Data Cleaning: Car Details Dataset

The goal of this project is to inspect and clean the dataset in a suitable form for further exploratory data analysis and predictive modelling

Importing Necessary Libraries

In [361]:

```
import numpy as np
import pandas as pd
from IPython.display import HTML, display
```

In [362]:

```
pd.options.display.float_format = "{:,.3f}".format
```

Reading in the Dataset

In [363]:

```
path = r"/content/Car details v3.csv"

df = pd.read_csv(path)
display(HTML(f"<h3>Data Shape: {df.shape}</h3>"))
df.head()
```

Data Shape: (8128, 13)

Out[363]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torc
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nı 2000r
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nı 15(2500r
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12. 2,700(kgı rţ
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm 17: 2750r
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11. 4,500(kgı rţ

In [364]:

```
df.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
```

```
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           8128 non-null   object 
 1   year           8128 non-null   int64  
 2   selling_price  8128 non-null   int64  
 3   km_driven      8128 non-null   int64  
 4   fuel           8128 non-null   object 
 5   seller_type    8128 non-null   object 
 6   transmission   8128 non-null   object 
 7   owner          8128 non-null   object 
 8   mileage        7907 non-null   object 
 9   engine         7907 non-null   object 
 10  max_power      7913 non-null   object 
 11  torque         7906 non-null   object 
 12  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 5.0 MB
```

- **Data has 8128 observations and 13 features**
- **Data utilizes about 5 mb of memory**
- **There're 4 numeric and 9 categorical columns**
- `mileage`**,** `engine`**,** `max_power` **and** `torque` **have** `object` **type, but they should be converted to numeric types**
- **Some of the columns have missing values**

Checking for Duplicates

In [365]:

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

Out[365]:

```
1202
```

In [366]:

```
df.duplicated(subset="name").sum()
```

Out[366]:

```
6070
```

In [367]:

```
(df
 .loc[df.duplicated(subset=["name", "year"], keep=False)]
 .sort_values("name"))
```

Out[367]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	to
7747	Audi A4 2.0 TDI	2014	1500000	60000	Diesel	Individual	Automatic	First Owner	16.55 kmpl	1968 CC	147.51 bhp	320I 1 250(
906	Audi A4 2.0 TDI	2014	1600000	44000	Diesel	Individual	Automatic	Second Owner	16.55 kmpl	1968 CC	147.51 bhp	320I 1 250(
7374	Audi A4 35 TDI Premium Plus	2016	2450000	30000	Diesel	Dealer	Automatic	First Owner	18.25 kmpl	1968 CC	187.74 bhp	400I 1 300(
5261	Audi A4 35 TDI Premium Plus	2016	1898999	46000	Diesel	Dealer	Automatic	First Owner	18.25 kmpl	1968 CC	187.74 bhp	400I 1 300(

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	to
4951	Audi A6 35 TFSI Matrix	2019	5923000	11500	Petrol	Dealer	Automatic	Test Drive Car	15.26 kmpl	1798 CC	187.74 bhp	320 1 410
...	...	...	...	...	...	...	...	...	...	...	...	
3520	Volvo XC40 D4 Inscription BSIV	2019	3800000	20000	Diesel	Individual	Automatic	First Owner	18.0 kmpl	1969 CC	190 bhp	40
374	Volvo XC40 D4 Inscription BSIV	2019	3800000	20000	Diesel	Individual	Automatic	First Owner	18.0 kmpl	1969 CC	190 bhp	40
6213	Volvo XC40 D4 Inscription BSIV	2019	3800000	20000	Diesel	Individual	Automatic	First Owner	18.0 kmpl	1969 CC	190 bhp	40
3251	Volvo XC40 D4 R-Design	2018	3400000	22000	Diesel	Dealer	Automatic	First Owner	18.0 kmpl	1969 CC	190 bhp	40
145	Volvo XC40 D4 R-Design	2018	3400000	22000	Diesel	Dealer	Automatic	First Owner	18.0 kmpl	1969 CC	190 bhp	40

5894 rows × 13 columns

- There're 1202 definite duplicate entries
- There're some vehicles of the same name, but these differ w.r.t. other features and can be considered valid entries
- The 1202 duplicate entries should be deleted from the dataset

Column-wise Inspection

- There're some irregularities and inconsistencies within the columns
- These issues will be inspected and handled accordingly

Name

In [368]:

```
df.name
```

Out[368]:

```
0             Maruti Swift Dzire VDI
1        Skoda Rapid 1.5 TDI Ambition
2            Honda City 2017-2020 EXi
3           Hyundai i20 Sportz Diesel
4              Maruti Swift VXI BSIII
                         ...
8123                Hyundai i20 Magna
8124            Hyundai Verna CRDi SX
8125           Maruti Swift Dzire ZDi
8126                  Tata Indigo CR4
8127                  Tata Indigo CR4
Name: name, Length: 8128, dtype: object
```

In [369]:

```
df.name.value_counts()
```

Out[369]:

```
Maruti Swift Dzire VDI                          129
Maruti Alto 800 LXI                              82
Maruti Alto LXi                                  71
BMW X4 M Sport X xDrive20d                       62
Maruti Swift VDI                                 61
                                               ...
Skoda Fabia 1.4 TDI Ambiente                      1
Mahindra Scorpio VLX 2WD AT BSIII                 1
Renault KWID Climber 1.0 AMT                      1
Mahindra XUV300 W8 Option Dual Tone Diesel BSIV   1
Toyota Innova 2.5 GX (Diesel) 8 Seater BS IV      1
Name: name, Length: 2058, dtype: int64
```

In [370]:

```
(df
 .name
 .str.split(" ", n=2, expand=True)
 .set_axis(["company", "model", "edition"], axis=1))
```

Out[370]:

	company	model	edition
0	Maruti	Swift	Dzire VDI
1	Skoda	Rapid	1.5 TDI Ambition
2	Honda	City	2017-2020 EXi
3	Hyundai	i20	Sportz Diesel
4	Maruti	Swift	VXI BSIII
...	...	...	...
8123	Hyundai	i20	Magna
8124	Hyundai	Verna	CRDi SX
8125	Maruti	Swift	Dzire ZDi
8126	Tata	Indigo	CR4
8127	Tata	Indigo	CR4

8128 rows × 3 columns

In [371]:

```
(df
 .pipe(lambda df_: pd.concat([df_
                              .name
                              .str.split(" ", n=2, expand=True)
                              .set_axis(["company", "model", "edition"], axis=1),
                              df_],
                             axis=1)))
```

Out[371]:

	company	model	edition	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage
0	Maruti	Swift	Dzire VDI	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmp
1	Skoda	Rapid	1.5 TDI Ambition	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmp
2	Honda	City	2017-2020 EXi	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmp

	company	model	edition	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage
3	Hyundai	i20	Sportz Diesel	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmp
4	Maruti	Swift	VXI BSIII	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmp
...	...	...	...	...	...	...	...	...	...	...	...	...
8123	Hyundai	i20	Magna	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.5 kmp
8124	Hyundai	Verna	CRDi SX	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.8 kmp
8125	Maruti	Swift	Dzire ZDi	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.3 kmp
8126	Tata	Indigo	CR4	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57 kmp
8127	Tata	Indigo	CR4	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57 kmp

8128 rows × 16 columns

Observations:

- `name` provides the complete name of each vehicle
- Contains period, paranthesis, Roman numerals and numbers
- Overall, the entries seem to be valid and accurate

Steps:

- The column can be broken into 2 or 3 parts, namely `company`, `model` and `edition` for brevity
- These new columns could be useful for further analysis

Year

In [372]:

```
df.year
```

Out[372]:

```
0       2014
1       2014
2       2006
3       2010
4       2007
        ...
8123    2013
8124    2007
8125    2009
8126    2013
8127    2013
Name: year, Length: 8128, dtype: int64
```

In [373]:

```
df.year.describe()
```

Out[373]:

```
count    8,128.000
mean     2,013.804
std          4.044
min      1,983.000
25%      2,011.000
50%      2,015.000
75%      2,017.000
max      2,020.000
Name: year, dtype: float64
```

Observations:

- **The column is of the right datatype**
- **The values seem to be valid and accurate**
- **No further cleaning steps required**

Selling Price

In [374]:

```
df.selling_price
```

Out[374]:

```
0       450000
1       370000
2       158000
3       225000
4       130000
         ...
8123    320000
8124    135000
8125    382000
8126    290000
8127    290000
Name: selling_price, Length: 8128, dtype: int64
```

In [375]:

```
df.selling_price.describe()
```

Out[375]:

```
count        8,128.000
mean       638,271.808
std        806,253.404
min         29,999.000
25%        254,999.000
50%        450,000.000
75%        675,000.000
max     10,000,000.000
Name: selling_price, dtype: float64
```

Observations:

- **The column is of the right datatype**
- **The values seem to be valid**
- **No further cleaning steps required**
- **This could be treated as the target feature for predictive modelling**

KM Driven

KM Driven

In [376]:

```
df.km_driven
```

Out[376]:

```
0       145500
1       120000
2       140000
3       127000
4       120000
         ...  
8123    110000
8124    119000
8125    120000
8126     25000
8127     25000
Name: km_driven, Length: 8128, dtype: int64
```

In [377]:

```
df.km_driven.describe()
```

Out[377]:

```
count       8,128.000
mean       69,819.511
std        56,550.555
min             1.000
25%        35,000.000
50%        60,000.000
75%        98,000.000
max     2,360,457.000
Name: km_driven, dtype: float64
```

Observations:

- **The column is of the right datatype**
- **The values seem to be valid**
- **No further cleaning steps required**

Fuel

In [378]:

```
df.fuel
```

Out[378]:

```
0       Diesel
1       Diesel
2       Petrol
3       Diesel
4       Petrol
         ...  
8123    Petrol
8124    Diesel
8125    Diesel
8126    Diesel
8127    Diesel
Name: fuel, Length: 8128, dtype: object
```

In [379]:

```
df.fuel.memory_usage(deep=True)
```

Out[379]:

511907

In [380]:

```
(df
 .fuel
 .astype("category")
 .memory_usage(deep=True))
```

Out[380]:

8674

In [381]:

```
511907 / 8674
```

Out[381]:

59.016255476135576

In [382]:

```
df.fuel.unique()
```

Out[382]:

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

In [383]:

```
(df
 .fuel
 .value_counts()
 .pipe(lambda ser: pd.concat([ser, df.fuel
                                  .value_counts(normalize=True)],
                             axis=1)))
```

Out[383]:

	fuel	fuel
Diesel	4402	0.542
Petrol	3631	0.447
CNG	57	0.007
LPG	38	0.005

In [384]:

```
(df
 .loc[df.fuel.isin(["Diesel", "Petrol"])]
 .mileage
 .str.split(" ")
 .str[1]
 .unique())
```

Out[384]:

array(['kmpl', nan], dtype=object)

In [385]:

```
(df
 .loc[df.fuel.isin(["CNG", "LPG"])]
 .mileage
 .str.split(" ")
 .str[1]
 .unique())
```

Out[385]:

```
array(['km/kg', nan], dtype=object)
```

Observations:

- `fuel` has 4 unique values
- Occupies about 511,900 bytes of memory
- CNG and LPG account for only 0.7% and 0.5% of the total observations respectively
- Vehicles operating on CNG and LPG have their mileage units in `km\kg` whereas those running on Petrol and Diesel have mileage measured on `kmpl`

Steps:

- The datatype could be converted to `category` for optimizing memory usage (about 60 times less) since the cardinality is very less
- CNG and LPG only account for 1.2% of total observations collectively, whether to drop these rows could be decided on further exploratory analysis

Seller Type

In [386]:

```
df.seller_type
```

Out[386]:

```
0       Individual
1       Individual
2       Individual
3       Individual
4       Individual
           ...
8123    Individual
8124    Individual
8125    Individual
8126    Individual
8127    Individual
Name: seller_type, Length: 8128, dtype: object
```

In [387]:

```
df.seller_type.unique()
```

Out[387]:

```
array(['Individual', 'Dealer', 'Trustmark Dealer'], dtype=object)
```

In [388]:

```
df.seller_type.value_counts()
```

Out[388]:

```
Individual          6766
Dealer              1126
Trustmark Dealer     236
Name: seller_type, dtype: int64
```

In [389]:

```
(df
 .loc[df.seller_type == "Trustmark Dealer"]
 .head())
```

Out[389]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque
375	Toyota Camry 2.5 Hybrid	2016	2000000	68089	Petrol	Trustmark Dealer	Automatic	First Owner	19.16 kmpl	2494 CC	157.7 bhp	213Nm@ 4500rpm
376	Maruti Wagon R LXI	2013	225000	58343	Petrol	Trustmark Dealer	Manual	First Owner	21.79 kmpl	998 CC	67.05 bhp	90Nm@ 3500rpm
378	Honda Jazz VX	2016	550000	56494	Petrol	Trustmark Dealer	Manual	First Owner	18.2 kmpl	1199 CC	88.7 bhp	110Nm@ 4800rpm
379	Toyota Innova 2.5 VX (Diesel) 7 Seater	2013	750000	79328	Diesel	Trustmark Dealer	Manual	Second Owner	12.99 kmpl	2494 CC	100.6 bhp	200Nm@ 1200-3600rpm
380	Maruti Swift AMT VVT VXI	2019	650000	5621	Petrol	Trustmark Dealer	Automatic	First Owner	22.0 kmpl	1197 CC	81.80 bhp	113Nm@ 4200rpm

In [390]:

```
(df
 .loc[df.seller_type == "Trustmark Dealer"]
 .mileage
 .str.split(" ")
 .str[1]
 .unique())
```

Out[390]:

```
array(['kmpl'], dtype=object)
```

In [391]:

```
(df
 .loc[df.seller_type == "Individual"]
 .mileage
 .str.split(" ")
 .str[1]
 .unique())
```

Out[391]:

```
array(['kmpl', 'km/kg', nan], dtype=object)
```

In [392]:

```
(df
 .loc[df.seller_type == "Dealer"]
 .mileage
 .str.split(" ")
 .str[1]
 .unique())
```

Out[392]:

```
array(['kmpl', nan, 'km/kg'], dtype=object)
```

In [393]:

```
(df
 .seller_type
 .memory_usage(deep=True))
```

Out[393]:

```
541616
```

In [394]:

```
(df
 .seller_type
 .astype("category")
 .memory_usage(deep=True))
```

Out[394]:

```
8567
```

In [395]:

```
541616 / 8567
```

Out[395]:

```
63.2211976187697
```

Observations:

- `seller_type` **has 3 unique values**
- **Occupies about 541,600 bytes of memory**
- **Vehicles of** `Trustmark Dealer` **have their mileages measured in** `kmpl`
- **Cardinality is less**

Steps:

- **The datatype could be converted to** `category` **for optimizing memory usage (about 63 times) since the cardinality is very less**

Transmission

In [396]:

```
df.transmission
```

Out[396]:

```
0       Manual
1       Manual
2       Manual
3       Manual
4       Manual
         ...
8123    Manual
8124    Manual
8125    Manual
8126    Manual
8127    Manual
Name: transmission, Length: 8128, dtype: object
```

In [397]:

```
df.transmission.unique()
```

Out[397]:

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

In [398]:

```
df.transmission.value_counts()
```

Out[398]:

```
Manual       7078
Automatic    1050
Name: transmission, dtype: int64
```

In [399]:

```
(df
 .transmission
 .memory_usage(deep=True))
```

Out[399]:

```
515342
```

In [400]:

```
(df
 .transmission
 .astype("category")
 .memory_usage(deep=True))
```

Out[400]:

```
8493
```

In [401]:

```
515342 / 8493
```

Out[401]:

```
60.67844106911574
```

Observations:

- `transmission` **has 2 unique values**
- **Occupies about 515,342 bytes of memory**
- **Cardinality is less**

Steps:

- **The datatype could be converted to** `category` **for optimizing memory usage (about 61 times less) since the cardinality is very less**

Owner

In [402]:

```
df.owner
```

Out[402]:

```
0                 First Owner
1                Second Owner
2                 Third Owner
3                 First Owner
4                 First Owner
                     ...
8123              First Owner
8124     Fourth & Above Owner
8125              First Owner
8126              First Owner
8127              First Owner
Name: owner, Length: 8128, dtype: object
```

In [403]:

```
df.owner.unique()
```

Out[403]:

```
array(['First Owner', 'Second Owner', 'Third Owner',
       'Fourth & Above Owner', 'Test Drive Car'], dtype=object)
```

In [404]:

```
df.owner.value_counts()
```

Out[404]:

```
First Owner             5289
Second Owner            2105
Third Owner              555
Fourth & Above Owner     174
Test Drive Car             5
Name: owner, dtype: int64
```

In [405]:

```
(df
 .owner
 .value_counts()
 .pipe(lambda ser: pd.concat([ser, df
                                   .owner
                                   .value_counts(normalize=True)],
                            axis=1)))
```

Out[405]:

	owner	owner
First Owner	5289	0.651
Second Owner	2105	0.259
Third Owner	555	0.068
Fourth & Above Owner	174	0.021
Test Drive Car	5	0.001

In [406]:

```
(df
 .owner
 .str.replace(" Owner", "")
 .unique())
```

Out[406]:

```
array(['First', 'Second', 'Third', 'Fourth & Above', 'Test Drive Car'],
      dtype=object)
```

In [407]:

```
(df
 .owner
 .memory_usage(deep=True))
```

Out[407]:

```
556518
```

In [408]:

```
(df
 .owner
 .astype("category")
 .memory_usage(deep=True))
```

Out[408]:

8781

In [409]:

```
556518 / 8781
```

Out[409]:

```
63.377519644687396
```

Observations:

- **There're 5 unique values**
- **The cardinality is less**
- **This column occupies about 556,518 bytes of memory**
- `Test Drive Car` **accounts for only 0.1% of the total observations**

Steps:

- **The word** `Owner` **could be stripped off from the categories as it seems redundant**
- **The datatype could be converted to** `category` **for optimizing memory usage (about 63 times less) since the cardinality is very less**

Mileage

In [410]:

```
df.mileage
```

Out[410]:

```
0         23.4 kmpl
1        21.14 kmpl
2         17.7 kmpl
3         23.0 kmpl
4         16.1 kmpl
           ...
8123      18.5 kmpl
8124      16.8 kmpl
8125      19.3 kmpl
8126     23.57 kmpl
8127     23.57 kmpl
Name: mileage, Length: 8128, dtype: object
```

In [411]:

```
(df
 .mileage
 .str.split(" ")
 .str[1]
 .unique())
```

Out[411]:

```
array(['kmpl', 'km/kg', nan], dtype=object)
```

In [412]:

```
(df
 .mileage
 .str.split(" ")
 .str[0]
 .astype("float")
 .describe())
```

Out[412]:

```
count    7,907.000
mean        19.419
std          4.037
min          0.000
25%         16.780
50%         19.300
75%         22.320
max         42.000
Name: mileage, dtype: float64
```

In [413]:

```
(df
 .dropna()
 .loc[lambda df_: df_.mileage.str.startswith("0")])
```

Out[413]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	
644	Tata Indica Vista Aura Safire Anniversary Edition	2009	135000	28900	Petrol	Individual	Manual	Second Owner	0.0 kmpl	1172 CC	65 bhp	3,(
785	Hyundai Santro Xing GL	2009	120000	90000	Petrol	Individual	Manual	Second Owner	0.0 kmpl	1086 CC	62 bhp	
1649	Hyundai Santro Xing GL	2008	105000	128000	Petrol	Individual	Manual	First Owner	0.0 kmpl	1086 CC	62 bhp	
1676	Mercedes-Benz M-Class ML 350 4Matic	2011	1700000	110000	Diesel	Individual	Automatic	Third Owner	0.0 kmpl	2987 CC	165 bhp	5
2137	Land Rover Freelander 2 TD4 HSE	2013	1650000	64788	Diesel	Dealer	Automatic	First Owner	0.0 kmpl	2179 CC	115 bhp	/
2366	Hyundai Santro Xing (Non-AC)	2010	110000	80000	Petrol	Individual	Manual	Second Owner	0.0 kmpl	1086 CC	62.1 bhp	
2725	Hyundai Santro Xing (Non-AC)	2013	184000	15000	Petrol	Individual	Manual	First Owner	0.0 kmpl	1086 CC	62.1 bhp	
4527	Mercedes-Benz M-Class ML 350 4Matic	2011	1700000	110000	Diesel	Individual	Automatic	Third Owner	0.0 kmpl	2987 CC	165 bhp	5
5276	Hyundai Santro Xing GL	2008	175000	40000	Petrol	Individual	Manual	First Owner	0.0 kmpl	1086 CC	62 bhp	
5843	Volkswagen Polo GT TSI BSIV	2014	574000	28080	Petrol	Dealer	Automatic	First Owner	0.0 kmpl	1197 CC	103.25 bhp	
5846	Volkswagen Polo GT TSI BSIV	2014	575000	28100	Petrol	Dealer	Automatic	First Owner	0.0 kmpl	1197 CC	103.25 bhp	
5900	Mahindra Bolero Pik-Up FB 1.7T	2020	679000	5000	Diesel	Individual	Manual	First Owner	0.0 kmpl	2523 CC	70 bhp	
6534	Hyundai Santro Xing GL	2010	150000	110000	Petrol	Individual	Manual	First Owner	0.0 kmpl	1086 CC	62 bhp	
	Mahindra Bolero Pik-							First	0.0	2523		

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power
6629	Bolero Pik-Up 1.7T	2019	722000	80000	Diesel	Individual	Manual	First Owner	0.0 kmpl	2523 CC	70 bhp
6824	Hyundai Santro Xing GL	2011	150000	40000	Petrol	Individual	Manual	Fourth & Above Owner	0.0 kmpl	1086 CC	62 bhp
7002	Hyundai Santro Xing (Non-AC)	2010	110000	80000	Petrol	Individual	Manual	Second Owner	0.0 kmpl	1086 CC	62.1 bhp
7337	Mercedes-Benz GLC 220d 4MATIC	2017	3300000	60000	Diesel	Dealer	Automatic	First Owner	0.0 kmpl	1950 CC	194 bhp

In [414]:

```
(df
 .mileage
 .isna()
 .sum())
```

Out[414]:

```
221
```

In [415]:

```
values = np.empty(len(df))

for i, value in df.mileage.items():
  try:
    splitted = value.split(" ")
  except AttributeError:
    values[i] = np.nan
  else:
    num = float(splitted[0])
    unit = splitted[1]
    if num == 0:
      new_num = np.nan
    elif unit == "kmpl":
      new_num = num * 2.35
    elif unit == "km/kg":
      new_num = num * 0.0016
    values[i] = new_num

pd.Series(values).rename("mileage_mpg").astype(np.float16)
```

Out[415]:

```
0       55.000
1       49.688
2       41.594
3       54.062
4       37.844
         ...
8123    43.469
8124    39.469
8125    45.344
8126    55.375
8127    55.375
Name: mileage_mpg, Length: 8128, dtype: float16
```

Observations:

- This is a numeric column but it is of `object` type
- The values are measured in 2 different units, `kmpl` and `km/kg`
- Some vehicles show a reading of 0 mileage: (maybe because)
  - The actual mileage values weren't recorded for these vehicles

- The actual mileage values weren't recorded for these vehicles
- The odometer may have been reset to zero (this is illegal)

Steps:

- We will extract the numeric values and store them as floats
- We will then convert all values to a common unit, i.e., `mpg`
    - 1 km/l = 2.35 mpg [reference]
    - 1 km/kg = 0.0016 mpg [reference]
- The values showing zero mileage will be considered missing
    - Replace with `nan`

Engine

In [416]:

```
df.engine
```

Out[416]:

```
0       1248 CC
1       1498 CC
2       1497 CC
3       1396 CC
4       1298 CC
          ...
8123    1197 CC
8124    1493 CC
8125    1248 CC
8126    1396 CC
8127    1396 CC
Name: engine, Length: 8128, dtype: object
```

In [417]:

```
df.engine.isna().sum()
```

Out[417]:

```
221
```

In [418]:

```
df.loc[df.engine.isna()]
```

Out[418]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque
13	Maruti Swift 1.3 VXi	2007	200000	80000	Petrol	Individual	Manual	Second Owner	NaN	NaN	NaN	NaN
31	Fiat Palio 1.2 ELX	2003	70000	50000	Petrol	Individual	Manual	Second Owner	NaN	NaN	NaN	NaN
78	Tata Indica DLS	2003	50000	70000	Diesel	Individual	Manual	First Owner	NaN	NaN	NaN	NaN
87	Maruti Swift VDI BSIV W ABS	2015	475000	78000	Diesel	Dealer	Manual	First Owner	NaN	NaN	NaN	NaN
119	Maruti Swift VDI	2010	300000	120000	Diesel	Individual	Manual	Second Owner	NaN	NaN	NaN	NaN

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque
...	...	...	...	...	...	...	...	...	...	...	...	...
7846	Toyota Qualis Fleet A3	2000	200000	100000	Diesel	Individual	Manual	First Owner	NaN	NaN	NaN	NaN
7996	Hyundai Santro LS zipPlus	2000	140000	50000	Petrol	Individual	Manual	Second Owner	NaN	NaN	NaN	NaN
8009	Hyundai Santro Xing XS eRLX Euro III	2006	145000	80000	Petrol	Individual	Manual	Second Owner	NaN	NaN	NaN	NaN
8068	Ford Figo Aspire Facelift	2017	580000	165000	Diesel	Individual	Manual	First Owner	NaN	NaN	NaN	NaN
8103	Maruti Swift 1.3 VXi	2006	130000	100000	Petrol	Individual	Manual	Second Owner	NaN	NaN	NaN	NaN

221 rows × 13 columns

In [419]:

```
df.loc[df.engine.isna()].isna().sum()
```

Out[419]:

```
name              0
year              0
selling_price     0
km_driven         0
fuel              0
seller_type       0
transmission      0
owner             0
mileage         221
engine          221
max_power       215
torque          221
seats           221
dtype: int64
```

In [420]:

```
df.loc[df.mileage.isna()].isna().sum()
```

Out[420]:

```
name              0
year              0
selling_price     0
km_driven         0
fuel              0
seller_type       0
transmission      0
owner             0
mileage         221
engine          221
max_power       215
torque          221
seats           221
dtype: int64
```

In [421]:

```
(df
 .engine
 .str.split(" ")
 .str[0]
 .pipe(lambda ser: pd.to_numeric(ser)))
```

Out[421]:

```
0       1,248.000
1       1,498.000
2       1,497.000
3       1,396.000
4       1,298.000
           ...
8123    1,197.000
8124    1,493.000
8125    1,248.000
8126    1,396.000
8127    1,396.000
Name: engine, Length: 8128, dtype: float64
```

In [422]:

```
(df
 .engine
 .str.split(" ")
 .str[0]
 .pipe(lambda ser: pd.to_numeric(ser))
 .describe())
```

Out[422]:

```
count   7,907.000
mean    1,458.625
std       503.916
min       624.000
25%     1,197.000
50%     1,248.000
75%     1,582.000
max     3,604.000
Name: engine, dtype: float64
```

In [423]:

```
(df
 .engine
 .str.split(" ")
 .str[1]
 .unique())
```

Out[423]:

```
array(['CC', nan], dtype=object)
```

Observations:

- **This should be a numeric column but is of `object` type**
- **The rows which are missing values for `engine`, also have values missing for:**
  - `mileage`
  - `torque`
  - `max_power`
  - `seats`
- **Upon inspection, these values seem to be missing due to not being recorded**
  - **Since these rows contain missing values in 5 columns, they could be deleted**
  - **Values could be imputed based on other features (multivariate imputation techniques)**
  - **Decision could be taken during modelling stage based on model performance**
- **Otherwise, the values seem to be valid**

Steps:

- **The unit should be stripped from the values and attached to column name for better readability**

Max Power

In [424]:

```
df.max_power
```

Out[424]:

```
0           74 bhp
1       103.52 bhp
2           78 bhp
3           90 bhp
4         88.2 bhp
           ...
8123     82.85 bhp
8124       110 bhp
8125      73.9 bhp
8126        70 bhp
8127        70 bhp
Name: max_power, Length: 8128, dtype: object
```

In [425]:

```
(df
 .max_power
 .str.split(" ")
 .str[1]
 .unique())
```

Out[425]:

```
array(['bhp', nan], dtype=object)
```

In [426]:

```
(df
 .max_power
 .str.split(" ")
 .str[0]
 .pipe(lambda ser: pd.to_numeric(ser)))
```

Out[426]:

```
0        74.000
1       103.520
2        78.000
3        90.000
4        88.200
          ...
8123     82.850
8124    110.000
8125     73.900
8126     70.000
8127     70.000
Name: max_power, Length: 8128, dtype: float64
```

In [427]:

```
(df
 .max_power
 .str.split(" ")
 .str[0]
 .pipe(lambda ser: pd.to_numeric(ser))
 .describe())
```

Out[427]:

```
count    7,912.000
mean        91.518
std         35.822
min          0.000
25%         68.050
50%         82.000
75%        102.000
max        400.000
Name: max_power, dtype: float64
```

In [428]:

```
df.max_power.str.startswith("0").sum()
```

Out[428]:

```
6
```

In [429]:

```
mask = np.zeros(len(df), dtype=bool)
for i, entry in df.max_power.items():
  try:
    splitted = entry.split(" ")
  except:
    mask[i] = False
  else:
    if entry.startswith("0"):
      mask[i] = True
    else:
      mask[i] = False
df[mask]
```

Out[429]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torqu
575	Maruti Alto K10 LXI	2011	204999	97500	Petrol	Individual	Manual	First Owner	NaN	NaN	0	Nal
576	Maruti Alto K10 LXI	2011	204999	97500	Petrol	Individual	Manual	First Owner	NaN	NaN	0	Nal
1442	Maruti Swift Dzire VDI Optional	2017	589000	41232	Diesel	Dealer	Manual	First Owner	NaN	NaN	0	Nal
1443	Maruti Swift Dzire VDI Optional	2017	589000	41232	Diesel	Dealer	Manual	First Owner	NaN	NaN	0	Nal
2549	Tata Indica Vista Quadrajet LS	2012	240000	70000	Diesel	Individual	Manual	First Owner	NaN	NaN	0	Nal
2550	Tata Indica Vista Quadrajet LS	2012	240000	70000	Diesel	Individual	Manual	First Owner	NaN	NaN	0	Nal

In [430]:

```
df.dropna().loc[lambda df_: df_.max_power.str.startswith("0")]
```

Out[430]:

name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats

In [431]:

```
df.max_power.isna().sum()
```

Out[431]:

```
215
```

In [432]:

```
(df
 .max_power
 .str.split(" ")
 .str[0]
 .pipe(lambda ser: pd.to_numeric(ser))
 .pipe(lambda ser: ser[ser == 0]))
```

Out[432]:

```
575     0.000
576     0.000
1442    0.000
1443    0.000
2549    0.000
2550    0.000
Name: max_power, dtype: float64
```

In [433]:

```
(df
 .max_power
 .str.split(" ")
 .str[0]
 .replace("0", np.nan)
 .pipe(lambda ser: pd.to_numeric(ser))
 .pipe(lambda ser: df.loc[ser.isna()])
 .pipe(lambda df_: df_.loc[df_.mileage.notna()]))
```

Out[433]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	s
4933	Maruti Omni CNG	2000	80000	100000	CNG	Individual	Manual	Second Owner	10.9 km/kg	796 CC	bhp	NaN	8

In [434]:

```
(df
 .max_power
 .str.split(" ")
 .str[0]
 .pipe(lambda ser: pd.to_numeric(ser))
 .loc[4933])
```

Out[434]:

```
nan
```

In [435]:

```
(df
 .max_power
 .str.split(" ")
 .str[0]
 .replace("0", np.nan)
 .pipe(lambda ser: pd.to_numeric(ser)))
```

Out[435]:

```
0        74.000
1       103.520
2        78.000
3        90.000
4        88.200
          ...
8123     82.850
8124    110.000
8125     73.900
8126     70.000
8127     70.000
Name: max_power, Length: 8128, dtype: float64
```

Observations:

- **This should be a numeric column but is of `object` type**
- **Values are measured in units of `brake horsepower (bhp)`**
- **Entry of index `4933` has an empty value**
- **Some entries have values as zero (invalid)**

Steps:

- **The unit should be stripped from the values and attached to column name for better readability**
- **The values having 0 should be replaced as nan**

Torque

In [436]:

```
df.torque
```

Out[436]:

```
0                   190Nm@ 2000rpm
1              250Nm@ 1500-2500rpm
2            12.7@ 2,700(kgm@ rpm)
3         22.4 kgm at 1750-2750rpm
4            11.5@ 4,500(kgm@ rpm)
                      ...
8123              113.7Nm@ 4000rpm
8124     24@ 1,900-2,750(kgm@ rpm)
8125                190Nm@ 2000rpm
8126           140Nm@ 1800-3000rpm
8127           140Nm@ 1800-3000rpm
Name: torque, Length: 8128, dtype: object
```

In [437]:

```
(df
 .torque
 .str.lower()
 .unique())
```

Out[437]:

```
array(['190nm@ 2000rpm', '250nm@ 1500-2500rpm', '12.7@ 2,700(kgm@ rpm)',
       '22.4 kgm at 1750-2750rpm', '11.5@ 4,500(kgm@ rpm)',
       '113.75nm@ 4000rpm', '7.8@ 4,500(kgm@ rpm)', '59nm@ 2500rpm',
       '170nm@ 1800-2400rpm', '160nm@ 2000rpm', '248nm@ 2250rpm',
       '78nm@ 4500rpm', nan, '84nm@ 3500rpm', '115nm@ 3500-3600rpm',
       '200nm@ 1750rpm', '62nm@ 3000rpm', '219.7nm@ 1500-2750rpm',
       '114nm@ 3500rpm', '115nm@ 4000rpm', '69nm@ 3500rpm',
       '172.5nm@ 1750rpm', '6.1kgm@ 3000rpm', '114.7nm@ 4000rpm',
       '60nm@ 3500rpm', '90nm@ 3500rpm', '151nm@ 4850rpm',
       '104nm@ 4000rpm', '320nm@ 1700-2700rpm', '250nm@ 1750-2500rpm',
```

```
'145nm@ 4600rpm', '146nm@ 4800rpm', '343nm@ 1400-3400rpm',
'200nm@ 1400-3400rpm', '200nm@ 1250-4000rpm',
'400nm@ 2000-2500rpm', '138nm@ 4400rpm', '360nm@ 1200-3400rpm',
'200nm@ 1200-3600rpm', '380nm@ 1750-2500rpm', '173nm@ 4000rpm',
'400nm@ 1750-3000rpm', '400nm@ 1400-2800rpm',
'200nm@ 1750-3000rpm', '111.7nm@ 4000rpm', '219.6nm@ 1500-2750rpm',
'112nm@ 4000rpm', '250nm@ 1500-3000rpm', '130nm@ 4000rpm',
'205nm@ 1750-3250rpm', '280nm@ 1350-4600rpm', '99.04nm@ 4500rpm',
'77nm@ 3500rpm', '110nm@ 3750rpm', '153nm@ 3800rpm',
'113.7nm@ 4000rpm', '114nm@ 4000rpm', '113nm@ 4200rpm',
'101nm@ 3000rpm', '290nm@ 1800-2800rpm', '120nm@ 4250rpm',
'250nm@ 1500~4500rpm', '96 nm at 3000 rpm', '360nm@ 1750-2800rpm',
'135nm@ 2500rpm', '259.8nm@ 1900-2750rpm', '200nm@ 1900rpm',
'259.9nm@ 1900-2750rpm', '91nm@ 4250rpm', '96.1nm@ 3000rpm',
'109nm@ 4500rpm', '202nm@ 3600-5200rpm', '430nm@ 1750-2500rpm',
'347nm@ 4300rpm', '382nm@ 1750-2250rpm', '620nm@ 1600-2400rpm',
'400nm@ 1750-2500rpm', '250@ 1250-5000rpm', '500nm@ 1600-1800rpm',
'250nm@ 1600-3600rpm', '400nm', '550nm@ 1750-2750rpm',
'490nm@ 1600rpm', '250 nm at 2750 rpm', '177.5nm@ 4700rpm',
'170nm@ 1750-4000rpm', '300nm@ 1200-4000rpm',
'300nm@ 1200-1400rpm', '260nm@ 1500-2750rpm', '213nm@ 4500rpm',
'224nm@ 4000rpm', '640nm@ 1740rpm', '113nm@ 4500rpm',
'95nm@ 3000-4300rpm', '13.1kgm@ 4600rpm', '205nm@ 1800-2800rpm',
'71nm@ 3500rpm', '190nm@ 1750-3000rpm', '146nm at 4800 rpm',
'14.9 kgm at 3000 rpm', '115nm@ 3200rpm', '117nm@ 4000rpm',
'320nm@ 1500-3000rpm', '72nm@ 4386rpm', '11.4 kgm at 4,000 rpm',
'140nm@ 1500-4000rpm', '134nm@ 4000rpm', '150nm@ 4500rpm',
'340nm@ 1800-3250rpm', '240nm@ 1600-2800rpm',
'330nm@ 1600-2800rpm', '12.5@ 3,500(kgm@ rpm)', '110nm@ 4800rpm',
'111.8nm@ 4000rpm', '11.8@ 3,200(kgm@ rpm)', '135.4nm@ 2500rpm',
'300nm@ 1750-2500rpm', '190.25nm@ 1750-2250rpm',
'140nm@ 1800-3000rpm', '20.4@ 1400-3400(kgm@ rpm)',
'247nm@ 1800-2000rpm', '223nm@ 1600-2200rpm',
'180 nm at 1440-1500rpm', '195nm@ 1400-2200rpm',
'154.9nm@ 4200rpm', '114.73nm@ 4000rpm', '160nm@ 1500-2750rpm',
'108nm@ 4400rpm', '190.24nm@ 1750-2250rpm', '200nm@ 2000-3500rpm',
'420nm@ 1400-2600rpm', '100nm@ 2700rpm', '51nm@ 4000rpm',
'250nm@ 1250-5300rpm', '132nm@ 3000rpm', '350nm@ 1500-2750rpm',
'218nm@ 4200rpm', '14.9@ 3,000(kgm@ rpm)',
'24@ 1,900-2,750(kgm@ rpm)', '13.5@ 2,500(kgm@ rpm)',
'85nm@ 3000rpm', '74.5nm@ 4000rpm', '160nm@ 1750rpm',
'180.4nm@ 1750-2500rpm', '230nm@ 1500-2500rpm',
'219.66nm@ 1500-2750rpm', '245nm@ 1750rpm', '360nm@ 1400-3200rpm',
'320nm@ 2000rpm', '135 nm at 2500  rpm ',
'24 kgm at 1900-2750 rpm', '190nm@ 1750-2250rpm',
'204nm@ 2000-2750rpm', '14.3@ 1,800-3,000(kgm@ rpm)',
'250nm@ 1500-2750rpm', '125nm@ 2000rpm', '172nm@ 4300rpm',
'150nm@ 1750rpm', '102nm@ 4000rpm', '85nm@ 2500rpm',
'8.5@ 2,500(kgm@ rpm)', '180nm@ 1440-1500rpm', '106.5nm@ 4400rpm',
'108.5nm@ 5000rpm', '350nm@ 1750-2500rpm', '144.15nm@ 4500rpm',
'104nm@ 4400rpm', '99nm@ 4500rpm', '200nm@ 2000rpm',
'280nm@ 1800-2800rpm', '142.5nm@ 1750rpm', '140nm@ 4400rpm',
'115@ 2,500(kgm@ rpm)', '196nm@ 5000rpm',
'260 nm at 1800-2200 rpm', '9.8@ 3,000(kgm@ rpm)',
'209nm@ 2000rpm', '135 nm at 2500 rpm', '140nm@ 4200rpm',
'220nm at 1400-2600 rpm', '48nm@ 3000rpm', '171nm@ 1800rpm',
'277.5nm@ 1700-2200rpm', '215nm@ 3600rpm', '219.6nm@ 1750-2750rpm',
'195nm@ 1440-2200rpm', '13@ 2,500(kgm@ rpm)', '180nm@ 2000rpm',
'200nm@ 1400-2200rpm', '380nm(38.7kgm)@ 2500rpm', '110nm@ 4400rpm',
'72nm@ 4388rpm', '263.7nm@ 2500rpm', '320nm@ 1600-2800rpm',
'25.5@ 1,500-3,000(kgm@ rpm)', '16.3@ 2,000(kgm@ rpm)',
'190 nm at 1750 rpm ', '94.14nm@ 3500rpm', '12@ 3,500(kgm@ rpm)',
'113nm@ 5000rpm', '280nm@ 2400-2800rpm', '96nm@ 3500rpm',
'16@ 2,000(kgm@ rpm)', '320nm@ 1750-3000rpm',
'320nm@ 1750-2500rpm', '190nm@ 1750rpm', '789nm@ 2250rpm',
'259.87nm@ 1900-2750rpm', '205nm@ 1750rpm',
'436.39nm@ 1800-2500rpm', '182.5nm@ 1500-1800rpm',
'90.3nm@ 4200rpm', '12.5@ 2,500(kgm@ rpm)', '215nm@ 1750-3000rpm',
'215nm@ 1750-3000', '305nm@ 2000rpm', '540nm@ 2000rpm',
'327nm@ 2600rpm', '300nm@ 1600-3000rpm', '620nm@ 2000-2500rpm',
'450nm@ 1600-2400rpm', '19@ 1,800(kgm@ rpm)',
'9.2@ 4,200(kgm@ rpm)', '145@ 4,100(kgm@ rpm)',
```

```
'51nm@ 4000+/-500rpm', '110nm@ 3000rpm', '148nm@ 3500rpm',
'116nm@ 4750rpm', '48@ 3,000+/-500(nm@ rpm)', '148nm@ 4000rpm',
'222nm@ 4300rpm', '135.3nm@ 5000rpm', '98nm@ 1600-3000rpm',
'170nm@ 1400-4500rpm', '343nm@ 1400-2800rpm',
'402nm@ 1600-3000rpm', '113nm@ 3300rpm', '99.07nm@ 4500rpm',
'210nm@ 1600-2200rpm', '190 nm at 1750  rpm ', '32.1kgm@ 2000rpm',
'224nm@ 1500-2750rpm', '215nm@ 1750-2500rpm',
'25@ 1,800-2,800(kgm@ rpm)', '197nm@ 1750rpm', '136.3nm@ 4200rpm',
'470nm@ 1750-2500rpm', '11@ 3,000(kgm@ rpm)', '142nm@ 4000rpm',
'145nm@ 4100rpm', '320nm@ 1500-2800rpm', '123nm@ 1000-2500rpm',
'218nm@ 1400-2600rpm', '510@ 1600-2400', '220nm@ 1500-2750rpm',
'380nm@ 2000rpm', '104nm@ 3100rpm', '292nm@ 2000rpm',
'20@ 3,750(kgm@ rpm)', '46.5@ 1,400-2,800(kgm@ rpm)',
'380nm@ 2500rpm', '15@ 3,800(kgm@ rpm)', '136nm@ 4250rpm',
'228nm@ 4400rpm', '149nm@ 4500rpm', '187nm@ 2500rpm',
'146nm@ 3400rpm', '8.6@ 3,500(kgm@ rpm)', '219.7nm@ 1750-2750rpm',
'190nm@ 2000-3000', '450nm@ 2000rpm', '300nm@ 2000rpm',
'230nm@ 1800-2000rpm', '42@ 2,000(kgm@ rpm)',
'110nm@ 3000-4300rpm', '110(11.2)@ 4800', '330nm@ 1800rpm',
'225nm@ 1500-2500rpm', '380nm@ 1750-2750rpm',
'28.3@ 1,700-2,200(kgm@ rpm)', '259.88nm@ 1900-2750rpm',
'580nm@ 1400-3250rpm', '400 nm /2000 rpm', '127nm@ 3500rpm',
'300nm@ 1500-2500rpm', '132.3nm@ 4000rpm', '113nm@ 4400rpm',
'153nm@ 3750-3800rpm', '10.7@ 2,500(kgm@ rpm)', '124.6nm@ 3500rpm',
'78nm@ 3500rpm', '219.9nm@ 1750-2750rpm', '420.7nm@ 1800-2500rpm',
'130nm@ 3000rpm', '424nm@ 2000rpm', '130@ 2500(kgm@ rpm)',
'99.8nm@ 2700rpm', '113nm@ 4,500rpm', '11.2@ 4,400(kgm@ rpm)',
'240nm@ 1850rpm', '16.1@ 4,200(kgm@ rpm)', '320nm@ 1750-2700rpm',
'115nm@ 4500rpm', '245nm@ 4000rpm', '321nm@ 1600-2400rpm',
'619nm@ 1600-2400rpm', '380nm@ 1750-3000rpm', '560nm@ 1500rpm',
'230nm@ 1500-2250rpm', '90nm@ 2650rpm', '260nm@ 1800-2200rpm',
'600nm@ 2000rpm', '259.87nm@ 1500-3000rpm',
'16.6@ 4,500(kgm@ rpm)', '12.5@ 3,000(kgm@ rpm)',
'620nm@ 1500-2500rpm', '250nm@ 1500-4500rpm',
'14.9@ 3,400(kgm@ rpm)', '25.5@ 1,900(kgm@ rpm)',
'33.7@ 1,800(kgm@ rpm)', '285nm@ 2400-4000rpm',
'10.7@ 2,600(kgm@ rpm)', '250nm@ 1000-2000rpm', '240nm@ 1750rpm',
'226nm@ 4400rpm', '510nm@ 1600-2800rpm', '155 nm at 1600-2800 rpm',
'240nm@ 2000rpm', '103nm@ 4500rpm', '13.5@ 4,800(kgm@ rpm)',
'400nm@ 1750-2750rpm', '175nm@ 1500-4100rpm', '72.9nm@ 2250rpm',
'135.4nm@ 2500', '245nm@ 5000rpm', '57nm@ 2500rpm',
'96nm@ 2500rpm', '10.4@ 3,200(kgm@ rpm)', '128nm@ 3100rpm',
'102nm@ 2600rpm', '131nm@ 4400rpm', '11.4@ 4,000(kgm@ rpm)',
'250nm@ 4250rpm', '343nm@ 1600-2800rpm', '185nm@ 1750-2750rpm',
'12@ 2500(kgm@ rpm)', '12.4@ 2,600(kgm@ rpm)', '170nm@ 4200rpm',
'176nm@ 1500rpm', '380nm@ 1800-2800rpm', '250nm@ 1600-2000rpm',
'24.5@ 3,500-4,500(kgm@ rpm)', '22.9@ 1,950-4,700(kgm@ rpm)',
'121nm@ 2800rpm', '210 / 1900', '250nm@ 1250-5000rpm',
'400nm@ 175-2750rpm', '350nm@ 1500-3500rpm', '175nm@ 1750-4000rpm',
'115@ 2500(kgm@ rpm)', '110nm@ 4500rpm', '190nm@ 2000-3000rpm',
'106nm@ 2200rpm', '21.4@ 1,750-4,600(kgm@ rpm)', '96nm@ 3000rpm',
'23.6@ 4,250(kgm@ rpm)', '11.3kgm@ 4700rpm', '450nm@ 1750-2500rpm',
'35.7@ 1,750-3,000(kgm@ rpm)', '6@ 2,500(kgm@ rpm)',
'13.9 kgm at 4200 rpm', '320nm@ 1400-4100rpm',
'150nm@ 1700-4500rpm', '113.8nm@ 4000rpm', '110@ 3,000(kgm@ rpm)',
'151nm@ 2400rpm', '62nm@ 2500rpm', '18@ 1,600-2,200(kgm@ rpm)',
'83nm@ 3000rpm', '124.5nm@ 3500rpm', '20@ 4,700(kgm@ rpm)',
'300nm@ 1600-4000rpm', '171.6nm@ 1500-4000rpm',
'21.4@ 1,900(kgm@ rpm)', '190@ 21,800(kgm@ rpm)',
'5.7@ 2,500(kgm@ rpm)', '88.4nm@ 4200rpm',
'250 nm at 1,500-3,000 rpm', '340nm@ 1750-3000rpm',
'36.6@ 1,750-2,500(kgm@ rpm)', '12.5kgm@ 3500rpm',
'6.1@ 3,000(kgm@ rpm)', '110nm@ 4000rpm', '350nm@ 1800-2600rpm',
'4.8kgm@ 3000rpm', '355nm@ 4500rpm', '51@ 1,750-3,000(kgm@ rpm)',
'119nm@ 4250rpm', '410nm@ 1600-2800rpm', '174nm@ 4300rpm',
'99.1nm@ 4500rpm', '385nm@ 1600-2500rpm', '180 nm at 2000rpm',
'190 nm at 1750 rpm', '53@ 2,000-2,750(kgm@ rpm)',
'360nm@ 1400-2600rpm', '420nm@ 2000rpm', '124nm@ 3500rpm',
'17.5@ 4,300(kgm@ rpm)', '360nm@ 2000rpm', '145nm@ 3750rpm',
'85nm@ 3500rpm', '190nm@ 4200rpm', '190 nm at 2000rpm',
'13.5@ 2500(kgm@ rpm)', '159.8nm@ 1500-2750rpm', '500nm@ 2000rpm',
'333nm@ 1600-3200rpm', '400nm@ 2800rpm',
```

```
       '33@ 2,000-2,680(kgm@ rpm)', '10.2@ 2,600(kgm@ rpm)', '480nm',
       '190nm@ 4300rpm', '320nm@ 1800-2800rpm', '380nm@ 1750rpm',
       '250.06nm@ 1500-2750rpm', '190nm@ 3700rpm',
       '436.4nm@ 1800-2500rpm', '96  nm at 3000  rpm '], dtype=object)
```

In [438]:

```
(df
 .torque
 .str.extract(r"(^[0-9.]+).", expand=False)
 .pipe(lambda ser: pd.to_numeric(ser)))
```

Out[438]:

```
0       190.000
1       250.000
2        12.700
3        22.400
4        11.500
          ...
8123    113.700
8124     24.000
8125    190.000
8126    140.000
8127    140.000
Name: torque, Length: 8128, dtype: float64
```

In [439]:

```
df.torque.isna().sum()
```

Out[439]:

```
222
```

In [440]:

```
(df
 .torque
 .str.extract(r"(^[0-9.]+).", expand=False)
 .pipe(lambda ser: pd.to_numeric(ser))
 .values)
```

Out[440]:

```
array([190. , 250. ,  12.7, ..., 190. , 140. , 140. ])
```

In [441]:

```
values = (df
          .torque
          .str.extract(r"(^[0-9.]+).", expand=False)
          .pipe(lambda ser: pd.to_numeric(ser))
          .values)

for i, entry in (df
                 .torque
                 .str.lower()
                 .items()):
  try:
    splitted = entry.split(" ")
  except AttributeError:
    pass
  else:
    if "nm" in entry:
      pass
    elif "kg" in entry:
      values[i] = values[i] * 9.80665

(pd.Series(values)
   .rename("torque_nm")
   .pipe(lambda ser: pd.concat([df.torque, ser], axis=1)))
```

Out[441]:

	torque	torque_nm
0	190Nm@ 2000rpm	190.000
1	250Nm@ 1500-2500rpm	250.000
2	12.7@ 2,700(kgm@ rpm)	124.544
3	22.4 kgm at 1750-2750rpm	219.669
4	11.5@ 4,500(kgm@ rpm)	112.776
...	...	...
8123	113.7Nm@ 4000rpm	113.700
8124	24@ 1,900-2,750(kgm@ rpm)	235.360
8125	190Nm@ 2000rpm	190.000
8126	140Nm@ 1800-3000rpm	140.000
8127	140Nm@ 1800-3000rpm	140.000

8128 rows × 2 columns

Observations:

- This should be nuemric column but is of `object` type
- The values are listed mainly in 2 units: `kgm` and `Nm`
    - The units are very messy and inconsistent

Steps:

- The unit should be stripped from the values and attached to column name for better readability
- The values will all be converted to a common unit of `Nm`
    - 1 kgm = 9.80665 Nm reference

Seats

In [442]:

```
df.seats
```

Out[442]:

```
0       5.000
1       5.000
2       5.000
3       5.000
4       5.000
         ...
8123    5.000
8124    5.000
8125    5.000
8126    5.000
8127    5.000
Name: seats, Length: 8128, dtype: float64
```

In [443]:

```
df.seats.unique()
```

Out[443]:

```
array([ 5.,  4., nan,  7.,  8.,  6.,  9., 10., 14.,  2.])
```

In [444]:

```
df.seats.value_counts()
```

Out[444]:

```
5.000     6254
7.000     1120
8.000      236
4.000      133
9.000       80
6.000       62
10.000      19
2.000        2
14.000       1
Name: seats, dtype: int64
```

Observations:

- **This column seems to be valid**
- **Its of type float due to missing values**
- **No further cleaning required**

Data Cleaning Function

In [445]:

```
def clean_data(data):
  # this function takes in the dataset and returns the cleaned version

  def convert_mileage_mpg(ser):
    # this function takes in the mileage column
    # and returns all values in the unit of mpg

    new_values = np.empty(len(ser))
    for i, entry in ser.items():
      try:
        splitted = entry.split(" ")
      except AttributeError:
        new_values[i] = np.nan
      else:
        value = float(splitted[0])
        unit = splitted[1]
        if num == 0:
          new_values[i] = np.nan
        elif unit == "kmpl":
          new_values[i] = value * 2.35
        elif unit == "km/kg":
          new_values[i] = value * 0.0016
    return (pd.Series(new_values)
              .astype(np.float16))


  def convert_torque_nm(ser):
    # this function takes in the torque column
    # and returns all values in unit of Nm

    values = (ser
              .str.extract(r"(^[0-9.]+).", expand=False)
              .pipe(lambda ser: pd.to_numeric(ser))
              .values)
    for i, entry in (ser
                     .str.lower()
                     .items()):
      try:
        # to handle nan values
        entry.split(" ")
      except AttributeError:
```

```
        pass
      else:
        if "nm" in entry:
          pass
        elif "kg" in entry:
          values[i] = values[i] * 9.80665
    return pd.Series(values)


  return (data
          .drop_duplicates(ignore_index=True)
          .apply(lambda ser: ser.str.strip() if ser.dtype == "object" else ser)
          .pipe(lambda df_: pd.concat([df_
                                       .name
                                       .str.split(" ", n=2, expand=True)
                                       .set_axis(["company", "model", "edition"], axis=1
),
                                       df_],
                                      axis=1))
          .assign(year=lambda df_: df_.year.astype(np.int16),
                  fuel=lambda df_: df_.fuel.astype("category"),
                  seller_type=lambda df_: df_.seller_type.astype("category"),
                  transmission=lambda df_: df_.transmission.astype("category"),
                  owner=lambda df_: df_.owner
                                       .str.replace(" Owner", "")
                                       .astype("category"),
                  mileage_mpg=lambda df_: convert_mileage_mpg(df_.mileage),
                  engine_cc=lambda df_: df_
                                         .engine
                                         .str.split(" ")
                                         .str[0]
                                         .pipe(lambda ser: pd.to_numeric(ser)),
                  max_power_bhp=lambda df_: df_
                                             .max_power
                                             .str.split(" ")
                                             .str[0]
                                             .replace("0", np.nan)
                                             .pipe(lambda ser: pd.to_numeric(ser,
                                                                             errors="co
erce")),
                  torque_nm=lambda df_: convert_torque_nm(df_.torque))
          .drop(columns=["mileage", "engine", "max_power", "torque"])
          .reindex(columns=["name",
                            "company",
                            "model",
                            "edition",
                            "year",
                            "owner",
                            "fuel",
                            "seller_type",
                            "transmission",
                            "km_driven",
                            "mileage_mpg",
                            "engine_cc",
                            "max_power_bhp",
                            "torque_nm",
                            "seats",
                            "selling_price"]))
```

Cleaned Data

In [446]:

```
df_cleaned = clean_data(df)
df_cleaned
```

Out[446]:

	name	company	model	edition	year	owner	fuel	seller_type	transmission	km_driven	mileage_mpg	engin

	name	company	model	edition	year	owner	fuel	seller_type	transmission	km_driven	mileage_mpg	engin
0	Maruti Swift Dzire VDI	Maruti	Swift	Dzire VDI	2014	First	Diesel	Individual	Manual	145500	55.000	1,248
1	Skoda Rapid 1.5 TDI Ambition	Skoda	Rapid	1.5 TDI Ambition	2014	Second	Diesel	Individual	Manual	120000	49.688	1,498
2	Honda City 2017-2020 EXi	Honda	City	2017-2020 EXi	2006	Third	Petrol	Individual	Manual	140000	41.594	1,497
3	Hyundai i20 Sportz Diesel	Hyundai	i20	Sportz Diesel	2010	First	Diesel	Individual	Manual	127000	54.062	1,396
4	Maruti Swift VXI BSIII	Maruti	Swift	VXI BSIII	2007	First	Petrol	Individual	Manual	120000	37.844	1,298
...	...	...	...	...	...	...	...	...	...	...	...	
6921	Maruti Wagon R VXI BS IV with ABS	Maruti	Wagon	R VXI BS IV with ABS	2013	Second	Petrol	Individual	Manual	50000	44.406	998
6922	Hyundai i20 Magna 1.4 CRDi	Hyundai	i20	Magna 1.4 CRDi	2014	Second	Diesel	Individual	Manual	80000	52.969	1,396
6923	Hyundai i20 Magna	Hyundai	i20	Magna	2013	First	Petrol	Individual	Manual	110000	43.469	1,197
6924	Hyundai Verna CRDi SX	Hyundai	Verna	CRDi SX	2007	Fourth & Above	Diesel	Individual	Manual	119000	39.469	1,493
6925	Maruti Swift Dzire ZDi	Maruti	Swift	Dzire ZDi	2009	First	Diesel	Individual	Manual	120000	45.344	1,248

6926 rows × 16 columns

Memory Comparison

In [447]:

```
df.memory_usage(deep=True).sum()
```

Out[447]:

5202999

In [448]:

```
df_cleaned.memory_usage(deep=True).sum()
```

Out[448]:

2302779

In [449]:

```
5202999 / 2302779
```

Out[449]:

```
2.2594434811156434
```

Final Remarks:

- **The dataset is now cleaned and ready for further exploratory data analysis**
- **The cleaned dataset utilizes nearly 2.25 times less memory than the original dataset**
- **The `name` column was split into 3 parts:**
  - **Further analysis should be done if it needs to be split into 2 or 3 parts for better model performance**
- **The columns measuring `mileage`, `max_power`, `torque`, `seats` and `engine` have values missing in the same rows**
  - **These rows could be deleted, or**
  - **Imputed based on other features (depending on model performance)**
- **Some columns contain very rare categories (<1% of total observations)**
  - **These categories should be handled appropriately**