Project Title: **Car Price Prediction Using Machine Learning**

Subtitle: **Accurate Price Estimation Using Data-Driven Approaches**

Team Members:

* **TRISHA NAG**
* **SANGITA DUTTA**
* **SUBHAM MONDAL**

Project Mentor**:**

**Prof. Arnab Chakraborty**

Contents:

* **Project Objective & Scope**
* **Data Description**
* **Methodology**
* **Data Preprocessing**
* **Models Used**
* **Accuracy Comparison**
* **Inference**
* **Future Scope of Improvements**

OBJECTIVES:

**1. Predict Accurate Car Prices**

* Develop a machine learning model to estimate the price of a car based on its features such as:
  + Make and model
  + Year of manufacture
  + Mileage
  + Fuel type
  + Transmission type
  + Owner type

**2. Provide a Decision Support Tool**

* Assist potential buyers and sellers in determining fair market values.
* Help users make informed decisions while purchasing or selling used cars.

**3. Analyze Key Factors Influencing Car Prices**

* Identify and evaluate the most important factors that influence car prices, such as:
  + Depreciation trends based on age.
  + Effect of mileage on car value.
  + Regional or brand-related pricing trends.

**4. Enhance Transparency in the Used Car Market**

* Create a platform where users can predict prices without the influence of middlemen or subjective appraisals.

**5. Build a Scalable Solution**

* Ensure the model is scalable for different car types, regions, and datasets.
* Incorporate features that allow integration with real-world systems, like car dealer platforms or auction sites.

**6. Simplify User Interaction**

* Develop an intuitive web application (e.g., via Streamlit) where users can input car details and instantly receive predictions.
* Ensure the tool is user-friendly for non-technical users.

**7. Leverage Data Insights for Business Opportunities**

* Provide dealerships or market analysts with trends and patterns from car price data for better inventory management and sales strategies.

SCOPES:

The scope of a **Car Price Detection Project** is extensive and versatile, as it addresses multiple aspects of the automotive industry, consumer decision-making, and machine learning applications. Below are detailed scopes categorized into technical, business, and academic domains:

**1. Technical Scope**

* **Machine Learning Applications**:
  + Implementation of regression algorithms like Linear Regression, Random Forest, Gradient Boosting, and K-Nearest Neighbors (KNN).
  + Model evaluation and optimization using performance metrics (e.g., R², RMSE).
  + Feature engineering techniques for improving prediction accuracy.
* **Data Handling and Preprocessing**:
* Cleaning datasets with missing values, duplicates, or inconsistencies.
* Encoding categorical data for compatibility with machine learning models.
* Scaling features to normalize input data for distance-based algorithms.
* **Scalability**:
* Designing systems that can handle larger datasets with additional features, such as engine specifications, insurance records, or geographic factors.
* Deploying predictive systems that can cater to real-time user input.

**2. Business Scope**

* **Car Dealership Optimization**:
  + Empower dealerships to provide instant price evaluations for trade-ins.
  + Help businesses optimize their pricing strategy by using predicted prices.
* **E-commerce Platforms**:
  + Integration with car sales platforms to assist buyers and sellers with fair pricing.
  + Real-time suggestions for listing prices based on market trends and historical data.
* **Market Analysis:**

 Insights into car depreciation rates and resale value trends.

 Help manufacturers and analysts understand factors influencing car prices.

* **Loan and Insurance Sectors**:
  + Provide financial institutions with accurate car valuation for approving loans or setting insurance premiums.
  + Assist in detecting overvalued or undervalued assets to minimize risks.

**3. Consumer Applications**

* **Fair Pricing for Buyers and Sellers**:
  + Equip car buyers with tools to determine if a car's price is reasonable.
  + Enable sellers to maximize profits by accurately pricing their vehicles.
* **User-Friendly Interfaces**:
  + Use web applications like Streamlit to allow users to input car attributes and instantly get price estimates.
* **Transparency in Transactions**:
  + Reduce information asymmetry between buyers and sellers by offering unbiased price predictions.

**4. Academic and Research Scope**

* **Model Comparison**:
  + Compare traditional regression techniques with modern approaches like ensemble learning.
  + Use this project to understand the trade-offs between complexity and accuracy in machine learning models.
* **Algorithm Development**:
  + Explore novel algorithms or hybrid techniques to improve prediction.
  + Analyze the impact of different datasets and features on model performance.
* **Educational Tool**:
  + Teach students about the end-to-end process of a machine learning project: from data preprocessing to model deployment.

**5. Social and Ethical Scope**

* **Sustainability**:
  + Encourage reuse of older vehicles by providing a transparent resale market.
  + Support circular economies in the automotive industry.
* **Ethical AI Implementation**:
  + Address biases in the model that could unfairly price vehicles based on limited or skewed data.
* **Accessibility**:
  + Ensure the tool is affordable and accessible to a wide audience, including small-scale car sellers.

METHODOLOGY:

The methodology for predicting car prices using machine learning typically involves the following steps:

Data collection: Gather a comprehensive dataset of car features and prices.

Data preprocessing: Handle missing values, duplicates, and convert categorical variables to numerical formats.

Feature engineering: Identify the most important features that influence car prices.

Model training: Split the dataset into training and testing sets, and train a model using the training set.

Model evaluation: Assess the model's predictive performance using the test dataset.

Model deployment: Deploy the model as a web application so users can input car details and get price predictions.

PURPOSE:

The purpose of a car price prediction project is to create a model that can accurately estimate the price of a car based on various factors:

Help buyers and sellers: The model can help buyers determine fair prices for cars and help sellers create competitive prices for their listings.

Improve efficiency: Machine learning can help the automotive industry become more efficient and customer-friendly.

Provide a transparent valuation process: The model can provide an objective valuation process for vehicle sales.

Car price prediction projects use machine learning to analyze large amounts of data, including a vehicle's brand, model, mileage, year, condition, and more. The model is trained on a large dataset to learn patterns and relationships within the data, and then use that information to make predictions.

PROBLEM STATEMENT:

A car price prediction project's problem statement can be to predict a car's price based on various factors:

Used car price prediction: Predict the price of a used car based on features like the manufacturer, mileage, engine, power, and number of seats.

Factors affecting car pricing: Understand the factors that affect car pricing in a specific market, such as the American market.

Machine learning model: Build a machine learning model to predict car prices.

The goal of car price prediction is to create a model that can reliably estimate car prices. This can help car dealerships, buyers, and other businesses make more informed decisions about used car pricing.

Here are some other things to consider when working on a car price prediction project:

Machine learning: Machine learning is a crucial application for car price prediction.

Features: Some factors that affect car prices include brand reputation, car features, horsepower, and fuel efficiency.

Data: Train the model on a dataset of historical car sales data.

Research questions: Research questions can include how significantly an independent feature affects the dependent variable, or if a linear regression model is effective for prediction.

SCOPE:

The scope of a car price prediction project can include:

Data collection: Gathering a comprehensive dataset of car features and prices

Data preprocessing: Cleaning, transforming, and preparing the data for modeling

Feature engineering: Identifying the most important features for price prediction

Model building: Creating a machine learning model capable of predicting car prices

Model evaluation: Assessing the model's accuracy and performance using appropriate metrics

Deployment: Making the trained model available for car price predictions

The goal of a car price prediction project is to create a model that can reliably estimate automobile prices based on many features. The project can help car dealerships, auction houses, and individual sellers in the following ways:

Competitive prices: Sellers can determine competitive prices for their listings

Fair prices: Buyers can determine fair prices for cars they are interested in buying

Optimal pricing strategies: Sellers can use data-driven insights to determine optimal pricing strategies.

This project aims to help users predict the price of a car based on input features such as the make, model, year, mileage, and more. It leverages machine learning techniques to provide accurate price predictions.

**Features**

* Predict car prices based on various input features.
* User-friendly interface built with Streamlit.
* Real-time predictions with interactive input fields.
* Hosted as a web application for easy access.

**Installation**

To run this project locally, follow these steps:

1. Clone the repository:
2. git clone https://github.com/your-username/car-price-prediction.git
3. cd car-price-prediction
4. Install the required packages:

pip install -r requirements.txt

1. Run the Streamlit application:

streamlit run car\_price\_prediction.py

**Usage**

1. Open the web application in your browser.
2. Enter the required car details into the input fields.
3. Click the 'Predict' button to get the estimated car price.

**Demo**

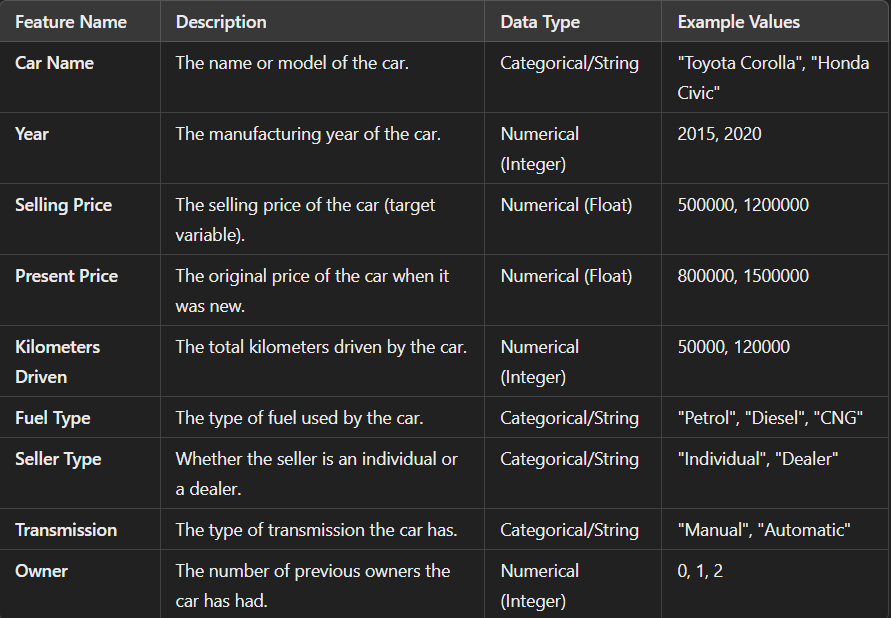
You can access the live demo of the Car Price Prediction System

**License**

This project is licensed under the GNU General Public License v2.0 License. See the LICENSE file for more details.

***Data description***

The description of the data with type and description of each Attribute is given/shown in the table.



Now we will pre-process the data. The methodology followed is given below:

• Checking for null values.

o If null values are present, we will fill them or drop the row containing the null

value based on the dataset.

• Checking for outliers.

o If outliers are present, they will either be removed or replaced by following a

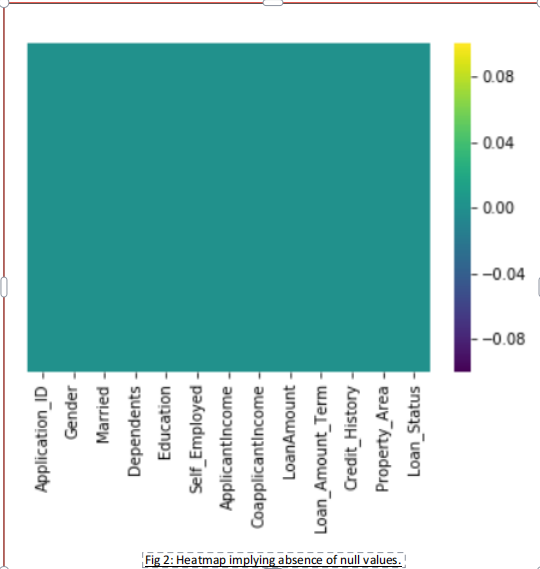
suitable method depending on the dataset.

Data Pre-processing

As the given dataset had Categorical and Non-categorical data mixed, we converted the

categorical data into non-categorical data accordingly. We converted the binary categories

into 0 and 1. We converted the other categorical attributes into suitable numerical values.



KNN Classifier

k -N N can be used for both classification and regression predictive probl e ms. However, it is

more widely used in classification problems in the industry. In pattern recognition, the k -

nearest neighbo rs algorithm (k -N N) is a non - parametric method used for classification and

regression. In both cases, the input consists of the k cl osest training examples in the feature

space. The output depends on whether k -N N is used for classification or regression.

In k -N N classification, the output is a class membership. An object is classified by a plurality

vote of its neighbors, with the o bject being assigned to the class most common among its k

nearest neighbors (k is a positive integer, typically small). For e. g. if k = 1, then the object is

simply assigned to the class of that single nearest neighbor. k -N N is a type of instance -b ased

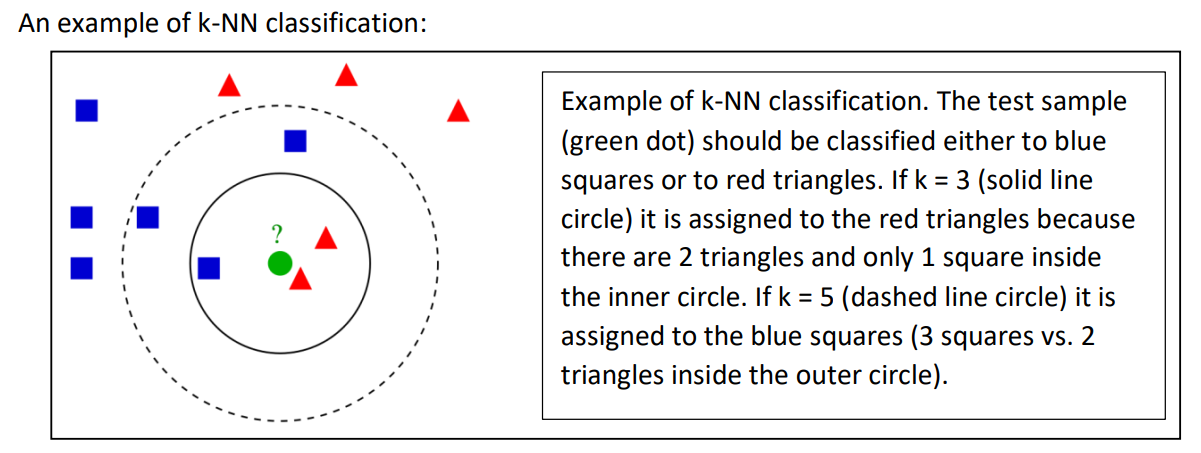
lea rning, or lazy learning, where the function is only approximated locally and all computation

is deferred until classification. The k -N N algorithm is among the simplest of all machine

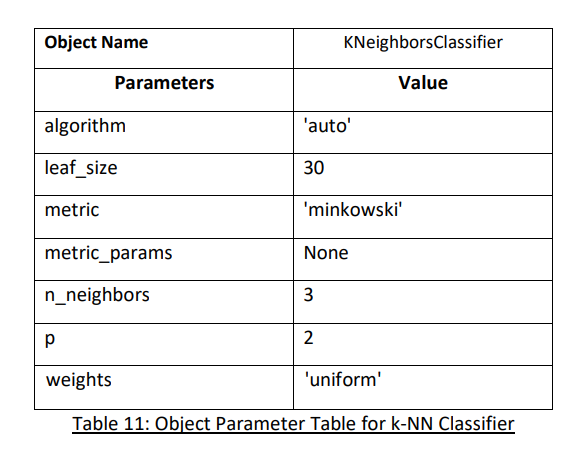
learning algorithms. The neighbors are taken from a set of objects for wh ich the class (for k -

NN classification) or the object property value (for k -N N regression) is known. This can be

thought of as the training set for the algorithm, though no explicit training step is required.

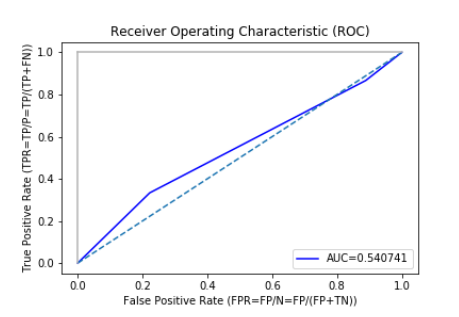


We used a GridSearchCV object to find the best optimum value of k. Our test results gave the value of k = 5. The object description of the k-NN Classifier used is given below:



Now we created a confusion matrix to view the actual and predicted test results.

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained k-NN model. The ROC curve is given below:



Naive Bayes Classifier

A Naive Bayes classifier is a probabilistic machine learning model that’s used for classification task. The crux of the classifier is based on the Bayes theorem. The fundamental Naive Bayes assumption is that each feature makes an:

• independent

• equal

contribution to the outcome. The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Bayes Theorem:

Bayes’ Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes’ theorem is stated mathematically as the following equation: 𝑃(𝐴|𝐵) = 𝑃(𝐴)𝑃(𝐵|𝐴) 𝑃(𝐵) where A and B are events and P(B) is the probability of occurrence of event B. Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence. P(A) is the priori of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance (here, it is event B). P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen. The class-data relation from the Bayes Theorem can be obtained as follows: 𝑃(𝐶𝑙𝑎𝑠𝑠|𝐷𝑎𝑡𝑎) = 𝑃(𝐶𝑙𝑎𝑠𝑠)𝑃(𝐷𝑎𝑡𝑎|𝐶𝑙𝑎𝑠𝑠) 𝑃(𝐷𝑎𝑡𝑎) Where, • P(Class|Data) = Posterior • P(Class) = Prior • P(Data|Class) = Likelihood • P(Data) = Marginal Probability In other words, it can be written as: 𝑃𝑜𝑠𝑡𝑒𝑟𝑖𝑜𝑟 = 𝑃𝑟𝑖𝑜𝑟 ∗ 𝐿𝑖𝑘𝑒𝑙𝑖ℎ𝑜𝑜𝑑 𝑀𝑎𝑟𝑔𝑖𝑛𝑎𝑙 𝑃𝑟𝑜𝑏𝑎𝑏𝑖𝑙𝑖𝑡𝑦 In application, we do not need to calculate the Marginal Probability for classification. We only need to calculate the numerator of the posterior for classification.

Types of Naive Bayes Classifier:

Multinomial Naive Bayes:

This is mostly used for document classification problem, i.e. whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

Bernoulli Naive Bayes:

This is similar to the multinomial Naive Bayes but the predictors are Boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

Gaussian Naive Bayes:

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.

Decision Tree:

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal, it’s also widely used in machine learning.

CODE:-

import pandas as pd

import datetime

import numpy as np

import streamlit

import xgboost as xgb

def main():

html\_temp="""

<div style ="background-color:pink; padding:10px; ">

<h2 style="color:black;text-align:center;"> Car Price Prediction Using ML</h2>

</div>

"""

model = xgb.XGBRegressor()

model.load\_model('xgb\_model.json')

streamlit.markdown(html\_temp,unsafe\_allow\_html=True)

streamlit.write('')

streamlit.write('')

streamlit.markdown("##### Are You Planning to sell your car ?\n###### So let's try to evaluating the price")

p1 = streamlit.number\_input("What is the current ex-showroom price of the Car (In Lakhs)",2.5,25.0,step=1.0)

p2 = streamlit.number\_input("What is the distance covered by the car in kilometers?",100,50000000,step=200)

s1 = streamlit.selectbox("What is the fuel type of the car?",('Petrol','Diesel','CNG'))

if s1 == 'Petrol':

p3=0

elif s1 == 'Diesel':

p3=1

elif s1 == 'CNG':

p3=2

s2 = streamlit.selectbox("Are U a Dealer or Individual?", ('Dealer', 'Individual'))

if s2 == 'Dealer':

p4=0

elif s2 == 'Individual':

p4=1

s3 = streamlit.selectbox("What is the transmission type?", ('Automatic', 'Manual'))

if s3 == 'Manual':

p5=0

elif s3 == 'Automatic':

p5=1

p6 = streamlit.slider("number of Owners the car previously had?",0,3)

date\_time = datetime.datetime.now()

years = streamlit.number\_input("In which year car was purchased", 1990,date\_time.year)

p7 = date\_time.year - years

data\_new = pd.DataFrame({

'Present\_Price': p1,

'Kms\_Driven': p2,

'Fuel\_Type': p3,

'Seller\_Type': p4,

'Transmission': p5,

'Owner': p6,

'Age': p7

}, index=[0])

try:

if streamlit.button('Predict'):

pred = model.predict(data\_new)

if pred >0:

streamlit.balloons()

streamlit.success("now you can sell your car for {:.2f} lakhs".format(pred[0]))

else:

streamlit.warning("You can't able to sell this car")

except:

streamlit.warning("Something Went Wrong please try again")

if \_\_name\_\_ == '\_\_main\_\_':

main()

ML CODE:

{

"cells": [

{

"cell\_type": "code",

"execution\_count": 3,

"metadata": {},

"outputs": [],

"source": [

"import warnings"

]

},

{

"cell\_type": "code",

"execution\_count": 4,

"metadata": {},

"outputs": [],

"source": [

"warnings.filterwarnings('ignore')"

]

},

{

"cell\_type": "code",

"execution\_count": 5,

"metadata": {},

"outputs": [],

"source": [

"import pandas as pd"

]

},

{

"cell\_type": "code",

"execution\_count": 6,

"metadata": {},

"outputs": [],

"source": [

"data = pd.read\_csv('car data.xls')"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 1. Display Top 5 Rows of The Dataset"

]

},

{

"cell\_type": "code",

"execution\_count": 7,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Car\_Name</th>\n",

" <th>Year</th>\n",

" <th>Selling\_Price</th>\n",

" <th>Present\_Price</th>\n",

" <th>Kms\_Driven</th>\n",

" <th>Fuel\_Type</th>\n",

" <th>Seller\_Type</th>\n",

" <th>Transmission</th>\n",

" <th>Owner</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>0</th>\n",

" <td>ritz</td>\n",

" <td>2014</td>\n",

" <td>3.35</td>\n",

" <td>5.59</td>\n",

" <td>27000</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <th>1</th>\n",

" <td>sx4</td>\n",

" <td>2013</td>\n",

" <td>4.75</td>\n",

" <td>9.54</td>\n",

" <td>43000</td>\n",

" <td>Diesel</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <th>2</th>\n",

" <td>ciaz</td>\n",

" <td>2017</td>\n",

" <td>7.25</td>\n",

" <td>9.85</td>\n",

" <td>6900</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <th>3</th>\n",

" <td>wagon r</td>\n",

" <td>2011</td>\n",

" <td>2.85</td>\n",

" <td>4.15</td>\n",

" <td>5200</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <th>4</th>\n",

" <td>swift</td>\n",

" <td>2014</td>\n",

" <td>4.60</td>\n",

" <td>6.87</td>\n",

" <td>42450</td>\n",

" <td>Diesel</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"</div>"

],

"text/plain": [

" Car\_Name Year Selling\_Price Present\_Price Kms\_Driven Fuel\_Type \\\n",

"0 ritz 2014 3.35 5.59 27000 Petrol \n",

"1 sx4 2013 4.75 9.54 43000 Diesel \n",

"2 ciaz 2017 7.25 9.85 6900 Petrol \n",

"3 wagon r 2011 2.85 4.15 5200 Petrol \n",

"4 swift 2014 4.60 6.87 42450 Diesel \n",

"\n",

" Seller\_Type Transmission Owner \n",

"0 Dealer Manual 0 \n",

"1 Dealer Manual 0 \n",

"2 Dealer Manual 0 \n",

"3 Dealer Manual 0 \n",

"4 Dealer Manual 0 "

]

},

"execution\_count": 7,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.head()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 2. Check Last 5 Rows of The Dataset"

]

},

{

"cell\_type": "code",

"execution\_count": 8,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Car\_Name</th>\n",

" <th>Year</th>\n",

" <th>Selling\_Price</th>\n",

" <th>Present\_Price</th>\n",

" <th>Kms\_Driven</th>\n",

" <th>Fuel\_Type</th>\n",

" <th>Seller\_Type</th>\n",

" <th>Transmission</th>\n",

" <th>Owner</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>296</th>\n",

" <td>city</td>\n",

" <td>2016</td>\n",

" <td>9.50</td>\n",

" <td>11.6</td>\n",

" <td>33988</td>\n",

" <td>Diesel</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <th>297</th>\n",

" <td>brio</td>\n",

" <td>2015</td>\n",

" <td>4.00</td>\n",

" <td>5.9</td>\n",

" <td>60000</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <th>298</th>\n",

" <td>city</td>\n",

" <td>2009</td>\n",

" <td>3.35</td>\n",

" <td>11.0</td>\n",

" <td>87934</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <th>299</th>\n",

" <td>city</td>\n",

" <td>2017</td>\n",

" <td>11.50</td>\n",

" <td>12.5</td>\n",

" <td>9000</td>\n",

" <td>Diesel</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" <tr>\n",

" <th>300</th>\n",

" <td>brio</td>\n",

" <td>2016</td>\n",

" <td>5.30</td>\n",

" <td>5.9</td>\n",

" <td>5464</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"</div>"

],

"text/plain": [

" Car\_Name Year Selling\_Price Present\_Price Kms\_Driven Fuel\_Type \\\n",

"296 city 2016 9.50 11.6 33988 Diesel \n",

"297 brio 2015 4.00 5.9 60000 Petrol \n",

"298 city 2009 3.35 11.0 87934 Petrol \n",

"299 city 2017 11.50 12.5 9000 Diesel \n",

"300 brio 2016 5.30 5.9 5464 Petrol \n",

"\n",

" Seller\_Type Transmission Owner \n",

"296 Dealer Manual 0 \n",

"297 Dealer Manual 0 \n",

"298 Dealer Manual 0 \n",

"299 Dealer Manual 0 \n",

"300 Dealer Manual 0 "

]

},

"execution\_count": 8,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.tail()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 3. Find Shape of Our Dataset (Number of Rows And Number of Columns)"

]

},

{

"cell\_type": "code",

"execution\_count": 9,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"(301, 9)"

]

},

"execution\_count": 9,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.shape"

]

},

{

"cell\_type": "code",

"execution\_count": 10,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Number of Rows 301\n",

"Number of Columns 9\n"

]

}

],

"source": [

"print(\"Number of Rows\",data.shape[0])\n",

"print(\"Number of Columns\",data.shape[1])"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 4. Get Information About Our Dataset Like the Total Number of Rows, Total Number of Columns, Datatypes of Each Column And Memory Requirement"

]

},

{

"cell\_type": "code",

"execution\_count": 11,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"<class 'pandas.core.frame.DataFrame'>\n",

"RangeIndex: 301 entries, 0 to 300\n",

"Data columns (total 9 columns):\n",

" # Column Non-Null Count Dtype \n",

"--- ------ -------------- ----- \n",

" 0 Car\_Name 301 non-null object \n",

" 1 Year 301 non-null int64 \n",

" 2 Selling\_Price 301 non-null float64\n",

" 3 Present\_Price 301 non-null float64\n",

" 4 Kms\_Driven 301 non-null int64 \n",

" 5 Fuel\_Type 301 non-null object \n",

" 6 Seller\_Type 301 non-null object \n",

" 7 Transmission 301 non-null object \n",

" 8 Owner 301 non-null int64 \n",

"dtypes: float64(2), int64(3), object(4)\n",

"memory usage: 21.3+ KB\n"

]

}

],

"source": [

"data.info()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 5. Check Null Values In The Dataset"

]

},

{

"cell\_type": "code",

"execution\_count": 13,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"Car\_Name 0\n",

"Year 0\n",

"Selling\_Price 0\n",

"Present\_Price 0\n",

"Kms\_Driven 0\n",

"Fuel\_Type 0\n",

"Seller\_Type 0\n",

"Transmission 0\n",

"Owner 0\n",

"dtype: int64"

]

},

"execution\_count": 13,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.isnull().sum()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 6. Get Overall Statistics About The Dataset"

]

},

{

"cell\_type": "code",

"execution\_count": 14,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Year</th>\n",

" <th>Selling\_Price</th>\n",

" <th>Present\_Price</th>\n",

" <th>Kms\_Driven</th>\n",

" <th>Owner</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>count</th>\n",

" <td>301.000000</td>\n",

" <td>301.000000</td>\n",

" <td>301.000000</td>\n",

" <td>301.000000</td>\n",

" <td>301.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>mean</th>\n",

" <td>2013.627907</td>\n",

" <td>4.661296</td>\n",

" <td>7.628472</td>\n",

" <td>36947.205980</td>\n",

" <td>0.043189</td>\n",

" </tr>\n",

" <tr>\n",

" <th>std</th>\n",

" <td>2.891554</td>\n",

" <td>5.082812</td>\n",

" <td>8.644115</td>\n",

" <td>38886.883882</td>\n",

" <td>0.247915</td>\n",

" </tr>\n",

" <tr>\n",

" <th>min</th>\n",

" <td>2003.000000</td>\n",

" <td>0.100000</td>\n",

" <td>0.320000</td>\n",

" <td>500.000000</td>\n",

" <td>0.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>25%</th>\n",

" <td>2012.000000</td>\n",

" <td>0.900000</td>\n",

" <td>1.200000</td>\n",

" <td>15000.000000</td>\n",

" <td>0.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>50%</th>\n",

" <td>2014.000000</td>\n",

" <td>3.600000</td>\n",

" <td>6.400000</td>\n",

" <td>32000.000000</td>\n",

" <td>0.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>75%</th>\n",

" <td>2016.000000</td>\n",

" <td>6.000000</td>\n",

" <td>9.900000</td>\n",

" <td>48767.000000</td>\n",

" <td>0.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>max</th>\n",

" <td>2018.000000</td>\n",

" <td>35.000000</td>\n",

" <td>92.600000</td>\n",

" <td>500000.000000</td>\n",

" <td>3.000000</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"</div>"

],

"text/plain": [

" Year Selling\_Price Present\_Price Kms\_Driven Owner\n",

"count 301.000000 301.000000 301.000000 301.000000 301.000000\n",

"mean 2013.627907 4.661296 7.628472 36947.205980 0.043189\n",

"std 2.891554 5.082812 8.644115 38886.883882 0.247915\n",

"min 2003.000000 0.100000 0.320000 500.000000 0.000000\n",

"25% 2012.000000 0.900000 1.200000 15000.000000 0.000000\n",

"50% 2014.000000 3.600000 6.400000 32000.000000 0.000000\n",

"75% 2016.000000 6.000000 9.900000 48767.000000 0.000000\n",

"max 2018.000000 35.000000 92.600000 500000.000000 3.000000"

]

},

"execution\_count": 14,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.describe()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 7. Data Preprocessing"

]

},

{

"cell\_type": "code",

"execution\_count": 16,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Car\_Name</th>\n",

" <th>Year</th>\n",

" <th>Selling\_Price</th>\n",

" <th>Present\_Price</th>\n",

" <th>Kms\_Driven</th>\n",

" <th>Fuel\_Type</th>\n",

" <th>Seller\_Type</th>\n",

" <th>Transmission</th>\n",

" <th>Owner</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>0</th>\n",

" <td>ritz</td>\n",

" <td>2014</td>\n",

" <td>3.35</td>\n",

" <td>5.59</td>\n",

" <td>27000</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"</div>"

],

"text/plain": [

" Car\_Name Year Selling\_Price Present\_Price Kms\_Driven Fuel\_Type \\\n",

"0 ritz 2014 3.35 5.59 27000 Petrol \n",

"\n",

" Seller\_Type Transmission Owner \n",

"0 Dealer Manual 0 "

]

},

"execution\_count": 16,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.head(1)"

]

},

{

"cell\_type": "code",

"execution\_count": 17,

"metadata": {},

"outputs": [],

"source": [

"import datetime"

]

},

{

"cell\_type": "code",

"execution\_count": 19,

"metadata": {},

"outputs": [],

"source": [

"date\_time = datetime.datetime.now()"

]

},

{

"cell\_type": "code",

"execution\_count": 22,

"metadata": {},

"outputs": [],

"source": [

"data['Age']=date\_time.year - data['Year']"

]

},

{

"cell\_type": "code",

"execution\_count": 23,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Car\_Name</th>\n",

" <th>Year</th>\n",

" <th>Selling\_Price</th>\n",

" <th>Present\_Price</th>\n",

" <th>Kms\_Driven</th>\n",

" <th>Fuel\_Type</th>\n",

" <th>Seller\_Type</th>\n",

" <th>Transmission</th>\n",

" <th>Owner</th>\n",

" <th>Age</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>0</th>\n",

" <td>ritz</td>\n",

" <td>2014</td>\n",

" <td>3.35</td>\n",

" <td>5.59</td>\n",

" <td>27000</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>8</td>\n",

" </tr>\n",

" <tr>\n",

" <th>1</th>\n",

" <td>sx4</td>\n",

" <td>2013</td>\n",

" <td>4.75</td>\n",

" <td>9.54</td>\n",

" <td>43000</td>\n",

" <td>Diesel</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>9</td>\n",

" </tr>\n",

" <tr>\n",

" <th>2</th>\n",

" <td>ciaz</td>\n",

" <td>2017</td>\n",

" <td>7.25</td>\n",

" <td>9.85</td>\n",

" <td>6900</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>5</td>\n",

" </tr>\n",

" <tr>\n",

" <th>3</th>\n",

" <td>wagon r</td>\n",

" <td>2011</td>\n",

" <td>2.85</td>\n",

" <td>4.15</td>\n",

" <td>5200</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>11</td>\n",

" </tr>\n",

" <tr>\n",

" <th>4</th>\n",

" <td>swift</td>\n",

" <td>2014</td>\n",

" <td>4.60</td>\n",

" <td>6.87</td>\n",

" <td>42450</td>\n",

" <td>Diesel</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>8</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"</div>"

],

"text/plain": [

" Car\_Name Year Selling\_Price Present\_Price Kms\_Driven Fuel\_Type \\\n",

"0 ritz 2014 3.35 5.59 27000 Petrol \n",

"1 sx4 2013 4.75 9.54 43000 Diesel \n",

"2 ciaz 2017 7.25 9.85 6900 Petrol \n",

"3 wagon r 2011 2.85 4.15 5200 Petrol \n",

"4 swift 2014 4.60 6.87 42450 Diesel \n",

"\n",

" Seller\_Type Transmission Owner Age \n",

"0 Dealer Manual 0 8 \n",

"1 Dealer Manual 0 9 \n",

"2 Dealer Manual 0 5 \n",

"3 Dealer Manual 0 11 \n",

"4 Dealer Manual 0 8 "

]

},

"execution\_count": 23,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.head()"

]

},

{

"cell\_type": "code",

"execution\_count": 24,

"metadata": {},

"outputs": [],

"source": [

"data.drop('Year',axis=1,inplace=True)"

]

},

{

"cell\_type": "code",

"execution\_count": 25,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Car\_Name</th>\n",

" <th>Selling\_Price</th>\n",

" <th>Present\_Price</th>\n",

" <th>Kms\_Driven</th>\n",

" <th>Fuel\_Type</th>\n",

" <th>Seller\_Type</th>\n",

" <th>Transmission</th>\n",

" <th>Owner</th>\n",

" <th>Age</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>0</th>\n",

" <td>ritz</td>\n",

" <td>3.35</td>\n",

" <td>5.59</td>\n",

" <td>27000</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>8</td>\n",

" </tr>\n",

" <tr>\n",

" <th>1</th>\n",

" <td>sx4</td>\n",

" <td>4.75</td>\n",

" <td>9.54</td>\n",

" <td>43000</td>\n",

" <td>Diesel</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>9</td>\n",

" </tr>\n",

" <tr>\n",

" <th>2</th>\n",

" <td>ciaz</td>\n",

" <td>7.25</td>\n",

" <td>9.85</td>\n",

" <td>6900</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>5</td>\n",

" </tr>\n",

" <tr>\n",

" <th>3</th>\n",

" <td>wagon r</td>\n",

" <td>2.85</td>\n",

" <td>4.15</td>\n",

" <td>5200</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>11</td>\n",

" </tr>\n",

" <tr>\n",

" <th>4</th>\n",

" <td>swift</td>\n",

" <td>4.60</td>\n",

" <td>6.87</td>\n",

" <td>42450</td>\n",

" <td>Diesel</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>8</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"</div>"

],

"text/plain": [

" Car\_Name Selling\_Price Present\_Price Kms\_Driven Fuel\_Type Seller\_Type \\\n",

"0 ritz 3.35 5.59 27000 Petrol Dealer \n",

"1 sx4 4.75 9.54 43000 Diesel Dealer \n",

"2 ciaz 7.25 9.85 6900 Petrol Dealer \n",

"3 wagon r 2.85 4.15 5200 Petrol Dealer \n",

"4 swift 4.60 6.87 42450 Diesel Dealer \n",

"\n",

" Transmission Owner Age \n",

"0 Manual 0 8 \n",

"1 Manual 0 9 \n",

"2 Manual 0 5 \n",

"3 Manual 0 11 \n",

"4 Manual 0 8 "

]

},

"execution\_count": 25,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.head()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"#### Outlier Removal"

]

},

{

"cell\_type": "code",

"execution\_count": 26,

"metadata": {},

"outputs": [],

"source": [

"import seaborn as sns"

]

},

{

"cell\_type": "code",

"execution\_count": 27,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<AxesSubplot:xlabel='Selling\_Price'>"

]

},

"execution\_count": 27,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"sns.boxplot(data['Selling\_Price'])"

]

},

{

"cell\_type": "code",

"execution\_count": 37,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"[35.0,\n",

" 33.0,\n",

" 23.5,\n",

" 23.0,\n",

" 23.0,\n",

" 23.0,\n",

" 20.75,\n",

" 19.99,\n",

" 19.75,\n",

" 18.75,\n",

" 18.0,\n",

" 17.0,\n",

" 16.0,\n",

" 14.9,\n",

" 14.73,\n",

" 14.5,\n",

" 14.25,\n",

" 12.9,\n",

" 12.5,\n",

" 11.75,\n",

" 11.5,\n",

" 11.45,\n",

" 11.25,\n",

" 11.25,\n",

" 11.25,\n",

" 10.9,\n",

" 10.25,\n",

" 10.11,\n",

" 9.7,\n",

" 9.65,\n",

" 9.5,\n",

" 9.25,\n",

" 9.25,\n",

" 9.25,\n",

" 9.15,\n",

" 9.1,\n",

" 8.99,\n",

" 8.75,\n",

" 8.65,\n",

" 8.55,\n",

" 8.5,\n",

" 8.4,\n",

" 8.4,\n",

" 8.35,\n",

" 8.25,\n",

" 8.25,\n",

" 7.9,\n",

" 7.75,\n",

" 7.75,\n",

" 7.75,\n",

" 7.5,\n",

" 7.5,\n",

" 7.5,\n",

" 7.45,\n",

" 7.45,\n",

" 7.45,\n",

" 7.4,\n",

" 7.25,\n",

" 7.25,\n",

" 7.2,\n",

" 7.05,\n",

" 6.95,\n",

" 6.85,\n",

" 6.75,\n",

" 6.7,\n",

" 6.6,\n",

" 6.5,\n",

" 6.5,\n",

" 6.45,\n",

" 6.4,\n",

" 6.25,\n",

" 6.25,\n",

" 6.15,\n",

" 6.1,\n",

" 6.0,\n",

" 6.0,\n",

" 6.0,\n",

" 6.0,\n",

" 5.95,\n",

" 5.95,\n",

" 5.9,\n",

" 5.85,\n",

" 5.85,\n",

" 5.8,\n",

" 5.75,\n",

" 5.75,\n",

" 5.65,\n",

" 5.5,\n",

" 5.5,\n",

" 5.5,\n",

" 5.5,\n",

" 5.5,\n",

" 5.4,\n",

" 5.4,\n",

" 5.35,\n",

" 5.3,\n",

" 5.3,\n",

" 5.25,\n",

" 5.25,\n",

" 5.25,\n",

" 5.25,\n",

" 5.25,\n",

" 5.25,\n",

" 5.25,\n",

" 5.2,\n",

" 5.15,\n",

" 5.11,\n",

" 5.0,\n",

" 4.95,\n",

" 4.95,\n",

" 4.9,\n",

" 4.9,\n",

" 4.85,\n",

" 4.8,\n",

" 4.8,\n",

" 4.75,\n",

" 4.75,\n",

" 4.75,\n",

" 4.75,\n",

" 4.75,\n",

" 4.75,\n",

" 4.65,\n",

" 4.6,\n",

" 4.5,\n",

" 4.5,\n",

" 4.5,\n",

" 4.5,\n",

" 4.5,\n",

" 4.5,\n",

" 4.5,\n",

" 4.4,\n",

" 4.4,\n",

" 4.4,\n",

" 4.35,\n",

" 4.15,\n",

" 4.1,\n",

" 4.1,\n",

" 4.0,\n",

" 4.0,\n",

" 4.0,\n",

" 4.0,\n",

" 4.0,\n",

" 3.95,\n",

" 3.95,\n",

" 3.9,\n",

" 3.9,\n",

" 3.8,\n",

" 3.75,\n",

" 3.75,\n",

" 3.65,\n",

" 3.6,\n",

" 3.51,\n",

" 3.5,\n",

" 3.5,\n",

" 3.49,\n",

" 3.45,\n",

" 3.35,\n",

" 3.35,\n",

" 3.25,\n",

" 3.25,\n",

" 3.25,\n",

" 3.15,\n",

" 3.1,\n",

" 3.1,\n",

" 3.1,\n",

" 3.1,\n",

" 3.0,\n",

" 3.0,\n",

" 3.0,\n",

" 3.0,\n",

" 2.95,\n",

" 2.95,\n",

" 2.9,\n",

" 2.9,\n",

" 2.9,\n",

" 2.85,\n",

" 2.85,\n",

" 2.85,\n",

" 2.75,\n",

" 2.75,\n",

" 2.7,\n",

" 2.65,\n",

" 2.65,\n",

" 2.65,\n",

" 2.55,\n",

" 2.55,\n",

" 2.5,\n",

" 2.5,\n",

" 2.35,\n",

" 2.25,\n",

" 2.25,\n",

" 2.25,\n",

" 2.1,\n",

" 2.0,\n",

" 1.95,\n",

" 1.95,\n",

" 1.75,\n",

" 1.7,\n",

" 1.65,\n",

" 1.5,\n",

" 1.45,\n",

" 1.35,\n",

" 1.35,\n",

" 1.35,\n",

" 1.25,\n",

" 1.25,\n",

" 1.2,\n",

" 1.2,\n",

" 1.2,\n",

" 1.15,\n",

" 1.15,\n",

" 1.15,\n",

" 1.15,\n",

" 1.11,\n",

" 1.1,\n",

" 1.1,\n",

" 1.1,\n",

" 1.05,\n",

" 1.05,\n",

" 1.05,\n",

" 1.05,\n",

" 1.05,\n",

" 1.0,\n",

" 0.95,\n",

" 0.9,\n",

" 0.9,\n",

" 0.8,\n",

" 0.78,\n",

" 0.75,\n",

" 0.75,\n",

" 0.75,\n",

" 0.75,\n",

" 0.72,\n",

" 0.65,\n",

" 0.65,\n",

" 0.65,\n",

" 0.65,\n",

" 0.6,\n",

" 0.6,\n",

" 0.6,\n",

" 0.6,\n",

" 0.6,\n",

" 0.6,\n",

" 0.6,\n",

" 0.6,\n",

" 0.55,\n",

" 0.55,\n",

" 0.52,\n",

" 0.51,\n",

" 0.5,\n",

" 0.5,\n",

" 0.5,\n",

" 0.5,\n",

" 0.5,\n",

" 0.48,\n",

" 0.48,\n",

" 0.48,\n",

" 0.48,\n",

" 0.45,\n",

" 0.45,\n",

" 0.45,\n",

" 0.45,\n",

" 0.45,\n",

" 0.45,\n",

" 0.45,\n",

" 0.45,\n",

" 0.42,\n",

" 0.42,\n",

" 0.4,\n",

" 0.4,\n",

" 0.4,\n",

" 0.4,\n",

" 0.4,\n",

" 0.38,\n",

" 0.38,\n",

" 0.35,\n",

" 0.35,\n",

" 0.35,\n",

" 0.35,\n",

" 0.31,\n",

" 0.3,\n",

" 0.3,\n",

" 0.3,\n",

" 0.27,\n",

" 0.25,\n",

" 0.25,\n",

" 0.25,\n",

" 0.25,\n",

" 0.25,\n",

" 0.2,\n",

" 0.2,\n",

" 0.2,\n",

" 0.2,\n",

" 0.2,\n",

" 0.2,\n",

" 0.18,\n",

" 0.17,\n",

" 0.16,\n",

" 0.15,\n",

" 0.12,\n",

" 0.1]"

]

},

"execution\_count": 37,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"sorted(data['Selling\_Price'],reverse=True)"

]

},

{

"cell\_type": "code",

"execution\_count": 41,

"metadata": {},

"outputs": [],

"source": [

"data = data[~(data['Selling\_Price']>=33.0) & (data['Selling\_Price']<=35.0)]"

]

},

{

"cell\_type": "code",

"execution\_count": 42,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"(299, 9)"

]

},

"execution\_count": 42,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.shape"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"#### Encoding the Categorical Columns"

]

},

{

"cell\_type": "code",

"execution\_count": 45,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Car\_Name</th>\n",

" <th>Selling\_Price</th>\n",

" <th>Present\_Price</th>\n",

" <th>Kms\_Driven</th>\n",

" <th>Fuel\_Type</th>\n",

" <th>Seller\_Type</th>\n",

" <th>Transmission</th>\n",

" <th>Owner</th>\n",

" <th>Age</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>0</th>\n",

" <td>ritz</td>\n",

" <td>3.35</td>\n",

" <td>5.59</td>\n",

" <td>27000</td>\n",

" <td>Petrol</td>\n",

" <td>Dealer</td>\n",

" <td>Manual</td>\n",

" <td>0</td>\n",

" <td>8</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"</div>"

],

"text/plain": [

" Car\_Name Selling\_Price Present\_Price Kms\_Driven Fuel\_Type Seller\_Type \\\n",

"0 ritz 3.35 5.59 27000 Petrol Dealer \n",

"\n",

" Transmission Owner Age \n",

"0 Manual 0 8 "

]

},

"execution\_count": 45,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.head(1)"

]

},

{

"cell\_type": "code",

"execution\_count": 47,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array(['Petrol', 'Diesel', 'CNG'], dtype=object)"

]

},

"execution\_count": 47,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data['Fuel\_Type'].unique()"

]

},

{

"cell\_type": "code",

"execution\_count": 48,

"metadata": {},

"outputs": [],

"source": [

"data['Fuel\_Type'] = data['Fuel\_Type'].map({'Petrol':0,'Diesel':1,'CNG':2})"

]

},

{

"cell\_type": "code",

"execution\_count": 49,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array([0, 1, 2], dtype=int64)"

]

},

"execution\_count": 49,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data['Fuel\_Type'].unique()"

]

},

{

"cell\_type": "code",

"execution\_count": 51,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array(['Dealer', 'Individual'], dtype=object)"

]

},

"execution\_count": 51,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data['Seller\_Type'].unique()"

]

},

{

"cell\_type": "code",

"execution\_count": 52,

"metadata": {},

"outputs": [],

"source": [

"data['Seller\_Type'] = data['Seller\_Type'].map({'Dealer':0,'Individual':1})"

]

},

{

"cell\_type": "code",

"execution\_count": 53,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array([0, 1], dtype=int64)"

]

},

"execution\_count": 53,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data['Seller\_Type'].unique()"

]

},

{

"cell\_type": "code",

"execution\_count": 55,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array(['Manual', 'Automatic'], dtype=object)"

]

},

"execution\_count": 55,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data['Transmission'].unique()"

]

},

{

"cell\_type": "code",

"execution\_count": 56,

"metadata": {},

"outputs": [],

"source": [

"data['Transmission'] =data['Transmission'].map({'Manual':0,'Automatic':1})"

]

},

{

"cell\_type": "code",

"execution\_count": 57,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array([0, 1], dtype=int64)"

]

},

"execution\_count": 57,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data['Transmission'].unique()"

]

},

{

"cell\_type": "code",

"execution\_count": 58,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Car\_Name</th>\n",

" <th>Selling\_Price</th>\n",

" <th>Present\_Price</th>\n",

" <th>Kms\_Driven</th>\n",

" <th>Fuel\_Type</th>\n",

" <th>Seller\_Type</th>\n",

" <th>Transmission</th>\n",

" <th>Owner</th>\n",

" <th>Age</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>0</th>\n",

" <td>ritz</td>\n",

" <td>3.35</td>\n",

" <td>5.59</td>\n",

" <td>27000</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>8</td>\n",

" </tr>\n",

" <tr>\n",

" <th>1</th>\n",

" <td>sx4</td>\n",

" <td>4.75</td>\n",

" <td>9.54</td>\n",

" <td>43000</td>\n",

" <td>1</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>9</td>\n",

" </tr>\n",

" <tr>\n",

" <th>2</th>\n",

" <td>ciaz</td>\n",

" <td>7.25</td>\n",

" <td>9.85</td>\n",

" <td>6900</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>5</td>\n",

" </tr>\n",

" <tr>\n",

" <th>3</th>\n",

" <td>wagon r</td>\n",

" <td>2.85</td>\n",

" <td>4.15</td>\n",

" <td>5200</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>11</td>\n",

" </tr>\n",

" <tr>\n",

" <th>4</th>\n",

" <td>swift</td>\n",

" <td>4.60</td>\n",

" <td>6.87</td>\n",

" <td>42450</td>\n",

" <td>1</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>0</td>\n",

" <td>8</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"</div>"

],

"text/plain": [

" Car\_Name Selling\_Price Present\_Price Kms\_Driven Fuel\_Type Seller\_Type \\\n",

"0 ritz 3.35 5.59 27000 0 0 \n",

"1 sx4 4.75 9.54 43000 1 0 \n",

"2 ciaz 7.25 9.85 6900 0 0 \n",

"3 wagon r 2.85 4.15 5200 0 0 \n",

"4 swift 4.60 6.87 42450 1 0 \n",

"\n",

" Transmission Owner Age \n",

"0 0 0 8 \n",

"1 0 0 9 \n",

"2 0 0 5 \n",

"3 0 0 11 \n",

"4 0 0 8 "

]

},

"execution\_count": 58,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"data.head()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 8. Store Feature Matrix In X and Response(Target) In Vector y"

]

},

{

"cell\_type": "code",

"execution\_count": 61,

"metadata": {},

"outputs": [],

"source": [

"X = data.drop(['Car\_Name','Selling\_Price'],axis=1)\n",

"y = data['Selling\_Price']"

]

},

{

"cell\_type": "code",

"execution\_count": 63,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"0 3.35\n",

"1 4.75\n",

"2 7.25\n",

"3 2.85\n",

"4 4.60\n",

" ... \n",

"296 9.50\n",

"297 4.00\n",

"298 3.35\n",

"299 11.50\n",

"300 5.30\n",

"Name: Selling\_Price, Length: 299, dtype: float64"

]

},

"execution\_count": 63,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"y"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 9. Splitting The Dataset Into The Training Set And Test Set"

]

},

{

"cell\_type": "code",

"execution\_count": 64,

"metadata": {},

"outputs": [],

"source": [

"from sklearn.model\_selection import train\_test\_split"

]

},

{

"cell\_type": "code",

"execution\_count": 65,

"metadata": {},

"outputs": [],

"source": [

"X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.20,random\_state=42)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 10. Import The models"

]

},

{

"cell\_type": "code",

"execution\_count": 67,

"metadata": {},

"outputs": [],

"source": [

"from sklearn.linear\_model import LinearRegression\n",

"from sklearn.ensemble import RandomForestRegressor\n",

"from sklearn.ensemble import GradientBoostingRegressor\n",

"from xgboost import XGBRegressor"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 11. Model Training"

]

},

{

"cell\_type": "code",

"execution\_count": 68,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"XGBRegressor(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,\n",

" colsample\_bynode=1, colsample\_bytree=1, gamma=0, gpu\_id=-1,\n",

" importance\_type='gain', interaction\_constraints='',\n",

" learning\_rate=0.300000012, max\_delta\_step=0, max\_depth=6,\n",

" min\_child\_weight=1, missing=nan, monotone\_constraints='()',\n",

" n\_estimators=100, n\_jobs=4, num\_parallel\_tree=1, random\_state=0,\n",

" reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=1, subsample=1,\n",

" tree\_method='exact', validate\_parameters=1, verbosity=None)"

]

},

"execution\_count": 68,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"lr = LinearRegression()\n",

"lr.fit(X\_train,y\_train)\n",

"\n",

"rf = RandomForestRegressor()\n",

"rf.fit(X\_train,y\_train)\n",

"\n",

"xgb = GradientBoostingRegressor()\n",

"xgb.fit(X\_train,y\_train)\n",

"\n",

"xg = XGBRegressor()\n",

"xg.fit(X\_train,y\_train)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 12. Prediction on Test Data"

]

},

{

"cell\_type": "code",

"execution\_count": 69,

"metadata": {},

"outputs": [],

"source": [

"y\_pred1 = lr.predict(X\_test)\n",

"y\_pred2 = rf.predict(X\_test)\n",

"y\_pred3 = xgb.predict(X\_test)\n",

"y\_pred4 = xg.predict(X\_test)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 13. Evaluating the Algorithm"

]

},

{

"cell\_type": "code",

"execution\_count": 70,

"metadata": {},

"outputs": [],

"source": [

"from sklearn import metrics"

]

},

{

"cell\_type": "code",

"execution\_count": 71,

"metadata": {},

"outputs": [],

"source": [

"score1 = metrics.r2\_score(y\_test,y\_pred1)\n",

"score2 = metrics.r2\_score(y\_test,y\_pred2)\n",

"score3 = metrics.r2\_score(y\_test,y\_pred3)\n",

"score4 = metrics.r2\_score(y\_test,y\_pred4)"

]

},

{

"cell\_type": "code",

"execution\_count": 72,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.6790884983129399 0.742728169066639 0.8814942870514745 0.8864839405756888\n"

]

}

],

"source": [

"print(score1,score2,score3,score4)"

]

},

{

"cell\_type": "code",

"execution\_count": 73,

"metadata": {},

"outputs": [],

"source": [

"final\_data = pd.DataFrame({'Models':['LR','RF','GBR','XG'],\n",

" \"R2\_SCORE\":[score1,score2,score3,score4]})"

]

},

{

"cell\_type": "code",

"execution\_count": 74,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>Models</th>\n",

" <th>R2\_SCORE</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>0</th>\n",

" <td>LR</td>\n",

" <td>0.679088</td>\n",

" </tr>\n",

" <tr>\n",

" <th>1</th>\n",

" <td>RF</td>\n",

" <td>0.742728</td>\n",

" </tr>\n",

" <tr>\n",

" <th>2</th>\n",

" <td>GBR</td>\n",

" <td>0.881494</td>\n",

" </tr>\n",

" <tr>\n",

" <th>3</th>\n",

" <td>XG</td>\n",

" <td>0.886484</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"</div>"

],

"text/plain": [

" Models R2\_SCORE\n",

"0 LR 0.679088\n",

"1 RF 0.742728\n",

"2 GBR 0.881494\n",

"3 XG 0.886484"

]

},

"execution\_count": 74,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"final\_data"

]

},

{

"cell\_type": "code",

"execution\_count": 75,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<AxesSubplot:xlabel='Models', ylabel='R2\_SCORE'>"

]

},

"execution\_count": 75,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"sns.barplot(final\_data['Models'],final\_data['R2\_SCORE'])"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 14. Save The Model"

]

},

{

"cell\_type": "code",

"execution\_count": 76,

"metadata": {},

"outputs": [],

"source": [

"xg = XGBRegressor()\n",

"xg\_final = xg.fit(X,y)"

]

},

{

"cell\_type": "code",

"execution\_count": 77,

"metadata": {},

"outputs": [],

"source": [

"import joblib"

]

},

{

"cell\_type": "code",

"execution\_count": 78,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"['car\_price\_predictor']"

]

},

"execution\_count": 78,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"joblib.dump(xg\_final,'car\_price\_predictor')"

]

},

{

"cell\_type": "code",

"execution\_count": 79,

"metadata": {},

"outputs": [],

"source": [

"model = joblib.load('car\_price\_predictor')"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### 15. Prediction on New Data"

]

},

{

"cell\_type": "code",

"execution\_count": 80,

"metadata": {},

"outputs": [],

"source": [

"import pandas as pd\n",

"data\_new = pd.DataFrame({\n",

" 'Present\_Price':5.59,\n",

" 'Kms\_Driven':27000,\n",

" 'Fuel\_Type':0,\n",

" 'Seller\_Type':0,\n",

" 'Transmission':0,\n",

" 'Owner':0,\n",

" 'Age':8\n",

"},index=[0])"

]

},

{

"cell\_type": "code",

"execution\_count": 81,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array([3.3666973], dtype=float32)"

]

},

"execution\_count": 81,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"model.predict(data\_new)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### GUI"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"from tkinter import \*\n",

"import joblib\n",

"\n",

"def show\_entry\_fields():\n",

" p1=float(e1.get())\n",

" p2=float(e2.get())\n",

" p3=float(e3.get())\n",

" p4=float(e4.get())\n",

" p5=float(e5.get())\n",

" p6=float(e6.get())\n",

" p7=float(e7.get())\n",

" \n",

" model = joblib.load('car\_price\_predictor')\n",

" data\_new = pd.DataFrame({\n",

" 'Present\_Price':p1,\n",

" 'Kms\_Driven':p2,\n",

" 'Fuel\_Type':p3,\n",

" 'Seller\_Type':p4,\n",

" 'Transmission':p5,\n",

" 'Owner':p6,\n",

" 'Age':p7\n",

"},index=[0])\n",

" result=model.predict(data\_new)\n",

" Label(master, text=\"Car Purchase amount\").grid(row=8)\n",

" Label(master, text=result).grid(row=10)\n",

" print(\"Car Purchase amount\", result[0])\n",

" \n",

"master = Tk()\n",

"master.title(\"Car Price Prediction Using Machine Learning\")\n",

"label = Label(master, text = \"Car Price Prediction Using Machine Learning\"\n",

" , bg = \"black\", fg = \"white\"). \\\n",

" grid(row=0,columnspan=2)\n",

"\n",

"\n",

"Label(master, text=\"Present\_Price\").grid(row=1)\n",

"Label(master, text=\"Kms\_Driven\").grid(row=2)\n",

"Label(master, text=\"Fuel\_Type\").grid(row=3)\n",

"Label(master, text=\"Seller\_Type\").grid(row=4)\n",

"Label(master, text=\"Transmission\").grid(row=5)\n",

"Label(master, text=\"Owner\").grid(row=6)\n",

"Label(master, text=\"Age\").grid(row=7)\n",

"\n",

"\n",

"e1 = Entry(master)\n",

"e2 = Entry(master)\n",

"e3 = Entry(master)\n",

"e4 = Entry(master)\n",

"e5 = Entry(master)\n",

"e6 = Entry(master)\n",

"e7 = Entry(master)\n",

"\n",

"\n",

"e1.grid(row=1, column=1)\n",

"e2.grid(row=2, column=1)\n",

"e3.grid(row=3, column=1)\n",

"e4.grid(row=4, column=1)\n",

"e5.grid(row=5, column=1)\n",

"e6.grid(row=6, column=1)\n",

"e7.grid(row=7, column=1)\n",

"\n",

"\n",

"Button(master, text='Predict', command=show\_entry\_fields).grid()\n",

"\n",

"mainloop()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

}

],

"metadata": {

"kernelspec": {

"display\_name": "Python 3",

"language": "python",

"name": "python3"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.7.9"

}

},

"nbformat": 4,

"nbformat\_minor": 4

}

Future Scope of Improvement:

Various banking institution can use these models and modify them according to their needs to use in their loan approval status. This will reduce the manual labour and time spent on determining whether to approve a loan application. • Customers who intend to take a loan can use these trained models to check whether their loan application will be approved or not. The trained models would be required to be implemented in a platform or interface easily accessible as well as with an easy GUI. • We saw a high value of correlation of "Married" attribute with our target attribute. But the feature importance of "Married" attribute was significantly lower. With more data and further analysis, it might be possible to describe the reason of this mismatch. • No loan application having value "Rural" in "Property\_Area" attribute was approved. With further research and more data for analysis, a more decisive conclusion can be made.

Certificate

This is to certify that Mr. Subham Mondal ,Asansol Engineering College; Roll number: 10800222003, has successfully completed a project on Car Price Prediction using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

--------------------------------------------------

Prof. Arnab Chakraborty

Certificate

This is to certify that Ms. Trisha Nag, Asansol Engineering College; Roll number: 10800222009, has successfully completed a project on Car Price Prediction using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

--------------------------------------------------

Prof. Arnab Chakraborty

Certificate

This is to certify that Ms. Sangita Dutta, Asansol Engineering College; Roll number: 10800222010, has successfully completed a project on Car Price Prediction using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

--------------------------------------------------

Prof. Arnab Chakraborty