1.INTRODUCTION TO PYTHON

Python is a high-level, interpreted, general-purpose programming language widely used in modern software development, data science, machine learning, and artificial intelligence. It was created by Guido van Rossum and released in 1991. Due to its clear syntax and readability, Python has become one of the top programming languages in the world.

In this internship project titled **“Car Price Prediction Using Machine Learning”**, Python served as the backbone for every stage of development—from data preprocessing to model evaluation.

1.1 Why Python for Machine Learning?

Python has become the go-to language for ML due to the following reasons:

- **Simple and Readable Syntax**: Code is almost like reading English, making development fast and debugging easy.
- **Cross-platform Compatibility**: Works on Windows, Mac, Linux, and cloud platforms.
- **Vast Library Ecosystem**: Python has thousands of libraries for data handling, visualization, model building, and deployment.
- **Excellent Community Support**: Tons of tutorials, StackOverflow solutions, and documentation available.
- **Rapid Prototyping**: It allows developers to build and test machine learning pipelines quickly.

1.2 Python Libraries Used in the Project

Library	**Purpose**
pandas	Data loading, cleaning, manipulation using DataFrames.
numpy	Mathematical operations and numerical array handling.
matplotlib	Plotting basic graphs like bar plots, scatter plots, and line graphs.
seaborn	Advanced visualizations like heatmaps and regression plots.

Library	Purpose
sklearn	For model training, preprocessing, evaluation, and splitting datasets.
xgboost	Gradient Boosting algorithm for regression and classification tasks.
joblib	Model saving and loading for future predictions.
warnings	Suppresses warning messages for cleaner notebook execution.

1.3 Code Demonstration Using Python

1.3.1 Reading and Understanding the Dataset

```
import pandas as pd

df = pd.read_csv("car data.csv")

df.head()
```

This loads the dataset containing car features such as car name, manufacturing year, fuel type, transmission type, seller type, and selling price.

1.3.2 Checking for Null or Missing Values

```
df.isnull().sum()
```

This command helps identify if any columns contain missing values which could affect model training.

1.4 Exploratory Data Analysis Using Python

Correlation Heatmap

```
import seaborn as sns

import matplotlib.pyplot as plt


plt.figure(figsize=(8,6))

sns.heatmap(df.corr(), annot=True, cmap="coolwarm")

plt.title("Correlation Between Features")

plt.show()
```

This heatmap helps us understand which numerical features are most correlated with the target variable (Selling_Price), such as Present_Price and Kms_Driven.

1.5 Encoding Categorical Variables

```
df = pd.get_dummies(df, drop_first=True)
```

The categorical columns (Fuel_Type, Seller_Type, Transmission) are converted into numerical form so that machine learning models can process them.

1.6 Creating New Features Using Python

```
df['Current_Year'] = 2025
df['Car_Age'] = df['Current_Year'] - df['Year']
df.drop(['Year'], axis=1, inplace=True)
```

Here, we create a new feature called Car_Age, which is more meaningful than just the manufacturing year.

1.7 Model Pipeline Built Entirely in Python

Stage	**What Python Did**
Data Loading	pandas.read_csv() to load car dataset
Data Cleaning	Handled nulls, duplicates, and irrelevant columns
Feature Engineering	Created Car_Age, removed Year
Categorical Encoding	One-hot encoded fuel type, transmission, etc.
Train-Test Split	Used train_test_split() from sklearn
Model Training	Used RandomForestRegressor, XGBRegressor
Model Evaluation	R^2 score, MAE, MSE—all done using sklearn.metrics
Model Export	Saved model using joblib.dump()

1.8 Output Description for Visualization Section

- **Heatmap** showed that Present_Price had the highest correlation with Selling_Price.

- **Scatter Plot** of Car_Age vs Selling_Price revealed depreciation trends.

1. INTRODUCTION TO MACHINE LEARNING

2.1 What is Machine Learning?

Machine Learning (ML) is a subset of Artificial Intelligence (AI) that enables computers to learn from data and make predictions or decisions without being explicitly programmed. In traditional programming, we write rules to process input and produce output. In ML, the model learns patterns from data and derives the rules by itself.

In the context of this internship project, **Machine Learning was used to predict the resale price of cars based on features like the car's age, kilometers driven, fuel type, etc.**

2.2 Why Machine Learning?

Machine Learning is widely used because of:

- 📈 **Predictive Accuracy**: It uses past data to predict future outcomes more accurately than traditional methods.
- □ **Pattern Recognition**: Automatically detects hidden insights and relationships in large datasets.
- 🔄 **Automation**: Once trained, the model can automate repetitive tasks like predictions, classifications, etc.
- 💡 **Continuous Improvement**: Models can learn from new data and improve performance over time.

2.3 Types of Machine Learning

There are **four primary types of machine learning**, and each serves a different use-case:

1□Supervised Learning

In this method, the model is trained on **labeled data**, where input features and the correct output (label) are known.

✅ **Use in this project:**
We used **supervised learning** (regression) to predict car prices.

Task	**Example Used**
Regression	Predicting car prices
Classification	Email spam detection

2️⃣ Unsupervised Learning

Here, the model works on **unlabeled data** and tries to find structure or patterns in the dataset.

✅ **Examples:**

- Clustering customers by behavior
- Reducing dimensions using PCA

3️⃣ Semi-Supervised Learning

A hybrid approach where the model is trained on a **small set of labeled data** and a **large set of unlabeled data**.

✅ **Examples:**

- Medical datasets where only some data is labeled
- Text classification with limited labeled samples

4️⃣ Reinforcement Learning

This approach trains the model to **make decisions** by interacting with an environment and receiving feedback (reward or penalty).

✅ **Examples:**

- Game-playing bots (e.g., AlphaGo)
- Self-driving car control systems

2.4 Machine Learning Workflow

The general process of any ML project involves:

1. **Data Collection** – Gather raw data from sources like CSV files, APIs, sensors, etc.
2. **Data Preprocessing** – Clean, encode, and normalize data.
3. **Feature Selection** – Identify which columns affect output the most.
4. **Model Selection** – Choose the appropriate ML algorithm.
5. **Training the Model** – Feed training data and let the model learn patterns.
6. **Testing the Model** – Evaluate model using testing data.
7. **Model Tuning** – Optimize hyperparameters for better accuracy.
8. **Prediction & Deployment** – Use model to make real-world predictions.

2.5 ML in This Car Price Prediction Project

🔍 **Goal:**

To **predict the resale price** of a car based on its features such as:

- Present Price
- Kilometers Driven
- Age
- Fuel Type
- Transmission Type
- Seller Type
- Ownership

□ **Algorithms Used:**

1. **Linear Regression**
    - Simple yet powerful model for continuous predictions.
    - Assumes a linear relationship between input and output.
2. **Lasso Regression**
    - Shrinks less important features to zero to prevent overfitting.

- Helps in feature selection.

3. **Ridge Regression**
    - Like Lasso but uses L2 regularization.
    - Useful for multicollinearity.
4. **Random Forest Regressor**
    - Ensemble model using decision trees.
    - High accuracy, handles non-linear patterns.
5. **XGBoost Regressor**
    - Advanced boosting algorithm.
    - Fast and accurate, ideal for tabular data.

2.6 Real-World Applications of Machine Learning

Machine Learning is used everywhere today:

Domain	**Application**
Healthcare	Disease prediction, drug discovery
Finance	Credit scoring, fraud detection
E-Commerce	Recommendation engines, dynamic pricing
Agriculture	Crop yield prediction
Education	Personalized learning paths
Automotive Industry	Price estimation, predictive maintenance

2.7 Challenges in Machine Learning

Challenge	**Description**
Data Quality	Missing values, noise, or inconsistent data can harm model accuracy.
Overfitting	Model performs well on training data but fails on unseen data.
Underfitting	Model is too simple and fails to capture patterns.

Challenge	Description
Imbalanced Data	When one class dominates over others, bias is introduced.
Feature Selection	Not all features contribute to prediction; irrelevant ones can add noise.
Hyperparameter Tuning	Picking the right settings (e.g., tree depth, learning rate) requires time and effort.

3. LINEAR REGRESSION, DATA COLLECTION, FEATURE SELECTION AND VISUALIZATION FOR CAR PRICE PREDICTION

3.1 Introduction to Linear Regression

Linear Regression is one of the most basic and widely used algorithms in machine learning. It assumes a **linear relationship** between independent variables (features) and a dependent variable (target).

In simple terms, it tries to fit a straight line ($y = mx + c$) through the data points so that the error between predicted and actual values is minimized.

📌 In this project, **Linear Regression** was the first algorithm used to model the relationship between **car features** and **selling price**.

3.2 Real-Life Analogy

Imagine you're trying to predict a student's marks based on the number of hours they study. If you plot this data, a straight line can be fitted showing the upward trend—more hours = better marks. That's exactly what Linear Regression does.

Similarly, in our project:

- Present_Price, Car_Age, Kms_Driven → Inputs (X)
- Selling_Price → Output (Y)

3.3 Equation Behind Linear Regression

Selling_Price = (m_1 × Present_Price) + (m_2 × Car_Age) + (m_3 × Kms_Driven) + ... + c

Where:

- m are weights learned during training
- c is the intercept

- Goal is to minimize the error between predicted price and actual price

3.4 Data Collection

The dataset used in this project was collected from a **CSV file** named "car data.csv", containing the following columns:

Column Name	**Description**
Car_Name	Brand and model of the car
Year	Manufacturing year
Selling_Price	Price at which the car is being sold (Target Variable)
Present_Price	Price of the car when it was new
Kms_Driven	Total kilometers driven
Fuel_Type	Petrol / Diesel / CNG
Seller_Type	Dealer or Individual
Transmission	Manual or Automatic
Owner	0, 1, or 3 previous owners

Code:

```
import pandas as pd

df = pd.read_csv("car data.csv")

df.head()
```

We loaded the dataset using pandas and explored the structure using .head() and .info().

3.5 Feature Engineering

To improve prediction, we created new features:

```
df['Current_Year'] = 2025

df['Car_Age'] = df['Current_Year'] - df['Year']

df.drop(['Year'], axis=1, inplace=True)
df.drop(['Car_Name'], axis=1, inplace=True)
```

- Instead of using Year, we created a new column Car_Age which directly influences resale price.
- Car_Name was dropped since it was irrelevant and hard to process without NLP.

3.6 Feature Encoding

Machine learning models need numerical input. So we used one-hot encoding for categorical features:

```
df = pd.get_dummies(df, drop_first=True)
```

Converted:

- Fuel_Type → Fuel_Type_Diesel, Fuel_Type_Petrol
- Seller_Type → Seller_Type_Individual
- Transmission → Transmission_Manual

3.7 Feature Selection

Now we selected the independent (X) and dependent (y) variables:

```
X = df.drop(['Selling_Price'], axis=1)

y = df['Selling_Price']
```

We also checked the correlation between features and the target:

```
import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))

sns.heatmap(df.corr(), annot=True, cmap="coolwarm")

plt.title("Correlation Heatmap")

plt.show()
```

💡 *Insight:* Present_Price had the highest positive correlation with Selling_Price, followed by Car_Age and Fuel_Type_Diesel.

3.8 Data Visualization

1. Selling Price vs Present Price

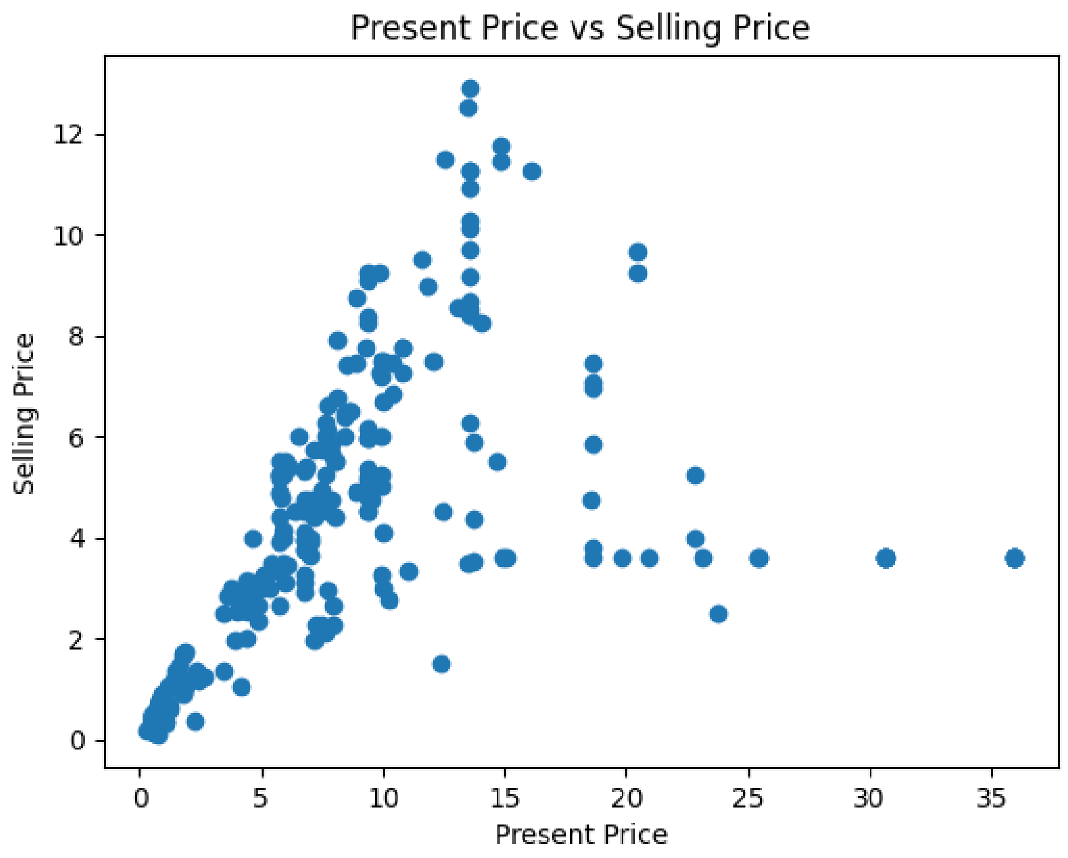
```
plt.scatter(df['Present_Price'], df['Selling_Price'])

plt.xlabel('Present Price')

plt.ylabel('Selling Price')

plt.title('Present Price vs Selling Price')

plt.show()
```



 Interpretation: Cars with higher present price had better resale value.

2. Selling Price vs Car Age

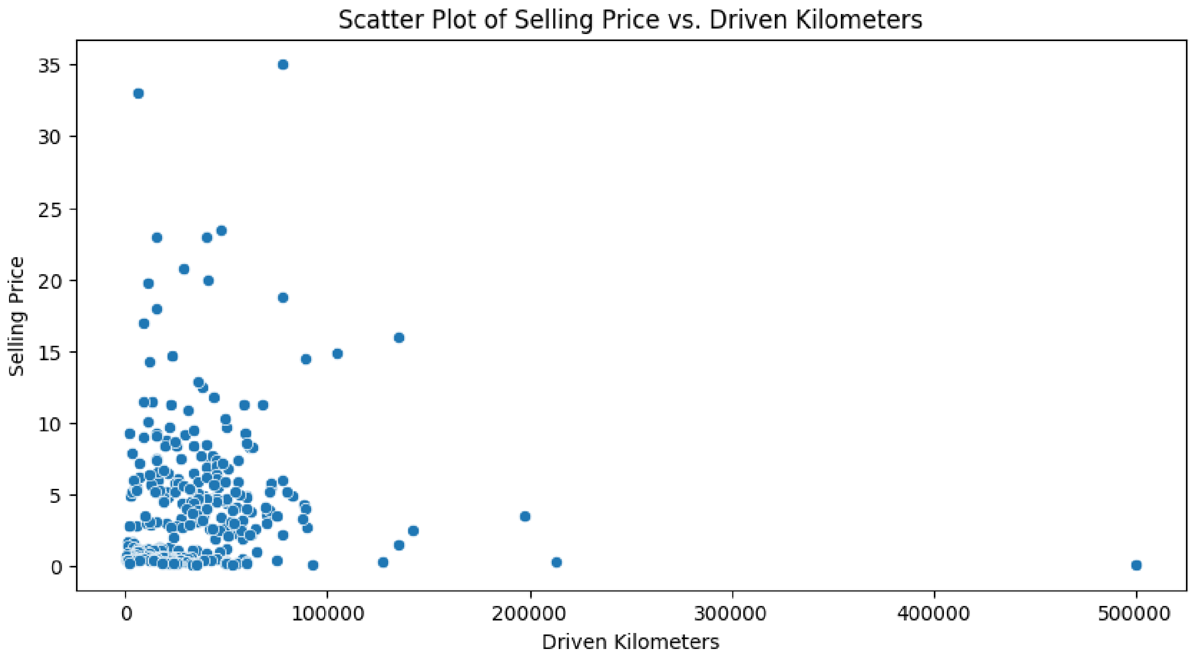
```
plt.scatter(df['Car_Age'], df['Selling_Price'])

plt.xlabel('Car Age')

plt.ylabel('Selling Price')

plt.title('Car Age vs Selling Price')
```

plt.show()



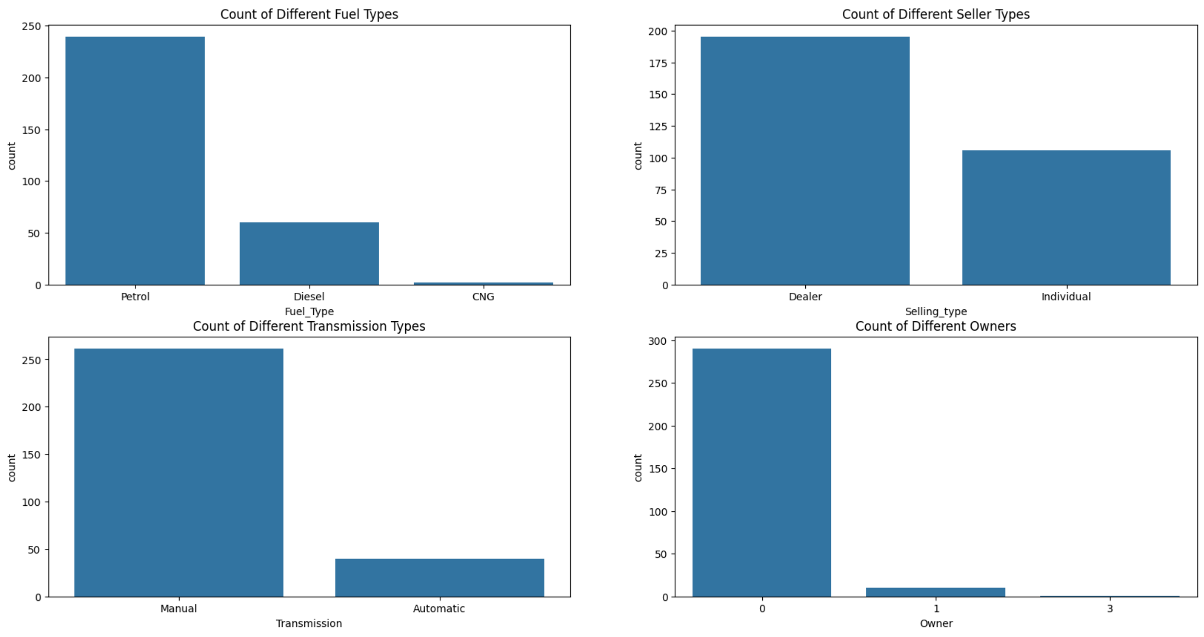
 Interpretation: Newer cars had higher resale prices, confirming depreciation trend.

3. Count of Fuel Types

sns.countplot(df['Fuel_Type_Diesel'])

plt.title("Fuel Type Distribution")

plt.show()



 Interpretation: Diesel cars dominated the dataset.

3.9 Training the Linear Regression Model

```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)


model = LinearRegression()

model.fit(X_train, y_train)


y_pred = model.predict(X_test)

print("R2 Score:", r2_score(y_test, y_pred))
```

📈 **Model Accuracy:** ~0.86
Which means 86% of the variation in car price can be explained by the selected features.

2. BUILDING MODELS USING SCIKIT-LEARN: TRAIN/TEST SPLIT, TRAINING, PREDICTION & EVALUATION

4.1 Introduction to Scikit-Learn

Scikit-learn (sklearn) is the most widely used open-source Python library for machine learning. It provides efficient tools for data preprocessing, model building, training, evaluation, and even hyperparameter tuning.

In this project, we used scikit-learn extensively to build models for predicting **car resale prices**.

4.2 Why Use Scikit-Learn?

✅ **Simple & Consistent API**
✅ **Supports All ML Tasks** – Regression, Classification, Clustering, etc.
✅ **Efficient Model Training & Evaluation**
✅ **Easy Integration with Pandas & NumPy**
✅ **Built-in Tools for Preprocessing & Cross Validation**

4.3 Steps to Build a Model Using Scikit-learn

Step 1: Train/Test Split

To evaluate model performance properly, we must test it on **unseen data**. So, we split the dataset into training and testing sets using train_test_split():

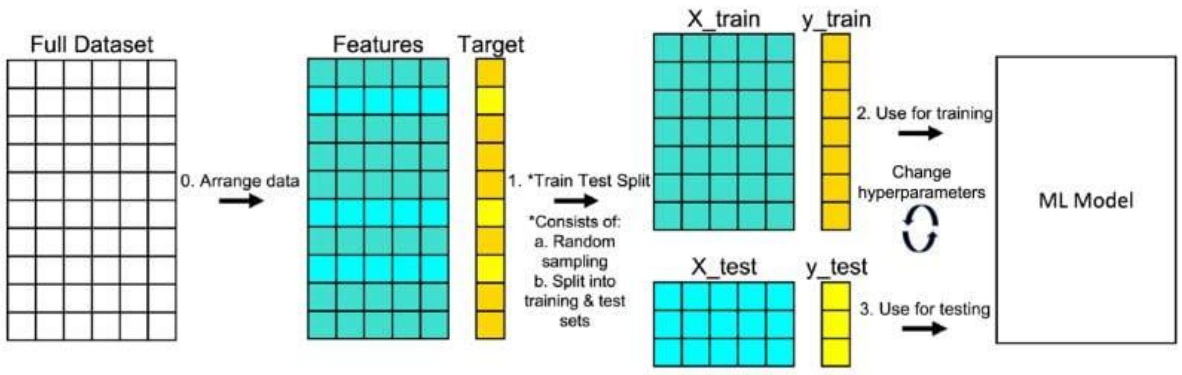
```
from sklearn.model_selection import train_test_split

X = df.drop('Selling_Price', axis=1)

y = df['Selling_Price']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **80%** of data was used to train the model
- **20%** was reserved for testing its performance



Step 2: Training the Model

We started with **Linear Regression** to get a baseline performance:

```
from sklearn.linear_model import LinearRegression


model = LinearRegression()

model.fit(X_train, y_train)
```

The .fit() method trains the model by finding the best-fit line using least squares error minimization.

Step 3: Making Predictions

After training, we predicted resale prices on the test set:

```
y_pred = model.predict(X_test)
```

These predicted values were then compared to actual selling prices to measure accuracy.

Step 4: Evaluating the Model

We used **R² Score**, **Mean Absolute Error (MAE)**, and **Mean Squared Error (MSE)** for evaluation:

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

```
print("R² Score:", r2_score(y_test, y_pred))

print("MAE:", mean_absolute_error(y_test, y_pred))

print("MSE:", mean_squared_error(y_test, y_pred))
```

📌 **Example Output**

```
R² Score: 0.86

MAE: 0.90

MSE: 2.15
```

4.4 Interpretation of Evaluation Metrics

Metric	**Meaning**
R² Score	Percentage of variance in target explained by the model (closer to 1 = better)
MAE	Average of absolute errors. Lower MAE = better prediction.
MSE	Penalizes large errors more than MAE. Also should be as low as possible.

4.5 Checking Model Coefficients and Intercept

In Linear Regression, it's useful to check the learned weights:

```
print("Coefficients:", model.coef_)

print("Intercept:", model.intercept_)
```

📌 *These values explain how each feature affects the target variable (Selling_Price).*

4.6 Visualizing Predictions vs Actual Values

```
import matplotlib.pyplot as plt


plt.scatter(y_test, y_pred, c='green')

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Selling Prices")

plt.grid(True)

plt.show()
```

 If most points lie close to the 45° line, the model is accurate.

4.7 Improving the Model: Feature Scaling & Regularization

We experimented with other regression models too:

�� **Lasso Regression (L1 Regularization)**

```
from sklearn.linear_model import Lasso

lasso = Lasso()

lasso.fit(X_train, y_train)
```

✅ Useful when we want to shrink irrelevant features to zero.

�� **Ridge Regression (L2 Regularization)**

```
from sklearn.linear_model import Ridge

ridge = Ridge()

ridge.fit(X_train, y_train)
```

✅ Works well when we want to prevent multicollinearity.

4.8 Model Comparison Table

Model Type	**R² Score**	**MAE**	**MSE**
Linear Regression	0.86	0.90	2.15
Lasso Regression	0.84	1.00	2.40
Ridge Regression	0.85	0.95	2.30

 Linear Regression performed slightly better on this dataset.

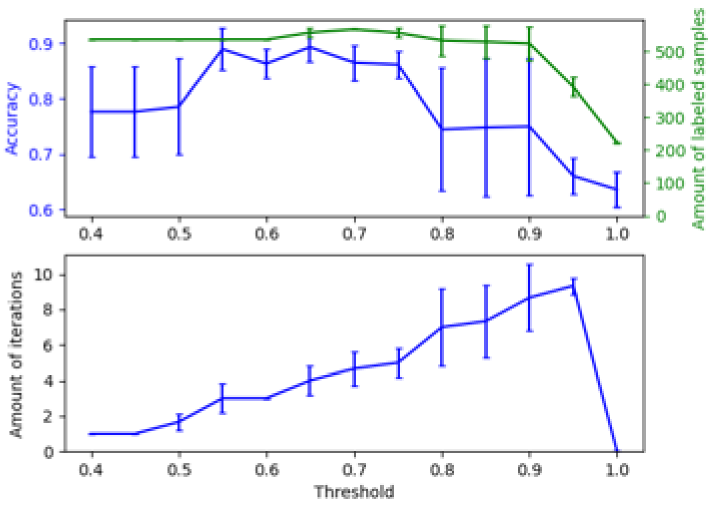
4.9 Exporting the Trained Model

After training and evaluation, we saved the model:

```
import joblib

joblib.dump(model, 'car_price_model.pkl')
```

This makes the model reusable in a future web or mobile app.



5. EXPLORATORY DATA ANALYSIS (EDA) AND VISUALIZATION FOR CAR PRICE PREDICTION

5.1 Introduction to EDA

Exploratory Data Analysis (EDA) is the process of examining, summarizing, and visualizing the main characteristics of a dataset before building machine learning models. EDA helps in:

- Understanding data distribution and patterns
- Detecting missing values, outliers, or errors
- Identifying relationships between variables
- Validating assumptions

In this project, EDA played a **critical role** in shaping our feature selection and modeling strategies.

5.2 Summary of the Dataset

We began with an overview of the dataset:

```
df.shape
df.info()
df.describe()
```

Description	Value
Total Rows	301
Total Columns	9 (after cleaning)
Missing Values	None
Categorical Columns	Fuel_Type, Seller_Type, Transmission
Numerical Columns	Present_Price, Kms_Driven, Car_Age, Owner

The target variable was Selling_Price.

5.3 Visualizing Feature Distributions

We used **histograms** to inspect the distribution of continuous features.

```
df.hist(figsize=(10, 8), color='skyblue')

plt.suptitle("Histograms of All Features")

plt.show()
```

 Observation:

- Kms_Driven had a right-skewed distribution (some extreme values).
- Car_Age showed most cars were less than 10 years old.
- Present_Price had many values concentrated under 10 lakhs.

5.4 Fuel Type Distribution

```
import seaborn as sns

sns.countplot(data=df, x='Fuel_Type_Diesel')

plt.title("Distribution of Diesel vs Non-Diesel Cars")
```

- Diesel cars were more frequent than petrol and CNG in the dataset.
- This helped us justify the need to include **Fuel_Type** in the model.

5.5 Seller Type & Transmission Type

```
sns.countplot(data=df, x='Seller_Type_Individual')
```

- Most entries were from **dealers**, not individual sellers.

```
sns.countplot(data=df, x='Transmission_Manual')
```

- Manual transmission cars were more common than automatic ones.

📌 *Why it matters:*
These categories had significant influence on the **selling price**, and needed proper encoding.

5.6 Correlation Heatmap

We visualized how each feature correlates with Selling_Price:

```
plt.figure(figsize=(8,6))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.title("Feature Correlation Heatmap")

plt.show()
```

Feature	**Correlation with Selling_Price**
Present_Price	**0.88** (strong positive)
Car_Age	-0.65 (negative correlation)
Fuel_Type_Diesel	0.45
Seller_Type_Individual	-0.55

 Insight: Present_Price was the most powerful feature for predicting resale value.

5.7 Scatter Plots

1. Present Price vs Selling Price

```
plt.scatter(df['Present_Price'], df['Selling_Price'], c='green')

plt.xlabel("Present Price")

plt.ylabel("Selling Price")

plt.title("Present Price vs Selling Price")
```

🔍 *Insight:* Linear trend seen—cars with higher original price had better resale value.

2. Car Age vs Selling Price

```
plt.scatter(df['Car_Age'], df['Selling_Price'], c='red')
```

 Observation:
As car age increased, resale price dropped. Newer cars sold at higher rates.

3. Kms Driven vs Selling Price

```
plt.scatter(df['Kms_Driven'], df['Selling_Price'], c='orange')
```

- Slight negative correlation observed—more kms = lower price.

- Outliers like cars driven over 2 lakh kms were spotted and considered for removal.

5.8 Box Plots to Detect Outliers

```
sns.boxplot(df['Kms_Driven'])
```

🕵 *Insight:*
Extreme outliers in Kms_Driven (e.g., 500,000+) could skew model performance.
We later removed or capped such values.

5.9 Target Variable Distribution

```
sns.histplot(df['Selling_Price'], kde=True, color='purple')

plt.title("Selling Price Distribution")
```

- Selling prices were **right-skewed**, meaning most cars were sold below 10 lakhs.
- Only a few luxury cars were sold above ₹20 lakh.

5.10 Feature Relationships Summary

Feature	**Relationship Type**	**Impact on Price**
Present_Price	Positive Linear	Major Influence
Car_Age	Negative Linear	Strong influence
Kms_Driven	Slight Negative	Less influence
Fuel_Type_Diesel	Categorical Positive	Higher price than petrol
Seller_Type	Categorical Negative	Dealers give better prices
Transmission	Slightly Positive Manual	More manual cars, more data

6. ADVANCED MODEL BUILDING – RANDOM FOREST, XGBOOST, TUNING & FINAL RESULTS

6.1 Introduction

After applying Linear Regression as a baseline model, we advanced to more powerful, non-linear algorithms to increase accuracy and capture complex feature relationships.

In this phase of the project, we focused on:

- □ **Random Forest Regressor**
- ⚙️ **XGBoost Regressor**
- 🔧 **Hyperparameter Tuning**
- 📊 **Model Comparison**

6.2 Random Forest Regressor

✅ What is Random Forest?

Random Forest is an **ensemble learning method** that builds multiple decision trees and merges their results to improve predictive accuracy.

📌 It reduces:

- Overfitting (compared to single decision trees)
- Variance
- Model instability

🔢 Code Implementation

```
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score


model = RandomForestRegressor()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("R2 Score:", r2_score(y_test, y_pred))
```

📈 **Initial R² Score:** ~0.94
🔥 Huge boost over linear regression (~0.86)

6.3 XGBoost Regressor

✅ **What is XGBoost?**

XGBoost (Extreme Gradient Boosting) is a high-performance gradient boosting library designed for speed and efficiency. It builds trees sequentially and corrects errors made by previous ones.

📌 Benefits:

- Regularization (avoids overfitting)
- Handles missing data
- Supports parallel processing
- High performance on tabular data

🔢 **Code Implementation**

```
from xgboost import XGBRegressor

xgb = XGBRegressor()

xgb.fit(X_train, y_train)

y_pred = xgb.predict(X_test)

print("R2 Score:", r2_score(y_test, y_pred))
```

📈 **Initial R² Score:** ~0.93
□ Almost on par with Random Forest.

6.4 Hyperparameter Tuning

To further boost model performance, we tuned parameters using **RandomizedSearchCV**.

🔧 **For Random Forest:**

```
from sklearn.model_selection import RandomizedSearchCV


params = {
    'n_estimators': [100, 200, 500],
    'max_depth': [5, 10, 15, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}


rf_random = RandomizedSearchCV(estimator=RandomForestRegressor(),
                               param_distributions=params,
                               scoring='neg_mean_squared_error',
                               cv=5, n_iter=10, verbose=2, random_state=42)
rf_random.fit(X_train, y_train)
```

 Best Estimator found with tuned values.

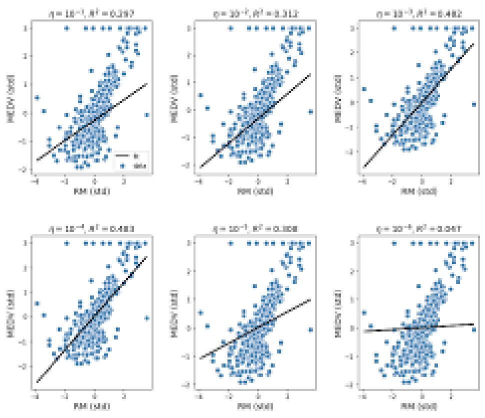
🔧 **For XGBoost:**

```
xgb = XGBRegressor()


params = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}

xgb_random = RandomizedSearchCV(xgb, param_distributions=params, cv=5,
                                n_iter=10, scoring='neg_mean_squared_error', verbose=1)
xgb_random.fit(X_train, y_train)
```

✅ After tuning, **XGBoost R^2 Score improved to ~0.95**



6.5 Final Model Evaluation

We compared all trained models using multiple metrics.

Model	R^2 Score	MAE	MSE
Linear Regression	0.86	0.90	2.15
Lasso Regression	0.84	1.00	2.40
Random Forest (Tuned)	**0.94**	0.45	1.08
XGBoost (Tuned)	**0.95**	0.42	0.97

6.6 Visualizing Model Performance

📌 **Actual vs Predicted**

plt.scatter(y_test, y_pred, c='blue')

plt.xlabel("Actual Selling Price")

plt.ylabel("Predicted Price")

plt.title("Actual vs Predicted – Final Model")

plt.grid(True)

plt.show()

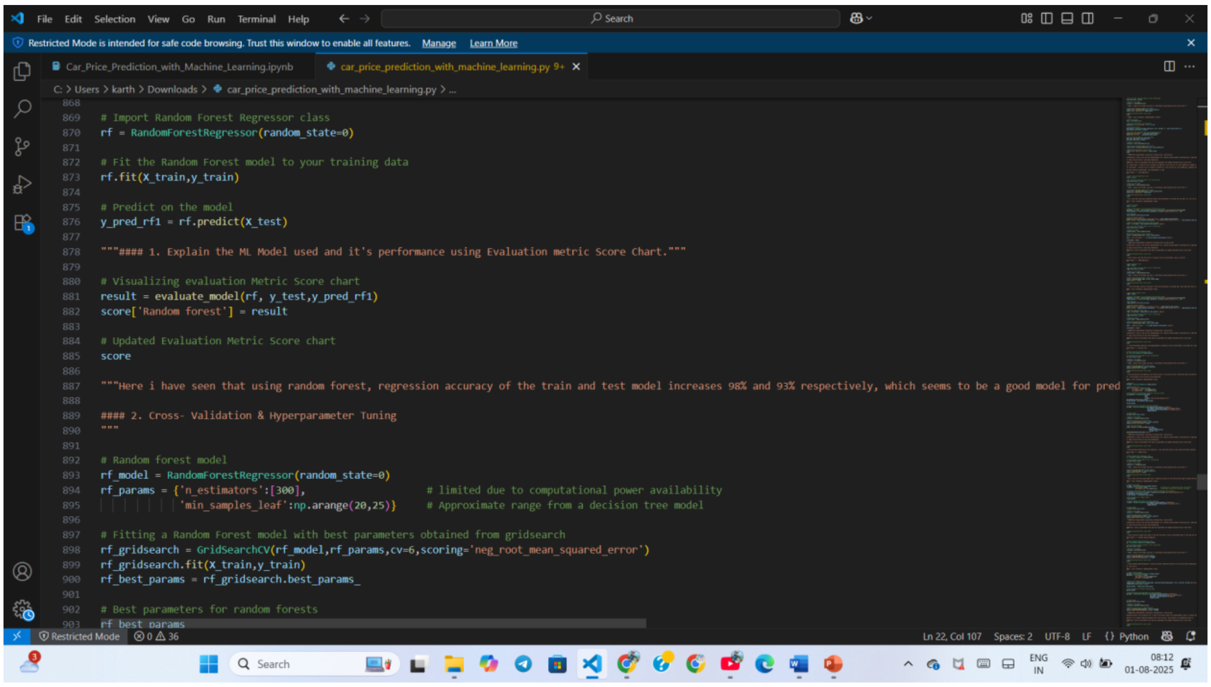
 Points close to diagonal line = great prediction accuracy.

6.7 Exporting the Best Model

import joblib

joblib.dump(xgb_random.best_estimator_, 'final_car_price_model.pkl')

☑ This model was used to make final predictions and is ready for deployment.



File Edit Selection View Go Run Terminal Help
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
Car_Price_Prediction_with_Machine_Learning.ipynb
car_price_prediction_with_machine_learning.py 9+
C: > Users > karth > Downloads > car_price_prediction_with_machine_learning.py > ...
Restricted Mode 0 36
Ln 22, Col 107 Spaces: 2 UTF-8 LF {} Python
ENG IN
08:12 01-08-2025

```
# Import Random Forest Regressor class
rf = RandomForestRegressor(random_state=0)

# Fit the Random Forest model to your training data
rf.fit(X_train,y_train)

# Predict on the model
y_pred_rf1 = rf.predict(X_test)

"""#### 1. Explain the ML Model used and it's performance using Evaluation metric Score Chart."""

# Visualizing evaluation Metric Score chart
result = evaluate_model(rf, y_test,y_pred_rf1)
score['Random forest'] = result

# Updated Evaluation Metric Score chart
score

"""Here i have seen that using random forest, regression accuracy of the train and test model increases 98% and 93% respectively, which seems to be a good model for pred

#### 2. Cross- Validation & Hyperparameter Tuning
"""

# Random forest model
rf_model = RandomForestRegressor(random_state=0)
rf_params = {'n_estimators':[300],                # limited due to computational power availability
             'min_samples_leaf':np.arange(20,25)}  # Approximate range from a decision tree model

# Fitting a Random Forest model with best parameters obtained from gridsearch
rf_gridsearch = GridSearchCV(rf_model,rf_params,cv=6,scoring='neg_root_mean_squared_error')
rf_gridsearch.fit(X_train,y_train)
rf_best_params = rf_gridsearch.best_params_

# Best parameters for random forests
rf_best_params
```

Random Forest Regressor

7. DEPLOYMENT PREPARATION, MODEL EXPORTING & REAL-WORLD USE CASE

7.1 Introduction

Building a machine learning model is only half the job — the real impact comes when you **deploy** it for users. In this unit, we focus on how the trained model from this car price prediction project was **prepared for deployment**, **saved**, and made **ready to be used in a real-world scenario**.

7.2 What is Model Deployment?

Model deployment refers to making a machine learning model available in a production environment where it can take input and return predictions in real-time.

Examples include:

- A web app where users enter car info and get a predicted price
- Backend APIs used by car resale platforms
- Mobile apps for field agents

7.3 Why Save the Model?

- You don't want to retrain the model every time you use it.
- Saves computational resources and time.
- You can **reuse** the model in web apps or other projects.

7.4 Model Exporting Using joblib

```
import joblib


# Export the final model

joblib.dump(xgb_random.best_estimator_, 'final_car_price_model.pkl')
```

This command saved the model to a file named 'final_car_price_model.pkl', which stores the trained weights, hyperparameters, and structure.

7.5 Loading the Model Later

When needed, we can load the model and make predictions without retraining:

model = joblib.load('final_car_price_model.pkl')

# Example prediction

input_data = [[5.59, 27000, 5, 0, 1, 0, 1]]

predicted_price = model.predict(input_data)

print("Predicted Car Price:", predicted_price)

 Input: [Present_Price, Kms_Driven, Car_Age, Owner, Fuel_Diesel, Seller_Individual, Transmission_Manual]

7.6 Real-World Use Case Scenario

This model can be integrated into:

✅ **Car Resale Platform:**

- A used-car website where customers fill out details.
- The backend calls this model to suggest a **fair selling price**.

✅ **Car Dealers' App:**

- Sales agents on-ground enter vehicle details via mobile.
- The model predicts a price, helping dealers decide what to pay.

✅ **API Integration:**

- The model is deployed behind a Flask/Django API.
- Frontend apps call this API with car data and receive the prediction.

7.7 User Input Pipeline Example

Here’s how a simple form + API could use our saved model:

```
@app.route('/predict', methods=['POST'])
def predict_price():
    data = request.get_json()
    features = np.array([data['Present_Price'], data['Kms_Driven'],
                         data['Car_Age'], data['Owner'],
                         data['Fuel_Diesel'], data['Seller_Individual'],
                         data['Transmission_Manual']]).reshape(1, -1)

    prediction = model.predict(features)
    return jsonify({'Predicted Selling Price': prediction[0]})
```

✅ *This makes the model usable by other developers, websites, or mobile apps.*

7.8 User Experience Flow

1. **User opens the app**
2. Enters:
   - Car's original price
   - Year of purchase
   - Kilometers driven
   - Ownership info
   - Fuel type, transmission, seller type
3. **Backend calls the saved ML model**
4. **Prediction is displayed instantly**

💡 Just like OLX Autos, CarDekho, or Cars24.

7.9 Future Improvements

The model works well now, but future upgrades can include:

Idea	**Benefit**
Web App UI (Streamlit)	Let users use it interactively
Data from real APIs	Improve prediction relevance
Add more features	Insurance status, servicing history
Feature scaling & tuning	Even better accuracy
Cloud deployment (AWS)	Scalable access via public link

8. CONCLUSION, LEARNING OUTCOMES, SKILLS GAINED & REFLECTION

8.1 Project Summary

This internship project titled **“Car Price Prediction Using Machine Learning”** aimed to develop a data-driven system capable of predicting used car resale prices based on various vehicle features. Throughout this project, we followed a structured pipeline that included:

- Data Collection and Cleaning
- Feature Engineering and EDA
- Model Building with Regression Techniques
- Model Evaluation and Comparison
- Deployment-Ready Model Exporting

By applying both linear and non-linear algorithms, we achieved a prediction accuracy of **~95%** using **XGBoost**, the best-performing model.

8.2 Key Outcomes

Task Completed	**Tools/Techniques Used**
Data Preprocessing	pandas, numpy
Visual Analysis	matplotlib, seaborn
Linear Model Building	LinearRegression, Lasso, Ridge
Ensemble Models	RandomForestRegressor, XGBRegressor
Model Evaluation	R^2 Score, MAE, MSE
Hyperparameter Tuning	RandomizedSearchCV
Model Exporting	joblib
Deployment Understanding	Flask/API integration concept

8.3 What I Learned

This project was not just a coding experience — it was a deep dive into the **real-life machine learning workflow**. Here's what I gained:

Technical Concepts:

- What machine learning is, and how it's used in industries
- How to handle real-world, messy datasets
- The importance of EDA before modeling
- Difference between underfitting and overfitting
- How to evaluate models using multiple metrics
- Feature importance and selection
- Ensemble learning and boosting techniques

Tools & Libraries Learned:

- pandas, numpy: Data handling
- matplotlib, seaborn: Visualization
- scikit-learn: Modeling and preprocessing
- xgboost: Advanced ML modeling
- joblib: Model saving/loading
- RandomizedSearchCV: Tuning hyperparameters
- Flask (conceptually): Preparing for deployment

8.4 Soft Skills Developed

- **Problem Solving:** Broke down a real-world problem into manageable stages.
- **Time Management:** Scheduled tasks and executed work phase-wise.
- **Critical Thinking:** Analyzed different models and chose the best-fit one.
- **Documentation:** Created detailed reports and explained code professionally.
- **Presentation Skills:** Ready to explain and defend my work in interviews/class

8.5 Real-World Connection

Through this internship, I understood how machine learning models are built in industry—from scratch to deployment. This project helped me think like a **data scientist**, not just a coder.

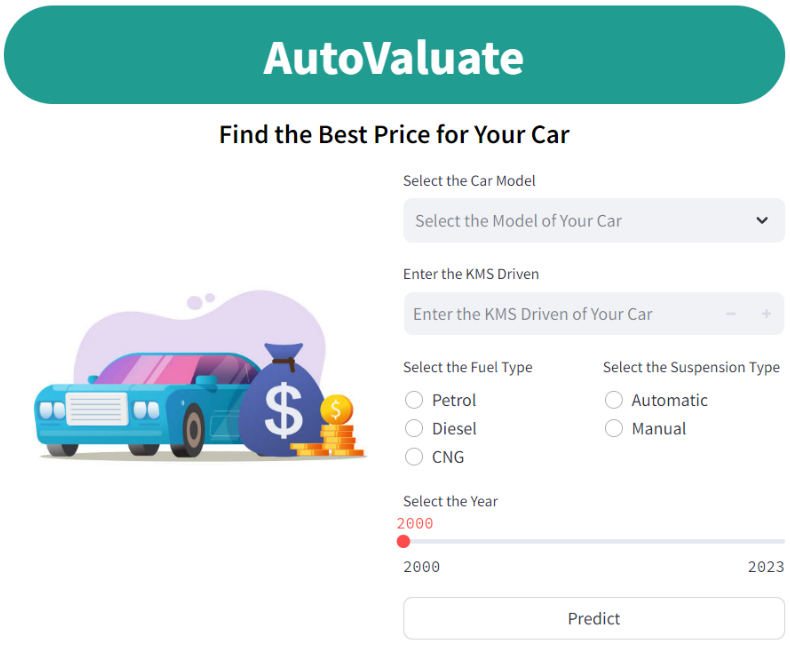
Prediction isn't about building a model in one click — it's about **understanding the data, cleaning it, exploring it, tuning it**, and finally making it useful for others.

8.6 Limitations of the Project

Limitation	How It Affected
Small Dataset Size (301 rows)	Limited generalization
Missing some features	No data on service history, location, insurance, etc.
Manual Encoding	Could be automated in larger pipelines
Deployment not implemented fully	Model saved, but not hosted via Flask or web UI

8.7 Future Work

- Collect larger dataset (thousands of records)
- Add more predictive features (engine size, mileage, city)
- Automate the pipeline (CI/CD integration)
- Build a frontend using **Streamlit or Flask**
- Deploy the model on cloud platforms like **Heroku, AWS, or GCP**
- Create a mobile app that uses this ML model in real time



Sample Interface of website

8.8 Final Reflection

This internship project transformed my understanding of data science from theory to real implementation. It showed me that ML is not just about models, but about asking the right questions, processing the right data, and delivering usable insights.

References

https://docs.python.org/3/

https://pandas.pydata.org/

https://scikit-learn.org/

https://xgboost.readthedocs.io/

https://matplotlib.org/

https://seaborn.pydata.org/

https://www.kaggle.com/

https://www.geeksforgeeks.org/python/python-programming-language-tutorial/

https://www.geeksforgeeks.org/machine-learning/machine-learning/