```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
cars_data = pd.read_csv('/content/Cardetails.csv')
```

```
cars_data.head()
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	sea
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5
2	Honda City 2017-2020	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5

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

```
cars_data.drop(columns=['torque'], inplace=True)
```

```
cars_data.head(2)
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	5.0
	Skoda											

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

```
cars_data.shape
```

```
(8128, 12)
```

Preprocessing and null value checking

```
cars_data.isnull().sum()
```

	0
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
seats	221

dtype: int64

```
cars_data.dropna(inplace=True)
```

```
cars_data.shape
```

```
(7907, 12)
```

Duplicate Checking

```
cars_data.duplicated().sum()
```

```
1189
```

```
cars_data.drop_duplicates(inplace=True)
```

```
cars_data.shape
```

```
(6718, 12)
```

```
cars_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6718 entries, 0 to 8125
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           6718 non-null   object 
 1   year           6718 non-null   int64  
 2   selling_price  6718 non-null   int64  
 3   km_driven      6718 non-null   int64  
 4   fuel           6718 non-null   object 
 5   seller_type    6718 non-null   object 
 6   transmission   6718 non-null   object 
 7   owner          6718 non-null   object 
 8   mileage        6718 non-null   object 
 9   engine         6718 non-null   object 
 10  max_power      6718 non-null   object 
 11  seats          6718 non-null   float64
dtypes: float64(1), int64(3), object(8)
memory usage: 682.3+ KB
```

Data analysis

```
for col in cars_data.columns:
  print('Unique values of ' + col)
  print(cars_data[col].unique)
  print("=================\n")
```

```
          ...
8121     18.9 kmpl
8122    22.54 kmpl
8123     18.5 kmpl
8124     16.8 kmpl
8125     19.3 kmpl
Name: mileage, Length: 6718, dtype: object>
==================

Unique values of engine
<bound method Series.unique of 0       1248 CC
1       1498 CC
2       1497 CC
3       1396 CC
4       1298 CC
         ...
8121     998 CC
8122    1396 CC
8123    1197 CC
8124    1493 CC
8125    1248 CC
Name: engine, Length: 6718, dtype: object>
==================

Unique values of max_power
<bound method Series.unique of 0           74 bhp
1       103.52 bhp
2           78 bhp
3           90 bhp
4         88.2 bhp
          ...
8121      67.1 bhp
8122     88.73 bhp
8123     82.85 bhp
8124       110 bhp
8125      73.9 bhp
Name: max_power, Length: 6718, dtype: object>
==================

Unique values of seats
<bound method Series.unique of 0       5.0
1       5.0
2       5.0
3       5.0
4       5.0
       ...
8121    5.0
8122    5.0
8123    5.0
8124    5.0
8125    5.0
Name: seats, Length: 6718, dtype: float64>
==================
```

Here we have split the data that we having unique name but we just have need of first name that is brand name of car

```
def get_brand_name(car_name):
  car_name = car_name.split(' ')[0]
  return car_name.strip()
```

```
def clean_data(value):
  value = value.split(' ')[0]
  value = value.strip()
  if value == '':
    value = 0
  return float(value)
```

```
get_brand_name('Maruti Suzuki Swift')
```

```
'Maruti'
```

```
cars_data['name'] = cars_data['name'].apply(get_brand_name)
```

```
cars_data['name'].unique()
```

```
array(['Maruti', 'Skoda', 'Honda', 'Hyundai', 'Toyota', 'Ford', 'Renault',
       'Mahindra', 'Tata', 'Chevrolet', 'Datsun', 'Jeep', 'Mercedes-Benz',
       'Mitsubishi', 'Audi', 'Volkswagen', 'BMW', 'Nissan', 'Lexus',
       'Jaguar', 'Land', 'MG', 'Volvo', 'Daewoo', 'Kia', 'Fiat', 'Force',
       'Ambassador', 'Ashok', 'Isuzu', 'Opel'], dtype=object)
```

Here we have take only numerical value in milage and max power and engine

```
cars_data['mileage'] = cars_data['mileage'].apply(clean_data)
```

```
cars_data['max_power'] = cars_data['max_power'].apply(clean_data)
```

```
cars_data['engine'] = cars_data['engine'].apply(clean_data)
```

```
for col in cars_data.columns:
  print('Unique values of ' + col)
  print(cars_data[col].unique)
  print("==================\n")
```

```
1       21.14
2       17.70
3       23.00
4       16.10
        ...
8121    18.90
8122    22.54
8123    18.50
8124    16.80
8125    19.30
Name: mileage, Length: 6718, dtype: float64>
==================

Unique values of engine
<bound method Series.unique of 0       1248.0
1       1498.0
2       1497.0
3       1396.0
4       1298.0
         ...
8121     998.0
8122    1396.0
8123    1197.0
8124    1493.0
8125    1248.0
Name: engine, Length: 6718, dtype: float64>
==================

Unique values of max_power
<bound method Series.unique of 0        74.00
1       103.52
2        78.00
3        90.00
4        88.20
         ...
8121     67.10
8122     88.73
8123     82.85
8124    110.00
8125     73.90
Name: max_power, Length: 6718, dtype: float64>
==================

Unique values of seats
<bound method Series.unique of 0       5.0
1       5.0
2       5.0
3       5.0
4       5.0
       ...
8121    5.0
8122    5.0
8123    5.0
8124    5.0
8125    5.0
Name: seats, Length: 6718, dtype: float64>
==================
```

Replacing names with numeric values

```
cars_data['name'].replace(['Maruti', 'Skoda', 'Honda', 'Hyundai', 'Toyota', 'Ford', 'Renault',
       'Mahindra', 'Tata', 'Chevrolet', 'Datsun', 'Jeep', 'Mercedes-Benz',
       'Mitsubishi', 'Audi', 'Volkswagen', 'BMW', 'Nissan', 'Lexus',
       'Jaguar', 'Land', 'MG', 'Volvo', 'Daewoo', 'Kia', 'Fiat', 'Force',
       'Ambassador', 'Ashok', 'Isuzu', 'Opel'],
```

```
                        [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29
                        inplace=True)
```

```
<ipython-input-176-d9734779e316>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chai
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are se

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = 

  cars_data['name'].replace(['Maruti', 'Skoda', 'Honda', 'Hyundai', 'Toyota', 'Ford', 'Renault',
<ipython-input-176-d9734779e316>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a fu
  cars_data['name'].replace(['Maruti', 'Skoda', 'Honda', 'Hyundai', 'Toyota', 'Ford', 'Renault',
```

```
cars_data['transmission'].unique()
```

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

```
cars_data['transmission'].replace(['Manual', 'Automatic'],[1,2], inplace=True)
```

```
<ipython-input-178-66fc237d138f>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chai
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are se

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = 

  cars_data['transmission'].replace(['Manual', 'Automatic'],[1,2], inplace=True)
<ipython-input-178-66fc237d138f>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a fu
  cars_data['transmission'].replace(['Manual', 'Automatic'],[1,2], inplace=True)
```

```
cars_data['seller_type'].unique()
```

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

```
cars_data['seller_type'].replace(['Individual', 'Dealer', 'Trustmark Dealer'],[1,2,3], inplace=True)
```

```
<ipython-input-180-9418936a2ba3>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chai
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are se

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = 

  cars_data['seller_type'].replace(['Individual', 'Dealer', 'Trustmark Dealer'],[1,2,3], inplace=True)
<ipython-input-180-9418936a2ba3>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a fu
  cars_data['seller_type'].replace(['Individual', 'Dealer', 'Trustmark Dealer'],[1,2,3], inplace=True)
```

```
cars_data['fuel'].unique()
```

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

```
cars_data['fuel'].replace(['Diesel', 'Petrol', 'LPG', 'CNG'],[1,2,3,4], inplace=True)
```

```
<ipython-input-182-84556209217c>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chai
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are se

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = 

  cars_data['fuel'].replace(['Diesel', 'Petrol', 'LPG', 'CNG'],[1,2,3,4], inplace=True)
<ipython-input-182-84556209217c>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a fu
  cars_data['fuel'].replace(['Diesel', 'Petrol', 'LPG', 'CNG'],[1,2,3,4], inplace=True)
```

```
cars_data['owner'].unique()
```

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

```
cars_data['owner'].replace(['First Owner', 'Second Owner', 'Third Owner',
       'Fourth & Above Owner', 'Test Drive Car'],[1,2,3,4,5], inplace=True)
```

```
<ipython-input-184-11d6af9151b3>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chai
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are se

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = 
```

```
  cars_data['owner'].replace(['First Owner', 'Second Owner', 'Third Owner',
<ipython-input-184-11d6af9151b3>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a fut
  cars_data['owner'].replace(['First Owner', 'Second Owner', 'Third Owner',
```

```
cars_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6718 entries, 0 to 8125
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           6718 non-null   int64  
 1   year           6718 non-null   int64  
 2   selling_price  6718 non-null   int64  
 3   km_driven      6718 non-null   int64  
 4   fuel           6718 non-null   int64  
 5   seller_type    6718 non-null   int64  
 6   transmission   6718 non-null   int64  
 7   owner          6718 non-null   int64  
 8   mileage        6718 non-null   float64
 9   engine         6718 non-null   float64
 10  max_power      6718 non-null   float64
 11  seats          6718 non-null   float64
dtypes: float64(4), int64(8)
memory usage: 682.3 KB
```

```
cars_data
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats
0	1	2014	450000	145500	1	1	1	1	23.40	1248.0	74.00	5.0
1	2	2014	370000	120000	1	1	1	2	21.14	1498.0	103.52	5.0
2	3	2006	158000	140000	2	1	1	3	17.70	1497.0	78.00	5.0
3	4	2010	225000	127000	1	1	1	1	23.00	1396.0	90.00	5.0
4	1	2007	130000	120000	2	1	1	1	16.10	1298.0	88.20	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...
8121	1	2013	260000	50000	2	1	1	2	18.90	998.0	67.10	5.0
8122	4	2014	475000	80000	1	1	1	2	22.54	1396.0	88.73	5.0
8123	4	2013	320000	110000	2	1	1	1	18.50	1197.0	82.85	5.0
8124	4	2007	135000	119000	1	1	1	4	16.80	1493.0	110.00	5.0
8125	1	2009	382000	120000	1	1	1	1	19.30	1248.0	73.90	5.0

6718 rows × 12 columns

Next steps: Generate code with `cars_data` View recommended plots New interactive sheet

```
cars_data.shape
```

```
(6718, 12)
```

```
cars_data.head(2)
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats
0	1	2014	450000	145500	1	1	1	1	23.40	1248.0	74.00	5.0
1	2	2014	370000	120000	1	1	1	2	21.14	1498.0	103.52	5.0

Next steps: Generate code with `cars_data` View recommended plots New interactive sheet

```
input_data = cars_data.drop(columns=['selling_price'])
output_data = cars_data['selling_price']
```

Splitting the dataset into training and testing dataset

```
X_train, X_test, y_train, y_test = train_test_split(input_data, output_data, test_size=0.2)
```

Model Creation

```
model = LinearRegression()
```

Train the model

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

```
LinearRegression
LinearRegression()
```

```
predict = model.predict(X_test)
```

```
predict
```

```
array([291710.95372437, 502527.35274504, 696530.68343687, ...,
       189288.10456821, 773982.32960714, 218078.91274624])
```

```
X_train.head(1)
```

	name	year	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats
2108	1	2008	112000	3	1	1	2	17.3	1061.0	57.5	5.0

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

```
input_data_model = pd.DataFrame(
    [[1,2023,1120,5,1,1,2,17.3,1061.0,57.5,6]],
    columns=['name','year','km_driven','fuel','seller_type','transmission','owner','mileage','engine','ma
```

```
input_data_model
```

	name	year	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats
0	1	2023	1120	5	1	1	2	17.3	1061.0	57.5	6

```
model.predict(input_data_model)
```

```
array([437909.78227536])
```