ml_module_end_project.ipynb

File Edit View Insert Runtime Tools Help

Share

+ Code + Text

RAM Disk

Gemini

```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

# Reading the dataset
cars = pd.read_csv('/content/CarPrice_Assignment.csv')

cars_df = cars.copy()

cars_df.head()
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495.0
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500.0
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500.0
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950.0
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450.0

5 rows × 26 columns

```
[2] cars_df.shape
```

```
(205, 26)
```

```
[3] # Summary of the dataset
cars_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
```

1m 27s completed at 18:06

```
# Summary of the dataset
cars_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

```
[4] # Description of the dataset
cars_df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
car_ID	205.0	103.000000	59.322565	1.00	52.00	103.00	154.00	205.00

[4]
```
# Description of the dataset
cars_df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
car_ID	205.0	103.000000	59.322565	1.00	52.00	103.00	154.00	205.00
symboling	205.0	0.834146	1.245307	-2.00	0.00	1.00	2.00	3.00
wheelbase	205.0	98.756585	6.021776	86.60	94.50	97.00	102.40	120.90
carlength	205.0	174.049268	12.337289	141.10	166.30	173.20	183.10	208.10
carwidth	205.0	65.907805	2.145204	60.30	64.10	65.50	66.90	72.30
carheight	205.0	53.724878	2.443522	47.80	52.00	54.10	55.50	59.80
curbweight	205.0	2555.565854	520.680204	1488.00	2145.00	2414.00	2935.00	4066.00
enginesize	205.0	126.907317	41.642693	61.00	97.00	120.00	141.00	326.00
boreratio	205.0	3.329756	0.270844	2.54	3.15	3.31	3.58	3.94
stroke	205.0	3.255415	0.313597	2.07	3.11	3.29	3.41	4.17
compressionratio	205.0	10.142537	3.972040	7.00	8.60	9.00	9.40	23.00
horsepower	205.0	104.117073	39.544167	48.00	70.00	95.00	116.00	288.00
peakrpm	205.0	5125.121951	476.985643	4150.00	4800.00	5200.00	5500.00	6600.00
citympg	205.0	25.219512	6.542142	13.00	19.00	24.00	30.00	49.00
highwaympg	205.0	30.751220	6.886443	16.00	25.00	30.00	34.00	54.00
price	205.0	13276.710571	7988.852332	5118.00	7788.00	10295.00	16503.00	45400.00

[5]
```
# Checking for null values
cars_df.isnull().sum()
```

	0
car_ID	0
symboling	0

```
# Checking for null values
cars_df.isnull().sum()
```

	0
car_ID	0
symboling	0
CarName	0
fueltype	0
aspiration	0
doornumber	0
carbody	0
drivewheel	0
enginelocation	0
wheelbase	0
carlength	0
carwidth	0
carheight	0
curbweight	0
enginetype	0
cylindernumber	0
enginesize	0
fuelsystem	0
boreratio	0
stroke	0
compressionratio	0

[5]

compressionratio	0
horsepower	0
peakrpm	0
citympg	0
highwaympg	0
price	0

dtype: int64

```
[6] # Checking for duplicates
cars_df.loc[cars.duplicated()]
```

car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price

0 rows × 26 columns

No duplicates found

```
# Dropping duplicated records (if any)
cars_df = cars_df.drop_duplicates()
```

```
# Dropping car_ID attribute
cars_df.drop('car_ID', axis=1, inplace=True)
```

```
[9] # Converting symbolin to categorical
cars_df['symboling'] = cars_df['symboling'].astype('object')
cars_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   symboling         205 non-null    object
```

```
cars_df.drop('car_ID', axis=1, inplace=True)
```

```
[9] # Converting symbolin to categorical
cars_df['symboling'] = cars_df['symboling'].astype('object')
cars_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   symboling         205 non-null    object
 1   CarName           205 non-null    object
 2   fueltype          205 non-null    object
 3   aspiration        205 non-null    object
 4   doornumber        205 non-null    object
 5   carbody           205 non-null    object
 6   drivewheel        205 non-null    object
 7   enginelocation    205 non-null    object
 8   wheelbase         205 non-null    float64
 9   carlength         205 non-null    float64
 10  carwidth          205 non-null    float64
 11  carheight         205 non-null    float64
 12  curbweight        205 non-null    int64
 13  enginetype        205 non-null    object
 14  cylindernumber    205 non-null    object
 15  enginesize        205 non-null    int64
 16  fuelsystem        205 non-null    object
 17  boreratio         205 non-null    float64
 18  stroke            205 non-null    float64
 19  compressionratio  205 non-null    float64
 20  horsepower        205 non-null    int64
 21  peakrpm           205 non-null    int64
 22  citympg           205 non-null    int64
 23  highwaympg        205 non-null    int64
 24  price             205 non-null    float64
dtypes: float64(8), int64(6), object(11)
memory usage: 40.2+ KB
```

```
[10] # Splitting company name from CarName column
CompanyName = cars_df['CarName'].apply(lambda x : x.split(' ')[0])

CompanyName
```

[10]
```
# Splitting company name from CarName column
CompanyName = cars_df['CarName'].apply(lambda x : x.split(' ')[0])

CompanyName
```

	CarName
0	alfa-romero
1	alfa-romero
2	alfa-romero
3	audi
4	audi
...	...
200	volvo
201	volvo
202	volvo
203	volvo
204	volvo

205 rows × 1 columns

dtype: object

[11]
```
# Adding Company name to DataFrame
cars_df.insert(3, 'CompanyName', CompanyName)
```

[12]
```
# Dropping CarName from the DataFrame
cars_df.drop(['CarName'], axis=1, inplace=True)

cars_df.head()
```

	symboling	fueltype	CompanyName	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price

```
[11] # Adding Company name to DataFrame
     cars_df.insert(3, 'CompanyName', CompanyName)
```

```
[12] # Dropping CarName from the DataFrame
     cars_df.drop(['CarName'], axis=1, inplace=True)

     cars_df.head()
```

	symboling	fueltype	CompanyName	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price
0	3	gas	alfa-romero	std	two	convertible	rwd	front	88.6	168.8	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495.0
1	3	gas	alfa-romero	std	two	convertible	rwd	front	88.6	168.8	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500.0
2	1	gas	alfa-romero	std	two	hatchback	rwd	front	94.5	171.2	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500.0
3	2	gas	audi	std	four	sedan	fwd	front	99.8	176.6	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950.0
4	2	gas	audi	std	four	sedan	4wd	front	99.4	176.6	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450.0

5 rows × 25 columns

```
# Unique Company Names
cars_df['CompanyName'].unique()
```

```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'maxda', 'mazda', 'buick', 'mercury',
       'mitsubishi', 'Nissan', 'nissan', 'peugeot', 'plymouth', 'porsche',
       'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyouta',
       'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)
```

```
[14] # Renaming the mis-spelled Car Brands
     cars_df['CompanyName'].replace('maxda', 'mazda', inplace=True)
     cars_df['CompanyName'].replace('Nissan', 'nissan', inplace=True)
     cars_df['CompanyName'].replace('porcshce', 'porsche', inplace=True)
     cars_df['CompanyName'].replace('toyouta', 'toyota', inplace=True)
     cars_df['CompanyName'].replace(['vokswagen', 'vw'], 'volkswagen', inplace=True)

     cars_df['CompanyName'].unique()
```

```
       'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyouta',
       'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)
```

[14]
```
# Renaming the mis-spelled Car Brands
cars_df['CompanyName'].replace('maxda', 'mazda', inplace=True)
cars_df['CompanyName'].replace('Nissan', 'nissan', inplace=True)
cars_df['CompanyName'].replace('porcshce', 'porsche', inplace=True)
cars_df['CompanyName'].replace('toyouta', 'toyota', inplace=True)
cars_df['CompanyName'].replace(['vokswagen', 'vw'], 'volkswagen', inplace=True)

cars_df['CompanyName'].unique()
```

```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mazda', 'buick', 'mercury', 'mitsubishi',
       'nissan', 'peugeot', 'plymouth', 'porsche', 'renault', 'saab',
       'subaru', 'toyota', 'volkswagen', 'volvo'], dtype=object)
```

[15]
```
# Unique fuel type
cars_df['fuelsystem'].unique()
```

```
array(['mpfi', '2bbl', 'mfi', '1bbl', 'spfi', '4bbl', 'idi', 'spdi'],
      dtype=object)
```

[16]
```
# Renaming mis-spelled fuel type
cars_df['fuelsystem'].replace('mfi', 'mpfi')
```

	fuelsystem
0	mpfi
1	mpfi
2	mpfi
3	mpfi
4	mpfi
...	...
200	mpfi

[16]
```
# Renaming mis-spelled fuel type
cars_df['fuelsystem'].replace('mfi', 'mpfi')
```

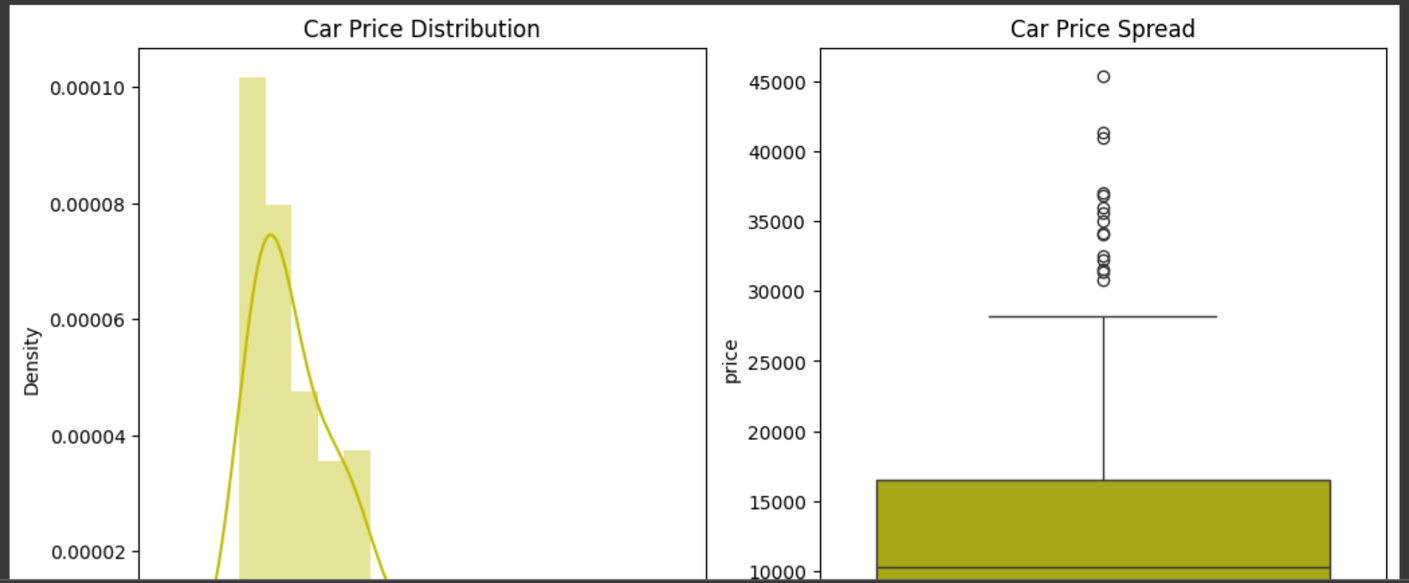
	fuelsystem
0	mpfi
1	mpfi
2	mpfi
3	mpfi
4	mpfi
...	...
200	mpfi
201	mpfi
202	mpfi
203	idi
204	mpfi

205 rows × 1 columns

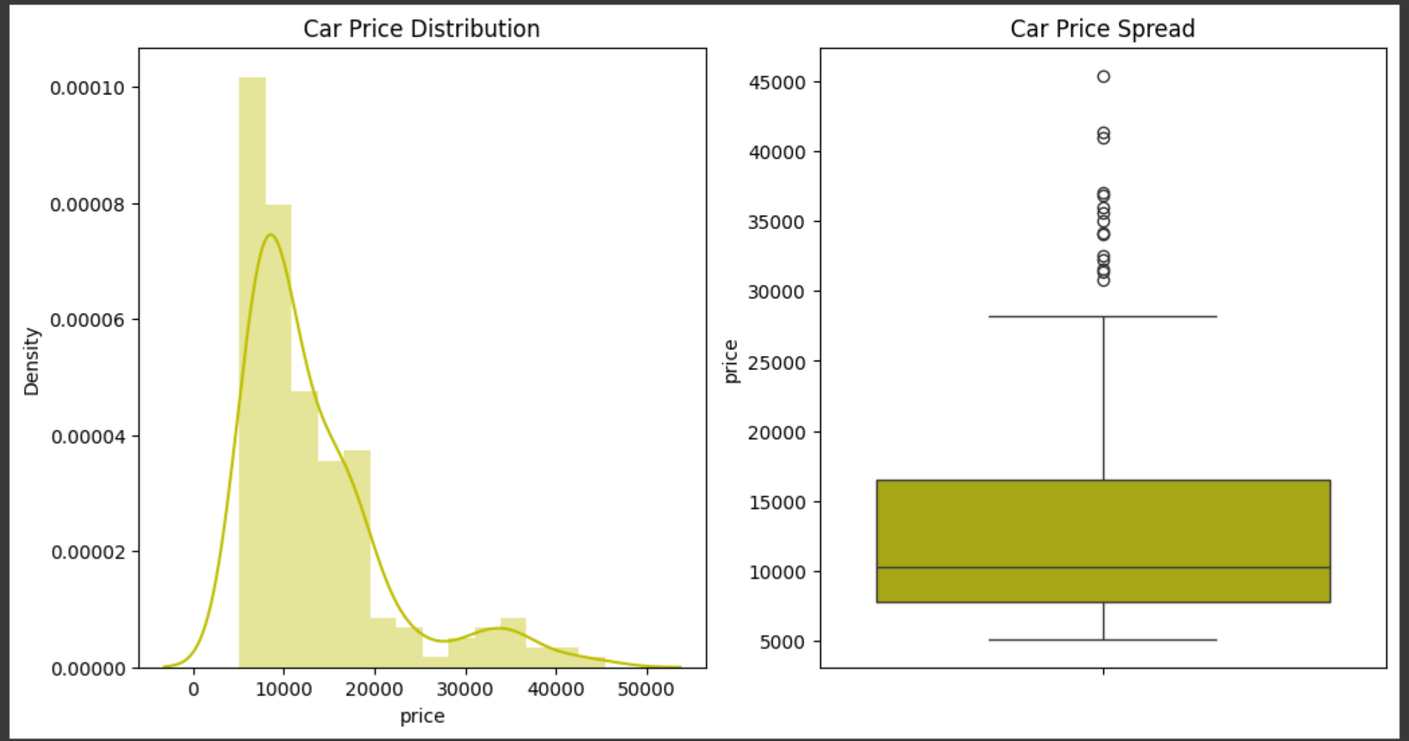
dtype: object

[17]
```
# Visalise the Data
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.title('Car Price Distribution')

sns.distplot(cars_df.price, color='y')

plt.subplot(1, 2, 2)
```

```
# Visalise the Data
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.title('Car Price Distribution')

sns.distplot(cars_df.price, color='y')

plt.subplot(1, 2, 2)
plt.title('Car Price Spread')
sns.boxplot(y=cars_df.price, color='y')

plt.show()
```

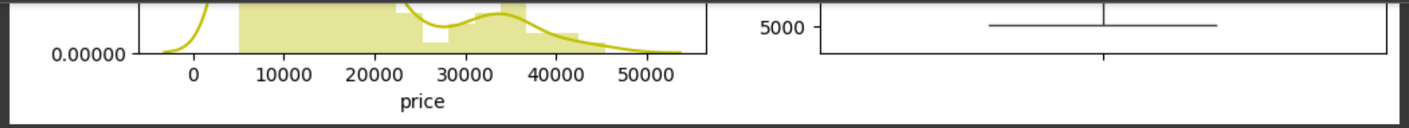


```
plt.title('Car Price Distribution')

sns.distplot(cars_df.price, color='y')

plt.subplot(1, 2, 2)
plt.title('Car Price Spread')
sns.boxplot(y=cars_df.price, color='y')

plt.show()
```




```
[18] # Checking car price distribution
```

[17]




```
# Checking car price distribution
cars_df.price.describe(percentiles = [0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90,1])
```

	price
count	205.000000
mean	13276.710571
std	7988.852332
min	5118.000000
10%	6657.000000
20%	7385.800000
30%	8022.000000
40%	9036.600000
50%	10295.000000
60%	12515.600000
70%	15458.000000
80%	17493.800000
90%	22563.000000
100%	45400.000000
max	45400.000000

dtype: float64

Insights:

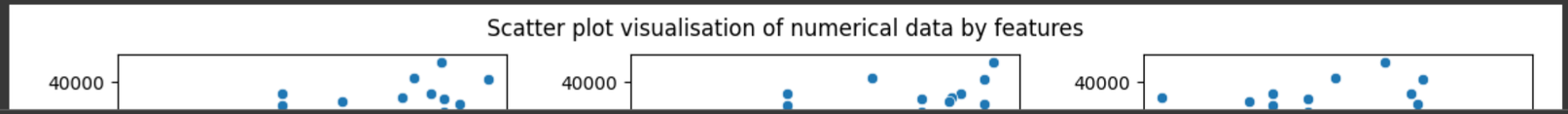
[18]

dtype: float64

Insights:

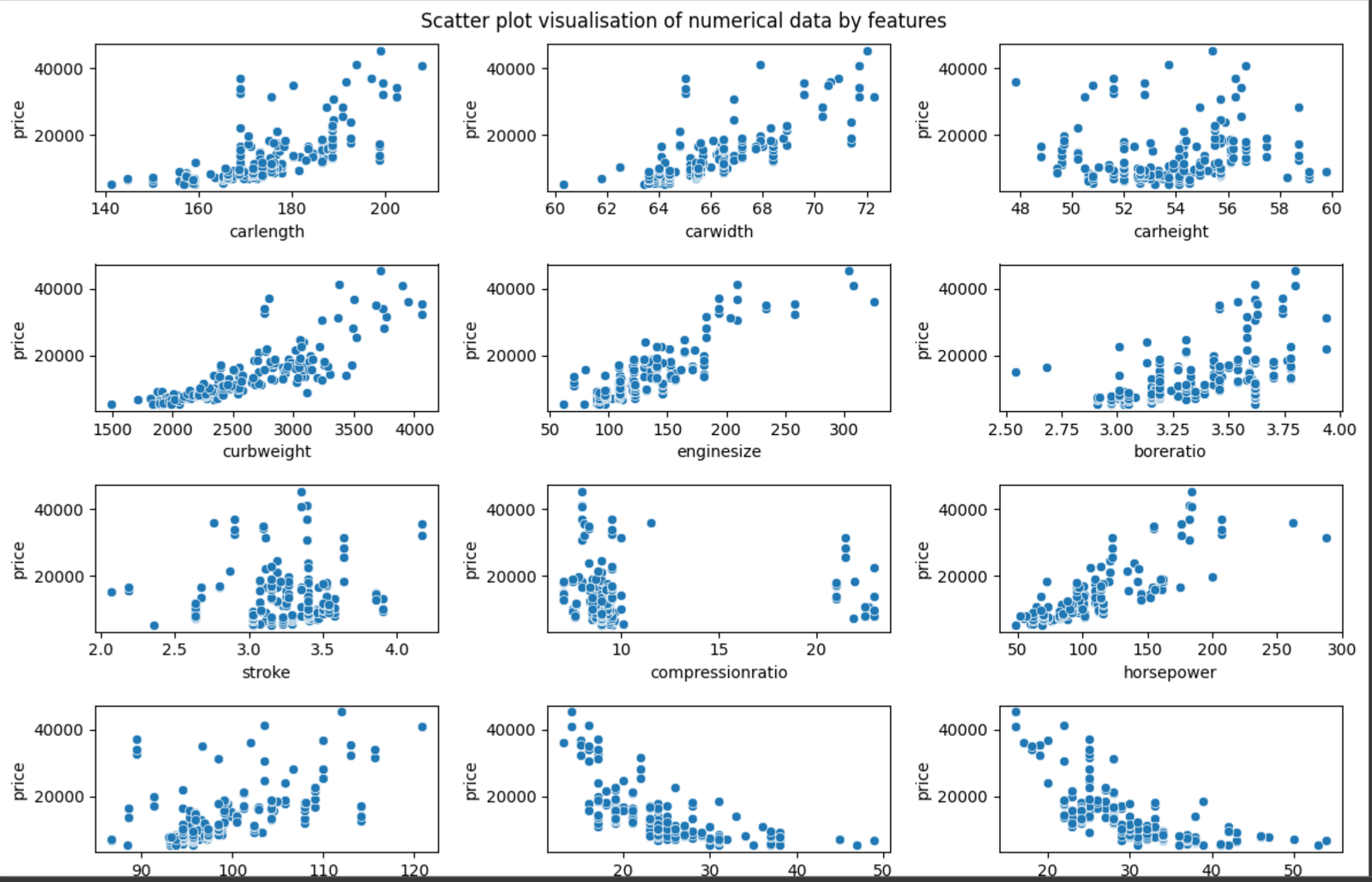
1. From the graphs, it seems that data is right skewed, which means most car prices in the dataset are below 17,493
2. Around 80% of the car price is below 17,493

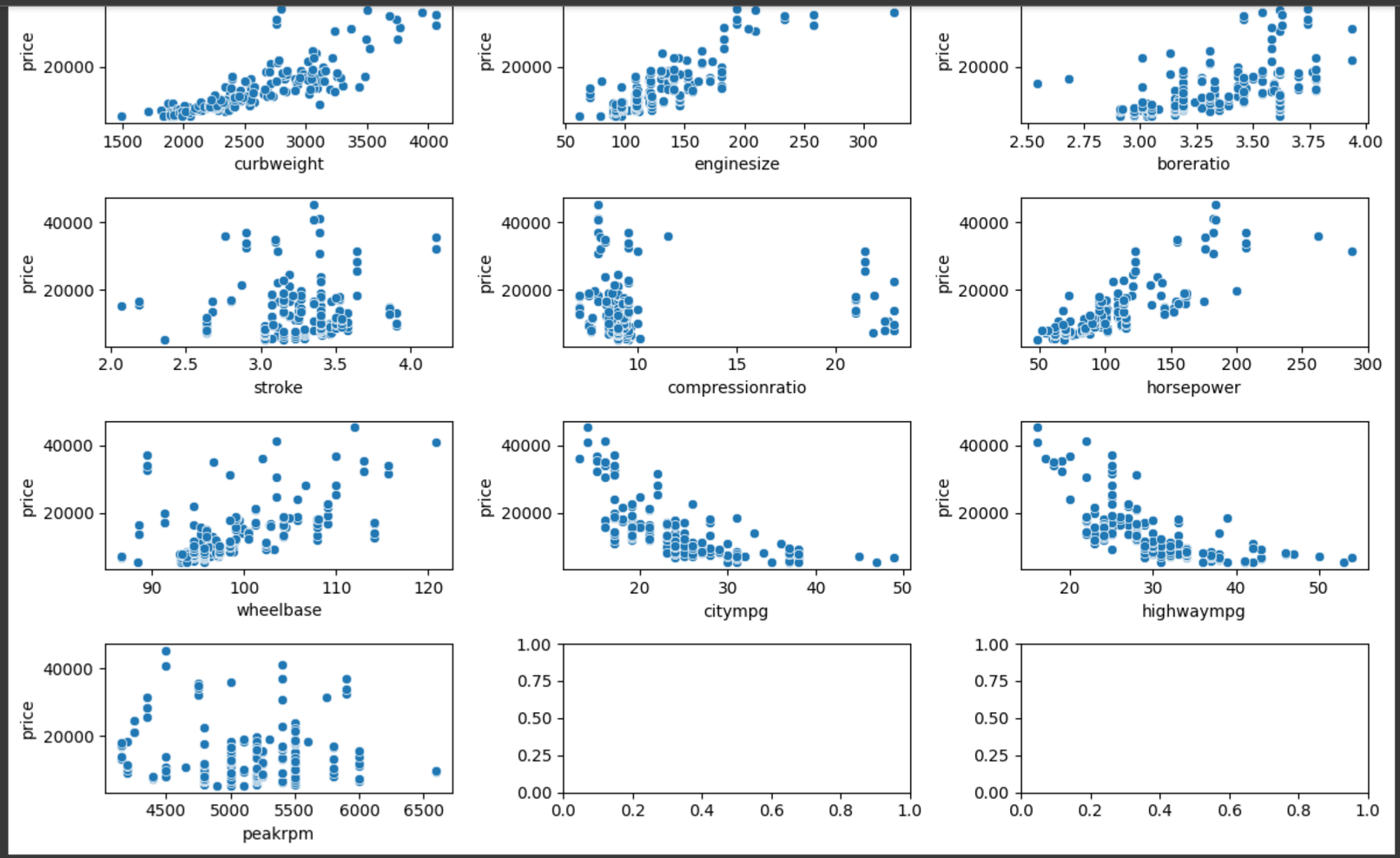
```
# Visualisation of numerical data by features
fig, axes = plt.subplots(5, 3, figsize=(12, 10))
fig.suptitle('Scatter plot visualisation of numerical data by features')

sns.scatterplot(ax=axes[0, 0], data=cars_df, x='carlength', y='price')
sns.scatterplot(ax=axes[0, 1], data=cars_df, x='carwidth', y='price')
sns.scatterplot(ax=axes[0, 2], data=cars_df, x='carheight', y='price')

sns.scatterplot(ax=axes[1, 0], data=cars_df, x='curbweight', y='price')
sns.scatterplot(ax=axes[1, 1], data=cars_df, x='enginesize', y='price')
sns.scatterplot(ax=axes[1, 2], data=cars_df, x='boreratio', y='price')

sns.scatterplot(ax=axes[2, 0], data=cars_df, x='stroke', y='price')
sns.scatterplot(ax=axes[2, 1], data=cars_df, x='compressionratio', y='price')
sns.scatterplot(ax=axes[2, 2], data=cars_df, x='horsepower', y='price')

sns.scatterplot(ax=axes[3, 0], data=cars_df, x='wheelbase', y='price')
sns.scatterplot(ax=axes[3, 1], data=cars_df, x='citympg', y='price')
sns.scatterplot(ax=axes[3, 2], data=cars_df, x='highwaympg', y='price')

sns.scatterplot(ax=axes[4, 0], data=cars_df, x='peakrpm', y='price')

plt.tight_layout()
plt.show()
```



[19]

Scatter plot visualisation of numerical data by features






```
[20] # Numerical features
```

peakrpm

```
# Numerical features
cars_numeric = cars_df.select_dtypes(exclude='object')

# Correlation matrix
cor = cars_numeric.corr()

cor
```

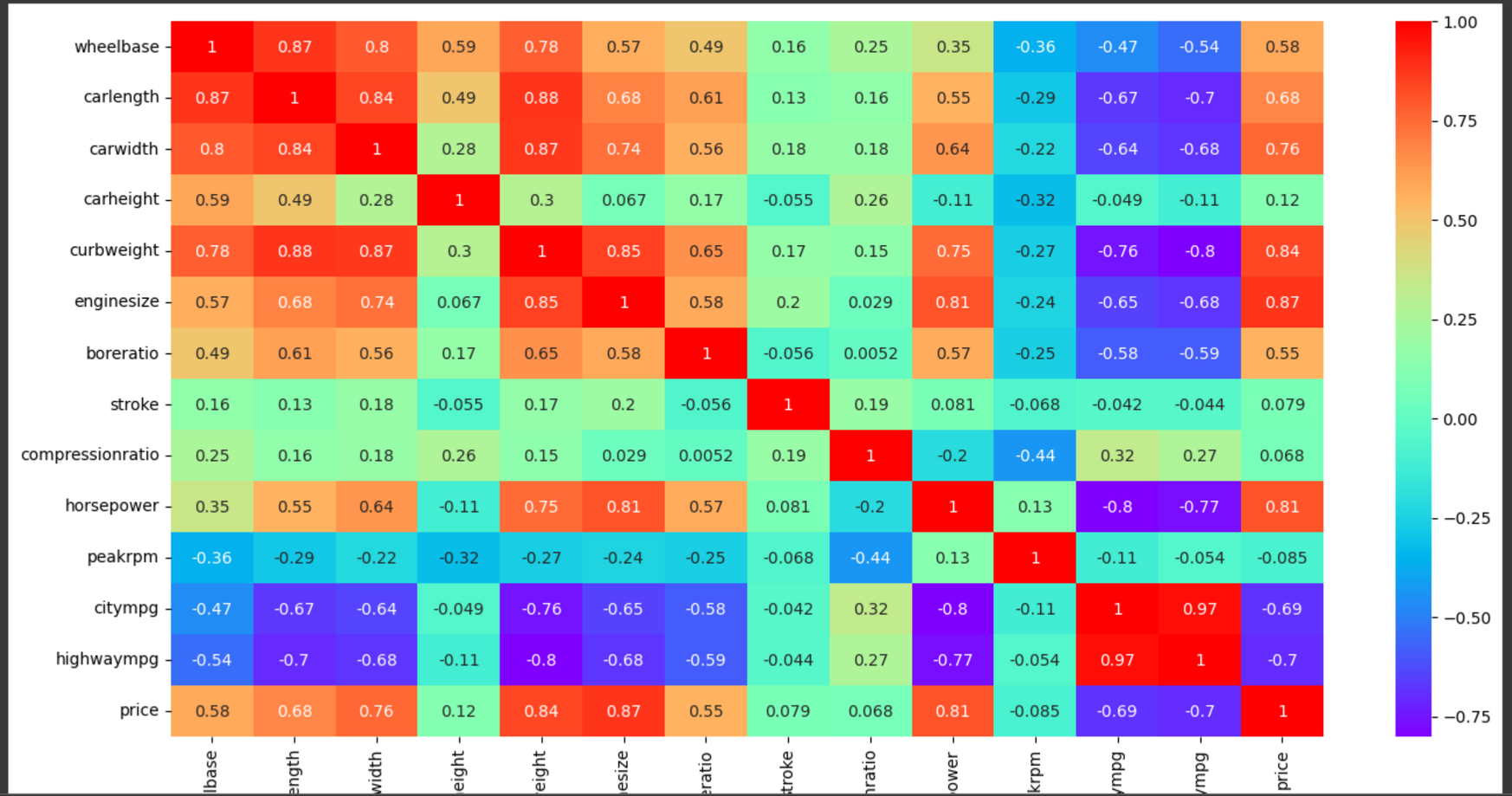
	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price
wheelbase	1.000000	0.874587	0.795144	0.589435	0.776386	0.569329	0.488750	0.160959	0.249786	0.353294	-0.360469	-0.470414	-0.544082	0.577816
carlength	0.874587	1.000000	0.841118	0.491029	0.877728	0.683360	0.606454	0.129533	0.158414	0.552623	-0.287242	-0.670909	-0.704662	0.682920
carwidth	0.795144	0.841118	1.000000	0.279210	0.867032	0.735433	0.559150	0.182942	0.181129	0.640732	-0.220012	-0.642704	-0.677218	0.759325
carheight	0.589435	0.491029	0.279210	1.000000	0.295572	0.067149	0.171071	-0.055307	0.261214	-0.108802	-0.320411	-0.048640	-0.107358	0.119336
curbweight	0.776386	0.877728	0.867032	0.295572	1.000000	0.850594	0.648480	0.168790	0.151362	0.750739	-0.266243	-0.757414	-0.797465	0.835305
enginesize	0.569329	0.683360	0.735433	0.067149	0.850594	1.000000	0.583774	0.203129	0.028971	0.809769	-0.244660	-0.653658	-0.677470	0.874145
boreratio	0.488750	0.606454	0.559150	0.171071	0.648480	0.583774	1.000000	-0.055909	0.005197	0.573677	-0.254976	-0.584532	-0.587012	0.553173
stroke	0.160959	0.129533	0.182942	-0.055307	0.168790	0.203129	-0.055909	1.000000	0.186110	0.080940	-0.067964	-0.042145	-0.043931	0.079443
compressionratio	0.249786	0.158414	0.181129	0.261214	0.151362	0.028971	0.005197	0.186110	1.000000	-0.204326	-0.435741	0.324701	0.265201	0.067984
horsepower	0.353294	0.552623	0.640732	-0.108802	0.750739	0.809769	0.573677	0.080940	-0.204326	1.000000	0.131073	-0.801456	-0.770544	0.808139
peakrpm	-0.360469	-0.287242	-0.220012	-0.320411	-0.266243	-0.244660	-0.254976	-0.067964	-0.435741	0.131073	1.000000	-0.113544	-0.054275	-0.085267
citympg	-0.470414	-0.670909	-0.642704	-0.048640	-0.757414	-0.653658	-0.584532	-0.042145	0.324701	-0.801456	-0.113544	1.000000	0.971337	-0.685751
highwaympg	-0.544082	-0.704662	-0.677218	-0.107358	-0.797465	-0.677470	-0.587012	-0.043931	0.265201	-0.770544	-0.054275	0.971337	1.000000	-0.697599
price	0.577816	0.682920	0.759325	0.119336	0.835305	0.874145	0.553173	0.079443	0.067984	0.808139	-0.085267	-0.685751	-0.697599	1.000000

Next steps: Generate code with cor | View recommended plots | New interactive sheet

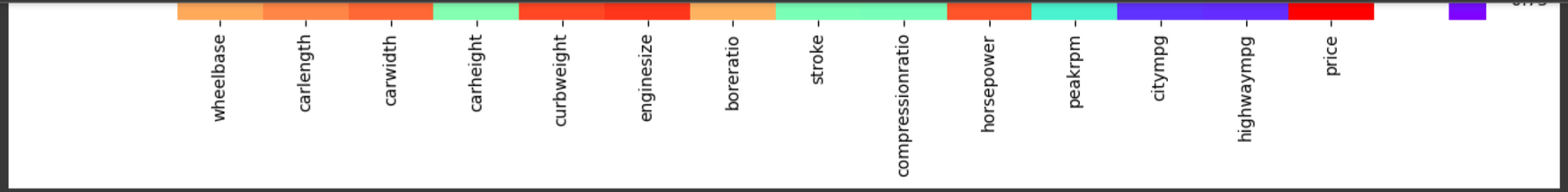
```
[21] # Plotting correlations on a heatmap
plt.figure(figsize=(16, 8))
```

```
# Plotting correlations on a heatmap
plt.figure(figsize=(16, 8))
sns.heatmap(cor, cmap='rainbow', annot=True)
```

<Axes: >



[21]



Insights:

1. Features 'carwidth', 'carlength', 'curbweight', 'enginesize', 'boreratio', 'horsepower' and 'wheelbase' have positive correlation with price
2. 'carheight', 'stroke', 'compressionratio', 'peakrpm' doesn't have any correlation with price.
3. 'citympg' and 'highwaympg' have negative correlation with price

[22]
```
# Removing 'carheight', 'stroke', 'compressionratio'
cars_df = cars_df.drop(columns=['carheight', 'stroke', 'compressionratio'])

cars_df.columns
```

```
Index(['symboling', 'fueltype', 'CompanyName', 'aspiration', 'doornumber',
       'carbody', 'drivewheel', 'enginelocation', 'wheelbase', 'carlength',
       'carwidth', 'curbweight', 'enginetype', 'cylindernumber', 'enginesize',
       'fuelsystem', 'boreratio', 'horsepower', 'peakrpm', 'citympg',
       'highwaympg', 'price'],
      dtype='object')
```

```
# Categorical features
cars_categorical = cars_df.select_dtypes(include=['object'])

cars_categorical.columns
```

```
Index(['symboling', 'fueltype', 'CompanyName', 'aspiration', 'doornumber',
       'carbody', 'drivewheel', 'enginelocation', 'enginetype',
       'cylindernumber', 'fuelsystem'],
      dtype='object')
```

```
[24] # Encoding categorical variables
     dummy = pd.get_dummies(cars_categorical, drop_first=True)

     # Dropping categorical variables
     cars_df = cars_df.drop(columns=cars_categorical)

     # Adding Encoded variables to DataFrame
     cars_df = pd.concat([cars_df, dummy], axis=1)

     cars_df.head()
```

	wheelbase	carlength	carwidth	curbweight	enginesize	boreratio	horsepower	peakrpm	citympg	highwaympg	...	cylindernumber_three	cylindernumber_twelve	cylindernumber_two	fuelsystem_2bbl	fuelsystem_4bbl	fuelsystem_idi	fuel
0	88.6	168.8	64.1	2548	130	3.47	111	5000	21	27	...	False	False	False	False	False	False	
1	88.6	168.8	64.1	2548	130	3.47	111	5000	21	27	...	False	False	False	False	False	False	
2	94.5	171.2	65.5	2823	152	2.68	154	5000	19	26	...	False	False	False	False	False	False	
3	99.8	176.6	66.2	2337	109	3.19	102	5500	24	30	...	False	False	False	False	False	False	
4	99.4	176.6	66.4	2824	136	3.19	115	5500	18	22	...	False	False	False	False	False	False	

5 rows × 66 columns

```
[25] # Separate features and target

     X = cars_df.drop('price', axis=1)

     y = cars_df['price']
```

```
[26] # Split the data into training and testing sets
     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, random_state=42)
```

```
[27] # Scaling features
     from sklearn.preprocessing import StandardScaler
```

```
[27] # Scaling features
     from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
```

Model implementation

```
[28] from sklearn.linear_model import LinearRegression
     from sklearn.tree import DecisionTreeRegressor
     from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
     from sklearn.svm import SVR

     # Define models
     models = {
         'Linear Regression' : LinearRegression(),
         'Decision Tree' : DecisionTreeRegressor(random_state=42),
         'Random Forest' : RandomForestRegressor(random_state=42),
         'Gradient Boosting' : GradientBoostingRegressor(random_state=42),
         'Support Vector Regressor' : SVR()
     }

     # Train and evaluate models
     results = {}
     for name, model in models.items():
       model.fit(X_train, y_train)
       y_pred = model.predict(X_test)
       results[name] = y_pred
```

Model Evaluation

```
[29] from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

     # Evaluate models
```

Model Evaluation

[29]
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# Evaluate models
evaluation = {}
for name, y_pred in results.items():
  evaluation[name] = {
      'R2' : r2_score(y_test, y_pred),
      'MSE' : mean_squared_error(y_test, y_pred),
      'MAE' : mean_absolute_error(y_test, y_pred)
  }

# Display results
for model, metrics in evaluation.items():
  print(f'Model : {model}')
  for metric, value in metrics.items():
    print(f'{metric} : {value}')
  print()
```

```
Model : Linear Regression
R2 : 0.903202735376224
MSE : 7641565.279553128
MAE : 1921.28435314806

Model : Decision Tree
R2 : 0.9043206861899992
MSE : 7553309.747168025
MAE : 1790.7113902439025

Model : Random Forest
R2 : 0.9554782571687123
MSE : 3514725.3956719316
MAE : 1305.399300813008

Model : Gradient Boosting
R2 : 0.924021153321103
MSE : 5998075.658631154
MAE : 1764.800292307468
```

```
[29] for metric, value in metrics.items():
       print(f'{metric} : {value}')
     print()
```

```
Model : Linear Regression
R2 : 0.903202735376224
MSE : 7641565.279553128
MAE : 1921.28435314806

Model : Decision Tree
R2 : 0.9043206861899992
MSE : 7553309.747168025
MAE : 1790.7113902439025

Model : Random Forest
R2 : 0.9554782571687123
MSE : 3514725.3956719316
MAE : 1305.399300813008

Model : Gradient Boosting
R2 : 0.924021153321103
MSE : 5998075.658631154
MAE : 1764.800292307468

Model : Support Vector Regressor
R2 : -0.1006541147030382
MSE : 86890061.41239779
MAE : 5701.703206690594
```

Best Performing Model : Random Forest Regressor

- R^2: 0.9554 (Highest, indicating strong predictive power)
- MSE : 3,514,725 (Lowest, showing minimal error in predictions)
- MAE : 1,305 (Lowest, reflecting the least average deviation from actual prices)

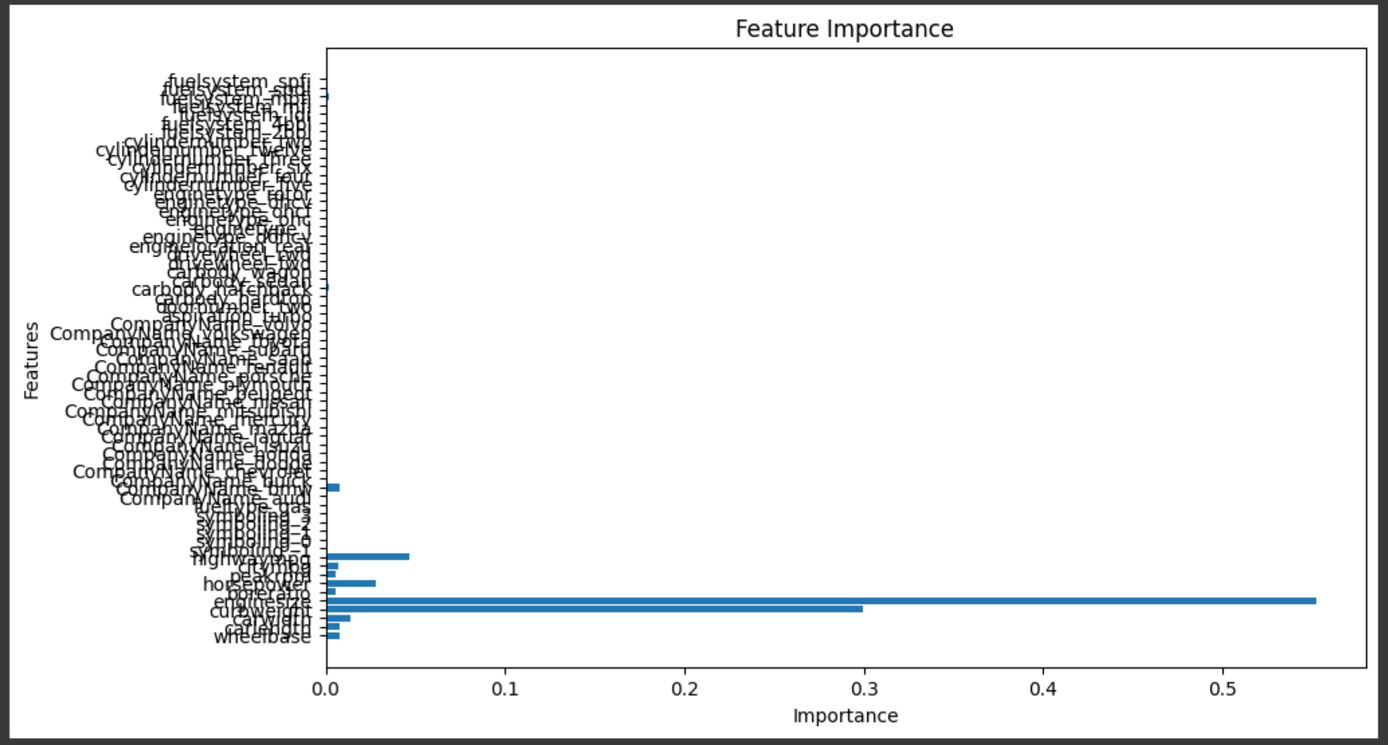
Feature Importance Analysis

```
[30] import matplotlib.pyplot as plt
```

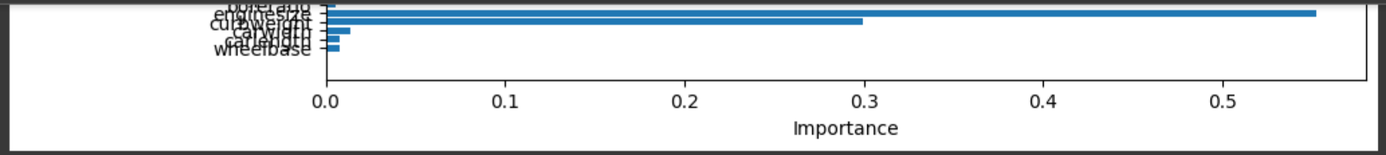
Feature Importance Analysis

```
[30] import matplotlib.pyplot as plt

     # Use the Random Forest Regressor
     feature_importances = models['Random Forest'].feature_importances_

     # Visualize feature importance
     features = X.columns
     plt.figure(figsize=(10, 6))
     plt.barh(features, feature_importances)
     plt.xlabel('Importance')
     plt.ylabel('Features')
     plt.title('Feature Importance')
     plt.show()
```

```python
# Visualize feature importance
features = X.columns
plt.figure(figsize=(10, 6))
plt.barh(features, feature_importances)
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Feature Importance')
plt.show()
```




```
[31] from sklearn.feature_selection import RFE, SelectKBest, mutual_info_regression
```



[31]
```
from sklearn.feature_selection import RFE, SelectKBest, mutual_info_regression

# Recursive Feature Elimiination (RFE) with Random Forest Regressor
rfe_selector_rf = RFE(estimator=RandomForestRegressor(random_state=42), n_features_to_select=5)
rfe_selector_rf.fit(X, y)

# Selected features using RFE
selected_features_rfe = X.columns[rfe_selector_rf.support_]
print('Top 5 features using RFE with Random Forest:')
print(list(selected_features_rfe))

# SelectKBest with mutual_info_regression
kbest_selector_rf = SelectKBest(score_func=mutual_info_regression, k=5)
kbest_selector_rf.fit(X, y)

# Selected features using SelectKBest
selected_features_kbest = X.columns[kbest_selector_rf.get_support()]
print('\nTop 5 features using SelectKBest with mutual_info_regression:')
print(list(selected_features_kbest))
```

```
Top 5 features using RFE with Random Forest:
['carwidth', 'curbweight', 'enginesize', 'horsepower', 'highwaympg']

Top 5 features using SelectKBest with mutual_info_regression:
['curbweight', 'enginesize', 'horsepower', 'citympg', 'highwaympg']
```

```
from sklearn.feature_selection import f_regression

# Splitting categorical and numerical features
categorical_cols = X.select_dtypes(include=['object', 'bool']).columns
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns
```

```
['carwidth', 'curbweight', 'enginesize', 'horsepower', 'highwaympg']
```

19s [31]

```
Top 5 features using SelectKBest with mutual_info_regression:
['curbweight', 'enginesize', 'horsepower', 'citympg', 'highwaympg']
```

0s [32]

```
from sklearn.feature_selection import f_regression

# Splitting categorical and numerical features
categorical_cols = X.select_dtypes(include=['object', 'bool']).columns
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns

X_categorical = X[categorical_cols]
X_numerical = X[numerical_cols]

# Mutual Information Regression for Categorical Features
mi_selector = SelectKBest(score_func=mutual_info_regression, k=5)
X_mi = mi_selector.fit_transform(X_categorical, y)
mi_selected_features = X_categorical.columns[mi_selector.get_support()]

# ANOVA F-test for Numerical Features
f_selector = SelectKBest(score_func=f_regression, k=5)
X_f = f_selector.fit_transform(X_numerical, y)
f_selected_features = X_numerical.columns[f_selector.get_support()]

# Display selected features
print('Mutual Information selected for Categorical features:')
print(list(mi_selected_features))
print('\nANOVA (F-test) selected for Numerical features:')
print(list((f_selected_features)))
```

```
Mutual Information selected for Categorical features:
['drivewheel_fwd', 'drivewheel_rwd', 'cylindernumber_four', 'fuelsystem_2bbl', 'fuelsystem_mpfi']

ANOVA (F-test) selected for Numerical features:
['carwidth', 'curbweight', 'enginesize', 'horsepower', 'highwaympg']
```

- Recursive Feature Elimination (RFE) with Random Forest Regressor to capture model-based importance
- Mutual Information Regression for categorical features to assess non-linear dependencies. It was specifically applied to categorical features to understand their influence on car prices.

[32]
```
# Splitting categorical and numerical features
categorical_cols = X.select_dtypes(include=['object', 'bool']).columns
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns

X_categorical = X[categorical_cols]
X_numerical = X[numerical_cols]

# Mutual Information Regression for Categorical Features
mi_selector = SelectKBest(score_func=mutual_info_regression, k=5)
X_mi = mi_selector.fit_transform(X_categorical, y)
mi_selected_features = X_categorical.columns[mi_selector.get_support()]

# ANOVA F-test for Numerical Features
f_selector = SelectKBest(score_func=f_regression, k=5)
X_f = f_selector.fit_transform(X_numerical, y)
f_selected_features = X_numerical.columns[f_selector.get_support()]

# Display selected features
print('Mutual Information selected for Categorical features:')
print(list(mi_selected_features))
print('\nANOVA (F-test) selected for Numerical features:')
print(list((f_selected_features)))
```

```
Mutual Information selected for Categorical features:
['drivewheel_fwd', 'drivewheel_rwd', 'cylindernumber_four', 'fuelsystem_2bbl', 'fuelsystem_mpfi']

ANOVA (F-test) selected for Numerical features:
['carwidth', 'curbweight', 'enginesize', 'horsepower', 'highwaympg']
```

- Recursive Feature Elimination (RFE) with Random Forest Regressor to capture model-based importance
- Mutual Information Regression for categorical features to assess non-linear dependencies. It was specifically applied to categorical features to understand their influence on car prices.
- ANOVA F-test for numerical features to determine statistical variance-based importance. It was applied to numerical features to quantify their contribution to the target variable.

Hyperparameter Tuning

Hyperparameter Tuning

```python
import optuna
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# Objective function for optimization
def objective(trial):
    n_estimators = trial.suggest_int('n_estimators', 100, 1000)
    max_depth = trial.suggest_int('max_depth', 5, 50)
    min_samples_split = trial.suggest_int('min_samples_split', 2, 10)
    min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 4)
    max_features = trial.suggest_categorical('max_features', ['sqrt', 'log2', None])

    rf = RandomForestRegressor(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        max_features=max_features,
        random_state=42
    )
    rf.fit(X_train, y_train)
    preds = rf.predict(X_test)

    r2 = r2_score(y_test, preds)
    mse = mean_squared_error(y_test, preds)
    mae = mean_absolute_error(y_test, preds)

    return r2

# Hyperparameter optimization
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)

# Best Parameters and Evaluation
print("Best Parameters:", study.best_params)
print("Best R² Score:", study.best_value)
```

```
    min_samples_split = trial.suggest_int('min_samples_split', 2, 10)
    min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 4)
    max_features = trial.suggest_categorical('max_features', ['sqrt', 'log2', None])

    rf = RandomForestRegressor(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        max_features=max_features,
        random_state=42
    )
    rf.fit(X_train, y_train)
    preds = rf.predict(X_test)

    r2 = r2_score(y_test, preds)
    mse = mean_squared_error(y_test, preds)
    mae = mean_absolute_error(y_test, preds)

    return r2

# Hyperparameter optimization
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)

# Best Parameters and Evaluation
print("Best Parameters:", study.best_params)
print("Best R² Score:", study.best_value)

# Evaluate with Best Model
best_model = RandomForestRegressor(**study.best_params, random_state=42)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

print(f"Optimized R²: {r2_score(y_test, y_pred):.4f}")
print(f"Optimized MSE: {mean_squared_error(y_test, y_pred):.2f}")
print(f"Optimized MAE: {mean_absolute_error(y_test, y_pred):.2f}")
```

```
[I 2025-02-05 12:35:14,646] A new study created in memory with name: no-name-b72a0a0c-2002-40a0-88e8-09152f26335a
[I 2025-02-05 12:35:15,731] Trial 0 finished with value: 0.908906246110166 and parameters: {'n_estimators': 745, 'max_depth': 23, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'log2'}. Best is trial 0 with value: 0.9
```

[34]
```
[I 2025-02-05 12:36:34,388] Trial 45 finished with value: 0.9562205667801746 and parameters: {'n_estimators': 985, 'max_depth': 22, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': None}. Best is trial 39 with value: 0.9
[I 2025-02-05 12:36:35,835] Trial 46 finished with value: 0.9564507347118711 and parameters: {'n_estimators': 545, 'max_depth': 32, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': None}. Best is trial 39 with value: 0.9
[I 2025-02-05 12:36:37,071] Trial 47 finished with value: 0.950580618586919 and parameters: {'n_estimators': 546, 'max_depth': 31, 'min_samples_split': 3, 'min_samples_leaf': 2, 'max_features': None}. Best is trial 39 with value: 0.95
[I 2025-02-05 12:36:38,554] Trial 48 finished with value: 0.9505739796513972 and parameters: {'n_estimators': 756, 'max_depth': 5, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': None}. Best is trial 39 with value: 0.95
[I 2025-02-05 12:36:40,092] Trial 49 finished with value: 0.9509998946461122 and parameters: {'n_estimators': 693, 'max_depth': 28, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': None}. Best is trial 39 with value: 0.9
Best Parameters: {'n_estimators': 815, 'max_depth': 21, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': None}
Best R² Score: 0.9566401816718533
Optimized R²: 0.9566
Optimized MSE: 3422998.40
Optimized MAE: 1287.83
```

For hyperparameter tuning, Optuna was used due to its efficient, automated optimization approach, leveraging a Tree-structured Parzen Estimator (TPE) for faster convergence compared to traditional grid or random search methods.

Performance of the model has increased

- R²: 0.9566
- MSE: 3422998.40
- MAE: 1287.83

> R² Score (Higher is Better):

- Initial Model: 0.9554
- Optimized Model: 0.9566 (Slight improvement)

> Mean Squared Error (MSE) (Lower is Better):

- Initial Model: 3,514,725
- Optimized Model: 3422998.40 (Reduced error)

> Mean Absolute Error (MAE) (Lower is Better):

- Initial Model: 1,305
- Optimized Model: 1287.83 (Less average deviation)