Used Cars Price Analysis

DR. Name: Ghada Hamed

Team ID: (General – Second year)

Team Members:

Omar Hesham Ali Abdelrazak – 20201700552 – Sec. 19
Enas Ahmed Abdelazem Rizk – 20201700169 – Sec. 7

Contents

1 Introduction

1.1 State Topic

How does the used car's *model*, *year* in which was bought, *distance* has driven in km, *fuel type*, *seller type*, *transmission type, number of owners* the car has previously had*, mileage, engine capacity, max power*, and *number of seats* affect its price?

We want to use Statistical Analysis, and Machine Learning concepts to display and explore the relations between car data by implementing Descriptive Statistics, Linear Regression, and Gradient Descent Algorithm, in addition, to performing estimation and inference on used cars' prices in the market.

1.2 Goals

- Predict used car price according to some information about it to help accurate pricing of the used cars.
- Provide both quantitative and visual evidence that the used car price affected with its (*model*, *year* in which was bought, *distance* has driven in km, *fuel type*, *seller type*, *transmission type, number of owners* the car has previously had, *mileage, engine capacity, max power*, and *number of seats*).

1.3 Direction

1. Gather, Clean, and prepare data.
2. Summarizing Data:
   a. Frequency Table.
3. Calculating Descriptive Statistical:
   a. Measures of Central Tendency.
   b. Measures of Variability.
4. Visualizing Data:
   a. Pie Charts.
   b. Bar Graphs.
   c. Box Plots.
   d. Histograms.
   e. X-Y Plots.
5. Between Pairs of data:
   a. Measures of Correlation.
   b. Implement Regression.
6. Perform Estimation and Inference.
7. Predict Price using Machine Learning techniques.

2

Data

2.1 Dataset

For our analysis, we conducted wide research to collect a suitable amount of data, to reach a high accuracy ratio.

We found a dataset containing 8128 records of used cars for each of them, twelve (12) pieces of information, which will be explained in the next section.

Dataset Sample:

```
In [23]: dataset = pd.read_csv(DATASET_FILE)
         dataset.head()
Out[23]:
```

	model	year	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	selling_price
0	Maruti Swift Dzire VDI	2014	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	5.0	450000
1	Skoda Rapid 1.5 TDI Ambition	2014	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	5.0	370000
2	Honda City 2017-2020 EXi	2006	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	5.0	158000
3	Hyundai i20 Sportz Diesel	2010	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	5.0	225000
4	Maruti Swift VXI BSIII	2007	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	5.0	130000

2.2 Car Model

This column should be filled with the model of the car.

Data Type: *String - Categorical (Nominal)*

Format: *COMPANY MODEL VERSION*

Unique Values: *[…] 2058 Values*

```
In [28]: dataset.model
```

```
Out[28]: 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: model, Length: 8128, dtype: object
```

```
In [31]: unique_model = dataset.model.unique()
         print("Num. of Unique Values: " + str(len(unique_model)) + "\n")
         print(unique_model)
```

```
         Num. of Unique Values: 2058

         ['Maruti Swift Dzire VDI' 'Skoda Rapid 1.5 TDI Ambition'
          'Honda City 2017-2020 EXi' ... 'Tata Nexon 1.5 Revotorq XT'
          'Ford Freestyle Titanium Plus Diesel BSIV'
          'Toyota Innova 2.5 GX (Diesel) 8 Seater BS IV']
```

2.3 Year of Bought

This column should be filled with the year in which the car was bought.

Data Type: *Integer - Quantitative (Discrete)*

Format: *YEAR*

Unique Values: *[…] 29 Values*

```
In [33]: dataset.year
```

```
Out[33]: 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 [34]: unique_year = dataset.year.unique()
         print("Num. of Unique Values: " + str(len(unique_year)) + "\n")
         print(unique_year)
```

```
Num. of Unique Values: 29

[2014 2006 2010 2007 2017 2001 2011 2013 2005 2009 2016 2012 2002 2015
 2018 2003 2019 2008 2020 1999 2000 1983 2004 1996 1994 1995 1998 1997
 1991]
```

2.4 Kms Driven

This column should be filled with the number of kilometers the car is driven.

Data Type: *Integer - Quantitative (Continuous)*

Format: *KM*

Unique Values: *[…] 921 Values*

```
In [35]: dataset.km_driven
```

```
Out[35]: 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 [36]: unique_km_driven = dataset.km_driven.unique()
         print("Num. of Unique Values: " + str(len(unique_km_driven)) + "\n")
         print(unique_km_driven)
```

```
Num. of Unique Values: 921

[ 145500  120000  140000  127000   45000  175000    5000   90000  169000
   68000  100000   80000   40000   70000   53000   50000   72000   35000
   28000   25000    2388   16200   10000   15000   42000   60000   76000
   28900   86300   23300   32600   10300   77000   99000   27800   49800
  151000   54700   64000   63000  127700   33900   59000  110000  147000
   30000  135000    9850   78000  170000   49000   32000   38000   44000
```

2.5 Fuel Type

This column should be filled with the fuel type of the car.

Data Type: *Integer - Categorical (Nominal)*

Format: *FUEL_TYPE*

Unique Values: *[Diesel, Petrol, LPG, CNG] 4 Values*

```
In [37]: dataset.fuel
```

```
Out[37]: 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 [38]: unique_fuel = dataset.fuel .unique()
         print("Num. of Unique Values: " + str(len(unique_fuel)) + "\n")
         print(unique_fuel)
```

```
Num. of Unique Values: 4

['Diesel' 'Petrol' 'LPG' 'CNG']
```

2.6 Seller Type

This column should define whether the seller is a dealer or an individual.

Data Type: *String - Categorical (Nominal)*

Format: *SELLER_TYPE*

Unique Values:

[Individual, Trustmark Dealer, Dealer] 3 Values

```
In [39]: dataset.seller_type

Out[39]: 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 [40]: unique_seller_type = dataset.seller_type.unique()
         print("Num. of Unique Values: " + str(len(unique_seller_type)) + "\n")
         print(unique_seller_type)

         Num. of Unique Values: 3

         ['Individual' 'Dealer' 'Trustmark Dealer']
```

2.7 Transmission

This column defines the gear transmission of the car.

Data Type: *String - Categorical (Nominal)*

Format: *TRANSMISSION*

Unique Values: *[Manual, Automatic] 2 Values*

```
In [25]: dataset.transmission

Out[25]: 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 [26]: unique_transmission = dataset.transmission.unique()
         print("Num. of Unique Values: " + str(len(unique_transmission)) + "\n")
         print(unique_transmission)

         Num. of Unique Values: 2

         ['Manual' 'Automatic']
```

2.8 Owner

This column Defines the number of owners the car has previously had.

Data Type: *String - Categorical (Ordinal)*

Format: *OWNER*

Unique Values: *[...] 5 Values*

```
In [27]: dataset.owner

Out[27]: 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 [29]: unique_owner = dataset.owner.unique()
         print("Num. of Unique Values: " + str(len(unique_owner)) + "\n")
         print(unique_owner)

         Num. of Unique Values: 5

         ['First Owner' 'Second Owner' 'Third Owner' 'Fourth & Above Owner'
          'Test Drive Car']
```

2.9 Mileage

This column defines the distance covered per unit of fuel by the vehicle under some specified conditions.

Data Type: *String - Quantitative (Continuous)*

Format: *MILEAGE **kmpl***

Unique Values: *[...] 394 Values*

```
In [30]: dataset.mileage
```

```
Out[30]: 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 [31]: unique_mileage = dataset.mileage.unique()
         print("Num. of Unique Values: " + str(len(unique_mileage)) + "\n")
         print(unique_mileage)
```

```
Num. of Unique Values: 394

['23.4 kmpl' '21.14 kmpl' '17.7 kmpl' '23.0 kmpl' '16.1 kmpl' '20.14 kmpl'
 '17.3 km/kg' '23.59 kmpl' '20.0 kmpl' '19.01 kmpl' '17.3 kmpl'
 '19.3 kmpl' nan '18.9 kmpl' '18.15 kmpl' '24.52 kmpl' '19.7 kmpl'
 '22.54 kmpl' '21.0 kmpl' '25.5 kmpl' '26.59 kmpl' '21.5 kmpl' '20.3 kmpl'
 '21.4 kmpl' '24.7 kmpl' '18.2 kmpl' '16.8 kmpl' '24.3 kmpl' '14.0 kmpl'
```

2.10 Engine Capacity

This column Defines the capacity of the car.

Data Type: *String - Quantitative (Discrete)*

Format: *ENGINE_CAPACITY* ***CC***

Unique Values: *[...] 122 Values*

```
In [33]: dataset.engine
```

```
Out[33]: 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 [34]: unique_engine = dataset.engine.unique()
         print("Num. of Unique Values: " + str(len(unique_engine)) + "\n")
         print(unique_engine)
```

```
Num. of Unique Values: 122

['1248 CC' '1498 CC' '1497 CC' '1396 CC' '1298 CC' '1197 CC' '1061 CC'
 '796 CC' '1364 CC' '1399 CC' '1461 CC' '993 CC' nan '1198 CC' '1199 CC'
 '998 CC' '1591 CC' '2179 CC' '1368 CC' '2982 CC' '2494 CC' '2143 CC'
 '2477 CC' '1462 CC' '2755 CC' '1968 CC' '1798 CC' '1196 CC' '1373 CC'
 '1598 CC' '1998 CC' '1086 CC' '1194 CC' '1172 CC' '1405 CC' '1582 CC'
 '999 CC' '2487 CC' '1999 CC' '3604 CC' '2987 CC' '1995 CC' '1451 CC'
```

2.11 Engine Max Power

This column Defines the max power of the engine.

Data Type: *String - Quantitative (Continuous)*

Format: *MAX_POWER **bhp***

Unique Values: *[…] 323 Values*

In [35]:
```
dataset.max_power
```

Out[35]:
```
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 [36]:
```
unique_max_power = dataset.max_power.unique()
print("Num. of Unique Values: " + str(len(unique_max_power)) + "\n")
print(unique_max_power)
```

```
Num. of Unique Values: 323

['74 bhp' '103.52 bhp' '78 bhp' '90 bhp' '88.2 bhp' '81.86 bhp' '57.5 bhp'
 '37 bhp' '67.1 bhp' '68.1 bhp' '108.45 bhp' '60 bhp' '73.9 bhp' nan
 '67 bhp' '82 bhp' '88.5 bhp' '46.3 bhp' '88.73 bhp' '64.1 bhp' '98.6 bhp'
 '88.8 bhp' '83.81 bhp' '83.1 bhp' '47.3 bhp' '73.8 bhp' '34.2 bhp'
 '35 bhp' '81.83 bhp' '40.3 bhp' '121.3 bhp' '138.03 bhp' '160.77 bhp'
 '117.3 bhp' '116.3 bhp' '83.14 bhp' '67.05 bhp' '168.5 bhp' '100 bhp'
 '120.7 bhp' '98.63 bhp' '175.56 bhp' '103.25 bhp' '171.5 bhp' '100.6 bhp'
```

2.12 Seats Number

This column Defines the number of seats in the car.

Data Type: *FLOAT - Quantitative (Discrete)*

Format: *SEATS_NUMBER*

Unique Values: *[…] 10 Values*

```
In [37]: dataset.seats
```

```
Out[37]: 0       5.0
         1       5.0
         2       5.0
         3       5.0
         4       5.0
                ...
         8123    5.0
         8124    5.0
         8125    5.0
         8126    5.0
         8127    5.0
         Name: seats, Length: 8128, dtype: float64
```

```
In [38]: unique_seats = dataset.seats.unique()
         print("Num. of Unique Values: " + str(len(unique_seats)) + "\n")
         print(unique_seats)
```

```
Num. of Unique Values: 10

[ 5.   4.  nan  7.   8.   6.   9.  10.  14.   2.]
```

2.13 Selling Price

This column should be filled with the price the owner wants to sell the car at.

Data Type: *INTEGER - Quantitative (Discrete)*

Format: *SELLING PRICE*

Unique Values: *[...] 677 Values*

In [39]:
```
dataset.selling_price
```

Out[39]:
```
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 [40]:
```
unique_selling_price= dataset.selling_price.unique()
print("Num. of Unique Values: " + str(len(unique_selling_price)) + "\n")
print(unique_selling_price)
```

```
Num. of Unique Values: 677

[  450000   370000   158000   225000   130000   440000    96000    45000
   350000   200000   500000    92000   280000   180000   400000   778000
   150000   680000   174000   950000   525000   600000   575000   275000
   300000   220000   254999   670000    70000   730000   650000   330000
   366000  1149000   425000  2100000   925000   675000   819999   390000
  1500000   700000  1450000  1090000   850000  1650000  1750000  1590000
```

3

Data Preparation

3.1 Generate Sub-columns

Preform studying on the model's column shows that it contains three different pieces of data and has 2058 unique values, which is a big number compared with the used dataset, so to perform better analysis on this column, we must split it into smaller pieces of data, which will help to reach stronger relations between variables.

As we mention in the last section its format is

COMPANY MODEL VERSION

so, splitting it into three sub-columns will be meaningful.

In [351]:
```
# Spliting Car Model into 3 sub-columns COMPANY, MODEL, VERSION
company = []
model = []
version = []

for record in dataset.model.iloc[:].values:
    company.append(record.split()[0])
    model.append(record.split()[1])
    version.append(' '.join(record.split()[2:]))

# Drop the old model column
dataset = dataset.drop(columns=['model'], axis=1)

# Insert the new three sub-columns
dataset.insert(0, "company", company, allow_duplicates = True)
dataset.insert(1, "model", model, allow_duplicates = True)
dataset.insert(2, "version", version, allow_duplicates = True)

dataset.head()
```

Out[351]:

	company	model	version	year	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	selling_price
0	Maruti	Swift	Dzire VDI	2014	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	5.0	450000
1	Skoda	Rapid	1.5 TDI Ambition	2014	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	5.0	370000
2	Honda	City	2017-2020 EXi	2006	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	5.0	158000
3	Hyundai	i20	Sportz Diesel	2010	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	5.0	225000
4	Maruti	Swift	VXI BSIII	2007	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	5.0	130000

3.2 Standardization

Each column should be measured with only one unit to be suitable for analyzing, but if we look at the Mileage Columns, we see that the column contains 2 units: *a) km/kg* *b) kmpl*

A quick search gave that:

1 liter of mileage = 710 to 775 grams ~= 742.5 (Avg)

The first step is to identify which unit we will use for the whole column, by counting how many rows follow each of these two units.

```
In [235]: records_in_kmkg = 0
          records_in_kmpl = 0

          for record in dataset.mileage:
              if str(record).endswith("kmpl"):
                  records_in_kmpl += 1
              elif str(record).endswith("km/kg"):
                  records_in_kmkg += 1

          print(f'The number of rows meaured by kmpl : {records_in_kmpl}')
          print(f'The number of rows meaured by km/kg : {records_in_kmkg}')

          The number of rows meaured by kmpl : 7819
          The number of rows meaured by km/kg : 88
```

As there are more rows measured by in kmpl, convert the other to this unit then, remove the unit label from the data.

In [339]:
```
mileage = dataset.mileage.values[:]

for i in range(len(mileage)):
    if 'kmpl' in str(mileage[i]):
        mileage[i] = float(mileage[i].replace('kmpl', ''))
    elif 'km/kg' in str(mileage[i]):
        mileage[i] = float(mileage[i].replace('km/kg', '')) * 1.3468

dataset['mileage'] = pd.to_numeric(mileage)
dataset.head()
```

Out[339]:

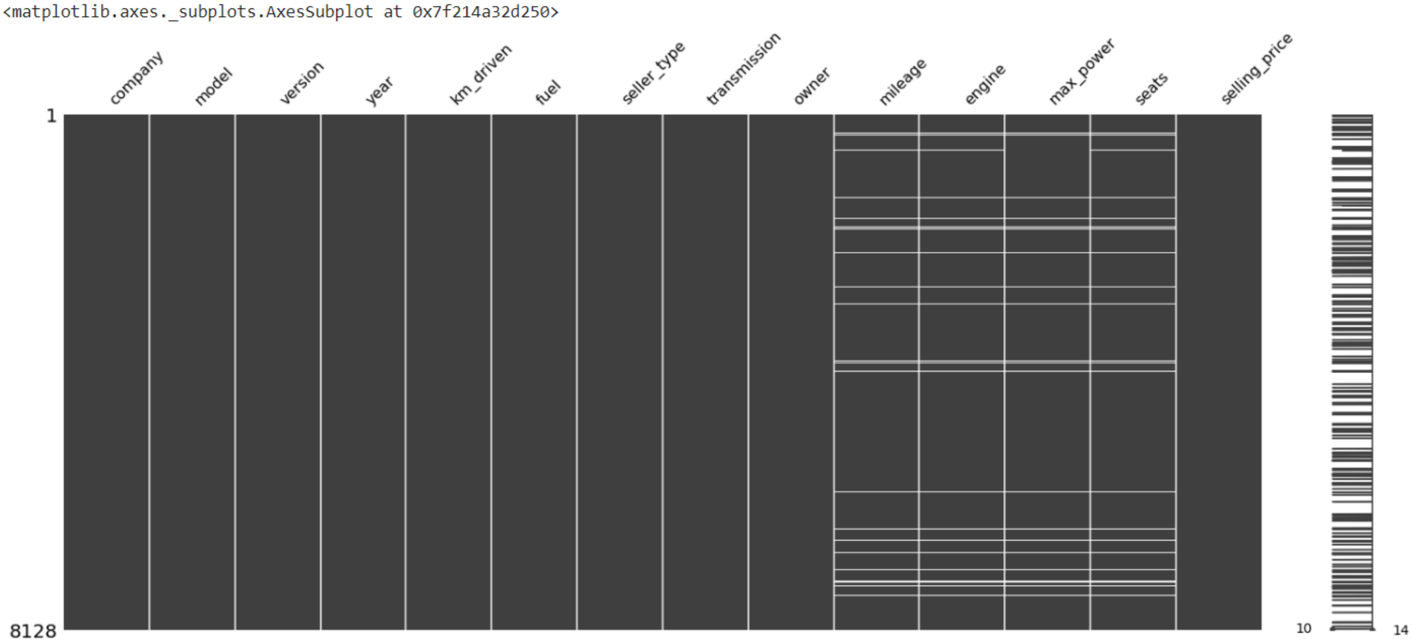
	company	model	version	year	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	selling_price
0	Maruti	Swift	Dzire VDI	2014	145500	Diesel	Individual	Manual	First Owner	23.40	1248 CC	74 bhp	5.0	450000
1	Skoda	Rapid	1.5 TDI Ambition	2014	120000	Diesel	Individual	Manual	Second Owner	21.14	1498 CC	103.52 bhp	5.0	370000
2	Honda	City	2017-2020 EXi	2006	140000	Petrol	Individual	Manual	Third Owner	17.70	1497 CC	78 bhp	5.0	158000
3	Hyundai	i20	Sportz Diesel	2010	127000	Diesel	Individual	Manual	First Owner	23.00	1396 CC	90 bhp	5.0	225000
4	Maruti	Swift	VXI BSIII	2007	120000	Petrol	Individual	Manual	First Owner	16.10	1298 CC	88.2 bhp	5.0	130000

3.3 Handling NULLs

Variables with missed values or in other words containing NULL values must be handled by removing them or using mean or median to give them a numeric value if we need complete statistical analysis.

```
msg.matrix(dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f214a32d250>
```



The figure represents NULL values in the dataset in white lines.

Find how many records with

a missing value (NULL)

```
dataset.isnull().sum()
```

```
company           0
model             0
version           0
year              0
km_driven         0
fuel              0
seller_type       0
transmission      0
owner             0
mileage         221
engine          221
max_power       215
seats           221
```

Then calculate the percentage of them compared to the whole dataset, which implies whether removing them will affect data or not.

```
[66] def calcNullsPercentage(column):
       return str(format((column.isnull().sum() / len(column)) * 100, '.2f')) +'%'

     print('NULLs percentage in: \n')
     print('mileage -> ' + calcNullsPercentage(dataset.mileage))
     print('engine -> ' + calcNullsPercentage(dataset.engine))
     print('max power -> ' + calcNullsPercentage(dataset.max_power))
     print('seats -> ' + calcNullsPercentage(dataset.seats))
```

```
NULLs percentage in:

mileage -> 2.72%
engine -> 2.72%
max power -> 2.65%
seats -> 2.72%
```

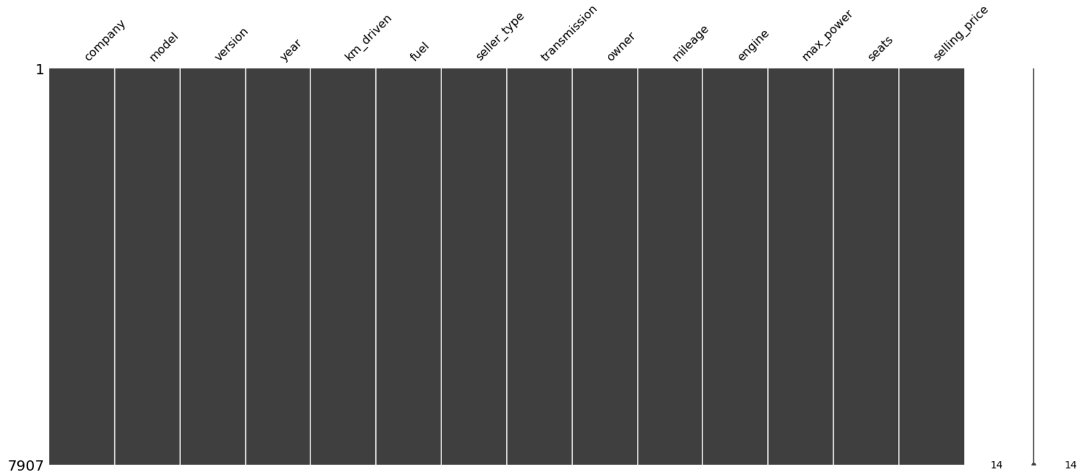
In our case, the NULL values represent only about 2.7% of the data, so removing them will not affect the dataset.

```
dataset.dropna(axis=0, subset=['mileage'], how= 'any',inplace=True )
dataset.dropna(axis=0, subset=['engine'], how= 'any',inplace=True )
dataset.dropna(axis=0, subset=['max_power'], how= 'any',inplace=True )
dataset.dropna(axis=0, subset=['seats'], how= 'any',inplace=True )

print(dataset.isnull().sum())
print('Cases in Dataset: ' + str(len(dataset)))
```

```
company          0
model            0
version          0
year             0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
mileage          0
engine           0
max_power        0
seats            0
selling_price    0
```

```
msg.matrix(dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe288f92c10>
```



The figure represents the dataset after dropping the NULLs values.

3.4 Cleaning Data

Take a quick look at our dataset, you will notice that there are two columns:

[Engine Capacity, Max Power]

that contain its measurement unit after the value, which is not helpful and will mislead the analysis process so let's clean them.

```
def cleanColumn(column, unit, column_header):
  values = column.values
  for i in range(len(values)):
    values[i] = str(values[i]).replace(unit, '')

  dataset[column_header] = pd.to_numeric(values)

cleanColumn(dataset.engine, 'CC', 'engine')

# print(dataset[(dataset == ' ').any(axis=1)])
dataset.drop(labels=4933, axis=0, inplace=True)
cleanColumn(dataset.max_power, 'bhp', 'max_power')

dataset.head()
```

	company	model	version	year	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	selling_price
0	Maruti	Swift	Dzire VDI	2014	145500	Diesel	Individual	Manual	First Owner	23.40	1248	74.00	5.0	450000
1	Skoda	Rapid	1.5 TDI Ambition	2014	120000	Diesel	Individual	Manual	Second Owner	21.14	1498	103.52	5.0	370000
2	Honda	City	2017-2020 EXi	2006	140000	Petrol	Individual	Manual	Third Owner	17.70	1497	78.00	5.0	158000
3	Hyundai	i20	Sportz Diesel	2010	127000	Diesel	Individual	Manual	First Owner	23.00	1396	90.00	5.0	225000
4	Maruti	Swift	VXI BSIII	2007	120000	Petrol	Individual	Manual	First Owner	16.10	1298	88.20	5.0	130000

3.5 Mapping String Values

To use formulas, descriptive analysis, and Machine Learning Techniques, the variables should be in a numerical form, so map strings *(Categorical Variables)* into integer values *(Quantitative Variables)* ***one label encoding***.

Note that those numbers don't have mathematical meaning.

To facilitate doing that, this function takes a column as parameter then return an equivalent numeric map.

```python
# Return a map of values from 0 ... n
def mapping_values(column):
    unique = column.unique()
    valuesMap = {}

    for i in range(len(unique)):
        valuesMap[unique[i]] = i

    print(len(valuesMap))
    if len(valuesMap) < 100:
      print(valuesMap)

    return valuesMap
```

Note:

There are 1855 unique values in the version column so it will not be so helpful in our analysis so let's drop it.

```python
mapping_values(dataset.version)

# Drop Version as it contains many unique values
dataset = dataset.drop(columns=['version'], axis=1)
```

```python
# Done all the mapping process
def perform_mapping():
  global company_map, model_map, fuel_map, seller_map, transmission_map, owner_map
  company_map = mapping_values(dataset.company)
  model_map = mapping_values(dataset.model)
  fuel_map = mapping_values(dataset.fuel)
  seller_map = mapping_values(dataset.seller_type)
  transmission_map = mapping_values(dataset.transmission)
  owner_map = mapping_values(dataset.owner)

  dataset.company.replace(company_map, inplace=True)
  dataset.model.replace(model_map, inplace=True)
  dataset.fuel.replace(fuel_map, inplace=True)
  dataset.seller_type.replace(seller_map, inplace=True)
  dataset.transmission.replace(transmission_map, inplace=True)
  dataset.owner.replace(owner_map, inplace=True)

perform_mapping()
dataset.head()
```

```
31
{'Maruti': 0, 'Skoda': 1, 'Honda': 2, 'Hyundai': 3, 'Toyota': 4, 'Ford': 5, 'Renault': 6, 'Mahindra': 7, 'Tata': 8, 'Chevrolet': 9

197
4
{'Diesel': 0, 'Petrol': 1, 'LPG': 2, 'CNG': 3}

3
{'Individual': 0, 'Dealer': 1, 'Trustmark Dealer': 2}

2
{'Manual': 0, 'Automatic': 1}

5
{'First Owner': 0, 'Second Owner': 1, 'Third Owner': 2, 'Fourth & Above Owner': 3, 'Test Drive Car': 4}
```

	company	model	year	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	selling_price
0	0	0	2014	145500	0	0	0	0	23.40	1248	74.00	5.0	450000
1	1	1	2014	120000	0	0	0	1	21.14	1498	103.52	5.0	370000
2	2	2	2006	140000	1	0	0	2	17.70	1497	78.00	5.0	158000
3	3	3	2010	127000	0	0	0	0	23.00	1396	90.00	5.0	225000

Encapsulate all the mapping process into one function to postpone calling it until finishing Descriptive Statistics with its original labels.

Reaching this point now the data is ready for analysis.

4 Summarizing Data

4.1 Frequency Table

It is a table used with categorical variables, shows how the values are distributed over the cases, It has two ways of implementation according to the variable's data type.

5.1.1 Categorical Variables

The variables ***company***, ***model***, ***fuel type***, ***seller type***, ***transmission***, and ***owner*** are categorical variables so let's illustrate the frequency table to them.

```python
def categorical_frequency_table(column_header):
  frequency = pd.crosstab(index=dataset[column_header], columns='Frequency')
  percentage = ((frequency / frequency.sum()) * 100)
  frequency.insert(1, "Percentage", percentage, allow_duplicates = True)
  frequency.insert(2, "Cumulative Percentage", percentage.cumsum(), allow_duplicates = True)
  return frequency
```

- Company

The table shows that most of the cars manufacture by ***Maruti*** about 30% of the cars**,** and about 17.2% by ***Hyundai.***

```
categorical_frequency_table('company')
```

col_0 company	Frequency	Percentage	Cumulative Percentage
Ambassador	4	0.050594	0.050594
Ashok	1	0.012649	0.063243
Audi	40	0.505945	0.569188
BMW	118	1.492537	2.061725
Chevrolet	230	2.909183	4.970908
Daewoo	3	0.037946	5.008854
Datsun	65	0.822160	5.831014
Fiat	41	0.518593	6.349608
Force	6	0.075892	6.425500
Ford	388	4.907665	11.333165
Honda	466	5.894258	17.227422
Hyundai	1360	17.202125	34.429547
Isuzu	5	0.063243	34.492790
Jaguar	71	0.898052	35.390842
Jeep	31	0.392107	35.782950
Kia	4	0.050594	35.833544
Land	6	0.075892	35.909436
Lexus	34	0.430053	36.339489
MG	3	0.037946	36.377435
Mahindra	758	9.587655	45.965090
Maruti	2367	29.939287	75.904376
Mercedes-Benz	54	0.683026	76.587402
Mitsubishi	14	0.177081	76.764483
Nissan	81	1.024538	77.789021
Opel	1	0.012649	77.801670
Renault	228	2.883886	80.685555
Skoda	104	1.315457	82.001012
Tata	719	9.094359	91.095371
Toyota	452	5.717177	96.812547
Volkswagen	185	2.339995	99.152542
Volvo	67	0.847458	100.000000

- ## Model

It contains 197 unique records so the frequency table will not be useful.

- ## Fuel Type

The table shows that more than half of the dataset (54%) use ***Diesel*** as fuel, and 44% use ***Petrol*** so there are 2% of the cars divided into using ***CNG*** and ***LPG***.

```
categorical_frequency_table('fuel')
```

col_0 fuel	Frequency	Percentage	Cumulative Percentage
CNG	52	0.657728	0.657728
Diesel	4299	54.376423	55.034151
LPG	35	0.442702	55.476853
Petrol	3520	44.523147	100.000000

• Seller Type

The table shows that most of the sellers (83%) are ***individual*** ones, with only 14% ***Dealers*** in the dataset.

```
categorical_frequency_table('seller_type')
```

col_0 seller_type	Frequency	Percentage	Cumulative Percentage
Dealer	1107	14.002024	14.002024
Individual	6563	83.012902	97.014925
Trustmark Dealer	236	2.985075	100.000000

• Transmission

It is obvious that about 90% of the used cars are ***Manual*** transmission, with the rest working in ***Automatic*** transmission.

```
categorical_frequency_table('transmission')
```

col_0 transmission	Frequency	Percentage	Cumulative Percentage
Automatic	1041	13.167215	13.167215
Manual	6865	86.832785	100.000000

- ## Owner

66% of the previous owners are the ***First Owner*** of the car, with 25% ***Second Owner***.

```
categorical_frequency_table('owner')
```

owner	Frequency	Percentage	Cumulative Percentage
First Owner	5215	65.962560	65.962560
Fourth & Above Owner	160	2.023779	67.986339
Second Owner	2016	25.499621	93.485960
Test Drive Car	5	0.063243	93.549203
Third Owner	510	6.450797	100.000000

5

Descriptive Statistical

5.1 Central Tendency

Besides summarizing data by means of tables, it can also be useful to describe the center of a distribution. The measures of central tendency show the central or middle values of datasets. There are several definitions of what's considered to be the center of a dataset.

We can do that by means of so-called measures of central tendency: the mode, median, and mean.

- ## Mean

The **sample mean**, also called simply the **average**, is the arithmetic average of all the items in a dataset, which identifies the balance point of the data.

The mean of a dataset x is mathematically expressed as $\Sigma_i x_i/n$, where i = 1, 2, …, n. In other words, it's the sum of all the values x_i divided by the number of observations in the dataset x.

```
dataset.mean()
```

```
year                 2013.983936
km_driven           69188.659752
mileage                19.504086
engine               1458.708829
max_power              91.587374
seats                   5.416393
selling_price      649813.720845
dtype: float64
```

we calculate the mean for only the quantitative variables.

5.1.2 Median

The ***sample median*** is the middle value of observations in a sorted dataset.

The dataset can be sorted in increasing or decreasing order. If the number of elements n of the dataset is **odd**, then the median is the value at the middle position: $0.5(n + 1)$. If n is **even**, then the median is the arithmetic mean of the two values in the middle, that is, the items at the positions $0.5n$ and $0.5n + 1$.

```
In [52]: dataset.median()

Out[52]: year                 2015.00
         km_driven           60000.00
         mileage                19.33
         engine               1248.00
         max_power              82.00
         seats                   5.00
         selling_price      450000.00
         dtype: float64
```

5.1.3 Mode

The ***sample mode*** is the value in the dataset that occurs most frequently.

If there isn't a single such value, then the set is multimodal since it has multiple modal values.

```
In [54]: dataset.mode()
```

Out[54]:

	company	model	year	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats	selling_price
0	Maruti	Swift	2017	120000	Diesel	Individual	Manual	First Owner	18.9	1248	74.0	5.0	300000

5.2 Measures of Dispersion

Central tendency is not sufficient for the right conclusion and decision, so we need more information. We also need information about the variability or dispersion of the data. In other words, measures of dispersion. Well-known measures of dispersion are the ***range***, the ***interquartile range***, the ***variance***, and the ***standard deviation***.

5.2.1 Range

The range of data is the difference between the maximum and minimum element in the dataset.

In [65]:
```
pd.DataFrame({
      'year' : [np.ptp(dataset.year)],
      'km_driven' : [np.ptp(dataset.km_driven)],
      'mileage' : [np.ptp(dataset.engine)],
      'engine' : [np.ptp(dataset.max_power)],
      'max_power' : [np.ptp(dataset.seats)],
      'selling_price' : [np.ptp(dataset.selling_price
})
```

Out[65]:

	year	km_driven	mileage	engine	max_power	selling_price
0	26	2360456	2980	367.2	12.0	9970001

5.2.2 Interquartile Range

The IQR of data is the difference between the third quartile **Q3** and the first quartile **Q1** which distribute the data into four equal parts, and leaves out extreme values (outliers).

In [74]:
```python
# np.percentile(, 25) --> Q1
# np.percentile(, 50) --> Q2
# np.percentile(, 75) --> Q3
def calc_IQR(column):
    return np.percentile(column, 75) - np.percentile(column, 25)

pd.DataFrame({
    'year' : [calc_IQR(dataset.year)],
    'km_driven' : [calc_IQR(dataset.km_driven)],
    'mileage' : [calc_IQR(dataset.mileage)],
    'engine' : [calc_IQR(dataset.engine)],
    'max_power' : [calc_IQR(dataset.max_power)],
    'selling_price' : [calc_IQR(dataset.selling_price)]
})
```

Out[74]:

	year	km_driven	mileage	engine	max_power	selling_price
0	5.0	60425.0	5.54	385.0	33.95	420000.0

5.2.3 Variance

The **sample variance** quantifies the spread of the data. It shows numerically how far the data points are from the mean.

You can express the sample variance of the dataset x with n elements mathematically as

$s^2 = \Sigma_i(x_i - \text{mean}(x))^2 / (n - 1)$, *where* $i = 1, 2, \ldots, n$

and mean(x) is the sample mean of x *(we already calculate).*

In [100]:
```
pd.DataFrame({
    'year' : [format(dataset.year.var(), '.5f')],
    'km_driven' : [format(dataset.km_driven.var(), '.5f')],
    'mileage' : [format(dataset.mileage.var(), '.5f')],
    'engine' : [format(dataset.engine.var(), '.5f')],
    'max_power' : [format(dataset.max_power.var(), '.5f')],
    'selling_price' : [format(dataset.selling_price.var(), '.5f')]
})
```

Out[100]:

	year	km_driven	mileage	engine	max_power	selling_price
0	14.92814	3225364923.94723	17.68095	253908.21274	1277.86346	661916888419.46741

5.2.4 Standard Deviation

The ***sample standard deviation*** is another measure of data spread. It's connected to the sample variance, as standard deviation, s, is the positive square root of the sample variance. The standard deviation is often more convenient than the variance because it has the same unit as the data points.

```
In [101]: dataset.std()

Out[101]: year                  3.863695
          km_driven         56792.296343
          mileage               4.204873
          engine              503.893057
          max_power            35.747216
          seats                 0.959208
          selling_price    813582.748354
          dtype: float64
```

6

Visualizing Data

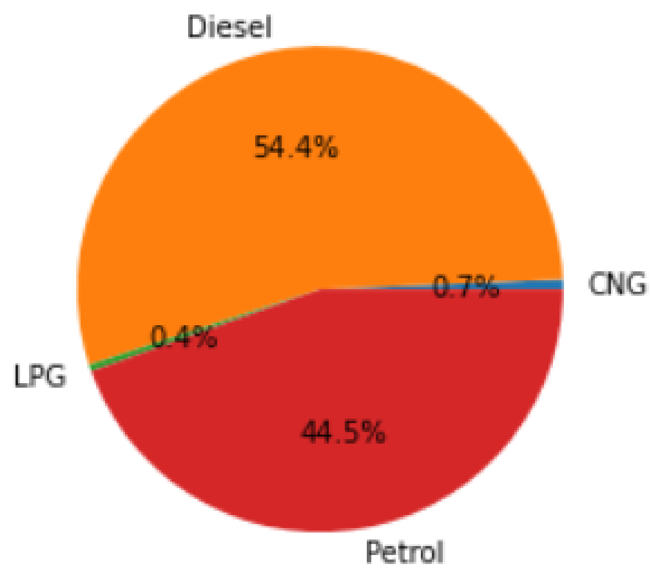
6.1 Pie Charts

Pie charts represent data with a small number of labels and given relative frequencies. They work well even with the labels that can't be ordered (like nominal data).

A pie chart is a circle divided into multiple slices. Each slice corresponds to a single distinct label from the dataset and has an area proportional to the relative frequency associated with that label.

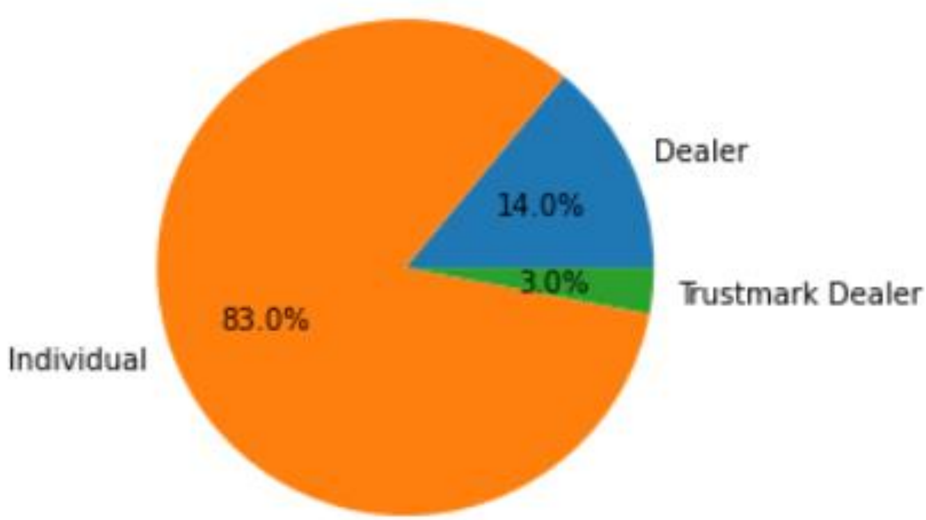
It used when we have small number of categories so, we will illustrate it for ***fuel type***, ***seller type, transmission***, and ***owner.***

```
pieChart(dataset.fuel, 'fuel')
```



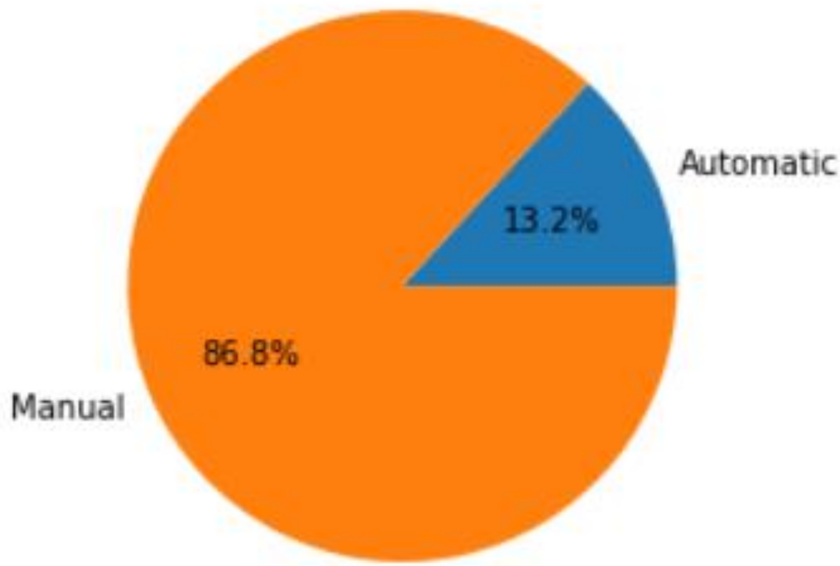
Pie Char of Fuel Type

```
pieChart(dataset.seller_type, 'seller_type')
```



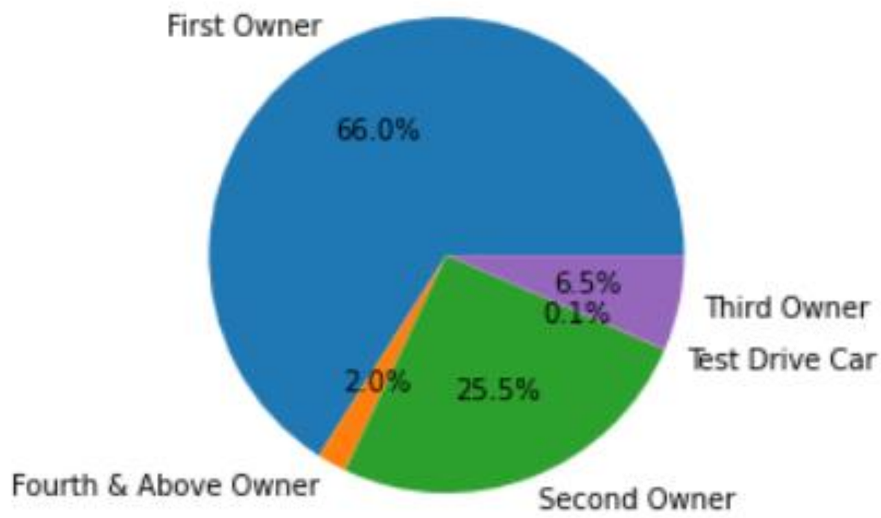
Pie Char of Seller Type

```
pieChart(dataset.transmission, 'transmission')
```



Pie Char of Transmission

```
pieChart(dataset.owner, 'owner')
```



Pie Char of Owner

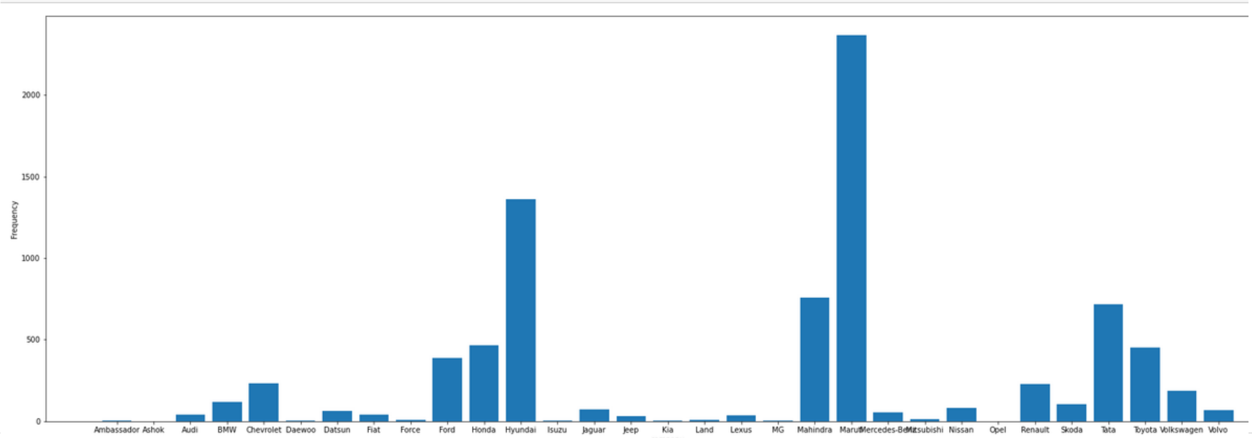
6.2 Bar Graphs

Bar charts also illustrate data that correspond to given labels or discrete numeric values. They can show the pairs of data from two datasets. Items of one set are the labels, while the corresponding items of the other are their frequencies.

The bar chart shows parallel rectangles called ***bars***. Each bar corresponds to a single label and has a height proportional to the frequency or relative frequency of its label.

It used when we have large number of categories so, we will illustrate it for ***company***.

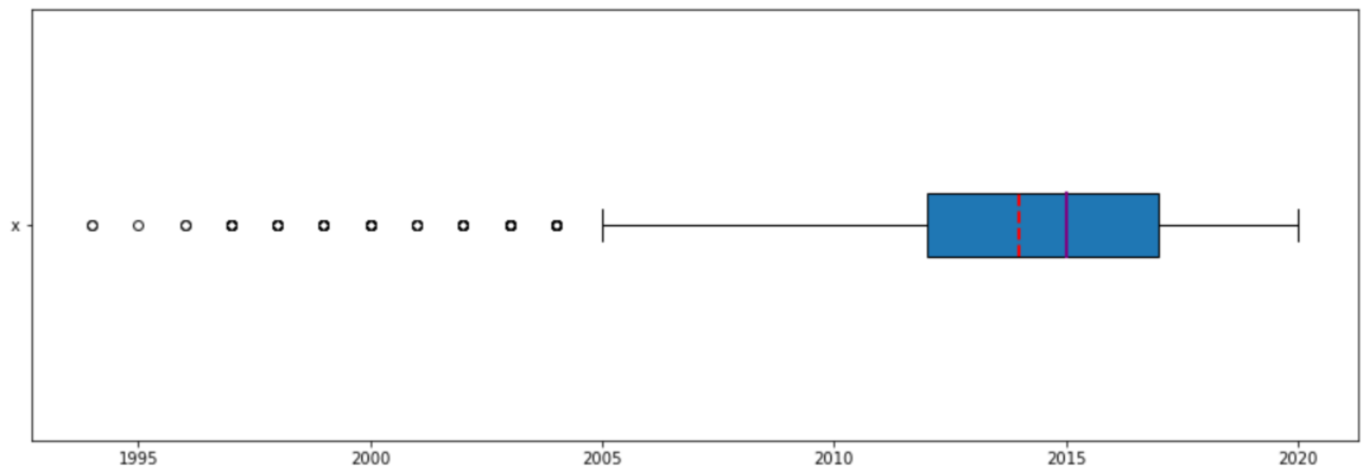
```
barGraph(dataset.company, 'company')
```



6.3 Box Plots

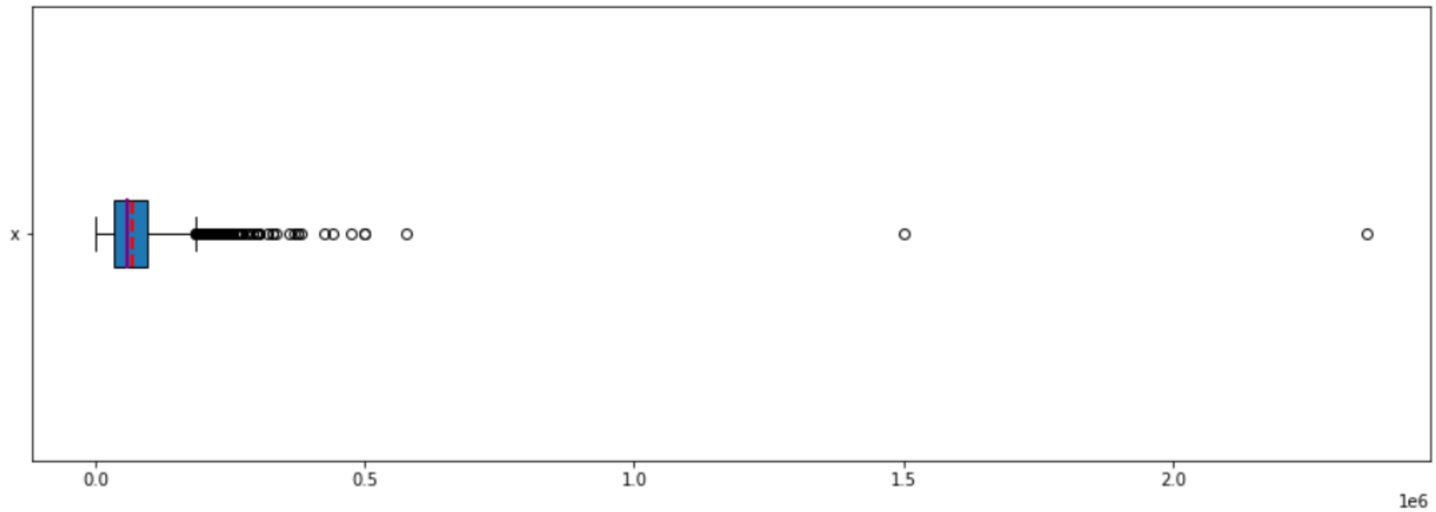
The box plot is an excellent tool to visually represent descriptive statistics of a given dataset. It can show the range, interquartile range, median, mode, outliers, and all quartiles.

```
fig, ax = plt.subplots(figsize=(15, 5))
ax.boxplot(dataset.year.values, vert=False, showmeans=True, meanline=True,
    labels='x', patch_artist=True,
    medianprops={'linewidth': 2, 'color': 'purple'},
    meanprops={'linewidth': 2, 'color': 'red'})
plt.show()
```



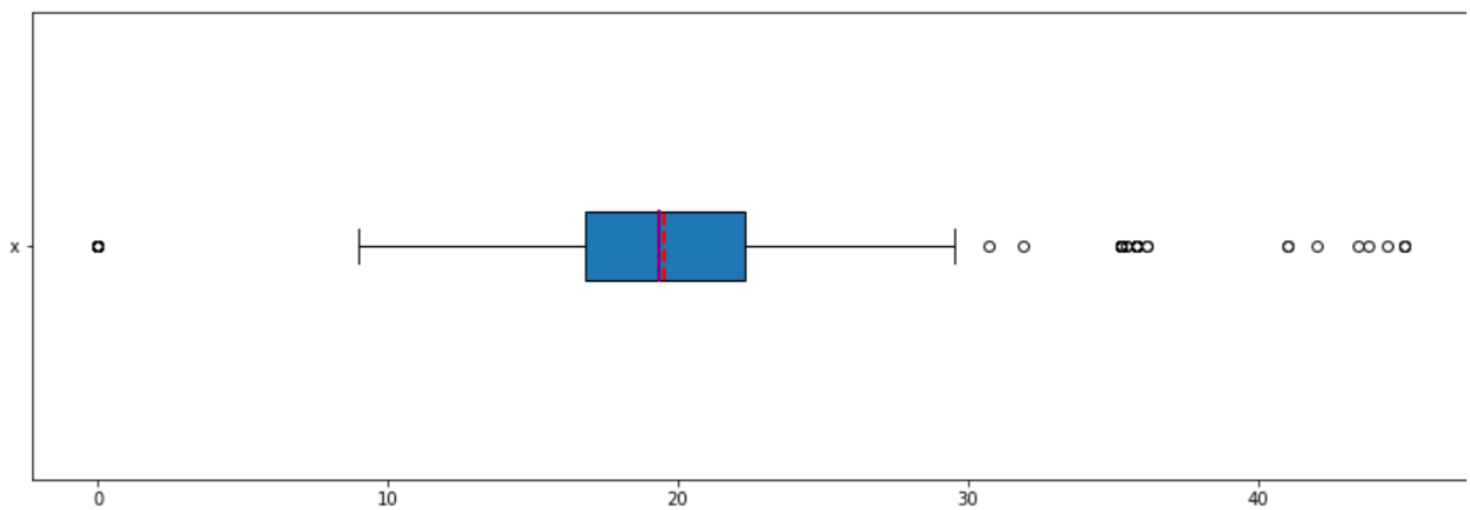
Box Plot for year

```
fig, ax = plt.subplots(figsize=(15, 5))
ax.boxplot(dataset.km_driven.values, vert=False, showmeans=True, meanline=True,
    labels='x', patch_artist=True,
    medianprops={'linewidth': 2, 'color': 'purple'},
    meanprops={'linewidth': 2, 'color': 'red'})
plt.show()
```



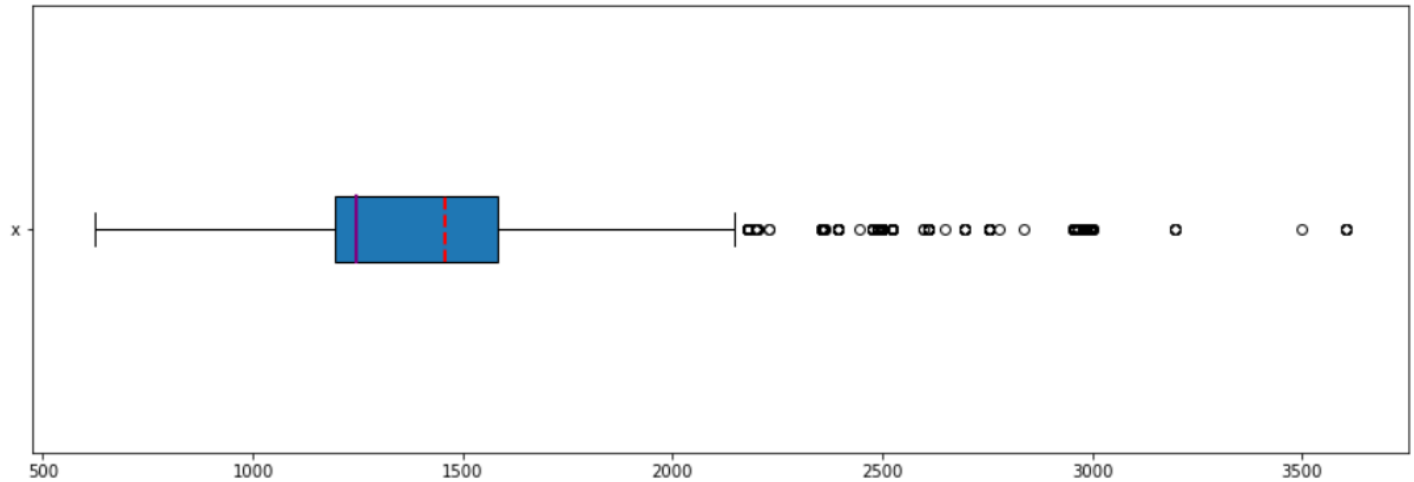
Box Plot for Km Driven

```
fig, ax = plt.subplots(figsize=(15, 5))
ax.boxplot(dataset.mileage.values, vert=False, showmeans=True, meanline=True,
    labels='x', patch_artist=True,
    medianprops={'linewidth': 2, 'color': 'purple'},
    meanprops={'linewidth': 2, 'color': 'red'})
plt.show()
```



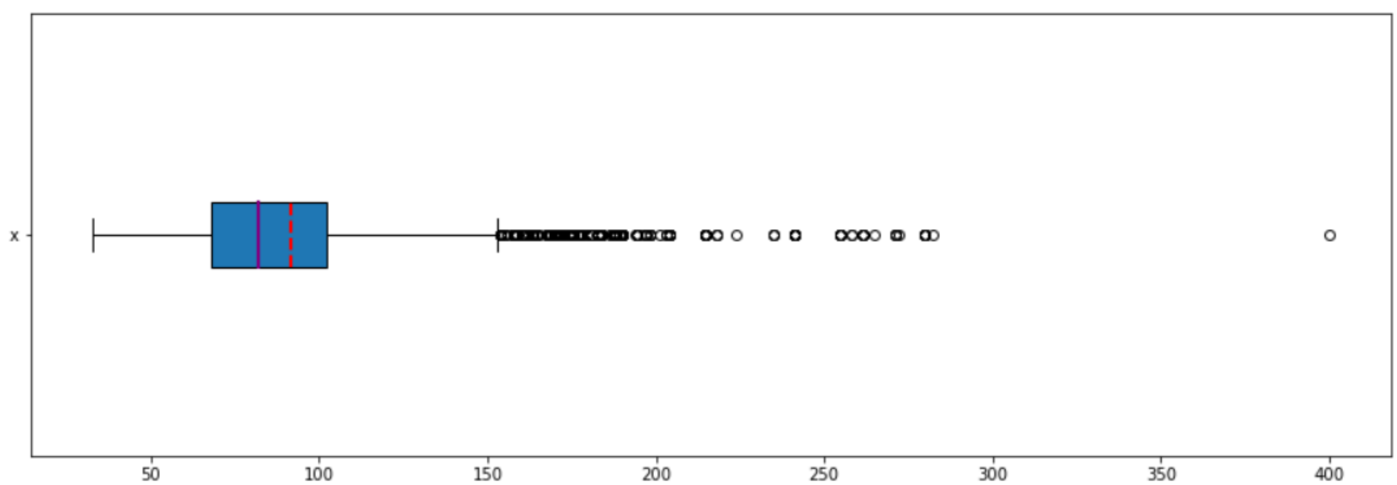
Box Plot for Mileage

```python
fig, ax = plt.subplots(figsize=(15, 5))
ax.boxplot(dataset.engine.values, vert=False, showmeans=True, meanline=True,
    labels='x', patch_artist=True,
    medianprops={'linewidth': 2, 'color': 'purple'},
    meanprops={'linewidth': 2, 'color': 'red'})
plt.show()
```



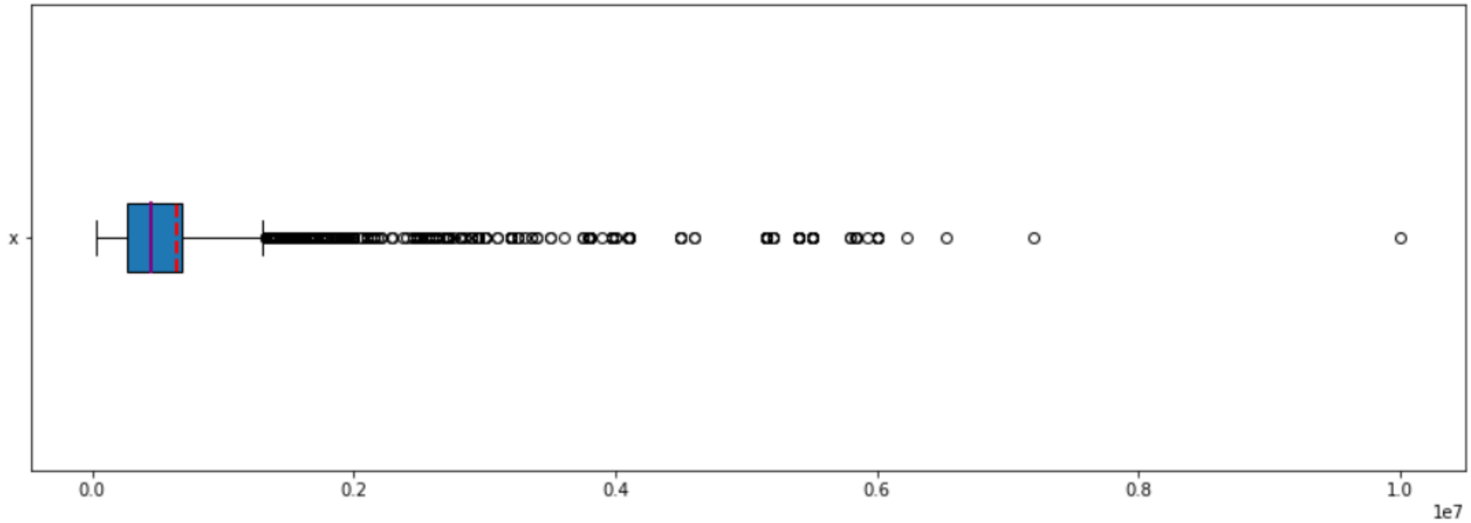
Box Plot for Engine Capacity

```python
fig, ax = plt.subplots(figsize=(15, 5))
ax.boxplot(dataset.max_power.values, vert=False, showmeans=True, meanline=True,
    labels='x', patch_artist=True,
    medianprops={'linewidth': 2, 'color': 'purple'},
    meanprops={'linewidth': 2, 'color': 'red'})
plt.show()
```



Box Plot for Max Power

```
fig, ax = plt.subplots(figsize=(15, 5))
ax.boxplot(dataset.selling_price.values, vert=False, showmeans=True, meanline=True,
    labels='x', patch_artist=True,
    medianprops={'linewidth': 2, 'color': 'purple'},
    meanprops={'linewidth': 2, 'color': 'red'})
plt.show()
```

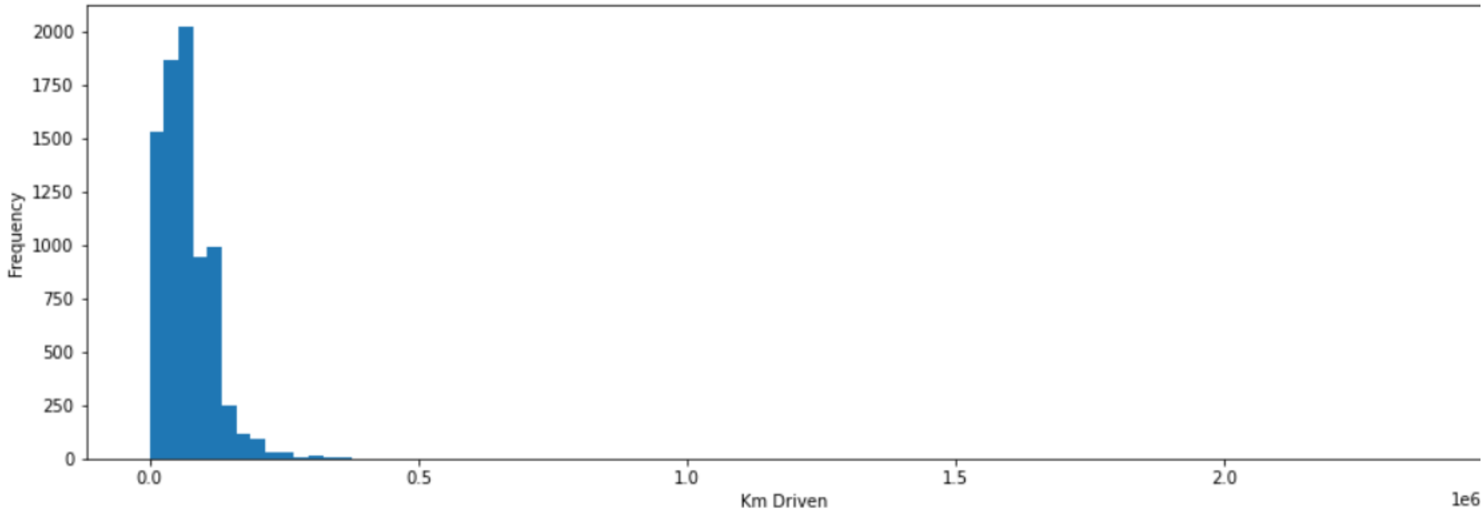


Box Plot for Selling Price

6.4 Histograms

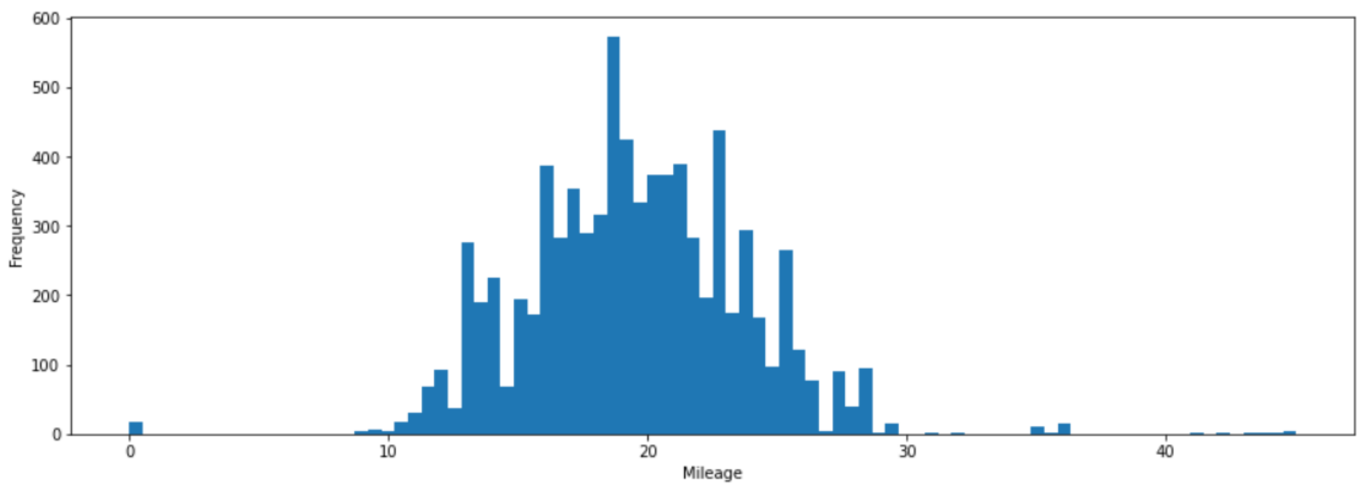
Histograms are particularly useful when there are a large number of unique values in a dataset. The histogram divides the values from a sorted dataset into intervals, also called bins. Often, all bins are of equal width, though this doesn't have to be the case.

```
hist, bin_edges = np.histogram(dataset.km_driven.values, bins=int(np.sqrt(len(dataset.km_driven))))

fig, ax = plt.subplots(figsize=(15, 5))
ax.hist(dataset.km_driven.values, bin_edges, cumulative=False)
ax.set_xlabel('Km Driven')
ax.set_ylabel('Frequency')
plt.show()
```



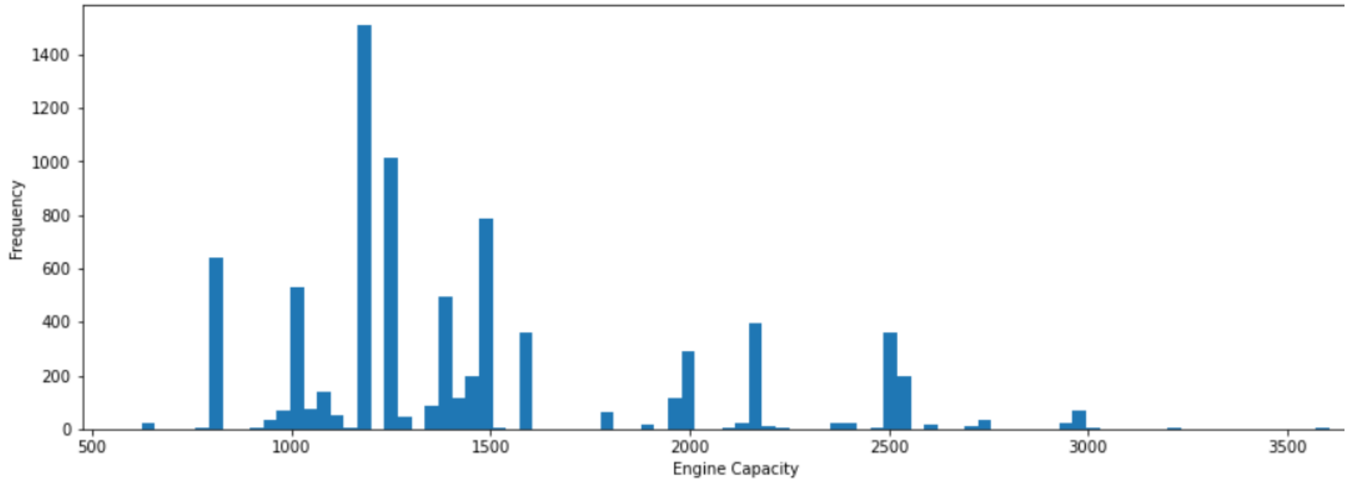
Histogram for Km Driven

```
hist, bin_edges = np.histogram(dataset.mileage.values, bins=int(np.sqrt(len(dataset.mileage))))

fig, ax = plt.subplots(figsize=(15, 5))
ax.hist(dataset.mileage.values, bin_edges, cumulative=False)
ax.set_xlabel('Mileage')
ax.set_ylabel('Frequency')
plt.show()
```



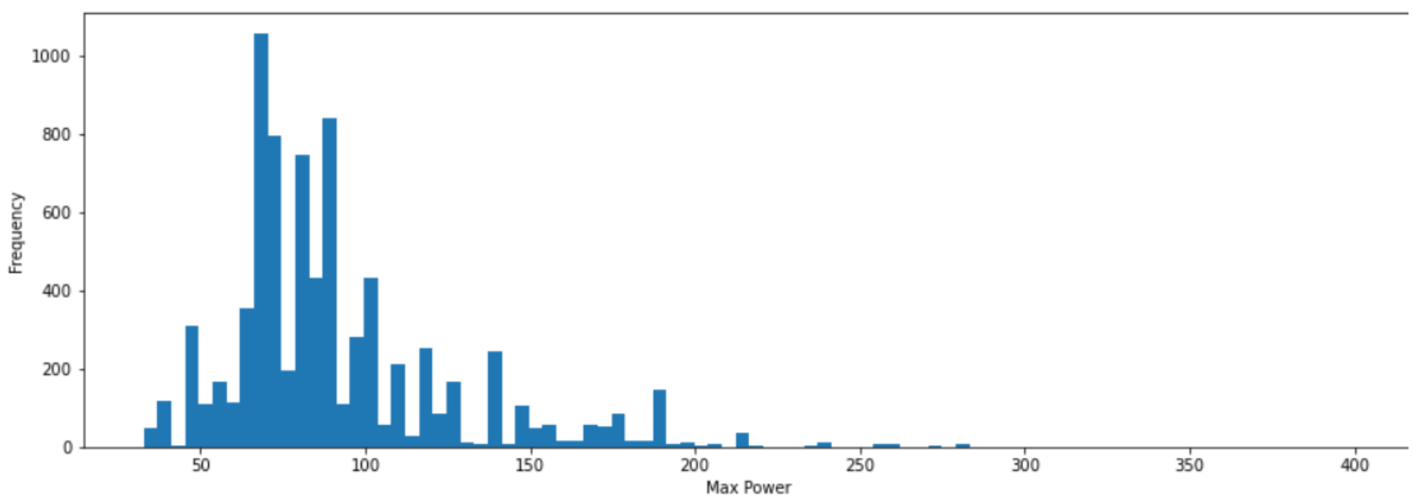
Histogram for Mileage

```
hist, bin_edges = np.histogram(dataset.engine.values, bins=int(np.sqrt(len(dataset.engine))))

fig, ax = plt.subplots(figsize=(15, 5))
ax.hist(dataset.engine.values, bin_edges, cumulative=False)
ax.set_xlabel('Engine Capacity')
ax.set_ylabel('Frequency')
plt.show()
```



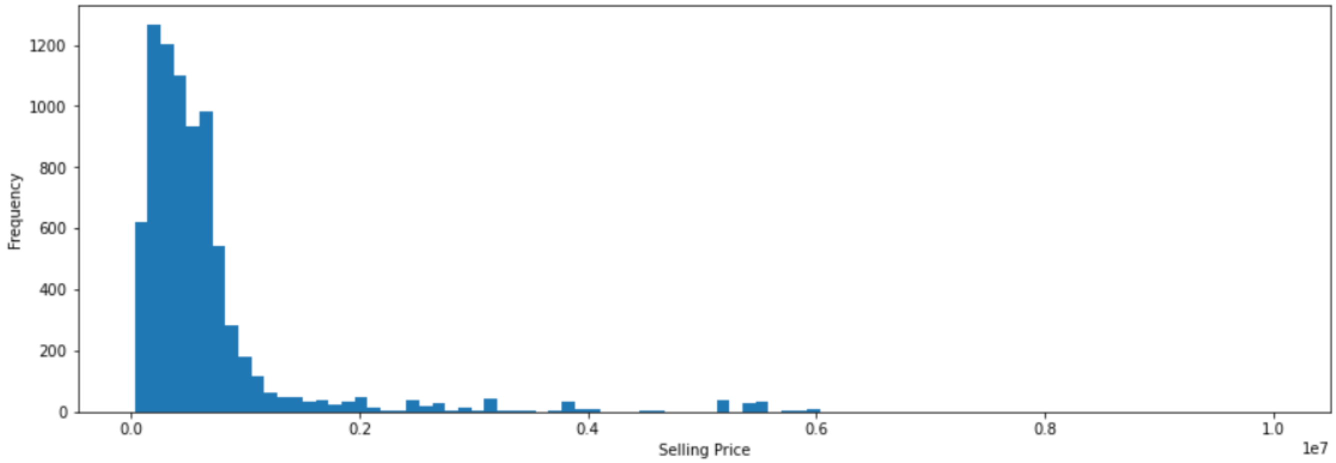
Histogram for Engine Capacity

```
hist, bin_edges = np.histogram(dataset.max_power.values, bins=int(np.sqrt(len(dataset.max_power))))

fig, ax = plt.subplots(figsize=(15, 5))
ax.hist(dataset.max_power.values, bin_edges, cumulative=False)
ax.set_xlabel('Max Power')
ax.set_ylabel('Frequency')
plt.show()
```



Histogram for Max Power

```
hist, bin_edges = np.histogram(dataset.selling_price.values, bins=int(np.sqrt(len(dataset.selling_price))))

fig, ax = plt.subplots(figsize=(15, 5))
ax.hist(dataset.selling_price.values, bin_edges, cumulative=False)
ax.set_xlabel('Selling Price')
ax.set_ylabel('Frequency')
plt.show()
```



Histogram for Selling Price

7

Measures of Correlation

As we want to predict selling price, we should find how other variables relate to it.

• Year, and Selling Price

```
In [278]: # Year , Selling Price
          dataset.year.corr(dataset.selling_price)

Out[278]: 0.41230155817117004
```

r shows that the relation ***Moderate*** and the direction is that the values increase or decrease *together*.

- ## KM Driven, and Selling Price

```
# km driven , Selling Price
dataset.km_driven.corr(dataset.selling_price)
```

```
-0.2221584753348377 6
```

r shows that the relation ***Weak*** and the direction is that when one value decreases the other decreases.

- ## Mileage, and Selling Price

```
# Mileage , Selling Price
dataset.mileage.corr(dataset.selling_price)
```

```
-0.1301727835191734
```

r shows that the relation ***Weak*** and the direction is that when one value decreases the other decreases.

- ## Engine, and Selling Price

```
# Engine , Selling Price
dataset.engine.corr(dataset.selling_price)
```

```
0.4556818000356144
```

r shows that the relation ***Moderate*** and the direction is that the values increase or decrease *together*.

- ## Max Power, and Selling Price

```
# Max power , Selling Price
dataset.max_power.corr(dataset.selling_price)
```

```
0.7496737800444901
```

r shows that the relation ***Strong*** and the direction is that the values increase or decrease *together*.

• Seats, and Selling Price

```
# seats , Selling Price
dataset.seats.corr(dataset.selling_price)
```

```
0.04161669383026344
```

r shows that no correlation (the values don't seem linked at all).

So, we will drop it from the dataset.

```
# Drop the seats column
dataset = dataset.drop(columns=['seats'], axis=1)
dataset.head()
```

	company	model	year	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	selling_price
0	Maruti	Swift	2014	145500	Diesel	Individual	Manual	First Owner	23.40	1248	74.00	450000
1	Skoda	Rapid	2014	120000	Diesel	Individual	Manual	Second Owner	21.14	1498	103.52	370000
2	Honda	City	2006	140000	Petrol	Individual	Manual	Third Owner	17.70	1497	78.00	158000
3	Hyundai	i20	2010	127000	Diesel	Individual	Manual	First Owner	23.00	1396	90.00	225000
4	Maruti	Swift	2007	120000	Petrol	Individual	Manual	First Owner	16.10	1298	88.20	130000

8

Sources

Used Cars' data:

https://www.kaggle.com/nehalbirla/vehicle-dataset-from-cardekho?select=car+data.csv