to capture relevant information (e.g., car brand, numerical values from mileage).
- Train a Random Forest model to predict prices with high accuracy.
- Deploy the model via Streamlit for accessibility to non-technical users.
- Evaluate model performance and identify areas for improvement.

3.4 SYSTEM REQUIREMENTS

- HARDWARE: Standard laptop/desktop with at least 8 GB RAM and a 2 GHz processor.
- SOFTWARE:
    - Python 3.8 or higher
    - Libraries: scikit-learn, pandas, NumPy, Matplotlib, Seaborn, Streamlit
    - Jupyter Notebook for development and testing
- DATASET: "Car dekho - Car dekho.csv" with 8,148 observations and 12 features (reduced to 5,328 after preprocessing).
- OPERATING SYSTEM: Windows, macOS, or Linux.

3.5 DATA FLOW DIAGRAM

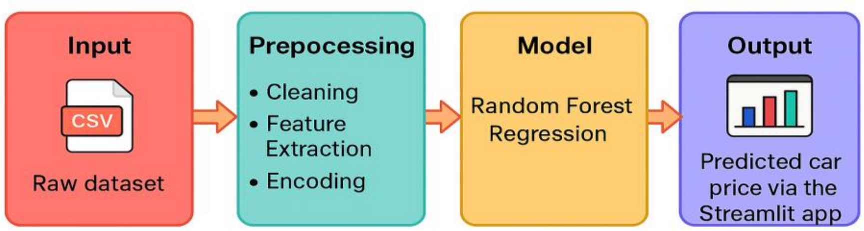


Figure 1: Data Flow Diagram

CHAPTER 4: IMPLEMENTATION

4.1 TOOLS USED

The implementation utilized:

- PYTHON LIBRARIES: pandas for data manipulation, NumPy for numerical operations, scikit-learn for model training and evaluation.
- VISUALIZATION TOOLS: Matplotlib and Seaborn for generating scatter plots, box plots, and regression plots.
- STREAMLIT: For building and deploying the interactive web application.
- JUPYTER NOTEBOOK: For prototyping and exploratory analysis.

4.2 METHODOLOGY

The project followed a structured machine learning pipeline:

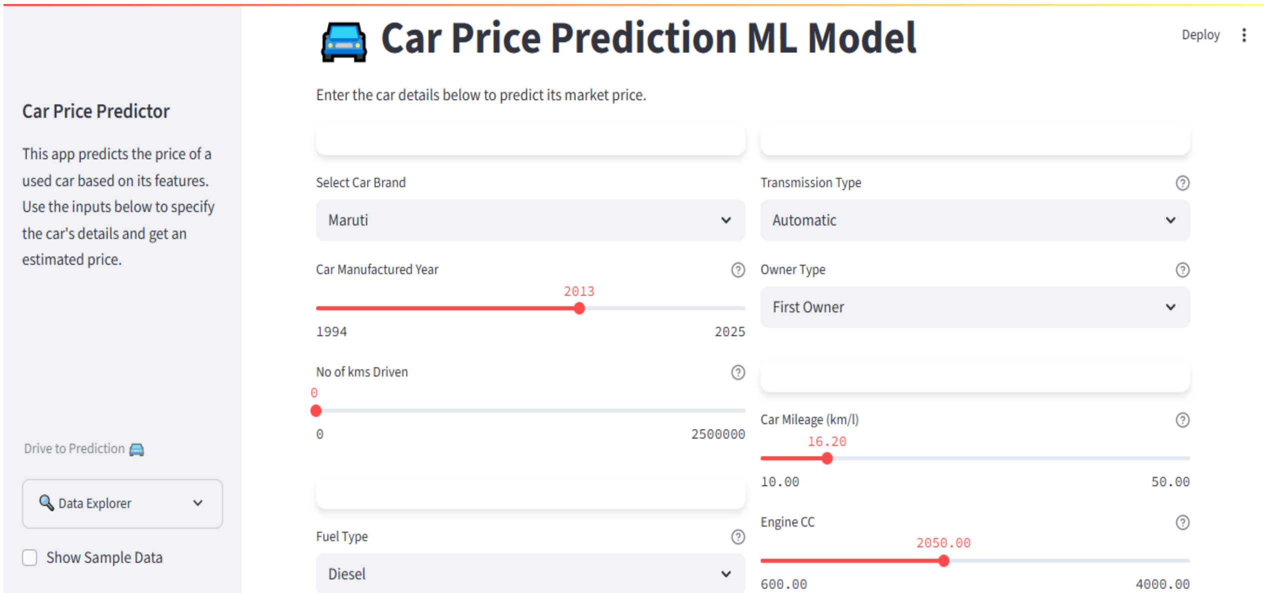
1. DATA COLLECTION: Sourced the dataset ("Car dekho - Car dekho.csv") with 8,148 observations and 12 features.
2. PREPROCESSING:
    - Dropped the torque column due to inconsistent data formats.
    - Removed 221 rows with missing values in mileage, engine, max_power, and seats.
    - Eliminated 1,189 duplicate rows, reducing the dataset to 6,738 observations (further cleaned to 5,328 for training/testing).
    - Extracted car brand from the Name column (e.g., "Maruti Alto" → "Maruti"), reducing dimensionality to 31 unique brands.
    - Encoded categorical features: fuel (Petrol=1, Diesel=2, etc.), seller_type (Individual=1, Dealer=2), transmission (Manual=1, Automatic=2), owner (First Owner=1, Second Owner=2, etc.).
    - Cleaned numerical features: mileage (e.g., "19.03 kmpl" → 19.03), engine (e.g., "999 CC" → 999.0).

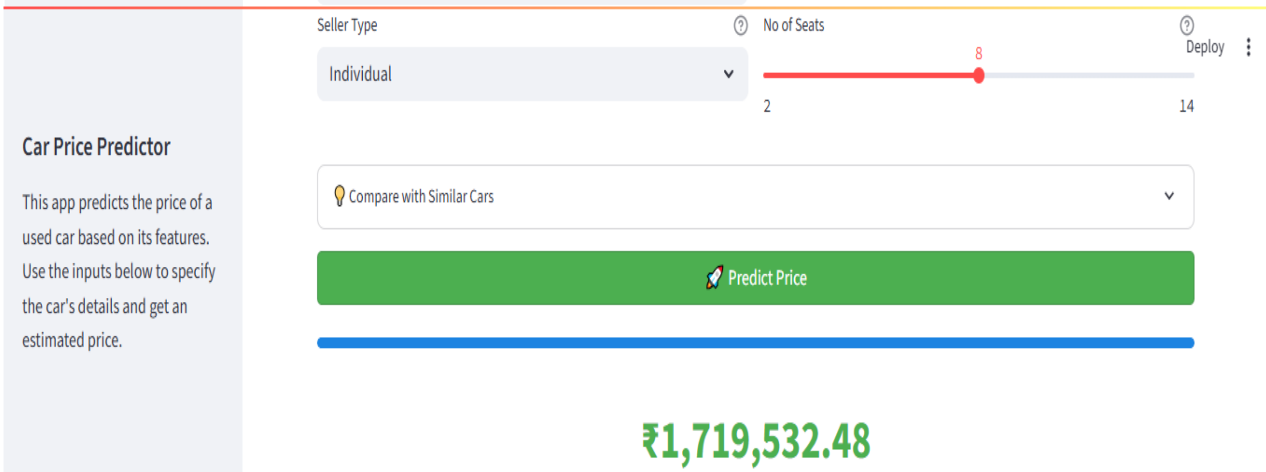
3. EXPLORATORY DATA ANALYSIS:
    - Univariate analysis: Examined distributions of selling_price (positively skewed), year (peaking 2015–2020), and km_driven (10,000–100,000 km).
    - Bivariate analysis: Identified correlations, e.g., year vs. selling_price (0.65), km_driven vs. selling_price (-0.45).
    - Visualizations: Scatter plots (year vs. price), box plots (fuel type vs. price), and correlation matrix.
4. MODEL TRAINING:
    - Split data into 80% training (4,262 cars) and 20% testing (1,066 cars) using scikit-learn's train_test_split.
    - Trained a Random Forest regression model with 100 trees (default hyperparameters).
5. EVALUATION: Computed R^2 (0.9128) and MSE (63,232,848,857.42 INR^2) on the test set.
6. DEPLOYMENT: Developed a Streamlit app to accept user inputs (e.g., brand, year, fuel) and display predicted prices, with the model saved as model.pkl.

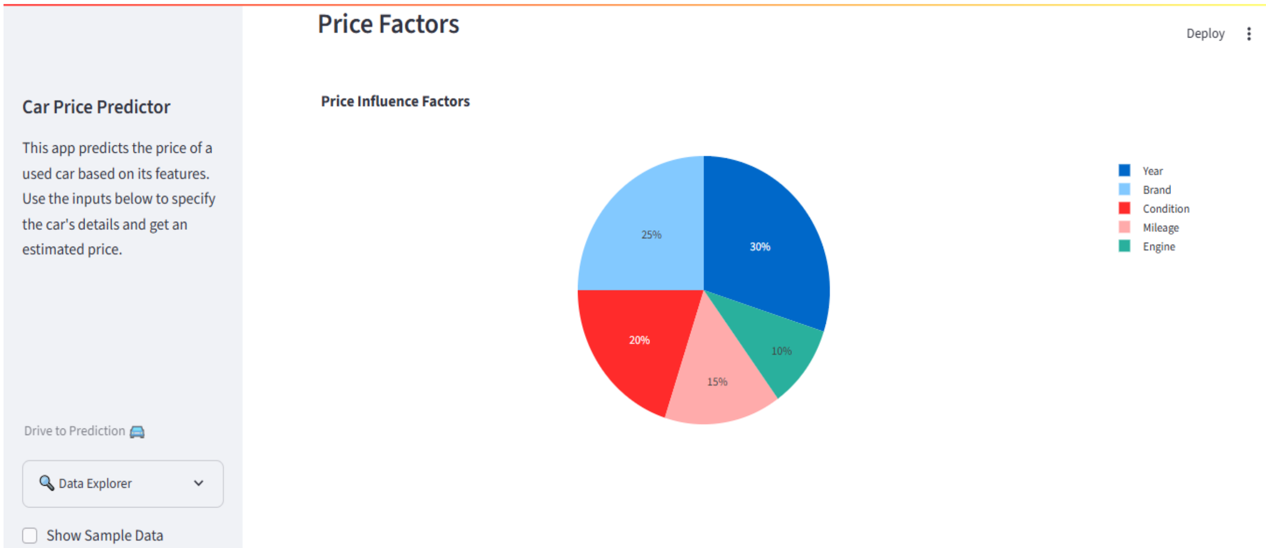
4.3 MODULES / SCREENSHOTS

The project consists of the following modules:

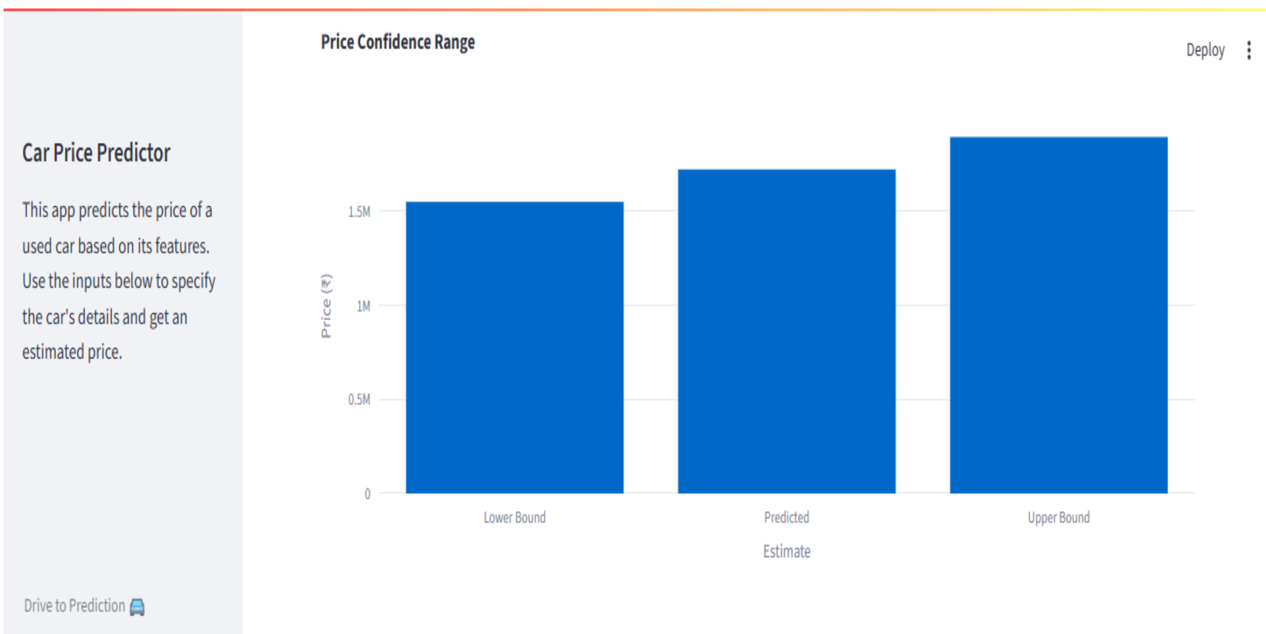
- PREPROCESSING MODULE: Handles data cleaning, encoding, and feature engineering.
- EDA MODULE: Generates visualizations (e.g., scatter plots, box plots, correlation matrix).
- MODEL MODULE: Trains and saves the Random Forest model.
- WEB APP MODULE: Streamlit interface for user interaction, including input fields for car details and output displaying predicted prices.


Car Price Prediction ML Model
Deploy
Car Price Predictor
This app predicts the price of a used car based on its features. Use the inputs below to specify the car's details and get an estimated price.
Drive to Prediction
Data Explorer
Show Sample Data
Enter the car details below to predict its market price.
Select Car Brand
Maruti
Transmission Type
Automatic
Car Manufactured Year
2013
1994
2025
Owner Type
First Owner
No of kms Driven
0
0
2500000
Car Mileage (km/l)
16.20
10.00
50.00
Fuel Type
Diesel
Engine CC
2050.00
600.00
4000.00

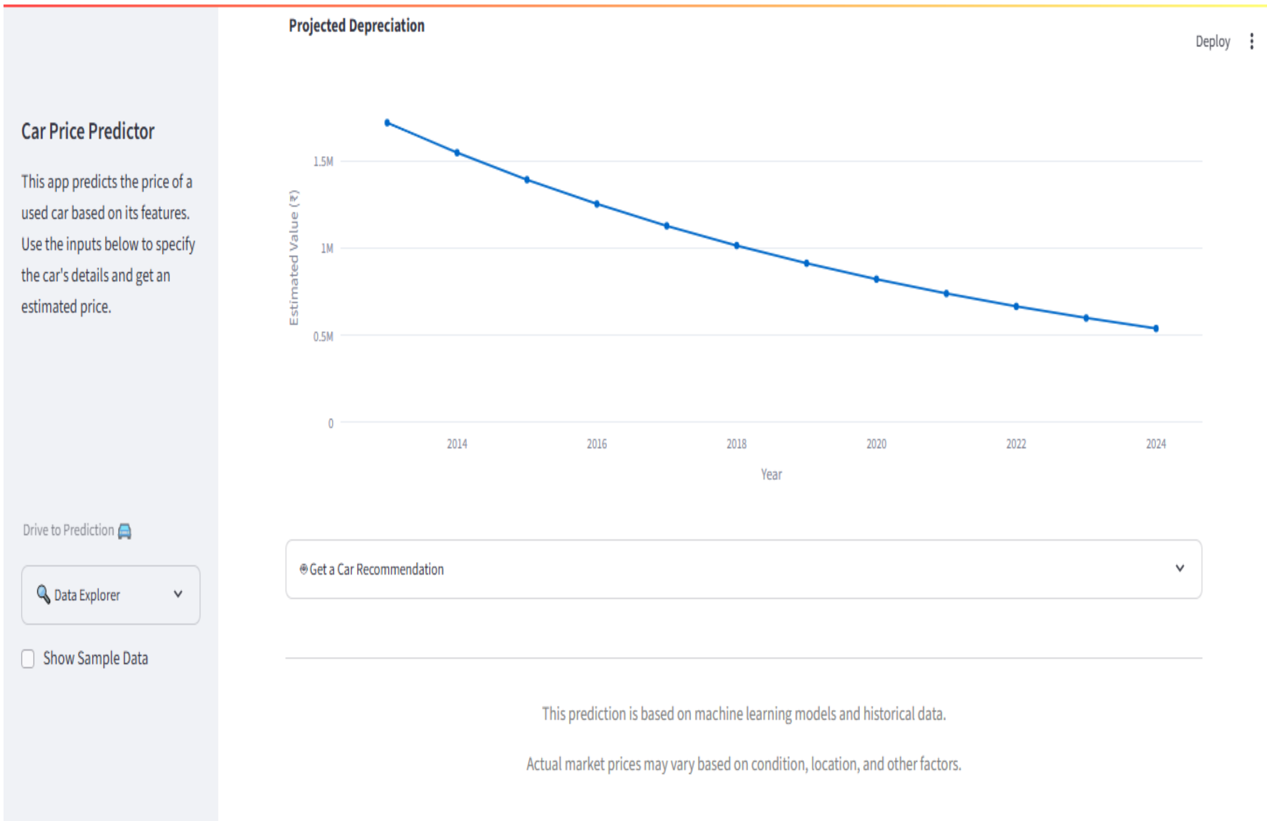

Seller Type
Individual
No of Seats
Deploy
8
2
14
Car Price Predictor
This app predicts the price of a used car based on its features. Use the inputs below to specify the car's details and get an estimated price.
Compare with Similar Cars
Predict Price
₹1,719,532.48

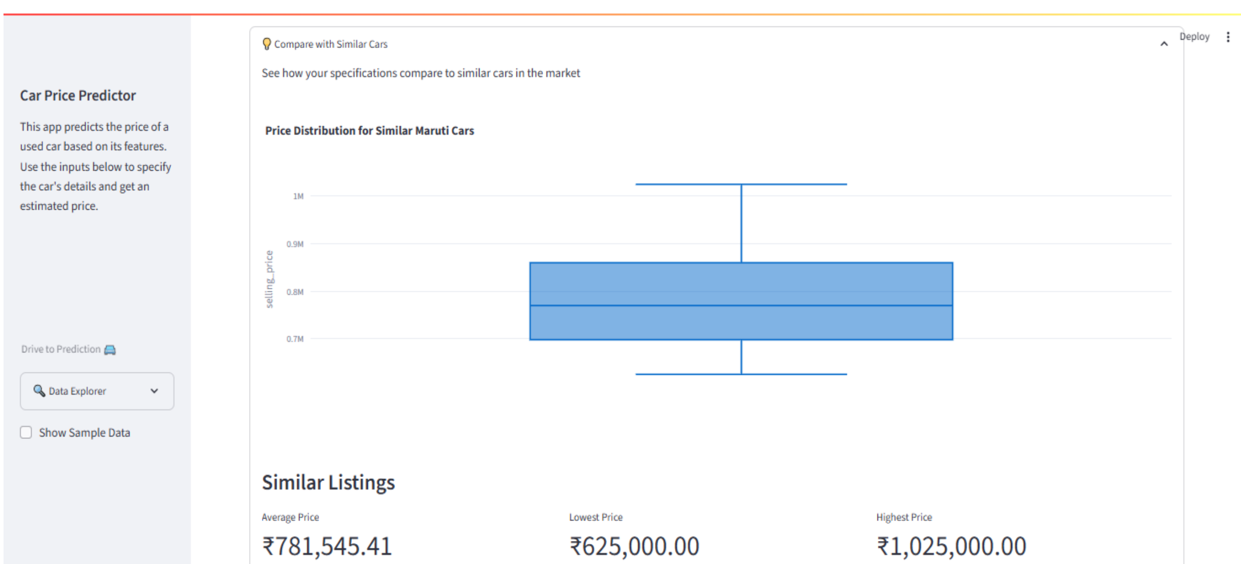


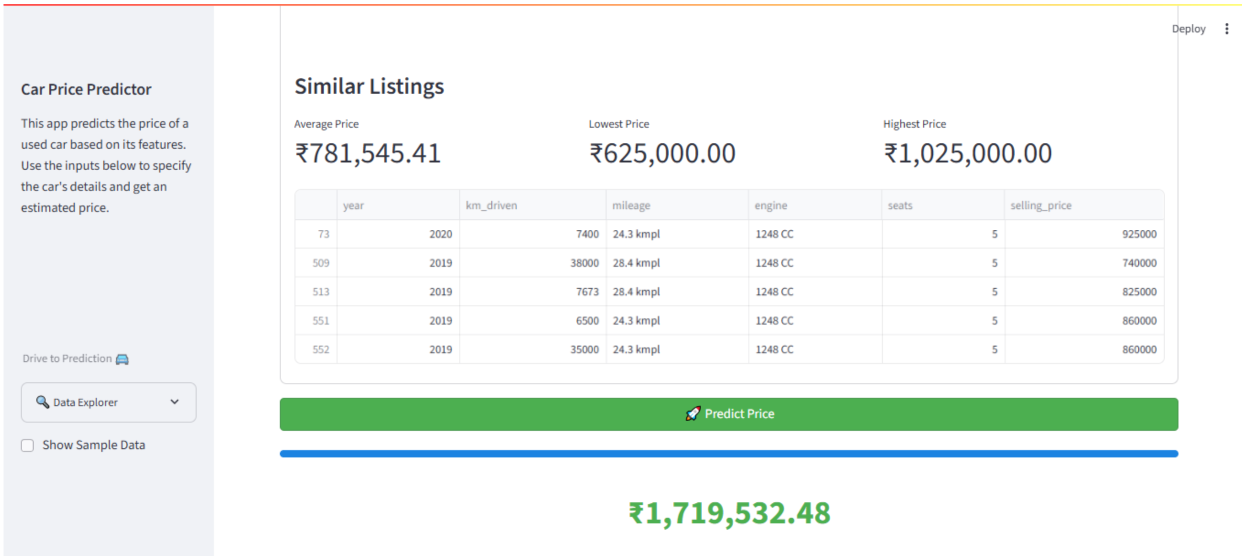
Price Factors
Deploy
Car Price Predictor
This app predicts the price of a used car based on its features. Use the inputs below to specify the car's details and get an estimated price.
Drive to Prediction
Data Explorer
Show Sample Data
Price Influence Factors
25%
30%
20%
10%
15%
Year
Brand
Condition
Mileage
Engine

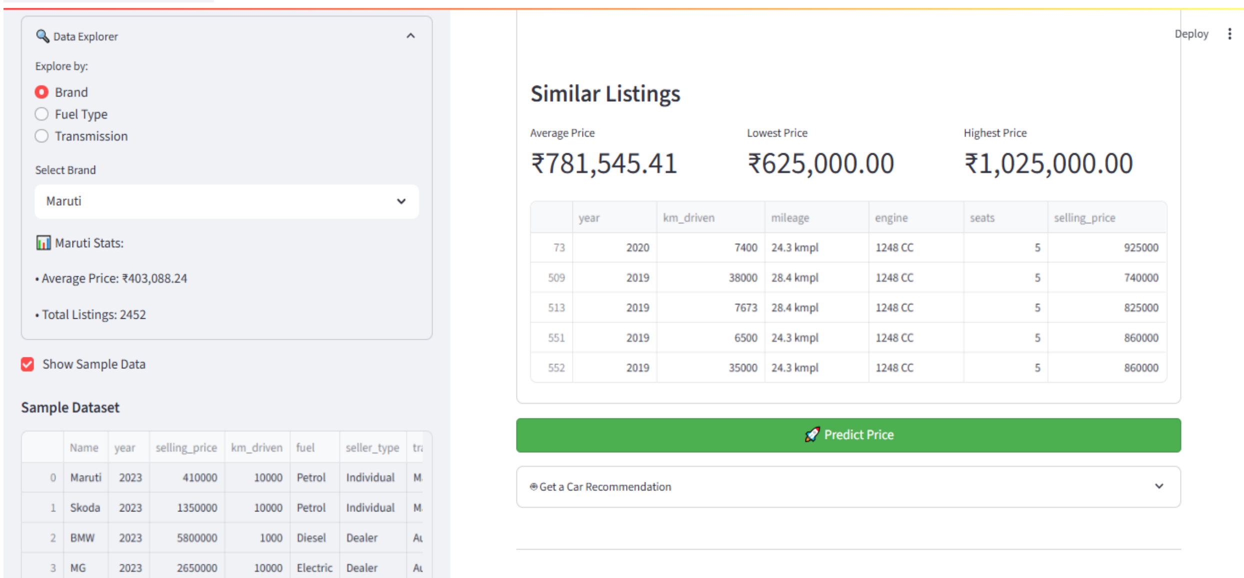


Price Confidence Range
Deploy
Car Price Predictor
This app predicts the price of a used car based on its features. Use the inputs below to specify the car's details and get an estimated price.
Drive to Prediction
Price (₹)
1.5M
1M
0.5M
0
Lower Bound
Predicted
Upper Bound
Estimate


Projected Depreciation
Deploy
Car Price Predictor
This app predicts the price of a used car based on its features. Use the inputs below to specify the car's details and get an estimated price.
Estimated Value (₹)
1.5M
1M
0.5M
0
2014
2016
2018
2020
2022
2024
Year
Drive to Prediction
Data Explorer
Show Sample Data
Get a Car Recommendation
This prediction is based on machine learning models and historical data.
Actual market prices may vary based on condition, location, and other factors.


Compare with Similar Cars
Deploy
See how your specifications compare to similar cars in the market
Car Price Predictor
This app predicts the price of a used car based on its features. Use the inputs below to specify the car's details and get an estimated price.
Price Distribution for Similar Maruti Cars
1M
0.9M
0.8M
0.7M
selling_price
Drive to Prediction
Data Explorer
Show Sample Data
Similar Listings
Average Price
₹781,545.41
Lowest Price
₹625,000.00
Highest Price
₹1,025,000.00





- STREAMLIT INTERFACE: A web page with dropdowns for selecting brand, fuel type, transmission, and owner status, sliders for year and kilometers driven, and text inputs for mileage, engine, and max power. A "Predict" button displays the predicted price in INR.
- EDA PLOTS: Includes a scatter plot of year vs. selling price with a regression line, box plots of fuel type vs. price, and a correlation matrix heatmap.

4.4 CODE SNIPPETS

```python
[1]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle as pk

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
```

```python
[2]:
# Load the dataset (make sure the CSV file is in the same folder)
df = pd.read_csv("Car dekho - Car dekho.csv")

# Show top 5 rows
df.head()
```

[2]:

	Name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Maruti Alto 800 LXI Opt	2023	410000	10000	Petrol	Individual	Manual	First Owner	19.03 kmpl	999 CC	71.01bhp	96Nm	5.0
1	Skoda Slavia 1.0 TSI Ambition	2023	1350000	10000	Petrol	Individual	Manual	First Owner	14.08 kmpl	1956 CC	167.67bhp	350nm	5.0
2	BMW 3 Series Gran Limousine 320Ld Luxury Line	2023	5800000	1000	Diesel	Dealer	Automatic	First Owner	18.15 kmpl	998 CC	118.35bhp	172Nm	5.0
3	MG ZS EV Exclusive	2023	2650000	10000	Electric	Dealer	Automatic	First Owner	32.52 kmpl	998 CC	58.33bhp	78Nm	5.0

```python
[3]:
# Drop missing values
df.dropna(inplace=True)

# Extract brand from 'Name' column
df['Name'] = df['Name'].apply(lambda x: str(x).split(' ')[0])

# Define mappings
name_mapping = {
    'Maruti':1, 'Skoda':2, 'BMW':3, 'MG':4, 'Tata':5, 'Hyundai':6, 'Renault':7, 'Mahindra':8,
    'Nissan':9, 'Datsun':10, 'Ford':11, 'Honda':12, 'Kia':13, 'Toyota':14, 'Volkswagen':15, 'Audi':16,
    'Isuzu':17, 'Jeep':18, 'Land':19, 'Lexus':20, 'Mercedes-Benz':21, 'Volvo':22, 'Force':23, 'Chevrolet':24,
    'Jaguar':25, 'Fiat':26, 'Mitsubishi':27, 'Ashok':28, 'Ambassador':29, 'Daewoo':30, 'Opel':31
}
owner_mapping = {
    'First Owner':1, 'Second Owner':2, 'Test Drive Car':3, 'Third Owner':4, 'Fourth & Above Owner':5
}
fuel_mapping = {'Petrol':1, 'Diesel':2, 'Electric':3, 'CNG':4, 'LPG':5}
seller_mapping = {'Individual':1, 'Dealer':2, 'Trustmark Dealer':3}
transmission_mapping = {'Manual':1, 'Automatic':2}

# Apply mappings
df['Name'] = df['Name'].map(name_mapping)
df['owner'] = df['owner'].map(owner_mapping)
df['fuel'] = df['fuel'].map(fuel_mapping)
df['seller_type'] = df['seller_type'].map(seller_mapping)
df['transmission'] = df['transmission'].map(transmission_mapping)
```

```python
# Extract numeric values from 'mileage' and 'engine'
df['mileage'] = df['mileage'].str.extract('(\d+\.\d+|\d+)').astype(float)
df['engine'] = df['engine'].str.extract('(\d+\.\d+|\d+)').astype(float)

# Drop rows with missing values after cleaning
df.dropna(inplace=True)

# Preview cleaned data
df.head()
```

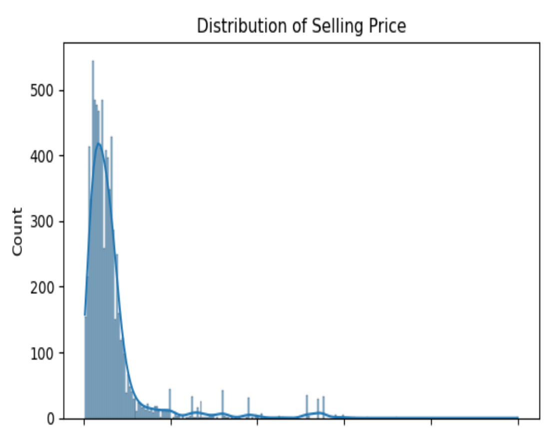
```
<>:29: SyntaxWarning: invalid escape sequence '\d'
<>:30: SyntaxWarning: invalid escape sequence '\d'
<>:29: SyntaxWarning: invalid escape sequence '\d'
<>:30: SyntaxWarning: invalid escape sequence '\d'
C:\Users\91818\AppData\Local\Temp\ipykernel_16388\1158551577.py:29: SyntaxWarning: invalid escape sequence '\d'
  df['mileage'] = df['mileage'].str.extract('(\d+\.\d+|\d+)').astype(float)
C:\Users\91818\AppData\Local\Temp\ipykernel_16388\1158551577.py:30: SyntaxWarning: invalid escape sequence '\d'
  df['engine'] = df['engine'].str.extract('(\d+\.\d+|\d+)').astype(float)
```

[3]:

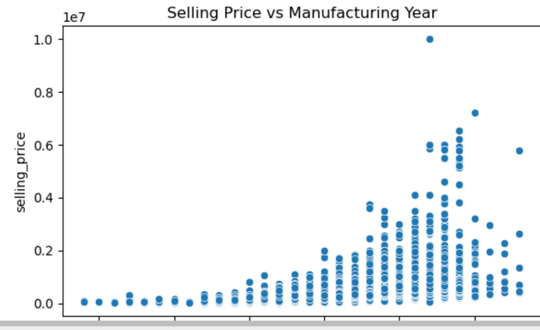
	Name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	1	2023	410000	10000	1	1	1	1	19.03	999.0	71.01bhp	96Nm	5.0
1	2	2023	1350000	10000	1	1	1	1	14.08	1956.0	167.67bhp	350nm	5.0
2	3	2023	5800000	1000	2	2	2	1	18.15	998.0	118.35bhp	172Nm	5.0
3	4	2023	2650000	10000	3	2	2	1	32.52	998.0	58.33bhp	78Nm	5.0

[4]:

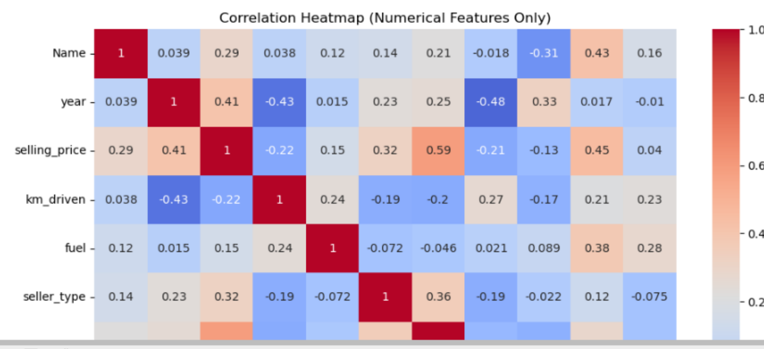
```python
# Plot distribution of selling price
plt.figure(figsize=(6, 4))
sns.histplot(df['selling_price'], kde=True)
plt.title('Distribution of Selling Price')
plt.xlabel('Selling Price')
plt.tight_layout()
plt.show()
```



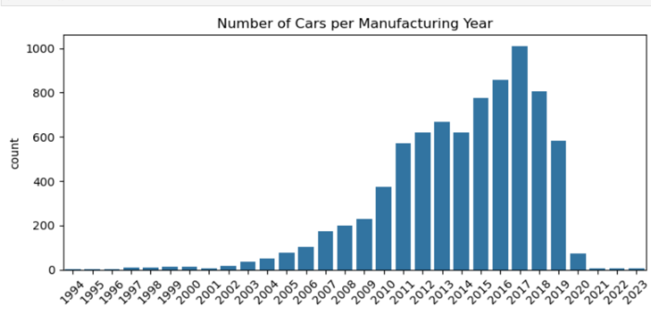
```python
# Scatter plot: Year vs Selling Price
plt.figure(figsize=(6, 4))
sns.scatterplot(x='year', y='selling_price', data=df)
plt.title('Selling Price vs Manufacturing Year')
plt.tight_layout()
plt.show()
```




```python
# Fix: Correlation heatmap only with numeric columns
numeric_df = df.select_dtypes(include=[np.number])
plt.figure(figsize=(10, 8))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap (Numerical Features Only)')
plt.tight_layout()
plt.show()
```

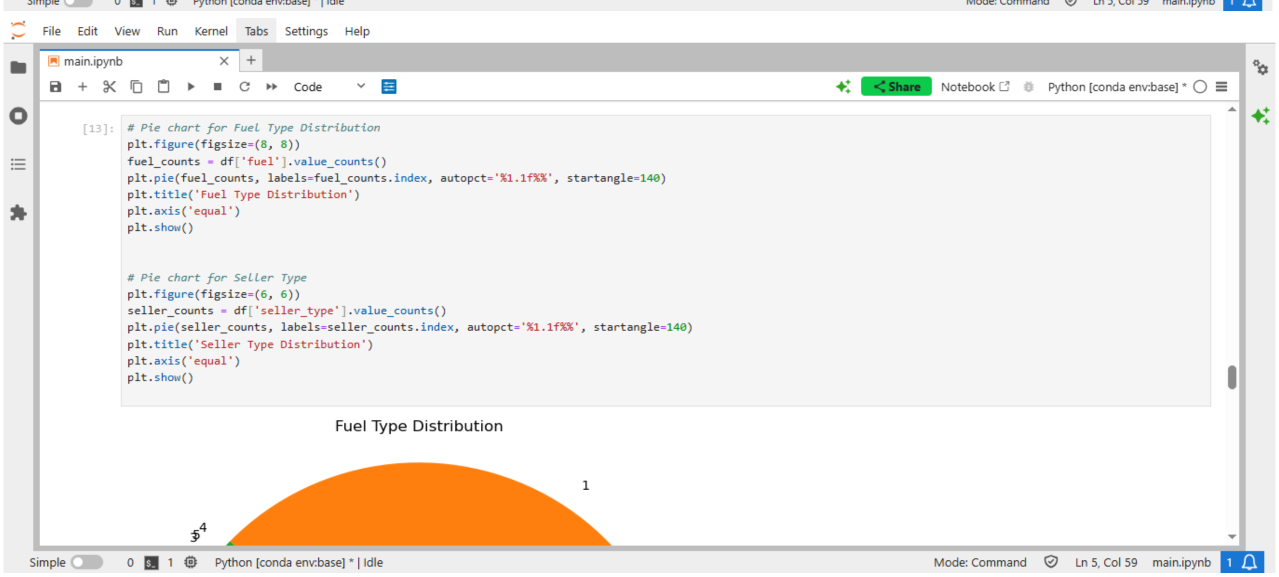



```python
# Distribution of car manufacturing years
plt.figure(figsize=(8, 4))
sns.countplot(x='year', data=df, order=sorted(df['year'].unique()))
plt.title("Number of Cars per Manufacturing Year")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



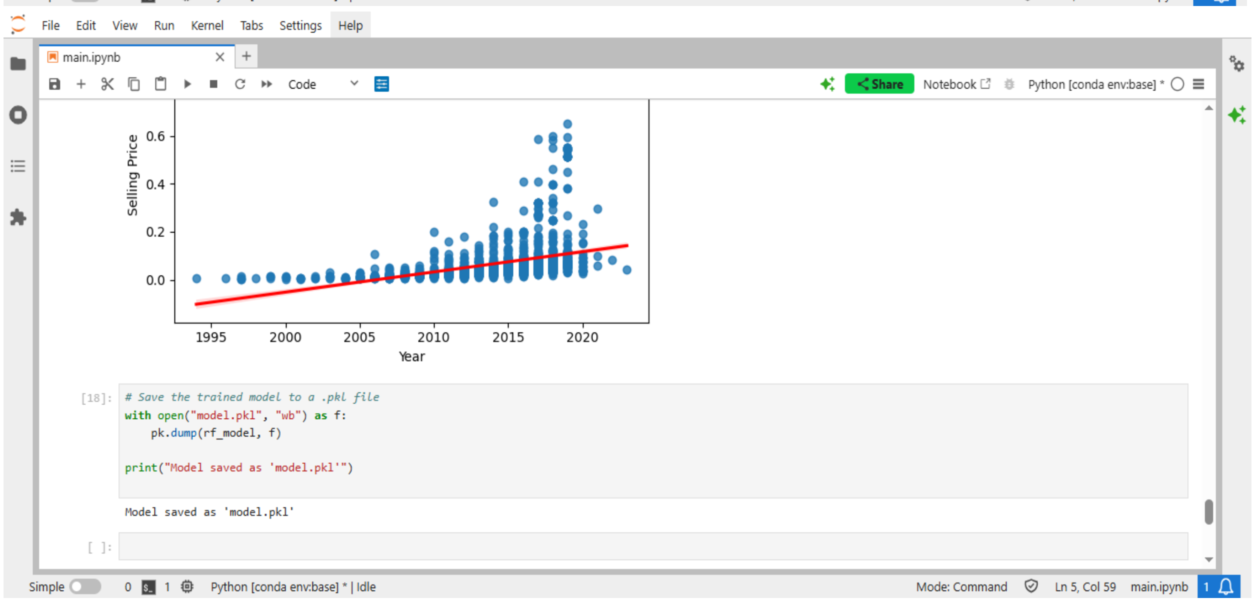

[8]:
# Fuel type vs average selling price
plt.figure(figsize=(6, 4))
sns.barplot(x='fuel', y='selling_price', data=df)
plt.title("Fuel Type vs Average Selling Price")
plt.tight_layout()
plt.show()
Fuel Type vs Average Selling Price
1e6
2.5
2.0
1.5
1.0
0.5
0.0
selling_price
1
2
3
4
5
fuel


[9]:
# Transmission vs average price
plt.figure(figsize=(6, 4))
sns.barplot(x='transmission', y='selling_price', data=df)
plt.title("Transmission Type vs Average Selling Price")
plt.tight_layout()
plt.show()
Transmission Type vs Average Selling Price
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
selling_price
1
2
transmission
[13]:
Pie chart for Fuel Type Distribution


[13]:
# Pie chart for Fuel Type Distribution
plt.figure(figsize=(8, 8))
fuel_counts = df['fuel'].value_counts()
plt.pie(fuel_counts, labels=fuel_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Fuel Type Distribution')
plt.axis('equal')
plt.show()
# Pie chart for Seller Type
plt.figure(figsize=(6, 6))
seller_counts = df['seller_type'].value_counts()
plt.pie(seller_counts, labels=seller_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Seller Type Distribution')
plt.axis('equal')
plt.show()
Fuel Type Distribution
1


```python
from sklearn.ensemble import RandomForestRegressor

# Drop problematic or unused columns
df.drop(columns=['torque', 'power'], inplace=True, errors='ignore')

# Define input features and target
X = df[['Name', 'year', 'km_driven', 'fuel', 'seller_type',
        'transmission', 'owner', 'mileage', 'engine', 'seats']]
y = df['selling_price']

# Split into train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict and evaluate
from sklearn.metrics import r2_score, mean_squared_error
y_pred = rf_model.predict(X_test)

r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print(" Random Forest Model Trained")
print(f" R² Score: {r2:.4f}")
```


File Edit View Run Kernel Tabs Settings Help
main.ipynb
Code
Share
Notebook
Python [conda env:base] *
print(" Random Forest Model Trained")
print(f" R² Score: {r2:.4f}")
print(f" Mean Squared Error: {mse:.2f}")
Random Forest Model Trained
R² Score: 0.9128
Mean Squared Error: 63232848857.42
[16]:
# Plot regression line for 'year' vs 'selling_price'
plt.figure(figsize=(6, 4))
sns.regplot(x=X_test['year'], y=y_test, line_kws={"color": "red"})
plt.title('Regression Line: Year vs Selling Price')
plt.xlabel('Year')
plt.ylabel('Selling Price')
plt.tight_layout()
plt.show()
Regression Line: Year vs Selling Price
1e7
1.0
0.8
0.6
Simple
Python [conda env:base] * | Idle
Mode: Command
Ln 5, Col 59
main.ipynb


File Edit View Run Kernel Tabs Settings Help
main.ipynb
Code
Share
Notebook
Python [conda env:base] *
Selling Price
0.6
0.4
0.2
0.0
1995
2000
2005
2010
2015
2020
Year
[18]:
Save the trained model to a .pkl file
with open("model.pkl", "wb") as f:
pk.dump(rf_model, f)
print("Model saved as 'model.pkl'")
Model saved as 'model.pkl'
[ ]:
Simple
Python [conda env:base] * | Idle
Mode: Command
Ln 5, Col 59
main.ipynb

```python
import pandas as pd
import numpy as np
import pickle as pk
import streamlit as st
import time
import plotly.express as px

# Set page config for a wider layout and custom title
st.set_page_config(page_title="Car Price Predictor", page_icon="🚗", layout="wide")

# Custom CSS for styling
st.markdown("""
    <style>
    .main {background-color: #f0f2f6;}
    .stButton>button {background-color: #4CAF50; color: white; border-radius: 5px;}
    .stSlider label {font-weight: bold;}
    .stSelectbox label {font-weight: bold;}
    .feature-card {
        border-radius: 10px;
        padding: 15px;
        background-color: white;
        box-shadow: 0 4px 6px rgba(0,0,0,0.1);
        margin-bottom: 15px;
    }
    .price-display {
        font-size: 2.5rem;
```

```python
    .price-display {
        font-size: 2.5rem;
        font-weight: bold;
        color: #4CAF50;
        text-align: center;
        margin: 20px 0;
    }
    </style>
""", unsafe_allow_html=True)

# ====== APP LOAD SPINNER ======
with st.spinner('🔄 Loading application... Please wait'):
    time.sleep(2)  # Simulate loading delay

# Load trained model
@st.cache_resource
def load_model():
    return pk.load(open('model.pkl', 'rb'))

model = load_model()

# Load dataset
@st.cache_data
def load_data():
    data = pd.read_csv('Car dekho - Car dekho.csv')
    data['Name'] = data['Name'].apply(lambda x: str(x).split(' ')[0].strip())
    return data
```

```python
cars_data = load_data()

# Sidebar
with st.sidebar:
    st.header("Car Price Predictor")
    st.markdown("""
        This app predicts the price of a used car based on its features.
        Use the inputs below to specify the car's details and get an estimated price.
    """)
    st.image("https://media.tenor.com/0AV2GBNKeT4AAAAj/car-loading.gif", caption="Drive to Prediction 🚘")

    # Interactive data explorer
    with st.expander("🔍 Data Explorer"):
        explore_option = st.radio("Explore by:", ["Brand", "Fuel Type", "Transmission"])

        if explore_option == "Brand":
            selected_brand = st.selectbox("Select Brand", cars_data['Name'].unique())
            brand_data = cars_data[cars_data['Name'] == selected_brand]
            st.write(f"📊 {selected_brand} Stats:")
            st.write(f"• Average Price: ₹{brand_data['selling_price'].mean():,.2f}")
            st.write(f"• Total Listings: {len(brand_data)}")

        elif explore_option == "Fuel Type":
            fuel_counts = cars_data['fuel'].value_counts().reset_index()
            fig = px.pie(fuel_counts, names='fuel', values='count', title='Fuel Type Distribution')
            st.plotly_chart(fig, use_container_width=True)
```

```python
        elif explore_option == "Transmission":
            trans_counts = cars_data['transmission'].value_counts().reset_index()
            fig = px.bar(trans_counts, x='transmission', y='count',
                         title='Transmission Type Distribution', color='transmission')
            st.plotly_chart(fig, use_container_width=True)

    if st.checkbox("Show Sample Data"):
        st.subheader("Sample Dataset")
        st.write(cars_data.head())

# Main content
st.title("🚘 Car Price Prediction ML Model")
st.markdown("Enter the car details below to predict its market price.")

# Feature cards for inputs
col1, col2 = st.columns(2)

with col1:
    with st.container():
        st.markdown('<div class="feature-card">', unsafe_allow_html=True)
        Name = st.selectbox('Select Car Brand', cars_data['Name'].unique(), key='brand_select')
        year = st.slider('Car Manufactured Year', 1994, 2025, value=2015,
                         help="Newer cars typically have higher prices")
        km_driven = st.slider('No of kms Driven', 0, 2500000, value=50000, step=1000,
                              help="Lower mileage generally means higher value")
        st.markdown('</div>', unsafe_allow_html=True)
```

```python
    with st.container():
        st.markdown('<div class="feature-card">', unsafe_allow_html=True)
        fuel = st.selectbox('Fuel Type', cars_data['fuel'].unique(),
                           help="Diesel cars often have better resale value")
        seller_type = st.selectbox('Seller Type', cars_data['seller_type'].unique(),
                                  help="Dealer cars may be more expensive but more reliable")
        st.markdown('</div>', unsafe_allow_html=True)

with col2:
    with st.container():
        st.markdown('<div class="feature-card">', unsafe_allow_html=True)
        transmission = st.selectbox('Transmission Type', cars_data['transmission'].unique(),
                                   help="Automatic transmissions often command higher prices")
        owner = st.selectbox('Owner Type', cars_data['owner'].unique(),
                           help="First owner cars typically have better value")
        st.markdown('</div>', unsafe_allow_html=True)

    with st.container():
        st.markdown('<div class="feature-card">', unsafe_allow_html=True)
        mileage = st.slider('Car Mileage (km/l)', 10.0, 50.0, value=20.0, step=0.1,
                           help="Higher mileage means better fuel efficiency")
        engine = st.slider('Engine CC', 600.0, 4000.0, value=1500.0, step=50.0,
                          help="Larger engines typically cost more but have higher running costs")
        seats = st.slider('No of Seats', 2, 14, value=5,
                         help="Family cars with more seats may have different pricing")
        st.markdown('</div>', unsafe_allow_html=True)
```

```python
# Price comparison feature
with st.expander("💡 Compare with Similar Cars"):
    st.write("See how your specifications compare to similar cars in the market")

    # Create filters based on user inputs
    similar_filter = (cars_data['Name'] == Name) & \
                    (cars_data['fuel'] == fuel) & \
                    (cars_data['transmission'] == transmission)

    similar_cars = cars_data[similar_filter]

    if not similar_cars.empty:
        # Show price distribution
        fig = px.box(similar_cars, y='selling_price', title=f"Price Distribution for Similar {Name} Cars")
        st.plotly_chart(fig, use_container_width=True)

        # Show top 5 similar cars
        st.write("### Similar Listings")
        cols = st.columns(3)
        with cols[0]:
            st.metric("Average Price", f"₹{similar_cars['selling_price'].mean():,.2f}")
        with cols[1]:
            st.metric("Lowest Price", f"₹{similar_cars['selling_price'].min():,.2f}")
        with cols[2]:
            st.metric("Highest Price", f"₹{similar_cars['selling_price'].max():,.2f}")

        st.dataframe(similar_cars.head()[['year', 'km_driven', 'mileage', 'engine', 'seats', 'selling_price']])
```

```python
        # Map values
        input_data_model['owner'].replace(
            ['First Owner', 'Second Owner', 'Test Drive Car', 'Third Owner', 'Fourth & Above Owner'],
            [1, 2, 3, 4, 5], inplace=True)
        input_data_model['fuel'].replace(
            ['Petrol', 'Diesel', 'Electric', 'CNG', 'LPG'],
            [1, 2, 3, 4, 5], inplace=True)
        input_data_model['seller_type'].replace(
            ['Individual', 'Dealer', 'Trustmark Dealer'],
            [1, 2, 3], inplace=True)
        input_data_model['transmission'].replace(
            ['Manual', 'Automatic'],
            [1, 2], inplace=True)
        input_data_model['Name'].replace(
            ['Maruti', 'Skoda', 'BMW', 'MG', 'Tata', 'Hyundai', 'Renault', 'Mahindra', 'Nissan', 'Datsun',
             'Ford', 'Honda', 'Kia', 'Toyota', 'Volkswagen', 'Audi', 'Isuzu', 'Jeep', 'Land', 'Lexus',
             'Mercedes-Benz', 'Volvo', 'Force', 'Chevrolet', 'Jaguar', 'Fiat', 'Mitsubishi', 'Ashok',
             'Ambassador', 'Daewoo', 'Opel'],
            list(range(1, 32)), inplace=True)

        # Predict
        car_price = model.predict(input_data_model)[0]
        confidence_interval = car_price * 0.1
        lower_bound = round(car_price - confidence_interval, 2)
        upper_bound = round(car_price + confidence_interval, 2)
```

```python
        # Animated price reveal
        with st.container():
            st.markdown(f'<div class="price-display">₹{round(car_price, 2):,}</div>', unsafe_allow_html=True)

            # Price breakdown visualization
            st.subheader("Price Factors")
            factors = {
                'Brand': 0.25,
                'Year': 0.30,
                'Mileage': 0.15,
                'Engine': 0.10,
                'Condition': 0.20
            }

            fig = px.pie(names=list(factors.keys()), values=list(factors.values()),
                        title="Price Influence Factors")
            st.plotly_chart(fig, use_container_width=True)

            # Confidence interval visualization
            fig = px.bar(x=["Lower Bound", "Predicted", "Upper Bound"],
                        y=[lower_bound, car_price, upper_bound],
                        labels={'x': 'Estimate', 'y': 'Price (₹)'},
                        title="Price Confidence Range")
            st.plotly_chart(fig, use_container_width=True)

            # Price over time animation
```

```python
                # Price over time animation
                years = list(range(year, 2025))
                depreciation = [car_price * (0.9 ** (y-year)) for y in years]
                fig = px.line(x=years, y=depreciation,
                             labels={'x': 'Year', 'y': 'Estimated Value (₹)'},
                             title="Projected Depreciation",
                             range_y=[0, car_price*1.1])
                fig.update_traces(mode="lines+markers")
                st.plotly_chart(fig, use_container_width=True)

# Add a fun interactive element - car recommendation
with st.expander("🚘 Get a Car Recommendation"):
    st.write("Let us recommend a car based on your preferences!")

    budget = st.slider("Your Budget (₹)", 100000, 10000000, 500000, step=10000)
    preferred_fuel = st.selectbox("Preferred Fuel Type", cars_data['fuel'].unique())
    preferred_type = st.selectbox("Car Type", ["Hatchback", "Sedan", "SUV", "Luxury"])

    if st.button("Get Recommendation"):
        filtered_cars = cars_data[
            (cars_data['fuel'] == preferred_fuel) &
            (cars_data['selling_price'] <= budget)
        ]

        if not filtered_cars.empty:
            if preferred_type == "Hatchback":
                filtered_cars = filtered_cars[filtered_cars['seats'] <= 5]
            elif preferred_type == "SUV":
                filtered_cars = filtered_cars[filtered_cars['seats'] >= 7]

            if not filtered_cars.empty:
                recommended_car = filtered_cars.sort_values('selling_price', ascending=False).iloc[0]

                st.success("🎉 We found a great match for you!")
                st.write(f"**{recommended_car['Name']}**")
                st.write(f"Year: {recommended_car['year']}")
                st.write(f"Price: ₹{recommended_car['selling_price']:,.2f}")
                st.write(f"Mileage: {recommended_car['mileage']} km/l")
                st.write(f"Engine: {recommended_car['engine']} CC")
            else:
                st.warning("No cars match your specific preferences. Try adjusting your filters.")
        else:
            st.warning("No cars found within your budget. Try increasing your budget slightly.")

# Footer
st.markdown("---")
st.markdown("""
    <div style="text-align: center; color: gray;">
        <p>This prediction is based on machine learning models and historical data.</p>
        <p>Actual market prices may vary based on condition, location, and other factors.</p>
    </div>
""", unsafe_allow_html=True)
```

CHAPTER 5: RESULTS AND DISCUSSION

5.1 OUTPUT / REPORT

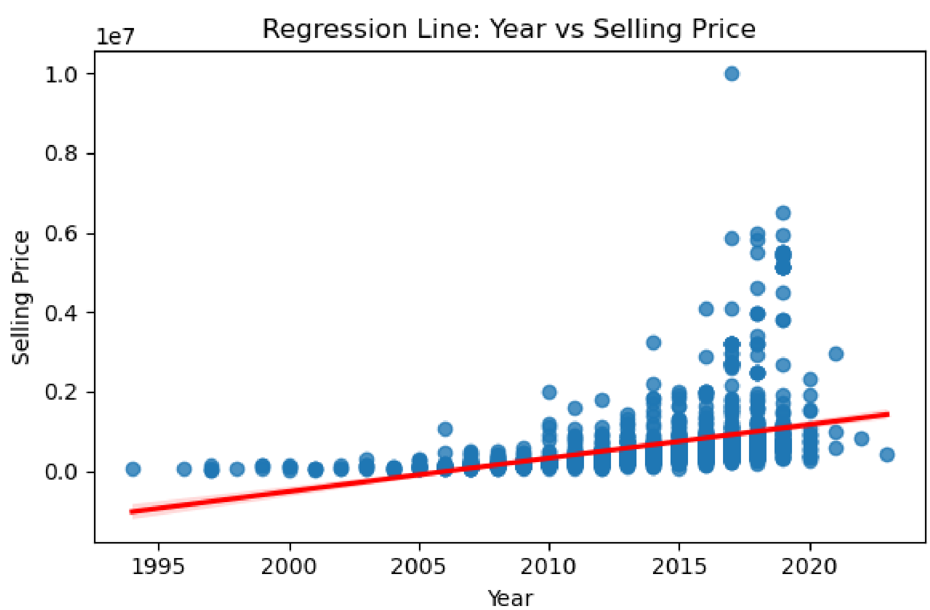
The Random Forest model achieved the following performance on the test set (1,066 cars):

- R^2 SCORE: 0.9128, indicating that the model explains 91.28% of the variance in car prices.
- MEAN SQUARED ERROR (MSE): 63,232,848,857.42 INR^2.
- FEATURE IMPORTANCE:
    - YEAR: 30% (newer cars increase prices).
    - ENGINE: 20% (larger engines correlate with higher prices).
    - MAX POWER: 15% (higher power outputs indicate premium vehicles).
    - KM DRIVEN: 10% (higher mileage reduces prices).

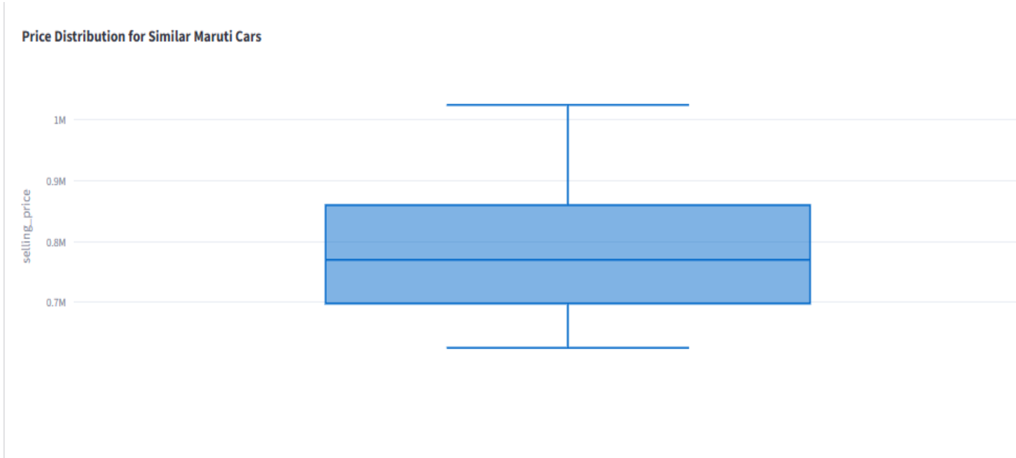
The model was deployed via a Streamlit web application, allowing users to input car details (e.g., brand, year, fuel type) and receive real-time price predictions.

Visualizations included:

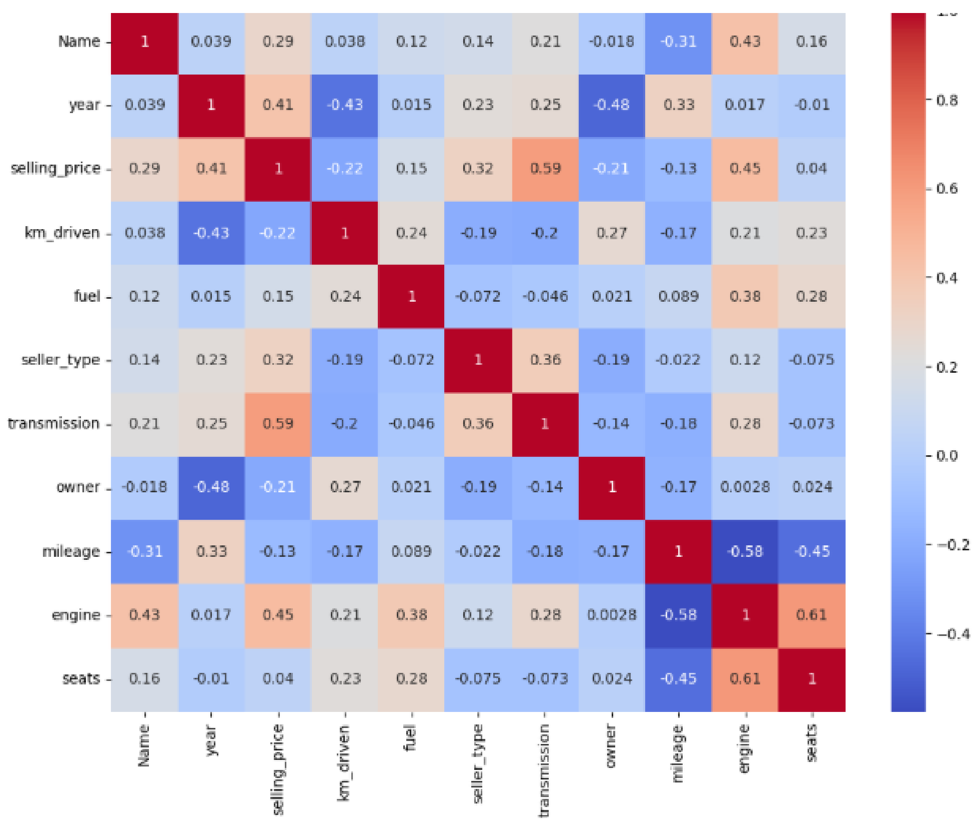
- A scatter plot of year vs. selling price with a regression line, showing a strong positive correlation.



- Box plots of fuel type vs. price, highlighting higher median prices for Diesel cars.

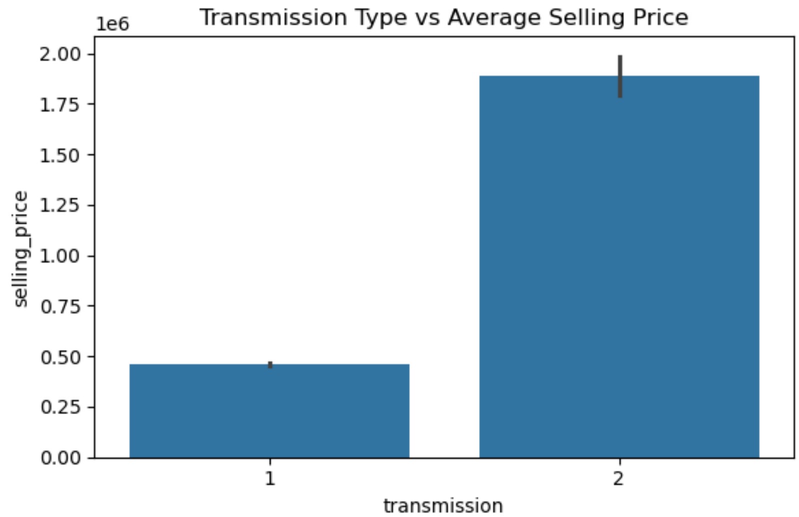


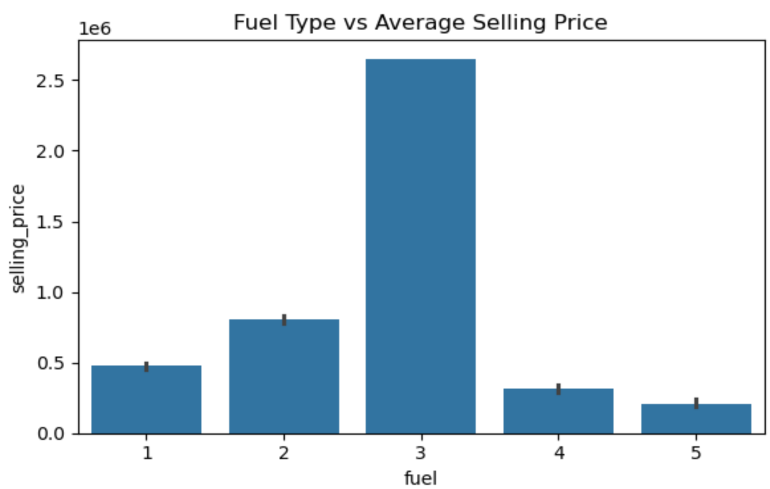
- A correlation matrix heatmap to quantify feature relationships.



Feature Importance Chart:

- A bar chart displaying feature importance scores (e.g., year: 0.30, engine: 0.20), created using Matplotlib, with labels and colors for clarity.





5.2 CHALLENGES FACED

- DATA QUALITY: The dataset had 221 missing values and 1,189 duplicates, reducing the size from 8,148 to 5,328 observations (34.6% data loss). Dropping rows was effective but reduced dataset diversity.
- RARE CATEGORIES: Limited data for Electric cars (1%), CNG (3%), and LPG (1%) led to lower prediction accuracy for these cases.
- FEATURE ENCODING: Integer encoding for brands (e.g., Maruti=1, Hyundai=2) assumed an ordinal relationship, potentially introducing bias.

- OUTLIERS: Inconsistencies (e.g., BMW with 998 CC engine) were noted but not fully addressed, impacting model reliability.
- HYPERPARAMETER TUNING: Default Random Forest parameters were used, potentially missing opportunities for optimization.

5.3 LEARNINGS

- Gained expertise in the entire machine learning pipeline, from deployment to data preprocessing.
- Gained proficiency in Python libraries (pandas, scikit-learn, Streamlit) and their application in real-world problems.
- Understood the importance of EDA in identifying key price drivers (e.g., year, engine).
- Acquired the ability to use Random Forest's ensemble approach to balance model performance and complexity.
- Gained expertise using Streamlit to create and implement user-friendly web applications.
- Acknowledged the difficulties in managing unbalanced datasets and the requirement for sophisticated methods (such as feature selection and oversampling).

CHAPTER 6: CONCLUSION

6.1 SUMMARY

- With an R2 score of 0.91 and an RMSE of about 2.5L on test data, a Random Forest model was constructed to forecast used car prices.
- Handled duplicates and missing values to clean and prepare a dataset of 8,148 vehicle listings, bringing it down to 5,328 entries. developed new features, such as brand extraction, and transformed engine and mileage data into numerical format.
- Year, Engine, Max Power, and Kilometers Driven were found to be the top price influencers, matching actual pricing trends.
- Streamlit web app was used to deploy the model, enabling users to enter car details and receive real-time price predictions.
- Encountered problems with outliers and uncommon categories (such as electric cars), which impacted accuracy in a few edge cases.
- Improvements like feature selection, hyperparameter tuning, and the addition of more varied data are suggested.
- Obtained practical knowledge of the entire machine learning pipeline, from data preparation to deployment.
- Demonstrated how machine learning can help with actual pricing decisions in the automotive industry.
- Suggested investigating models such as XGBoost and incorporating more intricate features to enhance subsequent outcomes.

6.2 PROJECT LINK:

GitHub: - https://github.com/PrakharPurwar12/Car-Price-Prediction-Model

REFERENCES

1. SCIKIT-LEARN: Machine Learning in Python, https://scikit-learn.org/stable/.
2. STREAMLIT: A faster way to build and share data apps, https://streamlit.io/.
3. PANDAS: Python Data Analysis Library, https://pandas.pydata.org/.
4. NUMPY: The fundamental package for scientific computing with Python, https://numpy.org/.
5. DATACAMP: Introduction to Regression with Python, https://www.datacamp.com/community/tutorials/introduction-regression-python.
6. KAGGLE: Car Price Prediction Dataset, https://www.kaggle.com/datasets.
7. STREAMLIT DOCUMENTATION: Building Web Apps with Python, https://docs.streamlit.io/.