MIDDLESEX UNIVERSITY - DUBAI

Title: Coursework 1 – Data Science and Machine Learning

Name: Aryan Kawade

Student ID: M01042658

Word Count: 1600+

Contents

1. Understanding of the data

1. Summarise the dataset and provide a link to it (or reference). How do you make sure that it is publicly available and do not require additional permissions for your study purpose? (5p)

- This project is an individual demonstration of how I use the dataset: OLX-Cars-Dataset which is available publicly on Kaggle.

  Origin – Indian / Pakistan
  Currency – Rupees
  URL – (https://www.kaggle.com/datasets/abdullahkhanuet22/olx-cars-dataset)

  Since the dataset is public and does not specify/ point-out personal information and connections. It does not require any additional permissions for my educational purpose. Furthermore, the dataset, having been sourced from public listings, does not contain personally identifiable information (PII) or sensitive private connections, which bypasses significant ethical hurdles and the need for institutional review board (IRB) approval or additional permissions for this study.

2. Are you considering regression, classification or clustering and state your goal in one sentence? (5p)

- According to my dataset, I personally felt that Regression would be a suitable model for my Machine Learning Analysis.

  My model's goal is to determine a cars price with respect to the columns included in the dataset i.e. Year, Kilometers Driven, Transmission Type, Fuel Type, etc.

3. Describe the dataset using descriptive statistics, e.g. df.info(), df.describe(), etc. (5p)

-

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9179 entries, 0 to 9178
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Make          9179 non-null   object
 1   Model         9179 non-null   object
 2   Year          9179 non-null   int64
 3   KM's driven   9179 non-null   int64
 4   Price         9179 non-null   int64
 5   Fuel          9179 non-null   object
 6   Transmission  9179 non-null   object
dtypes: int64(3), object(4)
memory usage: 502.1+ KB
```

The dataset for my analysis includes these columns.
The initial dataset included columns such as:

- Ad Id (Dropped: Not related to analysis and not unique)
- Car Name (Dropped Since the naming was too erratic)
- Make
- Model
- Fuel Type
- Year
- Price
- Kilometers Driven
- Registration City (Dropped: Format was not proper hence inconsequential)
- Car Documents (Dropped: Not related to analysis)
- Assembly (Dropped: Not related to analysis)
- Transmission
- Condition (Dropped: No relation to Price)
- Seller Location (Dropped: Not related to analysis)
- Description (Dropped: No format and too long to decipher)
- Car Features (Dropped: No proper format and too jumbled up)
- Images URL's (Dropped: Not needed for analysis)
- Car's profile (Dropped: Same reason as images)

As you can see majority of the columns were not suitable for regression type model analysis, hence was dropped. But, in the case this was a classification most of them could be used.

Price – The main focus of Regression. Could be influenced by various factors in the table.

The Non-Cleaned Data Set shows 9179 indexes which becomes 8795 indexes after cleaning.

2. Data preprocessing

The Preprocessing included encoding, changing the column names, dropping unnecessary columns.

```
# Drop unnecessary columns
df.drop(columns=['Ad ID','Car Name','Registration city','Car documents','Assembly','Condition','Seller Location',
                 'Description','Car Features',"Images URL's",'Car Profile'],inplace=True)
```

```
# Clean column names
df.columns = df.columns.str.replace("'",'').str.replace(' ','_')
df.rename(columns={'KMs_driven':'Km_driven'},inplace=True)
```

Data Type cleaning was also done:

The Integers were reinforced; the null/ NaN values were set to unknown. All the Values in the Make and Model were Capitalized.

```
# Convert datatypes and fill missing
df['Price'] = pd.to_numeric(df['Price'],errors='coerce').astype(int)
df['Km_driven'] = pd.to_numeric(df['Km_driven'],errors='coerce').astype(int)
df['Year'] = pd.to_numeric(df['Year'],errors='coerce').astype(int)

df['Fuel'] = df['Fuel'].fillna('Unknown')
df['Transmission'] = df['Transmission'].fillna('Unknown')

df['Make'] = df['Make'].str.title()
df['Model'] = df['Model'].str.title()
```

To Verify whether there were any null values, I checked the unique values in transmission and fuel column.

```
# Check fuel categories
df['Fuel'].unique()
```

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

```
# Check transmission categories
df['Transmission'].unique()
```

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

1. Check for missing values and handle them appropriately (you can manually insert NaN values, e.g. nan, N/a, N/A). (10p)

-

```
# Check null values
df.isnull().any()
```

	0
Make	False
Model	False
Year	False
Km_driven	False
Price	False
Fuel	False
Transmission	False

dtype: bool

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

	0
Make	0
Model	0
Year	0
Km_driven	0
Price	0
Fuel	0
Transmission	0

dtype: int64

caption - Initial Check for Missing Values

- Removing the (non-existant) null/NaN values

```
# Drop rows with nulls
df.dropna(axis=0,inplace=True)
df.isnull().sum().sum()
```

```
np.int64(0)
```

- Also Checking for duplicates and dropping them

```
df.duplicated().sum()
```

```
np.int64(384)
```

```
df[df.duplicated()]
```

	Make	Model	Year	Km_driven	Price	Fuel	Transmission
606	Honda	City Ivtec	2019	60000	3875000	Petrol	Automatic
817	Suzuki	Ravi	2022	2700	1850000	Petrol	Manual
883	Daihatsu	Terios Kid	2001	120000	1570000	Petrol	Automatic
1103	Daihatsu	Cuore	2000	15000	650000	Petrol	Manual
1254	Hyundai	Santro	2006	123521	798000	Petrol	Manual
...	...	...	...	...	...	...	...
9129	Suzuki	Mehran Vxr	2007	120000	690000	Petrol	Manual
9137	Toyota	Corolla Gli	2013	354720	2450000	Petrol	Manual
9148	Suzuki	Cultus Vxr	2010	1234	985000	Petrol	Manual
9159	Suzuki	Swift	2018	151000	2425000	Petrol	Manual
9160	Suzuki	Cultus Vxr	2005	100000	590000	Petrol	Manual

384 rows × 7 columns

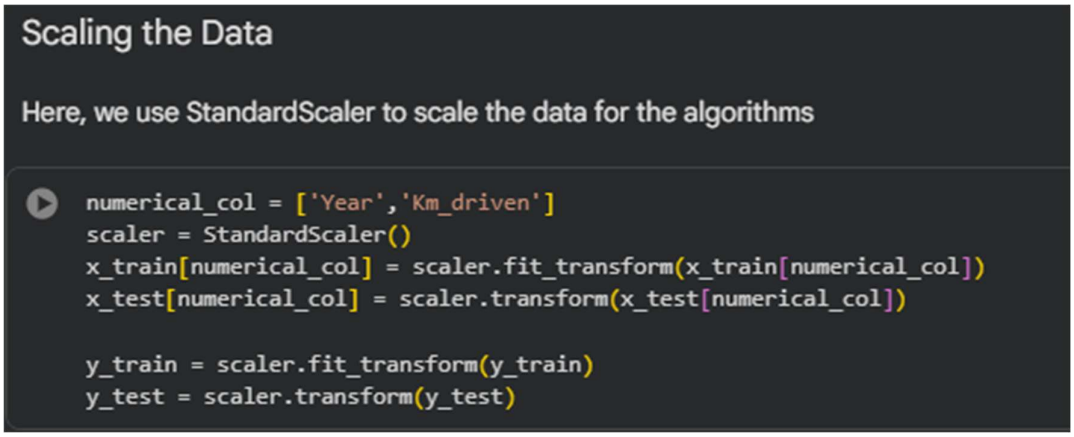
```
df = df.drop_duplicates().reset_index(drop=True)
df.duplicated().sum()
```

```
np.int64(0)
```

2. Standardise/normalise numerical features. (5p)

- I have chosen to standardise the numerical columns since I have many outliers.

  By Using the StandardScaler I can scale the data appropriately. Since my Target column Price had a very high numerical count, I Scaled it too.



caption – it has no display output

3. Encode categorical variables using, such as one-hot or label encoding. (5p)

- In my Model, I have chosen the Label Encoding since it was the one I understood the best.

```
print()

Mapping for : Make
Changan: 0
Chevrolet: 1
Daihatsu: 2
Faw: 3
Honda: 4
Hyundai: 5
Kia: 6
Mercedes: 7
Mitsubishi: 8
Suzuki: 9
Toyota: 10
```

```
Mapping for : Model
Alsvin: 0
Altis Grande: 1
Alto: 2
Baleno: 3
Bolan: 4
C Class: 5
Cervo: 6
City Aspire: 7
City Idsi: 8
City Ivtec: 9
City Vario: 10
Civic Exi: 11
Civic Oriel: 12
Civic Prosmetic: 13
Civic Vti: 14
```

(More in the .ipynb file)

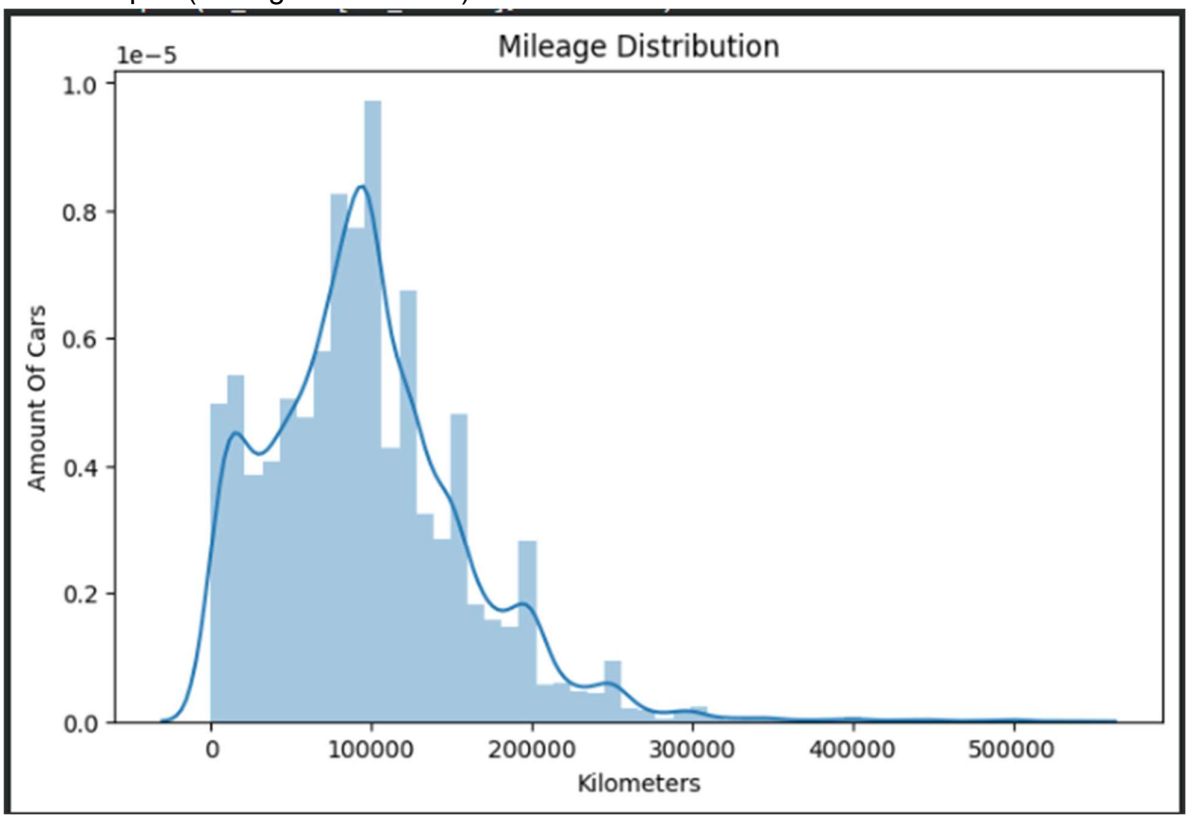
```
Mapping for : Fuel
CNG: 0
Diesel: 1
Hybrid: 2
Petrol: 3

Mapping for : Transmission
Automatic: 0
Manual: 1
```

3. Exploratory data analysis (EDA)

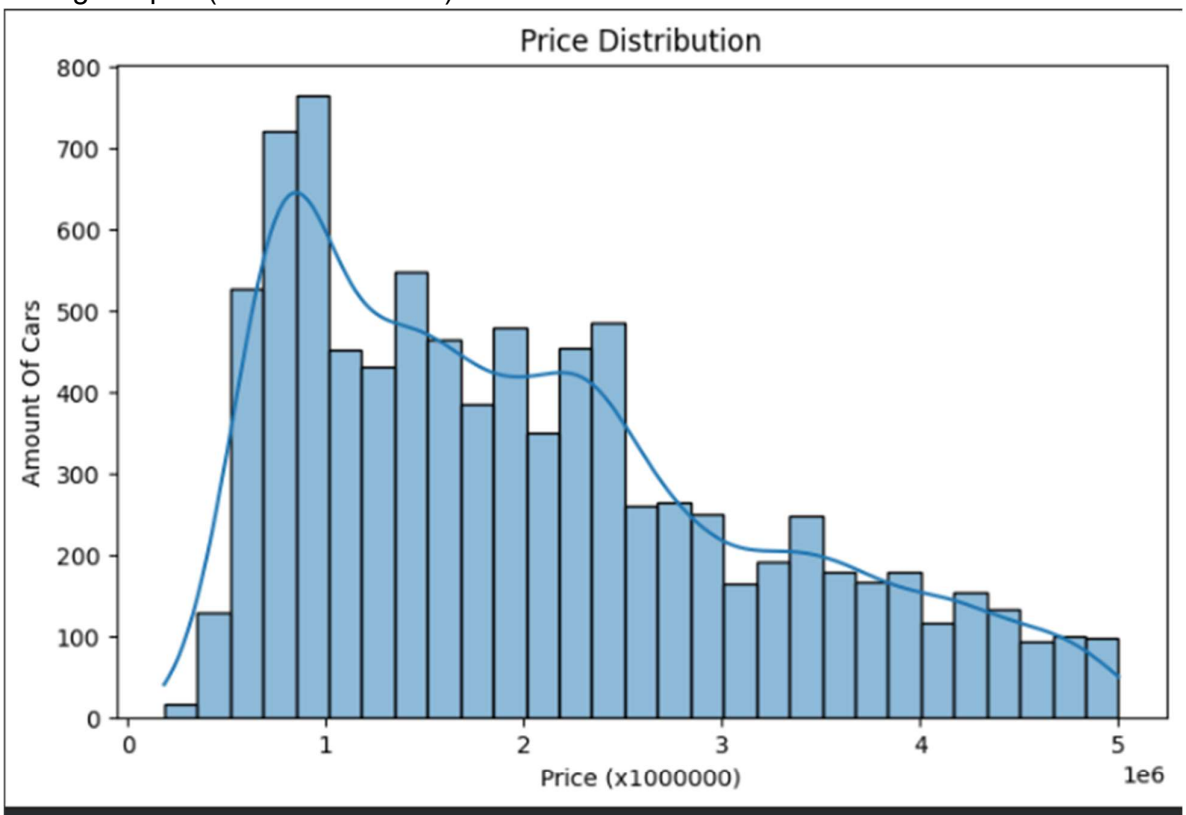
EDA visualizes the correlations between columns:

I. Distance plot (Mileage Distribution):



Signifies that the maximum number of cars drive at most 99000 – 100000 km's before deciding to sell the car. This graph is right-skewed meaning that majority of cars have low mileage.

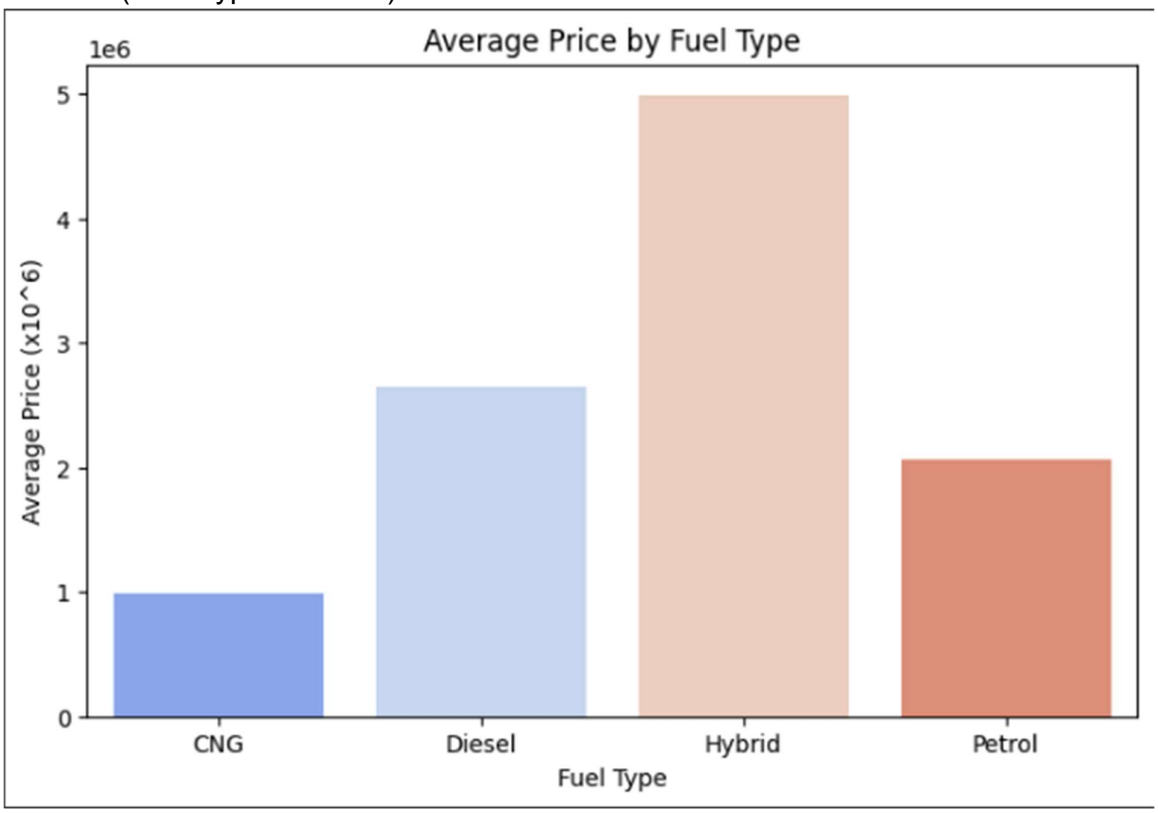
II. Histogram plot (Price Distribution):




```
These are some Price Statistics
Skewness: 0.68
Kurtosis: -0.46
Mean: 2036285.40
Median: 1815000.00
Standard Deviation: 1157100.49
```

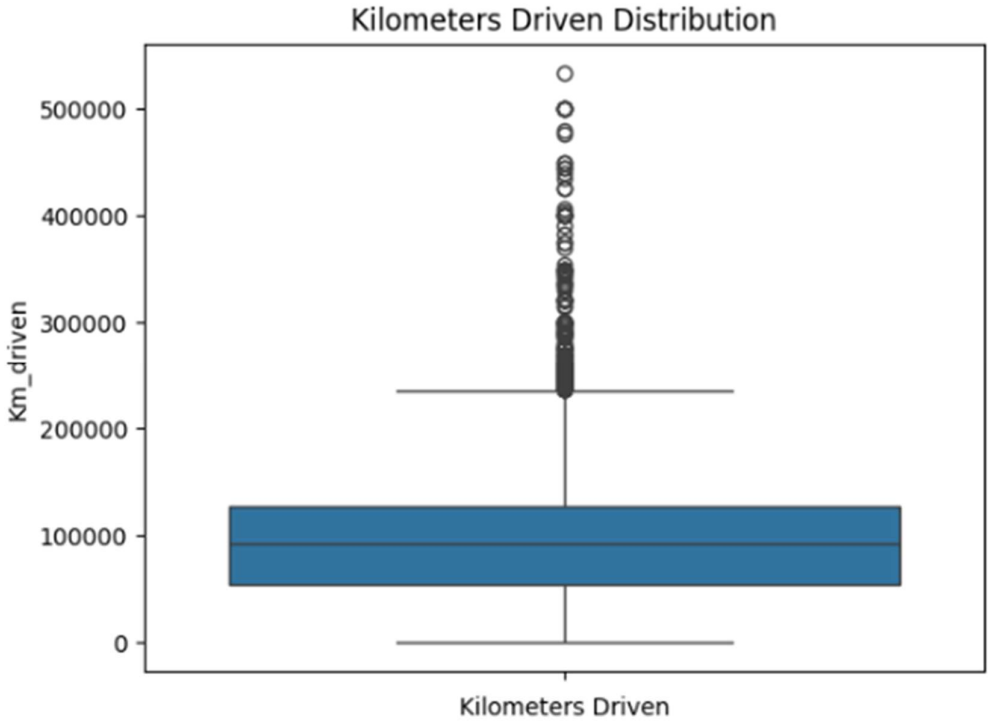
This graph visualizes the price point of the cars in the dataset. This graph is very right-skewed, which means that most of the cars have a low price point with few exceptions of expensive cars. The maximum number of cars have a price of 10^6 rupees.

III. Bar Plot (Fuel Type vs. Price)



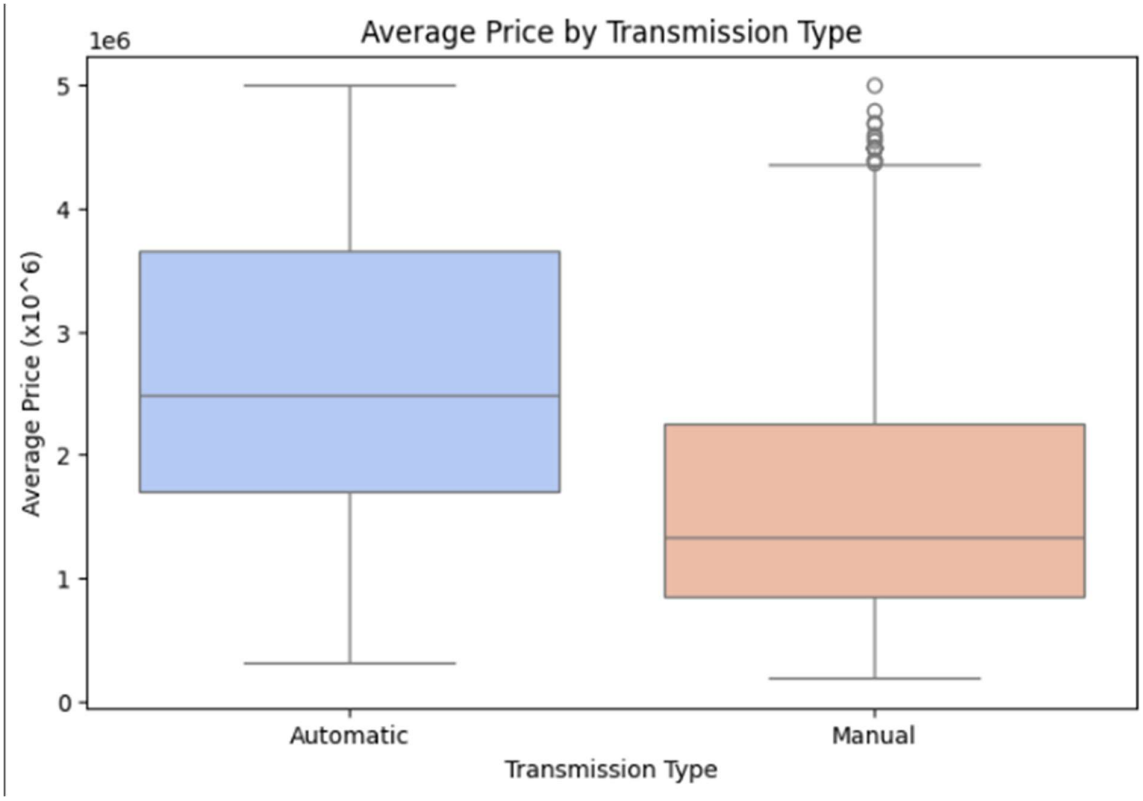
This graph visualizes the Avg. Price of the different kinds of fuel types. According to the graph we can see that hybrid cars spend the most on fuel, with the least money spent on CNG by cars.
Through this we can see the inefficiency of hybrid cars compared to traditional fuel cars.
This dataset may also contain slight bias since there are many cars that drive the longest are diesel, hence the data reflects that by showing a slightly higher price margin compared to petrol.

IV. Box Plot (Kilometers Driven)



This graph visualizes the Density of cars to kilometers they have driven. We can observe that the highest number of cars drive from 60k to 120k. The outliers are not less though, the maximum distance driven seems to be at 500k+ which is honestly respect worthy.
Through this box plot we can how the public perceives underdriven vehicles compared to over-driven vehicles on sale.
Underdriven (50k-)
Overdriven (230k+)

V. Multi-Box Plot (Transmission Type vs. Price)

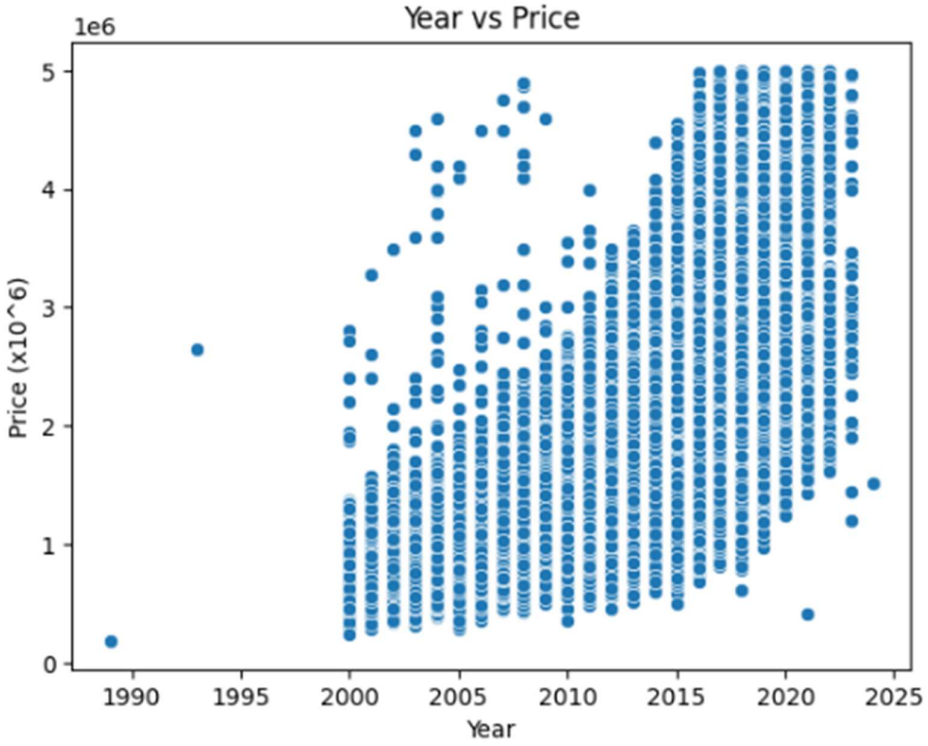


This graph compares the prices of an Automatic Transmission compared to a Manual Transmission. We can see that manuals are generally cheaper than automatics. This could also be because of the features the car includes, which was part of the dropped column which couldn't be included due to improper formatting of the column.

We can also see that the outliers of manual mean that even the manuals can reach the price of a very expensive automatic vehicle.

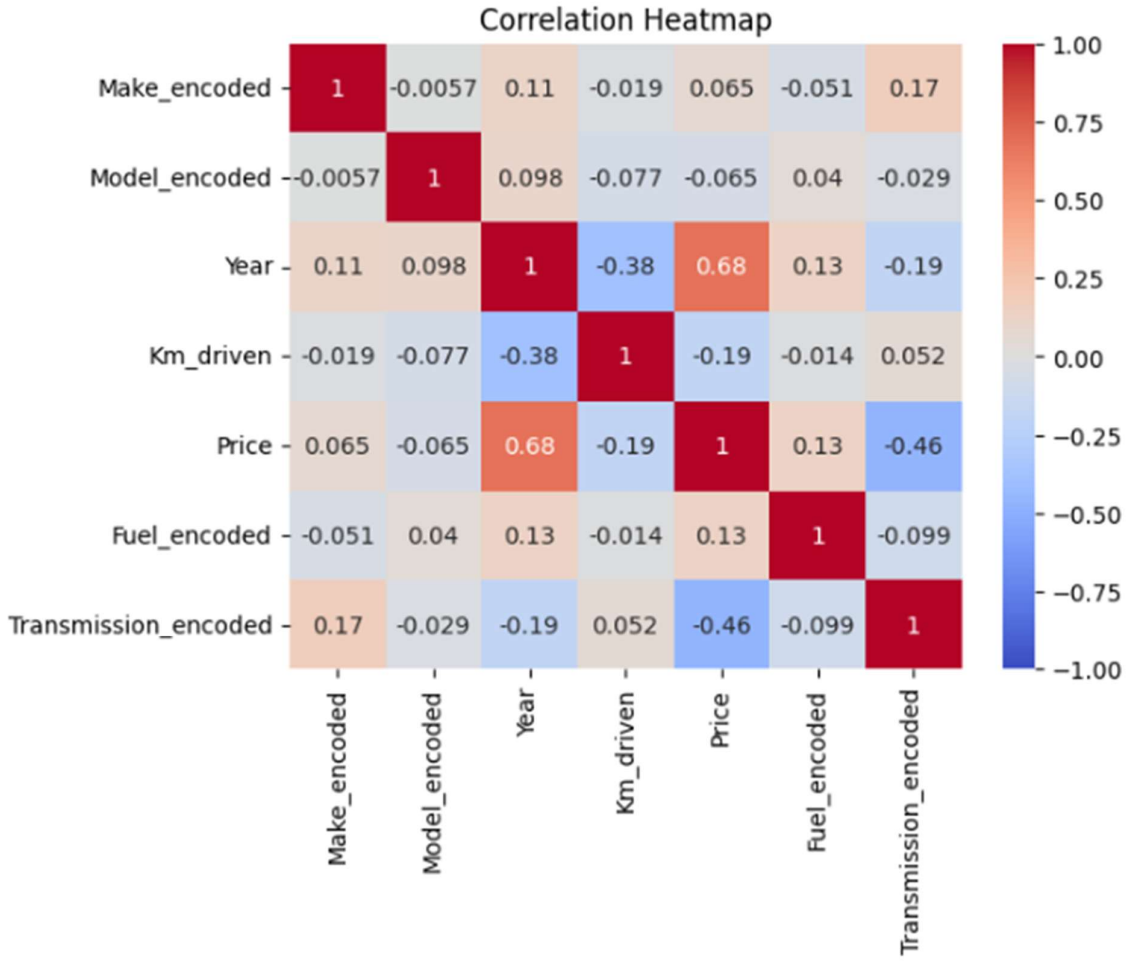
Since this dataset doesn't include professional race-cars which are generally manuals for various reasons, it means the price of a manual can even be higher. But this also mean that manuals are very flexible, can be both good and cheap.

VI. Scatter Plot (Year wise – Price Trends)



This graph visualizes the ever-increasing prices of cars. Through this we can analyse that as we go in the future the prices will keep getting higher (Nooo…)
The one point before 1990 the price was at 10k and in 2023 the minimum price can go up to 120k. A time-traveller would become insanely rich if he somehow gets hold of old currencies.
We can also see the outliers in the years 2000 to 2010, they are the luxury cars which were sold.

VII. Heatmap (Column Correlations)

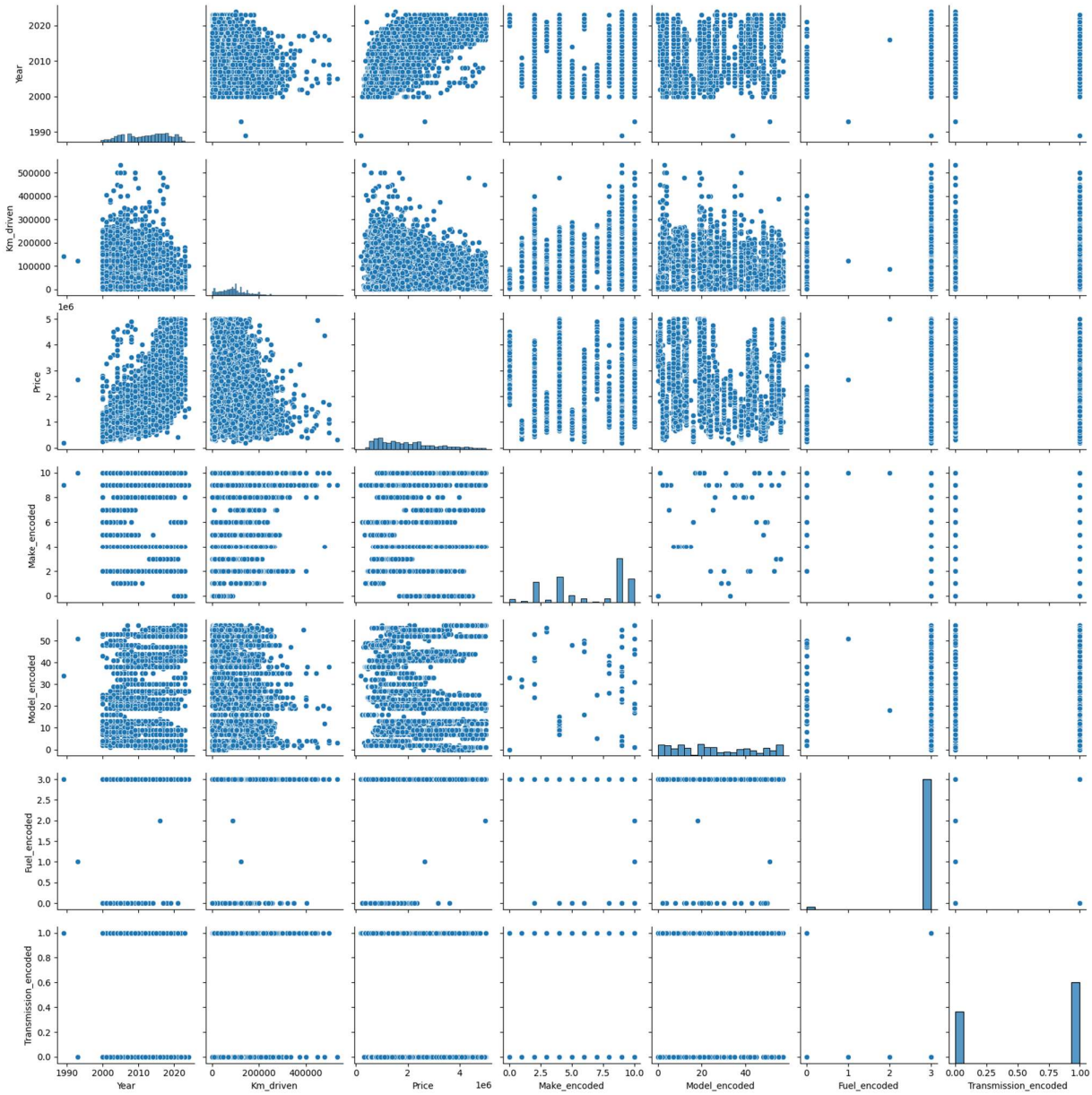


This graph compares each numerical column with each other. Since the character encoding had no meaningful connection, the most prominent correlations can be seen in Year, Km Driven, Price.

We can see that the price-year number is 0.68 which proves that as year increases the prices also go up.

The inverse can be observed in price-km driven; the number is -0.19 which means the opposite, that prices are higher the lower the distance driven. But this doesn't mean that prices are lower the higher the distance driven. You thought you could get a luxury for less huh...

VIII. Pair Plot (Ultimate Comparison between numerical columns and individual columns)



For a clearer picture please refer to the CW1.ipynb file. This graph proves the same as the heatmap but directly shows the data points between the individual columns. No witty comment this time.

1. Visualise the distribution of target variable and the correlation between features and the target variable. Identify the features that are most strongly correlated. (15p)

- Refer Individual Graph explanations and correlations.

2. How to interpret the obtained visuals in terms of disparities considering relevant features? Does EDA reveal any significant insights that directly inform your model or preprocessing? (10p)

- Refer Individual Graph explanations and correlations.

4. Model development and evaluation

1. Train the model using a **traditional** machine learning algorithm. (15p)

- I used 6 different regression models for this project:
    i. Linear Regression – The most basic regression model ever.

    ii. Decision Tree Regressor – Creates a tree like regression model which is used to predict accordingly.

    iii. Random Forrest Regression – Uses the same logic as Decision Tree but creates n types of similar models which get averaged out to give predictions.

    iv. Gradient Boosting Regressor – Uses a tree like neural structure which gets built on the data and gets overwritten if errors are detected, which creates data consistancy.

    v. Support Vector Regression (SVR) – Uses some technique to find an optimal ‘Hyperplane’ which in-turn returns the prediction values.

    vi. K-Nearest Neighbours Regression – Uses the data fed to create a map which predicts based to the nearest data points to the result. Hence the nearest neighbour.

2. Predict the target variable and evaluate the model performance, e.g. regression: MAE, RMSE; classification: precision/recall; clustering: silhouette score. Analyse and comment on the performance of the model. (15p)

- Since I am using a Regression model, I will evaluate the model's performance with Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), a R^2 score.

```
Linear Regression:
Mean Absolute Error: 0.50
Mean Squared Error: 0.40
Root Mean Squared Error: 0.63
R^2: 0.60

Random Forest:
Mean Absolute Error: 0.18
Mean Squared Error: 0.07
Root Mean Squared Error: 0.27
R^2: 0.93

Decision Tree:
Mean Absolute Error: 0.23
Mean Squared Error: 0.12
Root Mean Squared Error: 0.34
R^2: 0.88

Support Vector Machine:
Mean Absolute Error: 0.38
Mean Squared Error: 0.31
Root Mean Squared Error: 0.56
R^2: 0.68

K-Nearest Neighbors:
Mean Absolute Error: 0.19
Mean Squared Error: 0.08
Root Mean Squared Error: 0.28
R^2: 0.92

Gradient Boosting:
Mean Absolute Error: 0.20
Mean Squared Error: 0.08
Root Mean Squared Error: 0.28
R^2: 0.92
```

These models use a set function to calculate their results.
Named #regression_models(model_type).
Refer to CW1.ipynb

The common function regression_models(model_type) looks like this:

```python
#Regression Model
def regression_models(model_type):
    model_type.fit(x_train, y_train.ravel())
    y_pred = model_type.predict(x_test)

    model_type.score(x_test, y_test)

    MAE = mean_absolute_error(y_test, y_pred)
    MSE = mean_squared_error(y_test, y_pred)
    RMSE = np.sqrt(MSE)
    r2 = r2_score(y_test, y_pred)

    return [f'{MAE:.2f}', f'{MSE:.2f}', f'{RMSE:.2f}', f'{r2:.2f}']
```

Below I called this function using each models as a parameter:

```python
#Results for the different models
lm_result = regression_models(LinearRegression())
rfs_result = regression_models(RandomForestRegressor(n_estimators=100,random_state=69420))
dtr_result = regression_models(DecisionTreeRegressor(random_state=420))
svr_result = regression_models(SVR())
knr_result = regression_models(KNeighborsRegressor())
gbr_result = regression_models(GradientBoostingRegressor())

display_results({
    'Linear Regression': lm_result,
    'Random Forest': rfs_result,
    'Decision Tree': dtr_result,
    'Support Vector Machine': svr_result,
    'K-Nearest Neighbors': knr_result,
    'Gradient Boosting': gbr_result
})
```

Since the results feel to separated out. I decided to make a table to easily compare the results.Included in the .py notebook.

	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	R^2
Linear Regression	0.50	0.40	0.63	0.60
Random Forest	0.18	0.07	0.27	0.93
Decision Tree	0.23	0.12	0.34	0.88
Support Vector Machine	0.38	0.31	0.56	0.68
K-Nearest Neighbors	0.19	0.08	0.28	0.92
Gradient Boosting	0.20	0.08	0.28	0.92

The R^2 denotes the winner clearly:

1) Random Forest (🏆)
2) Tie (K-nearest Neighbours, Gradient Boosting)
3) Decision Tree
4) Support Vector
5) Linear Regression

The lowest error margin was also showed with Random Forest.
On the other hand, Linear Regression is too simple an algorithm for a multi-factor database, which is why its R^2 score is low while the error margin being too high.

5. References

Kaggle Dataset - https://www.kaggle.com/datasets/abdullahkhanuet22/olx-cars-dataset

AI Usage – Used for Error Fixing and learning.