Price Prediction Software

Based on a Log-Linear Regression Model

Price Prediction via Regression Models

Objective:

Create a **predictive model** that accurately estimates car prices using selected **features**, showcasing the **practical application of regression models** in predictive tasks.

Key Components

1. Setup and Dataset
2. Data Exploration and Cleaning
3. Feature Selection
4. Data Splitting
5. Choosing a Model
6. Evaluating Model Performance
7. Feature Importance Analysis
8. Implementation

1. Setup and Dataset

- Jupyter Notebooks
- Pandas
- Numpy
- Matplotlib
- Statsmodels
- SKlearn
- Seaborn
- Tkinter

Dataset exists in CSV form, downloaded from Kaggle.

Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import seaborn as sns
import tkinter as tk
from tkinter import ttk
sns.set()
```

```
raw_data = pd.read_csv('./UserCarData.csv')
```

2. Data Exploration and Cleaning

- No NaN values found
- All columns have same number of entries - 7906
- Data types as expected

```
raw_data.shape
```

```
(7906, 18)
```

```
# Check for NaN values
raw_data.isna().sum()
```

```
Sales_ID             0
name                 0
year                 0
selling_price        0
km_driven            0
Region               0
State or Province    0
City                 0
fuel                 0
seller_type          0
transmission         0
owner                0
mileage              0
engine               0
max_power            0
torque               0
seats                0
sold                 0
dtype: int64
```

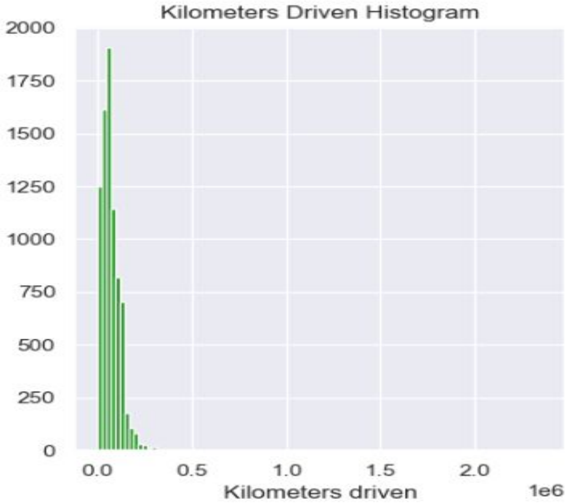
```
raw_data.describe(include = 'all').round(2)
```

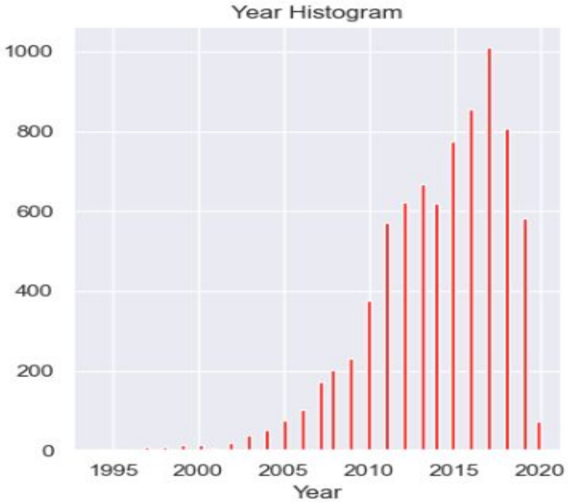
	Sales_ID	name	year	selling_price	km_driven	Region	State or Province	City	fuel	seller_type	transmission
count	7906.00	7906	7906.00	7906.00	7906.00	7906	7906	7906	7906	7906	7906

owner	mileage	engine	max_power	torque	seats	sold
7906	7906.00	7906.00	7906.00	7906	7906.00	7906

2. Data Exploration and Cleaning

- Outliers in selling_price, km_driven, year



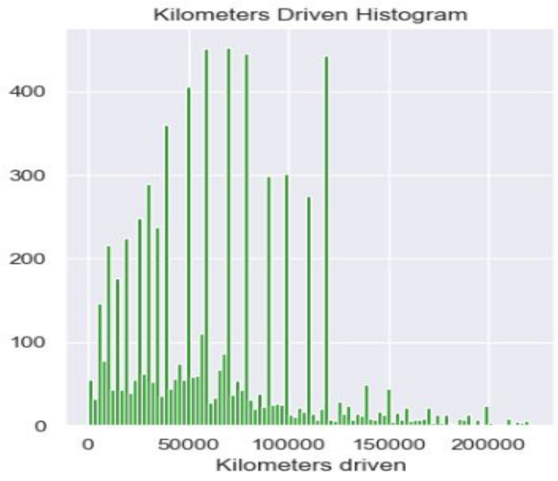


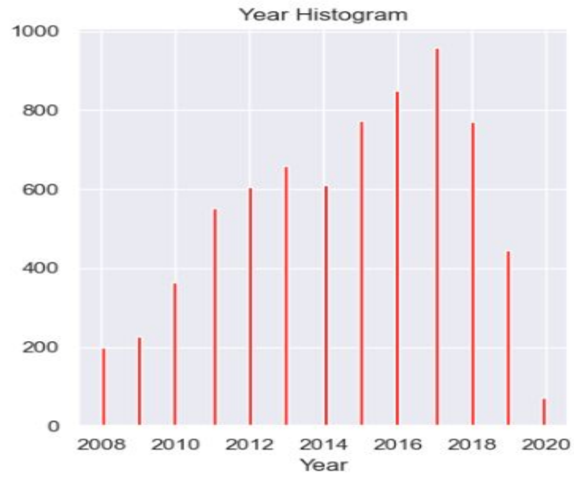
2. Data Exploration and Cleaning

- Remove selling_price; highest 3%
- Remove km_driven; highest 1%
- Remove year; lowest 5%

```
# Create the boolean mask
mask = (
    (raw_data['km_driven'] < top_1_percent_km) &
    (raw_data['selling_price'] < top_3_percent_price) &
    (raw_data['year'] > bottom_5_percent_year)
)
```

```
data_1 = raw_data[mask]
```





3. Feature Selection (and a bit of cleaning)

- Selected variables to base model off of;
- Dependent variable - | **selling_price** |
- Independent variables - | **name** | **km_driven** | **engine** | **year** |

Exclusions to highlight: **Torque** and **max power** are left out due to potential **collinearity concerns** with the **engine** variable. Additionally, **mileage** is omitted due to **insufficient details regarding the calculation methodology** for this variable.

<u>After dropping the relevant columns:</u>

‘**name**’: car brand

‘**year**’: car manufacture date

‘**engine**’: cubic capacity(engine size)

‘**selling_price**’: Indian Rupees

```
data_2.head()
```

	name	year	selling_price	km_driven	engine
0	Maruti	2014	450000	145500	1248
1	Skoda	2014	370000	120000	1498
2	Hyundai	2010	225000	127000	1396
3	Hyundai	2017	440000	45000	1197
4	Toyota	2011	350000	90000	1364

```
Sales_ID
name
year
selling_price
km_driven
Region
State or Province
City
fuel
seller_type
transmission
owner
mileage
engine
max_power
torque
seats
sold
```

3. Feature Selection (and a bit of cleaning)

- Encode categorical variables: one-hot encoding
- One-hot encoding is simple to do and is generally suitable for the regressions I will test later, namely Linear Regression, Decision Trees, Random Forest, and Gradient Boosting.

```
data_3 = pd.get_dummies(data_2, drop_first = True)
```

```
data_3.head()
```

	year	selling_price	km_driven	engine	name_Ashok	name_Audi	name_BMW	name_Chevrolet	name_Datsun	name_Fiat	...	name_Maruti	name_Mercedes
0	2014	450000	145500	1248	0	0	0	0	0	0	...	1	0
1	2014	370000	120000	1498	0	0	0	0	0	0	...	0	0
2	2010	225000	127000	1396	0	0	0	0	0	0	...	0	0
3	2017	440000	45000	1197	0	0	0	0	0	0	...	0	0
4	2011	350000	90000	1364	0	0	0	0	0	0	...	0	0

3. Feature Selection (and a bit of cleaning)

- Transforming dependent variable?

  Relationship between DV and IV:

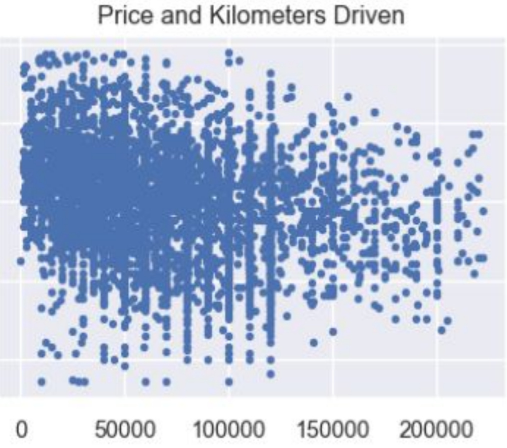
Seems slightly **exponential** as opposed to **linear.**

3. Feature Selection (and a bit of cleaning)

- Perform log transformation on DV:

  Relationship between DV and IV:

```
log_price = np.log(data_3['selling_price'])
data_3['log_price'] = log_price
```



Relationship is more linear, so we will keep add log_price to the dataframe and drop selling_price.

3. Feature Selection (and a bit of cleaning)

Dataframe depicting log_price instead of selling_price

```
data_4.describe()
```

	log_price	year	km_driven	engine	name_Ashok	name_Audi	name_BI
count	7085.000000	7085.000000	7085.000000	7085.000000	7085.000000	7085.000000	7085.000
mean	13.019159	2014.533522	65831.629922	1433.625265	0.000141	0.003952	0.004
std	0.652128	2.994284	41565.783587	474.937273	0.011880	0.062745	0.066
min	10.714418	2008.000000	1.000000	624.000000	0.000000	0.000000	0.000
25%	12.611538	2012.000000	34000.000000	1197.000000	0.000000	0.000000	0.000
50%	13.060488	2015.000000	60000.000000	1248.000000	0.000000	0.000000	0.000
75%	13.422468	2017.000000	90000.000000	1498.000000	0.000000	0.000000	0.000
max	14.893920	2020.000000	222300.000000	3498.000000	1.000000	1.000000	1.000

4. Data Splitting

- Select target and inputs
- Targets will be the dependent variable 'log_price'
- Inputs will be the dependent variables left over above
- We need to scale the inputs
- 80-20 split between training and testing data
- Need to keep random state consistent between models

Set DV as target

```
targets_DV = data_4['log_price']
```

Set IV's as inputs and then scale

```
inputs_IV = data_4.drop(['log_price'], axis = 1)

scaler = StandardScaler()
scaler.fit(inputs_IV)

inputs_IV_scaled = scaler.transform(inputs_IV)
```

Data splitting - training and testing data

```
x_train, x_test, y_train, y_test = train_test_split(inputs_IV_scaled, targets_DV, test_size = 0.2, random_state = 365)
```

5. Choosing a Model

- Choice between Linear Regression, Decision Trees, Random Forest, Gradient Boosting
- Evaluate each model with the training data for quick, basic accuracy check
- Use MAPE and R-squared as metrics

I'm choosing **Linear Regression** for its efficiency, simplicity, and familiarity. Despite slightly lower accuracy, its scores are acceptable for practical use.

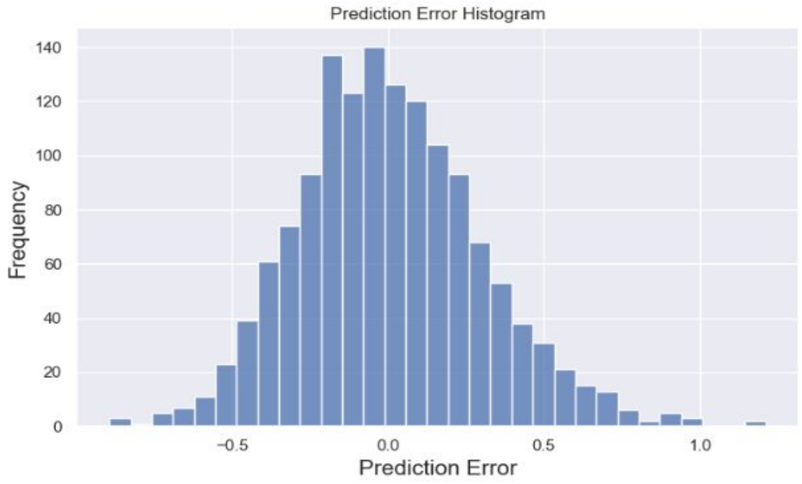
```python
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(random_state = 365),
    'Random Forest': RandomForestRegressor(random_state = 365),
    'Gradient Boosting': GradientBoostingRegressor(random_state = 365)
}

# Iterate through the models
for model_name, model in models.items():

    # Train the model with the training data
    model.fit(x_train, y_train)

    # Make predictions on the testing data
    y_predicted = model.predict(x_test)

    # Calculate MAPE
    mape = np.mean(np.abs((y_test - y_predicted) / y_test)) * 100

    # Calculate R2
    r2 = r2_score(y_test, y_predicted)

    # Print the results
    print(f'{model_name}: MAPE = {round(mape, 3)} %, R2 = {round(r2, 5)}')
```

```
Linear Regression: MAPE = 1.794 %, R2 = 0.80871
Decision Tree: MAPE = 1.38 %, R2 = 0.85426
Random Forest: MAPE = 1.207 %, R2 = 0.89715
Gradient Boosting: MAPE = 1.35 %, R2 = 0.88214
```

6. Evaluating Model Performance

- Train the model with training data
- Use model on testing data to get **y_predicted**
- Compare **y_predicted** with **y_train**
- **Difference resembles a normal distribution**

```
model = LinearRegression().fit(x_train, y_train)
y_predicted = model.predict(x_test)
```



7. Feature Importance Analysis

- Analyze the weights of each independent variable on the dependent variable
- For the categorical variables, the benchmark is the omitted car brand, Lexus

```
model_summary = pd.DataFrame(inputs_IV.columns.values, columns = ['Ind. Vars'])
model_summary['Weights'] = model.coef_
model_summary
```

	Ind. Vars	Weights
0	year	0.413753
1	km_driven	-0.009416
2	engine	0.378965
3	name_Ashok	0.001994
4	name_Audi	0.073115
5	name_BMW	0.084498
6	name_Chevrolet	0.009402
7	name_Datsun	-0.010974
8	name_Fiat	0.023975
9	name_Force	-0.002802
10	name_Ford	0.069865
11	name_Honda	0.123476
12	name_Hyundai	0.166693
13	name_Isuzu	0.007459
14	name_Jaguar	0.092477
15	name_Jeep	0.050887
16	name_Kia	0.015358
17	name_Land	0.021318
18	name_MG	0.021384
19	name_Mahindra	0.035224
20	name_Maruti	0.200829
21	name_Mercedes	0.080609
22	name_Mitsubishi	0.014995
23	name_Nissan	0.037380
24	name_Renault	0.054411
25	name_Skoda	0.045203
26	name_Tata	-0.003956
27	name_Toyota	0.097923
28	name_Volkswagen	0.064035
29	name_Volvo	0.077105

7. Implementation - Basic GUI interface

```python
def predict_price():

    # Get input values from the GUI
    year = int(year_var.get())
    km_driven = int(km_entry.get())
    engine = float(engine_entry.get())
    model_name = 'name_' + model_var.get()  # Concatenate 'name_' prefix

    # Create dictionary to save car information in
    specific_car = {
        'year': year, 'km_driven': km_driven, 'engine': engine,
        'name_Ashok': 0, 'name_Audi': 0, 'name_BMW': 0,
        'name_Chevrolet': 0, 'name_Datsun': 0, 'name_Fiat': 0,
        'name_Force': 0, 'name_Ford': 0, 'name_Honda': 0,
        'name_Hyundai': 0, 'name_Isuzu': 0, 'name_Jaguar': 0,
        'name_Jeep': 0, 'name_Kia': 0, 'name_Land': 0,
        'name_MG': 0, 'name_Mahindra': 0, 'name_Maruti': 0,
        'name_Mercedes': 0, 'name_Mitsubishi': 0, 'name_Nissan': 0,
        'name_Renault': 0, 'name_Skoda': 0, 'name_Tata': 0,
        'name_Toyota': 0, 'name_Volkswagen': 0, 'name_Volvo': 0
    }

    # Set the model to 1 for the selected model from the dropdown
    specific_car[model_name] = 1

    # Convert the dictionary to a DataFrame
    specific_car_data = pd.DataFrame([specific_car])

    # Scale the input data
    specific_car_inputs_scaled = scaler.transform(specific_car_data)

    # Run the model on the scaled data
    predicted_log_price = model.predict(specific_car_inputs_scaled)

    # Convert the predicted log price back original value before log transformation
    predicted_price = np.exp(predicted_log_price)

    # Display the predicted price
    result_label.config(text=f"Predicted Price: IDR {round(predicted_price[0], 2)}")
```