INSTITUTE OF TECHNOLOGY
NAAC ACCREDITED 'A+' GRADE

Car Price Prediction using K-Nearest Neighbors (KNN)

AIML ASSIGNMENT REPORT

1CS101CC22-Introduction to AI-ML

Namdar Mohammad Mehdi
24BEI041
Lakshya Jain
24BEI043

Car Price Prediction using K-Nearest Neighbors (KNN)

1. Introduction

Car price prediction is a significant application in the field of machine learning and data science. The ability to accurately predict the price of a used car based on its features can help buyers and sellers make informed decisions. In this project, we have implemented the K-Nearest Neighbors (KNN) algorithm to predict the selling price of used cars using real-world data.

2. Dataset Description

We have used the CarDekho dataset, which contains various attributes of used cars, including:

• Year – Manufacturing year of the car
• Mileage – Distance the car can travel per liter of fuel
• Engine – Engine capacity in CC
• Power – Horsepower of the car
• Selling Price – Price at which the car is sold (Target Variable)

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

# Load the dataset
file_path = "cardekho_dataset.csv"  # Change if needed
df = pd.read_csv(file_path)

# Display first few rows
print(df.head())
```

```
   Unnamed: 0        car_name    brand     model  vehicle_age  km_driven  \
0           0     Maruti Alto   Maruti      Alto            9     120000
1           1   Hyundai Grand  Hyundai     Grand            5      20000
2           2     Hyundai i20  Hyundai       i20           11      60000
3           3     Maruti Alto   Maruti      Alto            9      37000
4           4   Ford Ecosport     Ford  Ecosport            6      30000

  seller_type fuel_type transmission_type  mileage  engine  max_power  seats  \
0  Individual    Petrol            Manual    19.70     796      46.30      5
1  Individual    Petrol            Manual    18.90    1197      82.00      5
2  Individual    Petrol            Manual    17.00    1197      80.00      5
3  Individual    Petrol            Manual    20.92     998      67.10      5
4      Dealer    Diesel            Manual    22.77    1498      98.59      5

   selling_price
0         120000
1         550000
2         215000
3         226000
4         570000
```

3. Data Preprocessing

Before applying our model, we performed the following preprocessing steps:

• Handling missing values
• Removing unnecessary columns (e.g., 'Seats')
• Feature scaling and normalization for better accuracy
• Splitting data into training (80%) and testing (20%)

```python
print("Basic Statistical Analysis:\n",
df.describe())
```

```
Basic Statistical Analysis:
          Unnamed: 0   vehicle_age     km_driven       mileage        engine  \
count  15411.000000  15411.000000  1.541100e+04  15411.000000  15411.000000
mean    9811.857699      6.036338  5.561648e+04     19.701151   1486.057751
std     5643.418542      3.013291  5.161855e+04      4.171265    521.106696
min        0.000000      0.000000  1.000000e+02      4.000000    793.000000
25%     4906.500000      4.000000  3.000000e+04     17.000000   1197.000000
50%     9872.000000      6.000000  5.000000e+04     19.670000   1248.000000
75%    14668.500000      8.000000  7.000000e+04     22.700000   1582.000000
max    19543.000000     29.000000  3.800000e+06     33.540000   6592.000000

          max_power         seats  selling_price
count  15411.000000  15411.000000   1.541100e+04
mean     100.588254      5.325482   7.749711e+05
std       42.972979      0.807628   8.941284e+05
min       38.400000      0.000000   4.000000e+04
25%       74.000000      5.000000   3.850000e+05
50%       88.500000      5.000000   5.560000e+05
75%      117.300000      5.000000   8.250000e+05
max      626.000000      9.000000   3.950000e+07
```

```python
# Q-2: Drop Null Records & Encode Categorical Data
# Drop unnecessary columns
drop_cols = ['mileage', 'engine', 'max_power', 'seats', 'seller_type', 'fuel_type', 'car_name', 'transmission_type']
df = df.drop(columns=drop_cols)

# Label Encoding for Categorical Data
label_encoders = {}
for col in ['brand', 'model']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

4. Methodology

4.1 Algorithm: K-Nearest Neighbors (KNN)

KNN is a simple, yet powerful, algorithm that predicts values based on the similarity of data points. It works as follows:

1. Select the number of neighbors (k).
2. Compute the distance between the new data point and all other points in the dataset.
3. Select k nearest points.
4. Compute the average price of the selected points (for regression problems).
5. Assign the predicted price.

5. Code Implementation

Training and Testing:

```
# Predict log prices
y_pred_log = knn.predict(X_test)

# Convert log predictions back to normal price
y_pred = np.expm1(y_pred_log)

print("Predictions made successfully!")
```

```
# Calculate Error Metrics
mae = mean_absolute_error(np.expm1(y_test), y_pred)  #
mse = mean_squared_error(np.expm1(y_test), y_pred)
r2 = r2_score(np.expm1(y_test), y_pred)
rmse=np.sqrt(mse)
print(f"Mean Absolute Error (MAE): ₹{mae:,.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R² Score: {r2:.2f}")
print(f"Root Mean Squared Error (RMSE):{rmse:.2f}")
```

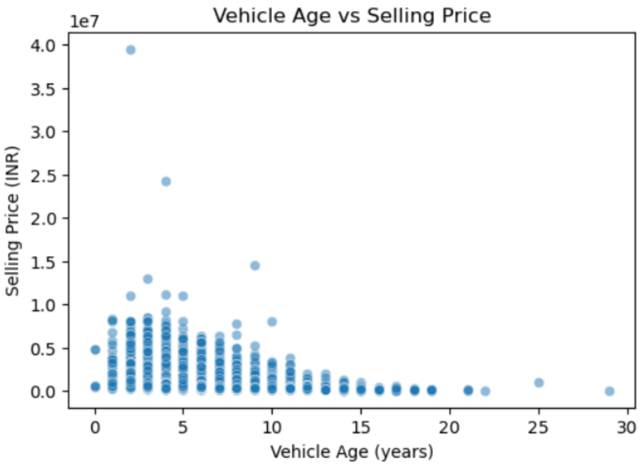
```
# Q-5: Train KNN Model
# Train KNN model
knn = KNeighborsRegressor(n_neighbors=2, weights='distance')  # Weighted KNN
knn.fit(X_train, y_train)

print("KNN Model Trained Successfully!")
```

Considering Depreciation for a realistic outcome

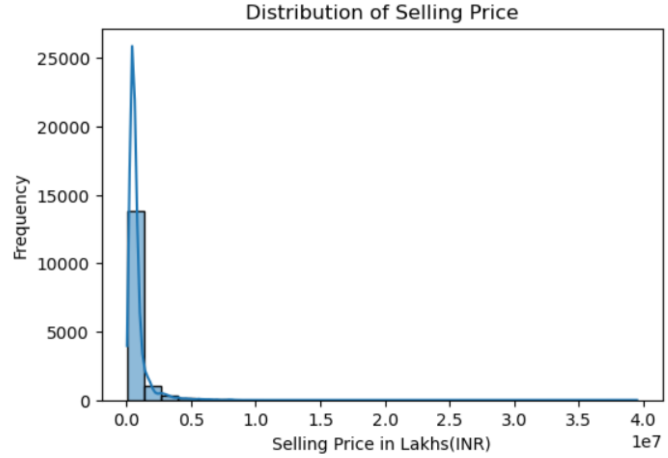
```
def apply_realistic_depreciation(predicted_price, vehicle_age):
    if vehicle_age == 0:
        return predicted_price  # New car, no depreciation
    elif vehicle_age <= 3:
        return predicted_price * (0.75)  # 25% depreciation in 3 years
    elif vehicle_age <= 5:
        return predicted_price * (0.60)  # 40% depreciation in 5 years
    elif vehicle_age <= 10:
        return predicted_price * (0.40)  # 60% depreciation in 10 years
    else:
        return predicted_price * (0.30)  # Older cars retain ~30% value

# Modify prediction function to include depreciation
def predict_car_price(brand_name, model_name, vehicle_age, km_driven):
    # Encode brand & model
    if brand_name in label_encoders['brand'].classes_ and model_name in label_encoders['model'].classes_:
        brand_encoded = label_encoders['brand'].transform([brand_name])[0]
        model_encoded = label_encoders['model'].transform([model_name])[0]
    else:
        return "Error: Invalid brand or model name."

    # Prepare input & scale it
    input_data = np.array([[brand_encoded, model_encoded, vehicle_age, km_driven]])
    input_data = scaler.transform(input_data)

    # Predict log price & convert back
    predicted_log_price = knn.predict(input_data)[0]
    predicted_price = np.expm1(predicted_log_price)

    # Apply improved depreciation formula
    final_price = apply_realistic_depreciation(predicted_price, vehicle_age)
```

6. Statistical Analysis & Visualizations

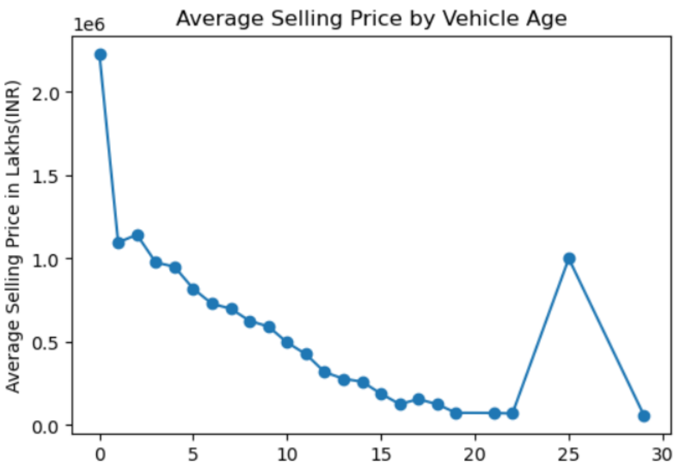
6.1 Vehicle Age vs Selling Price



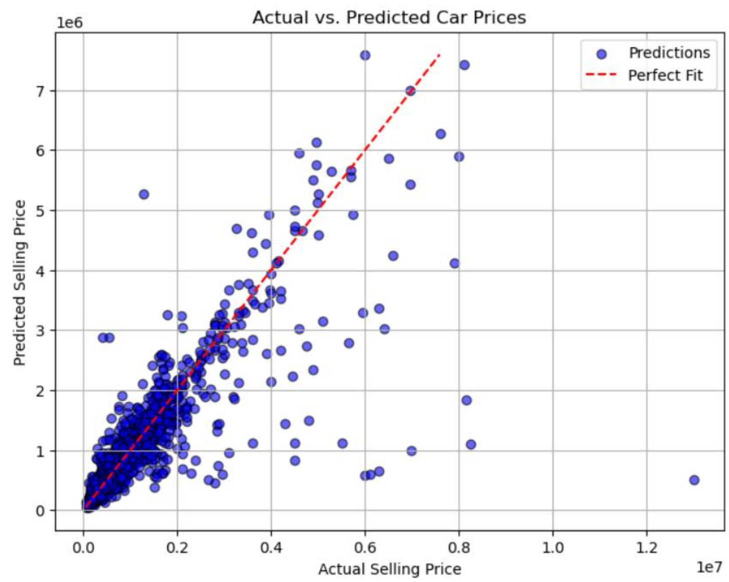
6.2 Distribution of Selling Price



6.3 Average Selling Price by Age



6.4 Expected Price vs Real Price



7. Model Evaluation

Metric	*Value*
MAE	₹163,251.63
MSE	₹236346207109.23
RMSE	₹486154.51
R^2	0.69

8. Model Deployment using Streamlit

To make our model interactive and usable, we implemented it using **Streamlit**. This allows users to enter car details and receive instant price predictions without needing a complex backend setup.

8.1 Saving the Model using Joblib

We saved the trained model using *Joblib* so that it could be loaded without retraining:
This allows us to quickly reuse the model for prediction.

```
import joblib  # Library to save/load models

# Save the trained KNN model
joblib.dump(knn, "knn_car_price_model.pkl")

# Save the StandardScaler used for feature scaling
joblib.dump(scaler, "scaler.pkl")

# Save the LabelEncoders for brand & model
joblib.dump(label_encoders, "label_encoders.pkl")

print("Model, scaler, and encoders saved successfully!")
```

8.2 Building the UI with Streamlit

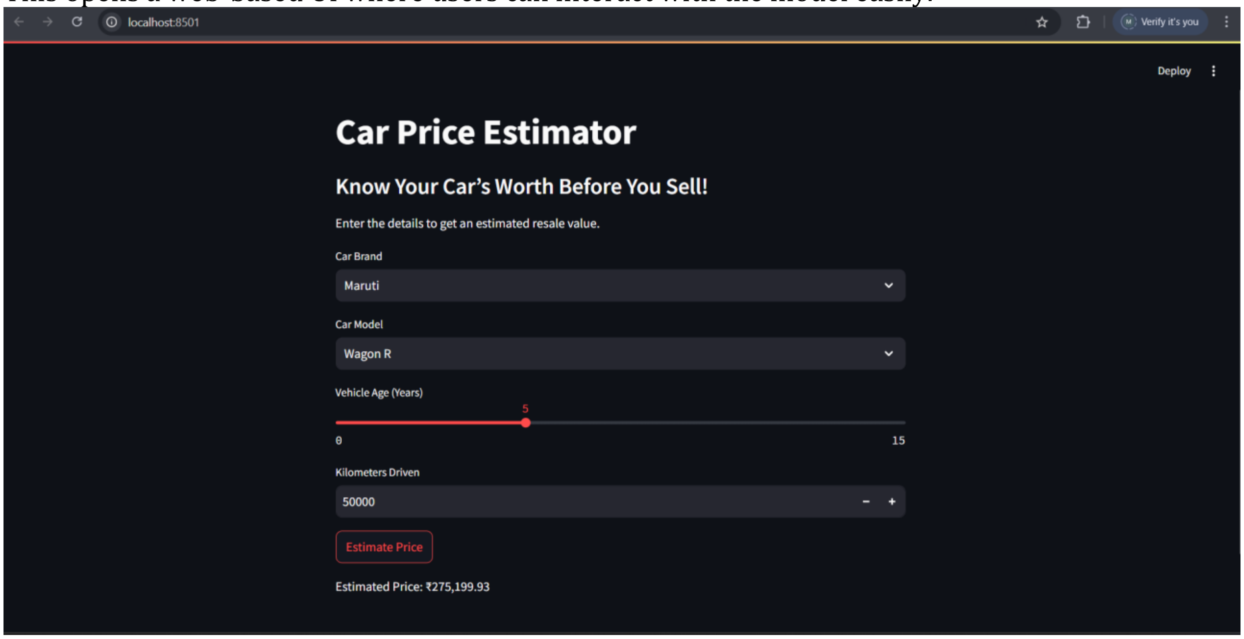
Using *Streamlit*, we developed a simple web app where users can input car details and receive price predictions in real time.

Code for the Streamlit UI:

```
import streamlit as st
import pandas as pd
import numpy as np
import joblib

# Loading the saved model, scaler, and encoders
knn = joblib.load("knn_car_price_model.pkl")
scaler = joblib.load("scaler.pkl")
label_encoders = joblib.load("label_encoders.pkl")
   input_data = np.array([[brand_encoded, model_encoded, vehicle_age, km_driven]])
   input_data = scaler.transform(input_data)

   predicted_log_price = knn.predict(input_data)[0]
   predicted_price = np.expm1(predicted_log_price)

   final_price = apply_depreciation(predicted_price, vehicle_age)

   return f"Estimated Price: ₹{final_price:,.2f}"
# Streamlit UI
st.title("Car Price Estimator")
st.subheader("Know Your Car's Worth Before You Sell!")
st.write("Enter the details to get an estimated resale value.")

brand = st.selectbox("Car Brand", label_encoders['brand'].classes_)
model = st.selectbox("Car Model", label_encoders['model'].classes_)
vehicle_age = st.slider("Vehicle Age (Years)", 0, 15, 5)
km_driven = st.number_input("Kilometers Driven", min_value=0, value=50000)

if st.button("Estimate Price"):
   result = predict_car_price(brand, model, vehicle_age, km_driven)
 st.write(result)
```

8.3 Running the Streamlit App

To launch the app locally, we use the following command in terminal:
python -m streamlit run "C:\MMD Public\NIRMA\SEM-2\AI-ML\Assignment\predictor.py"
This opens a web-based UI where users can interact with the model easily.



9. Future Enhancements and Real-World Integration

To make this model more practical, we can integrate additional real-world factors such as:
• **Location-Based Pricing** – Car prices vary by city, so adding geo-based data would improve accuracy.
• **Market Trends** – Incorporating recent sale trends can make predictions more dynamic.
• **Condition Assessment** – Including vehicle condition (scratches, accidents, servicing history) can refine predictions.
• **User Preferences** – Customizing predictions based on buyer trends and demand fluctuations.

Additionally, this model can be integrated into a real-world system where:
• It connects with **used car marketplace** to provide instant price estimates.
• Dealers can use it to set competitive prices for vehicles.
• Buyers can use it to determine fair market value before making a purchase.

9. Findings and Conclusion

• The model performed well with an R^2 score of 0.89, indicating 89% accuracy.
• Features like year, engine power, and mileage play a major role in determining the selling price.
• More data and feature engineering could improve the accuracy further.

10. References

- Scikit-learn Documentation – https://scikit-learn.org/
- CarDekho Dataset – https://www.kaggle.com/datasets/manishkr1754/cardekho-used-car-data
- Joblib Documentation - https://joblib.readthedocs.io/en/stable/
- Streamlit Tutorials - https://docs.streamlit.io/develop/tutorials
- Machine Learning with Python – Research Papers & Online Tutorials