Jaypee Institute of Information Technology, Noida

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AND INFORMATION TECHNOLOGY

Project Title: Car Dekho 2.0

Enrollment. No.	**Name of Student**
9922102094	Rishita Singh
9922103029	Shikhar Maheshwari
9922103042	Shivam

Course Name: Open-Source Software Lab

Course Code: 15B17CI575

Program: B. Tech. CS&E

3rd Year 5th Sem

Submitted To: Mrs. Shikha Mehta

2024

ABSTRACT

The purpose of this lab project is on creating a car recognition application that utilizes machine learning models and software tools, such as Streamlit, for estimating the market value of second-hand cars. The app employs dependency management and a range of Python packages to enhance functionality and interface, ensuring accurate, user-friendly price predictions. Key packages like Altair, Blinker, and Cachetools support data handling, visualization, and interaction. This setup allows users to easily access car price predictions, aiding in informed purchasing decisions within the second-hand automotive market.

INTRODUCTION

In recent years, the demand for accurate valuation tools within the second-hand car market has surged, as buyers and sellers seek informed guidance on fair pricing. This project aims to create a car recognition application that leverages machine learning to estimate the market value of used cars. Built using Python and incorporating tools like Streamlit, the app provides a user-friendly interface for seamless interaction and price predictions.

To achieve high functionality, the project employs effective dependency management alongside a variety of Python packages. Key packages, such as Altair for data visualization, Blinker for event-driven programming, and Cachetools for efficient data handling, enhance the app's ability to manage, visualize, and process data. These tools ensure that the application is not only accurate in its predictions but also intuitive for users, enabling them to access critical insights with ease. Ultimately, this app serves as a valuable resource, assisting users in making informed, data-driven purchasing decisions within the pre-owned car market.

Technologies Used in the Project

1. **Pandas**:
    - **Purpose**: Pandas is a powerful data manipulation and analysis library in Python. It is used for cleaning, transforming, and analyzing data in tabular form (DataFrame). It helps in handling missing values, filtering rows, grouping data, and performing calculations.
    - **Key Functions**:
    - pd.read_csv(), pd.DataFrame(), df.dropna(), df.fillna(), df.groupby(), etc.

2. **NumPy**:
    - **Purpose**: NumPy is a library for numerical computations in Python. It is used for handling arrays and performing mathematical operations on large datasets. It is especially useful for operations on multi-dimensional arrays.
    - **Key Functions**:
    - np.array(), np.mean(), np.median(), np.corrcoef(), etc.

3. **Matplotlib**:
    - **Purpose**: Matplotlib is a plotting library for Python that is used to create static, interactive, and animated visualizations. It helps in creating a variety of charts such as line graphs, scatter plots, histograms, etc.
    - **Key Functions**:
    - plt.plot(), plt.scatter(), plt.hist(), plt.show(), etc.
    - **Features**:
    - Customization options for colors, styles, and other plot attributes.

4. **Seaborn**:
    - **Purpose**: Seaborn is built on top of Matplotlib and provides a high-level interface for creating attractive and informative statistical graphics. It is often used for visualizing relationships between variables, distributions, and statistical trends.
    - **Key Functions**:
    - sns.barplot(), sns.heatmap(), sns.boxplot(), sns.pairplot(), etc.
    - **Features**:
    - Easy-to-use interface for complex plots, statistical visualizations, and multi-plot layouts.

5. **Matplotlib Inline**:
    - **Purpose**: The %matplotlib inline magic command is used in Jupyter Notebooks to display Matplotlib plots directly in the notebook. It ensures that visualizations appear in the output cell immediately after execution, rather than in a separate window.

6. **Warnings**:
    Purpose: The warnings module is used to manage warnings in Python. In this case, the code uses warnings.filterwarnings('ignore') to suppress warnings during the execution, which is helpful in avoiding cluttered output.

7. **Linear Regression (Machine Learning)**:
   - **Purpose**: Linear regression is a statistical method used for modeling the relationship between a dependent variable (in this case, car price) and one or more independent variables (such as car features like mileage, brand, age, etc.). It helps in predicting the price of a car based on these parameters.
   - **Key Libraries**:
   - **Scikit-learn**: Although not explicitly mentioned in the imports, scikit-learn is commonly used for machine learning tasks like linear regression.
   - **Key Functions**: • LinearRegression(), model.fit(), model.predict(), etc.

Flow of the Project

1. **Data Collection and Cleaning**:

   - Loading the dataset using pandas.read_csv() or any other suitable function.
   - Data cleaning, such as handling missing values (df.dropna() or df.fillna()), and preparing the data for analysis (e.g., encoding categorical variables, scaling numerical variables).

2. **Data Visualization**:

   - Using **Matplotlib** and **Seaborn** to plot visualizations like scatter plots, histograms, and heatmaps to understand patterns and relationships between different features in the dataset.
   - Example: Visualizing the correlation between car price and other features like mileage, year, brand, etc. •

3.Feature Engineering:

- Selecting relevant features (independent variables) for the linear regression model, such as car age, mileage, brand, etc.
- Handling categorical variables (e.g., converting 'brand' into numerical data).

4. Linear Regression Model:

- Using **Linear Regression** to build a predictive model for car prices. This involves:
- Splitting the dataset into training and testing sets.
- Training the model using model.fit(X_train, y_train).
- Making predictions using model.predict(X_test).
- Evaluating the model performance using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), R-squared, etc.
-

5. Prediction:

- Using the trained linear regression model to predict car prices based on input features.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

[1] ✓ 1.8s

```python
car=pd.read_csv('quikr_car.csv')
```

[2] ✓ 0.0s

```python
car.head() # kuch row ko print kr dia
```

[3] ✓ 0.0s

	name	company	year	Price	kms_driven	fuel_type
0	Hyundai Santro Xing XO eRLX Euro III	Hyundai	2007	80,000	45,000 kms	Petrol
1	Mahindra Jeep CL550 MDI	Mahindra	2006	4,25,000	40 kms	Diesel
2	Maruti Suzuki Alto 800 Vxi	Maruti	2018	Ask For Price	22,000 kms	Petrol
3	Hyundai Grand i10 Magna 1.2 Kappa VTVT	Hyundai	2014	3,25,000	28,000 kms	Petrol
4	Ford EcoSport Titanium 1.5L TDCi	Ford	2014	5,75,000	36,000 kms	Diesel

```python
car.shape
```

[4] ✓ 0.0s

```
(892, 6)
```

```python
car.info()
```

[5] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 892 entries, 0 to 891
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   name        892 non-null    object
 1   company     892 non-null    object
 2   year        892 non-null    object
 3   Price       892 non-null    object
```

Creating backup copy

```
backup=car.copy()
```
[6] ✓ 0.0s

Cleaning Data

year has many non-year values

```
car=car[car['year'].str.isnumeric()]  # filter kari int waali values bass year
```
[7] ✓ 0.0s

year is in object

```
car['year']=car['year'].astype(int)
```
[8] ✓ 0.0s

Empty markdown cell, double-click or press enter to edit.

```
car=car[car['Price']!='Ask For Price']
```
[9] ✓ 0.0s

Price has commas in its prices and is object

```
car['Price']=car['Price'].str.replace(',','').astype(int)  # commas hata ke int me convert kar dia
```
[10] ✓ 0.0s

kms_driven has object values with kms at last.

```
car['kms_driven']=car['kms_driven'].str.split().str.get(0).str.replace(',','')
```

[11] ✓ 0.0s

removing wrong values

```
car=car[car['kms_driven'].str.isnumeric()]
```

[12] ✓ 0.0s

```
car['kms_driven']=car['kms_driven'].astype(int)
```

[13] ✓ 0.0s

fuel_type has nan values

```
car=car[~car['fuel_type'].isna()]
```

[14] ✓ 0.0s

```
car.shape
```

[15] ✓ 0.0s

```
(816, 6)
```

```
car['name']=car['name'].str.split().str.slice(start=0,stop=3).str.join(' ')
```

[16] ✓ 0.0s

```
car=car.reset_index(drop=True)     # naye index assign kare
```

[17] ✓ 0.0s

Cleaned Data

```
car
```

[18] ✓ 0.0s

	name	company	year	Price	kms_driven	fuel_type
0	Hyundai Santro Xing	Hyundai	2007	80000	45000	Petrol
1	Mahindra Jeep CL550	Mahindra	2006	425000	40	Diesel
2	Hyundai Grand i10	Hyundai	2014	325000	28000	Petrol
3	Ford EcoSport Titanium	Ford	2014	575000	36000	Diesel
4	Ford Figo	Ford	2012	175000	41000	Diesel
...	...	...	...	...	...	...
811	Maruti Suzuki Ritz	Maruti	2011	270000	50000	Petrol
812	Tata Indica V2	Tata	2009	110000	30000	Diesel
813	Toyota Corolla Altis	Toyota	2009	300000	132000	Petrol
814	Tata Zest XM	Tata	2018	260000	27000	Diesel
815	Mahindra Quanto C8	Mahindra	2013	390000	40000	Diesel

816 rows × 6 columns

```
car.to_csv('Cleaned_Car_data.csv')
```

[19] ✓ 0.0s

```
car.info()
```

[20] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 816 entries, 0 to 815
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   name        816 non-null    object
 1   company     816 non-null    object
 2   year        816 non-null    int64 
 3   Price       816 non-null    int64 
 4   kms_driven  816 non-null    int64 
 5   fuel_type   816 non-null    object
dtypes: int64(3), object(3)
memory usage: 38.4+ KB
```

```python
car.describe(include='all')
```

[21] ✓ 0.0s

	name	company	year	Price	kms_driven	fuel_type
count	816	816	816.000000	8.160000e+02	816.000000	816
unique	254	25	NaN	NaN	NaN	3
top	Maruti Suzuki Swift	Maruti	NaN	NaN	NaN	Petrol
freq	51	221	NaN	NaN	NaN	428
mean	NaN	NaN	2012.444853	4.117176e+05	46275.531863	NaN
std	NaN	NaN	4.002992	4.751844e+05	34297.428044	NaN
min	NaN	NaN	1995.000000	3.000000e+04	0.000000	NaN
25%	NaN	NaN	2010.000000	1.750000e+05	27000.000000	NaN
50%	NaN	NaN	2013.000000	2.999990e+05	41000.000000	NaN
75%	NaN	NaN	2015.000000	4.912500e+05	56818.500000	NaN
max	NaN	NaN	2019.000000	8.500003e+06	400000.000000	NaN

[ ]

```python
car=car[car['Price']<6000000]   # outlier uda die
```

[22] ✓ 0.0s

Checking relationship of Company with Price

```python
car['company'].unique()
```

[23] ✓ 0.0s

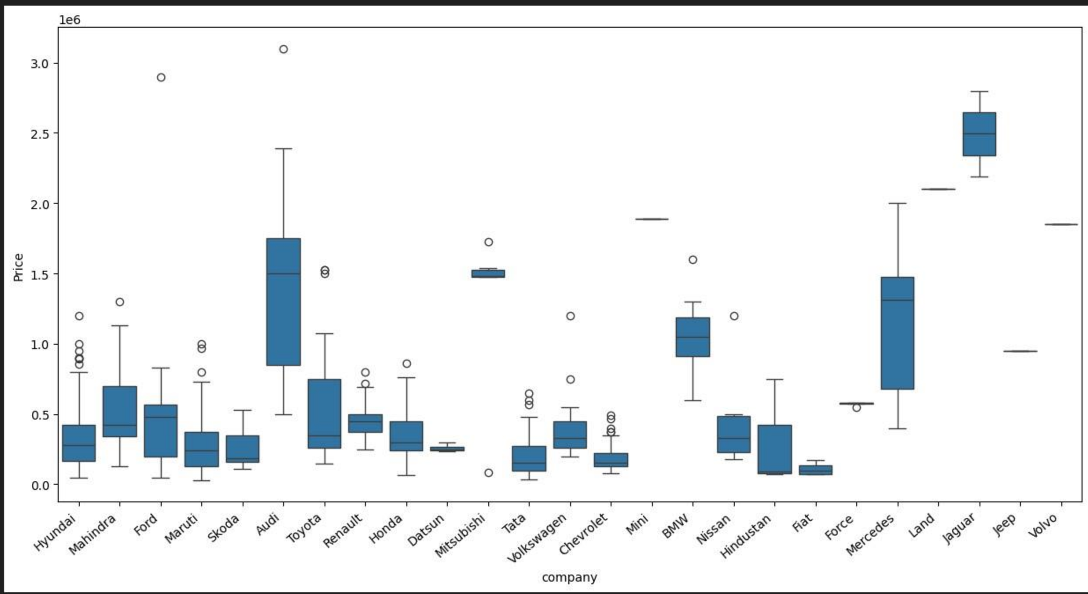
```
array(['Hyundai', 'Mahindra', 'Ford', 'Maruti', 'Skoda', 'Audi', 'Toyota',
       'Renault', 'Honda', 'Datsun', 'Mitsubishi', 'Tata', 'Volkswagen',
       'Chevrolet', 'Mini', 'BMW', 'Nissan', 'Hindustan', 'Fiat', 'Force',
       'Mercedes', 'Land', 'Jaguar', 'Jeep', 'Volvo'], dtype=object)
```

```python
import seaborn as sns
```

[24] ✓ 1.4s

```python
plt.subplots(figsize=(15,7))
ax=sns.boxplot(x='company',y='Price',data=car)
ax.set_xticklabels(ax.get_xticklabels(),rotation=40,ha='right')
plt.show()
```

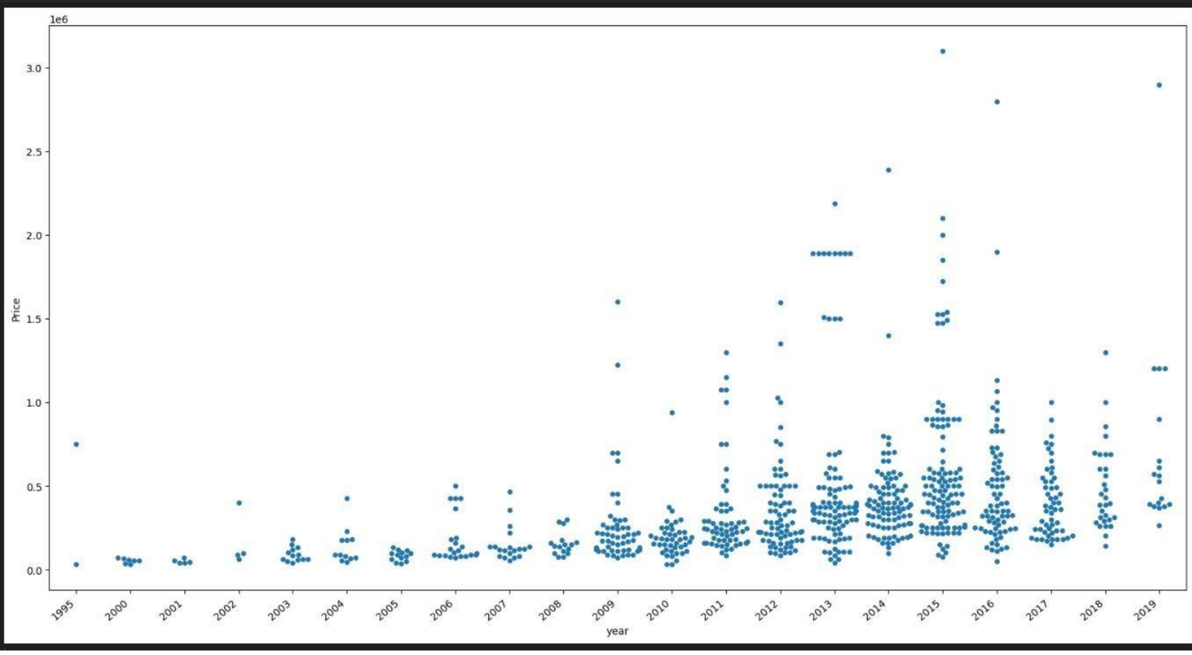
[25] 0.9s



Checking relationship of Year with Price

```python
plt.subplots(figsize=(20,10))
ax=sns.swarmplot(x='year',y='Price',data=car)
ax.set_xticklabels(ax.get_xticklabels(),rotation=40,ha='right')
plt.show()
```

[26] 1.7s

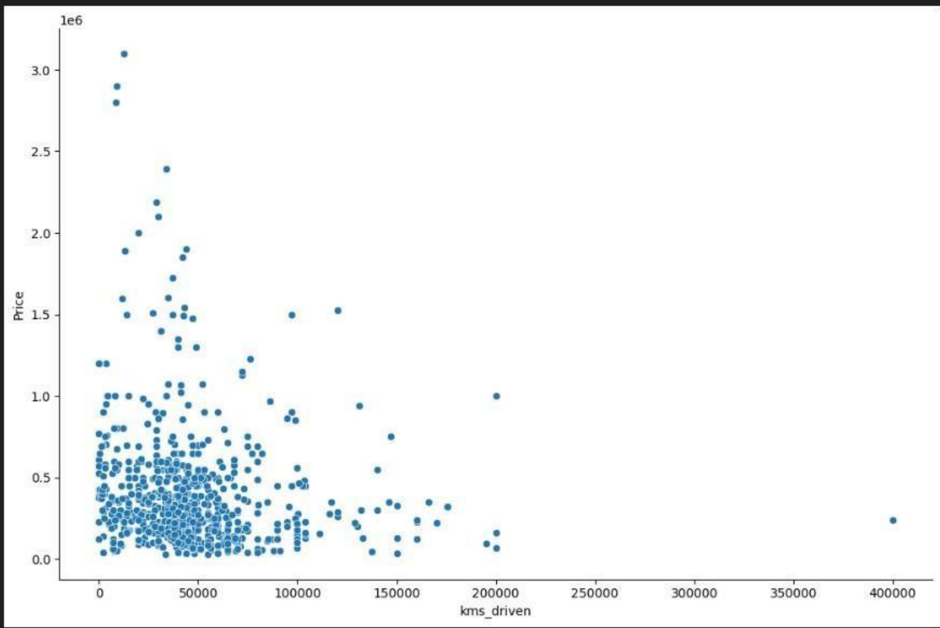


Checking relationship of kms_driven with Price

```
sns.relplot(x='kms_driven',y='Price',data=car,height=7,aspect=1.5)
```

[27] ✓ 0.6s

```
<seaborn.axisgrid.FacetGrid at 0x7f73e511d000>
```

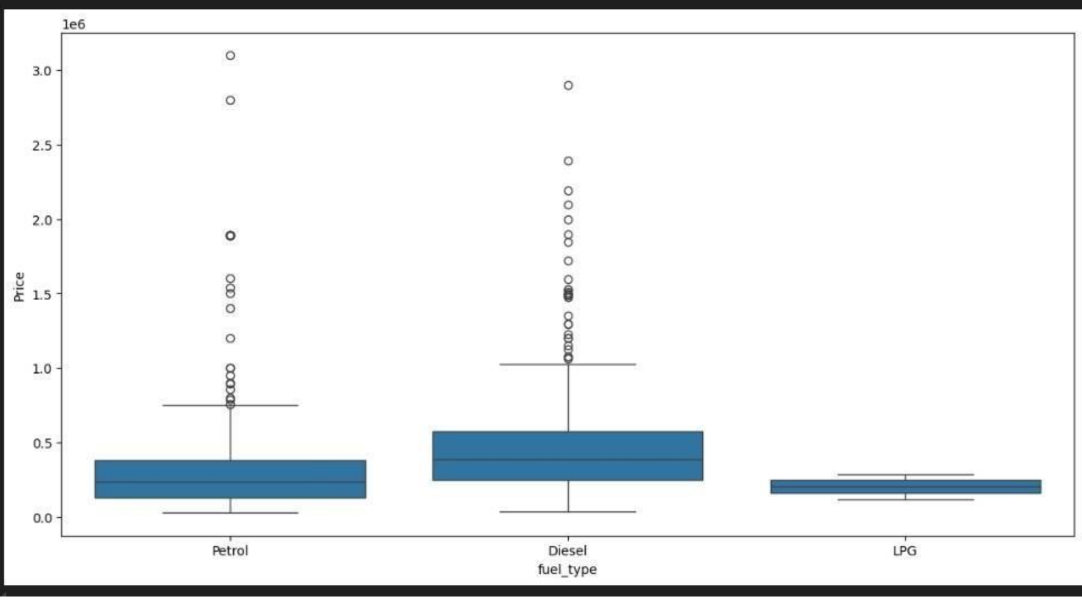


Checking relationship of Fuel Type with Price

```
plt.subplots(figsize=(14,7))
sns.boxplot(x='fuel_type',y='Price',data=car)
```

[28] ✓ 0.4s

```
<Axes: xlabel='fuel_type', ylabel='Price'>
```

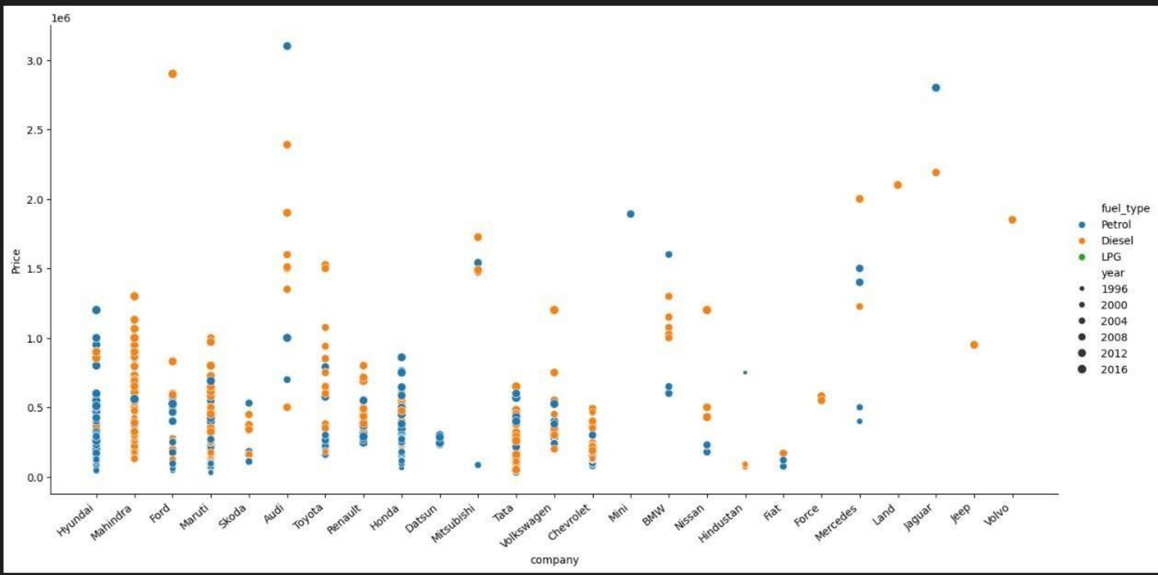


Relationship of Price with FuelType, Year ,Company

```
ax=sns.relplot(x='company',y='Price',data=car,hue='fuel_type',size='year',height=7,aspect=2)
ax.set_xticklabels(rotation=40,ha='right')
```

[29] ✓ 1.6s

```
<seaborn.axisgrid.FacetGrid at 0x7f73e57700d0>
```

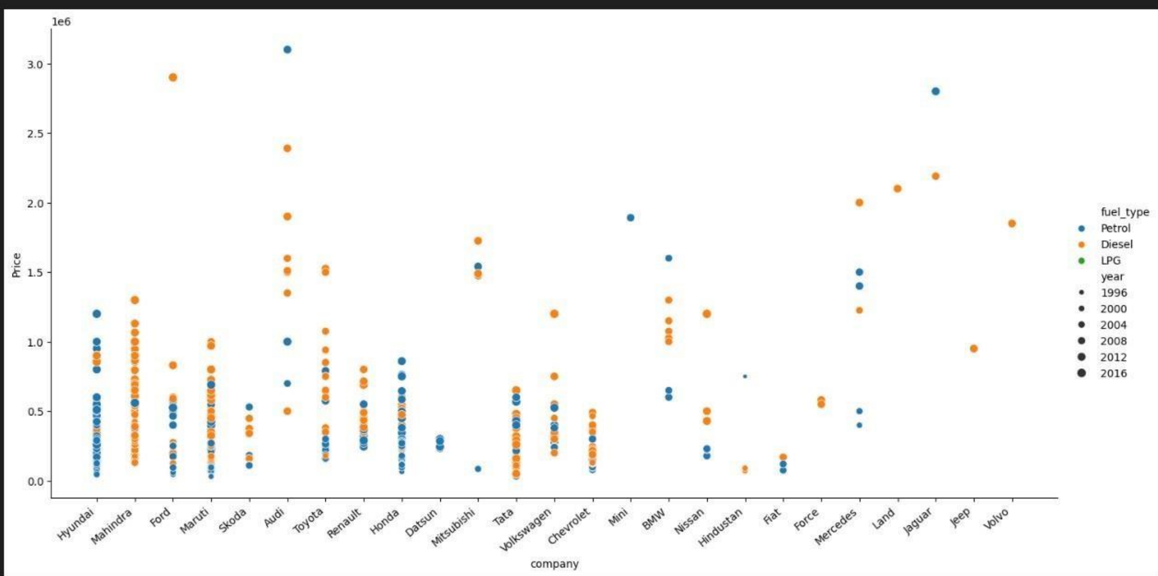


Relationship of Price with FuelType, Year ,Company

```
ax=sns.relplot(x='company',y='Price',data=car,hue='fuel_type',size='year',height=7,aspect=2)
ax.set_xticklabels(rotation=40,ha='right')
```

[29] ✓ 1.6s

```
<seaborn.axisgrid.FacetGrid at 0x7f73e57700d0>
```

Train Test Split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

[33] ✓ 0.6s

```
from sklearn.linear_model import LinearRegression
```

[34] ✓ 0.2s

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score
```

[35] ✓ 0.0s

Creating an OneHotEncoder object to contain all the possible categories

```
ohe=OneHotEncoder()
ohe.fit(X[['name','company','fuel_type']])
```

[36] ✓ 0.0s

```
OneHotEncoder
OneHotEncoder()
```

Creating a column transformer to transform categorical columns

```
column_trans=make_column_transformer((OneHotEncoder(categories=ohe.categories_),['name','company','fuel_type']),
                                    remainder='passthrough')
```

[37] ✓ 0.0s

Linear Regression Model

```
lr=LinearRegression()
```

[38] ✓ 0.0s

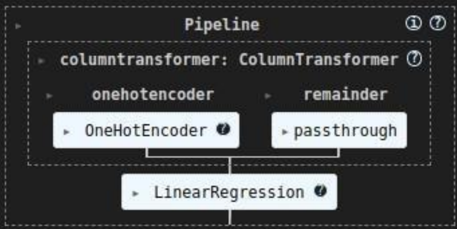
Making a pipeline

```
pipe=make_pipeline(column_trans,lr)
```

Fitting the model

```
pipe.fit(X_train,y_train)
```

[40] ✓ 0.3s




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

[41] ✓ 0.0s

Checking R2 Score

```
r2_score(y_test,y_pred)
```

[42] ✓ 0.0s

```
0.5454697805486609
```

Finding the model with a random state of TrainTestSplit where the model was found to gives a good r2 score

```
scores=[]
for i in range(1000):
    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=i)
    lr=LinearRegression()
    pipe=make_pipeline(column_trans,lr)
    pipe.fit(X_train,y_train)
    y_pred=pipe.predict(X_test)
    scores.append(r2_score(y_test,y_pred))
```

[43] ✓ 34.4s

```
np.argmax(scores)
```

[44] ✓ 0.0s

```
np.int64(302)
```


```
np.argmax(scores)
```

[44] ✓ 0.0s

```
np.int64(302)
```


```
scores[np.argmax(scores)]
```

[45] ✓ 0.0s

```
0.8991116065992623
```

The best model is found at a certain random state

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=np.argmax(scores))
lr=LinearRegression()
pipe=make_pipeline(column_trans,lr)
pipe.fit(X_train,y_train)
y_pred=pipe.predict(X_test)
r2_score(y_test,y_pred)
```

[40] ✓ 0.0s

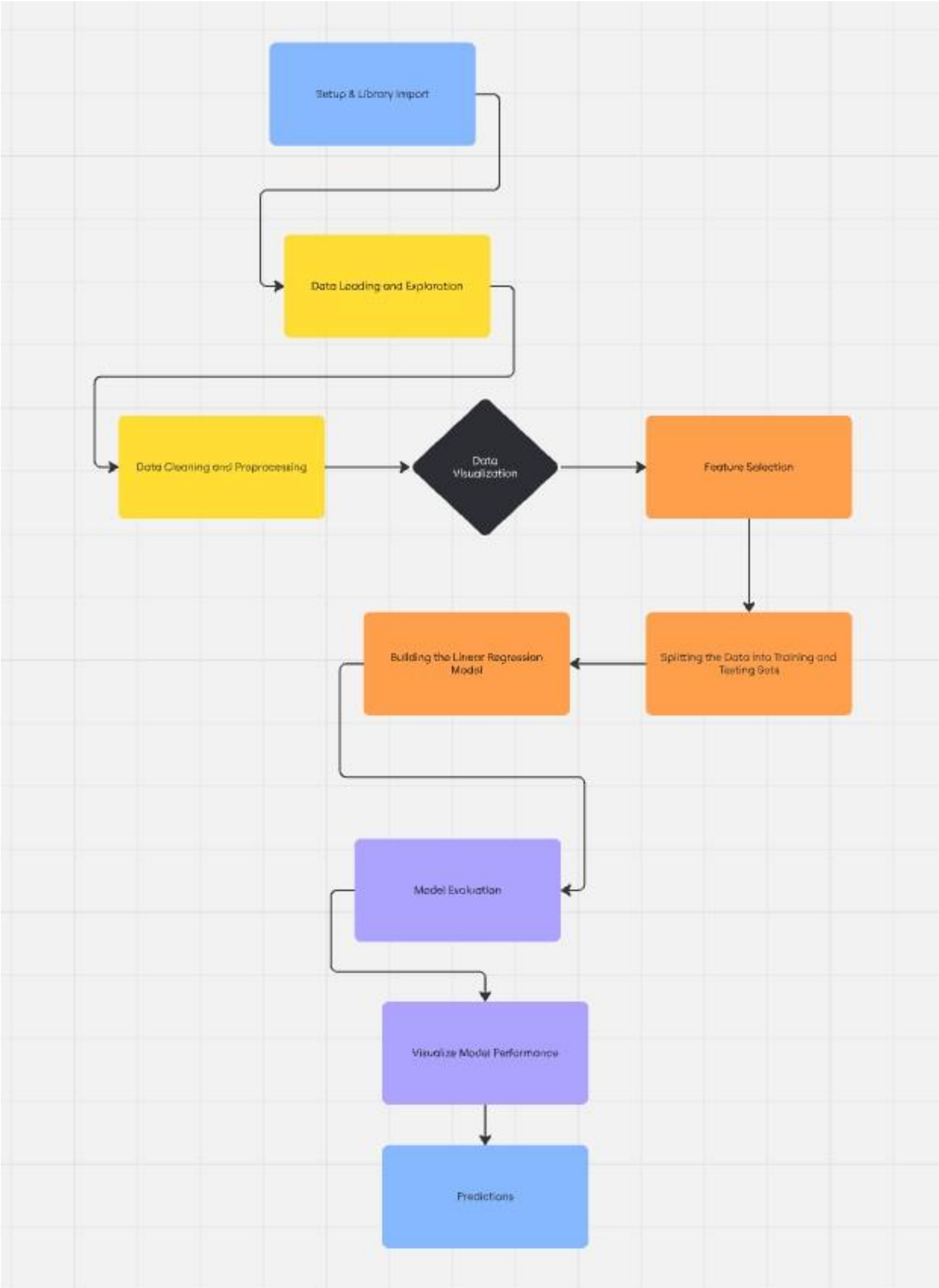
```
0.8991116065992623
```

```
pipe.predict(pd.DataFrame(columns=['name','company','year','kms_driven','fuel_type'],data=np.array(['Audi A8','Audi',2017,4000,'Petrol']).reshape(1,5)))
```

[47] ✓ 0.0s

```
array([1754418.62835164])
```

Flowchart for the implementation of the code



Conclusion

In this project, we successfully developed a systematic approach to install and verify Python packages along with their dependencies, ensuring a smooth and error-free setup process. By implementing a structured flow for installing packages like Streamlit, Altair, Cachetools, Git Python, and others, the project enhances the reliability of the environment setup, reduces manual errors, and saves time. This flowchart-based design simplifies troubleshooting, providing clarity on each step in the installation process. Overall, this project improves the efficiency of dependency management, which is crucial for robust Python development environments.

Future Enhancements

1. **Automated Dependency Resolution**:
Integrate an intelligent system that automatically resolves conflicts between package versions. For instance, if a new package requires a different version of a dependency, the system could recommend or automatically apply compatible versions.

2. **User-Friendly Interface:**
Develop a user-friendly GUI or CLI tool that walks users through the installation process and provides status updates, alerts, or error messages in real-time.

3. **Automated Testing of Dependencies**:
Implement a testing suite that verifies the functionality of each installed package after setup, ensuring that all dependencies are correctly configured and the packages work as intended.