Data Analysis

- This dataset contains information about used motorcycles This data can be used for a lot of purposes such as price prediction to exemplify the use of linear regression in Machine Learning. The columns in the given dataset are as follows:

- name
- selling price
- year
- seller type
- owner
- km driven
- ex showroom price, For used car datasets please go to https://www.kaggle.com/nehalbirla/vehicle-dataset-from-cardekho

In [3]:
```
from PIL import Image
jpeg = Image.open("C:/Users/Shubh/Downloads/paulo-freitas-qqqyKGveYoc-unsplash.jpg")
jpeg
```

Out[3]:

Import Libraries

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import sklearn
```

In [5]:
```
URL = "C:/Users/Shubh/Downloads/BIKE DETAILS.csv"
bike = pd.read_csv(URL)
```

In [6]:
```
bike.head()
```

Out[6]:

	name	selling_price	year	seller_type	owner	km_driven	ex_showroom_price
0	Royal Enfield Classic 350	175000	2019	Individual	1st owner	350	NaN
1	Honda Dio	45000	2017	Individual	1st owner	5650	NaN
2	Royal Enfield Classic Gunmetal Grey	150000	2018	Individual	1st owner	12000	148114.0
3	Yamaha Fazer FI V 2.0 [2016-2018]	65000	2015	Individual	1st owner	23000	89643.0
4	Yamaha SZ [2013-2014]	20000	2011	Individual	2nd owner	21000	NaN

In [7]:
```
bike.tail()
```

Out[7]:

	name	selling_price	year	seller_type	owner	km_driven	ex_showroom_price
1056	Activa 3g	17000	2010	Individual	1st owner	500000	52000.0
1057	Honda CB twister	16000	2012	Individual	1st owner	33000	51000.0
1058	Bajaj Discover 125	15000	2013	Individual	2nd owner	35000	57000.0
1059	Honda CB Shine	12000	2009	Individual	1st owner	53000	58000.0
1060	Bajaj Pulsar 150	10000	2008	Individual	1st owner	92233	75000.0

In [8]:
```
Total_Rows,Total_columns=bike.shape
print("Records:",Total_Rows)
print("Attributes",Total_columns)
```

```
Records: 1061
Attributes 7
```

In [9]:
```
path = 'C:/Users/Shubh/bike_price.csv'
bike.to_csv(path)
```

In [10]:
```
bike.dtypes
```

Out[10]:
```
name                 object
selling_price         int64
year                  int64
seller_type          object
owner                object
km_driven             int64
ex_showroom_price    float64
dtype: object
```

In [11]:
```
bike.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1061 entries, 0 to 1060
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   name               1061 non-null   object 
 1   selling_price      1061 non-null   int64  
 2   year               1061 non-null   int64  
 3   seller_type        1061 non-null   object 
 4   owner              1061 non-null   object 
 5   km_driven          1061 non-null   int64  
 6   ex_showroom_price  626 non-null    float64
dtypes: float64(1), int64(3), object(3)
memory usage: 58.1+ KB
```

Connect the MySQL to the Python

In [12]:
```
pip install mysql.connector.python
```

```
Requirement already satisfied: mysql.connector.python in c:\users\shubh\anaconda3\lib\site
-packages (8.0.31)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: protobuf<=3.20.1,>=3.11.0 in c:\users\shubh\anaconda3\lib\s
ite-packages (from mysql.connector.python) (3.20.1)
```

In [13]:
```
import mysql.connector

mydb = mysql.connector.connect(
    host = "localhost",
    user = "root",
    password = "Shubh@$123",
    database = "bike_data"
)

cursor = mydb.cursor()

# Creating a table called 'bike_info' in the
# 'bike_Data' database
cursor.execute("select*from basic_info")
```

In [14]:
```
bike.describe(include = 'all')
```

Out[14]:

	name	selling_price	year	seller_type	owner	km_driven	ex_showroom_price
count	1061	1061.000000	1061.000000	1061	1061	1061.000000	6.260000e+02
unique	279	NaN	NaN	2	4	NaN	NaN
top	Bajaj Pulsar 150	NaN	NaN	Individual	1st owner	NaN	NaN
freq	41	NaN	NaN	1055	924	NaN	NaN
mean	NaN	59638.151744	2013.867107	NaN	NaN	34359.833176	8.795871e+04
std	NaN	56304.291973	4.301191	NaN	NaN	51623.152702	7.749659e+04
min	NaN	5000.000000	1988.000000	NaN	NaN	350.000000	3.049000e+04
25%	NaN	28000.000000	2011.000000	NaN	NaN	13500.000000	5.485200e+04
50%	NaN	45000.000000	2015.000000	NaN	NaN	25000.000000	7.275250e+04
75%	NaN	70000.000000	2017.000000	NaN	NaN	43000.000000	8.703150e+04

	name	selling_price	year	seller_type	owner	km_driven	ex_showroom_price
max	NaN	760000.000000	2020.000000	NaN	NaN	880000.000000	1.278000e+06

In [15]:
```
bike.describe()
```

Out[15]:

	selling_price	year	km_driven	ex_showroom_price
count	1061.000000	1061.000000	1061.000000	6.260000e+02
mean	59638.151744	2013.867107	34359.833176	8.795871e+04
std	56304.291973	4.301191	51623.152702	7.749659e+04
min	5000.000000	1988.000000	350.000000	3.049000e+04
25%	28000.000000	2011.000000	13500.000000	5.485200e+04
50%	45000.000000	2015.000000	25000.000000	7.275250e+04
75%	70000.000000	2017.000000	43000.000000	8.703150e+04
max	760000.000000	2020.000000	880000.000000	1.278000e+06

In [16]:
```
categorical_features = bike.select_dtypes(include=object)
categorical_features.columns
categorical_features.describe()
```

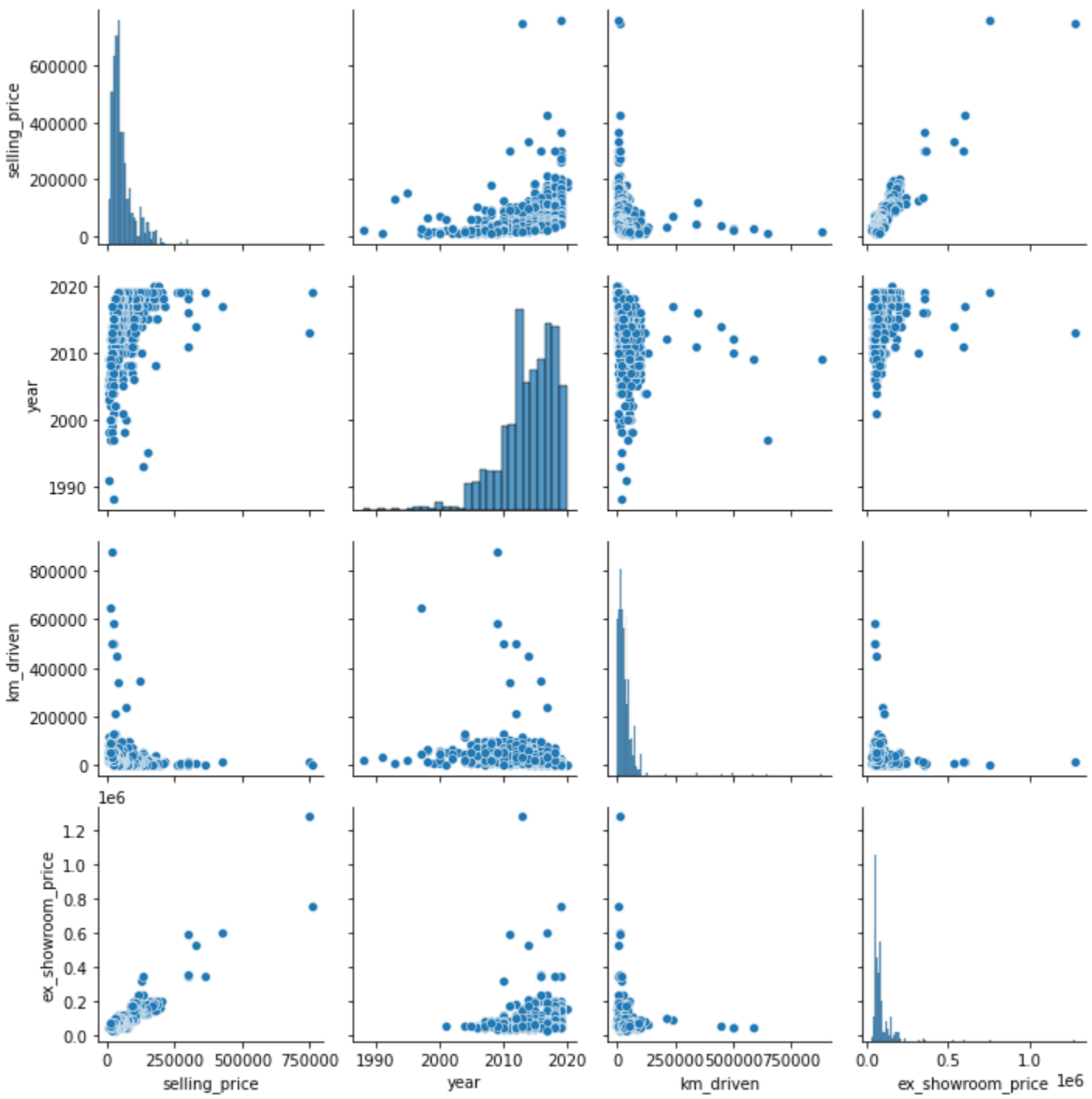
Out[16]:

	name	seller_type	owner
count	1061	1061	1061
unique	279	2	4
top	Bajaj Pulsar 150	Individual	1st owner
freq	41	1055	924

DATA VISUALIZATION

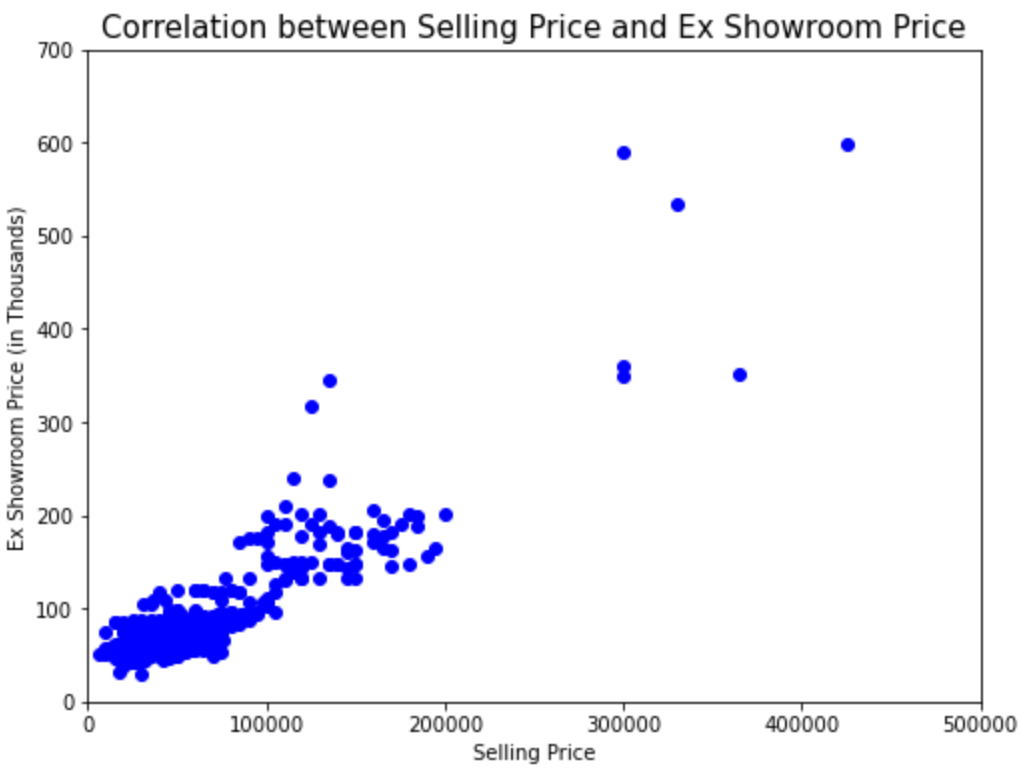
In [17]:
```
sns.pairplot(bike)
```

Out[17]:
```
<seaborn.axisgrid.PairGrid at 0x2019c9fcee0>
```



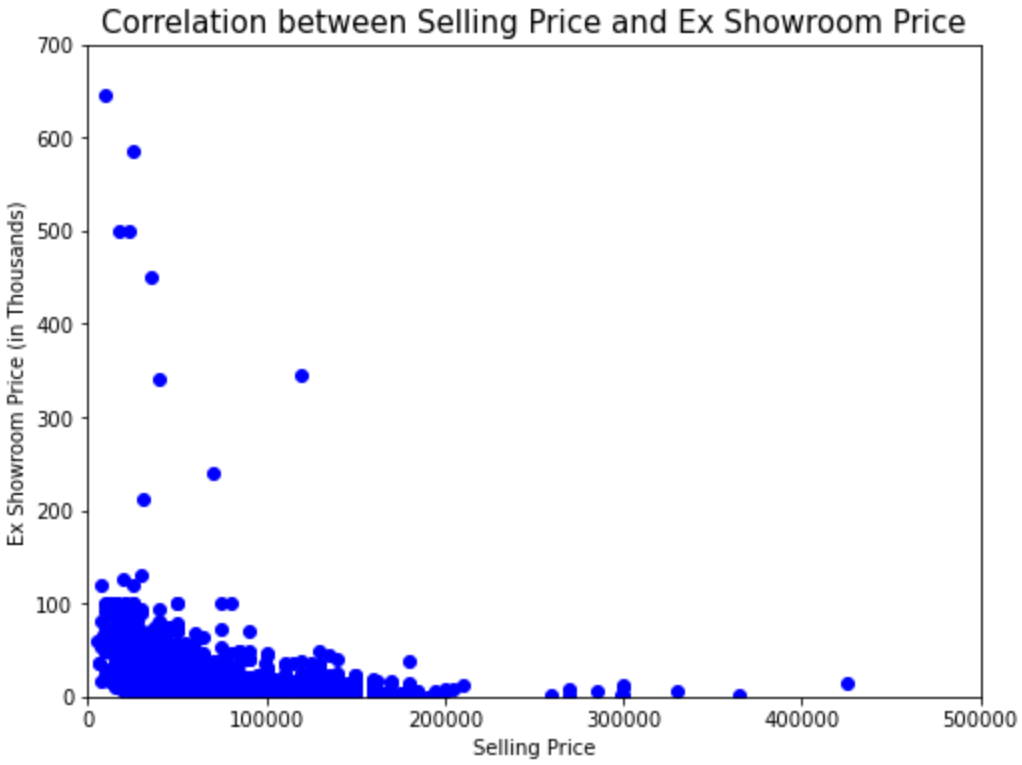
In [21]:

```
plt.figure(figsize=(8, 6))

plt.scatter(bike['selling_price'], bike['ex_showroom_price'],color = 'b')
plt.title('Correlation between Selling Price and Ex Showroom Price', fontsize=15)
plt.xlabel('Selling Price')
plt.ylabel('Ex Showroom Price (in Thousands)')
plt.xlim(xmin=0, xmax=500000)
plt.ylim(ymin=0, ymax=700000 )
labels, locations = plt.yticks()
plt.yticks(labels, (labels/1000).astype(int))
plt.show()
```



In [23]:

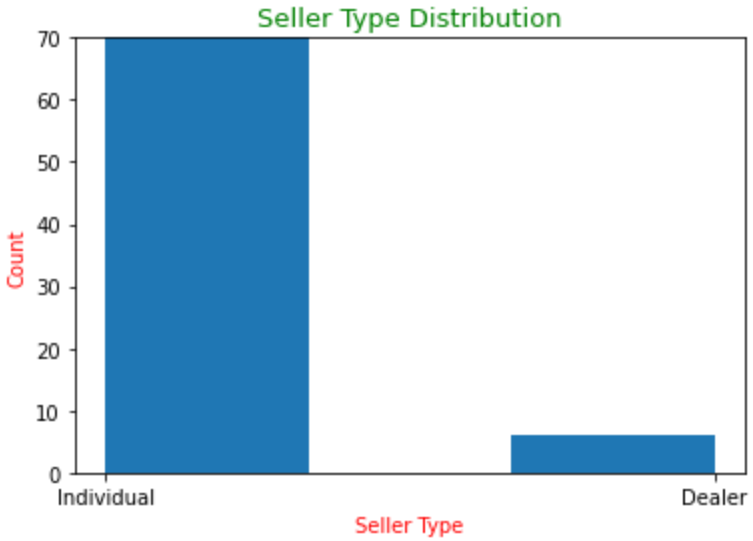
```
plt.figure(figsize=(8, 6))

plt.scatter(bike['selling_price'], bike['km_driven'],color = 'b')
plt.title('Correlation between Selling Price and Ex Showroom Price', fontsize=15)
plt.xlabel('Selling Price')
plt.ylabel('Ex Showroom Price (in Thousands)')
plt.xlim(xmin=0, xmax=500000)
plt.ylim(ymin=0, ymax=700000 )
labels, locations = plt.yticks()
plt.yticks(labels, (labels/1000).astype(int))
plt.show()
```



In [24]:

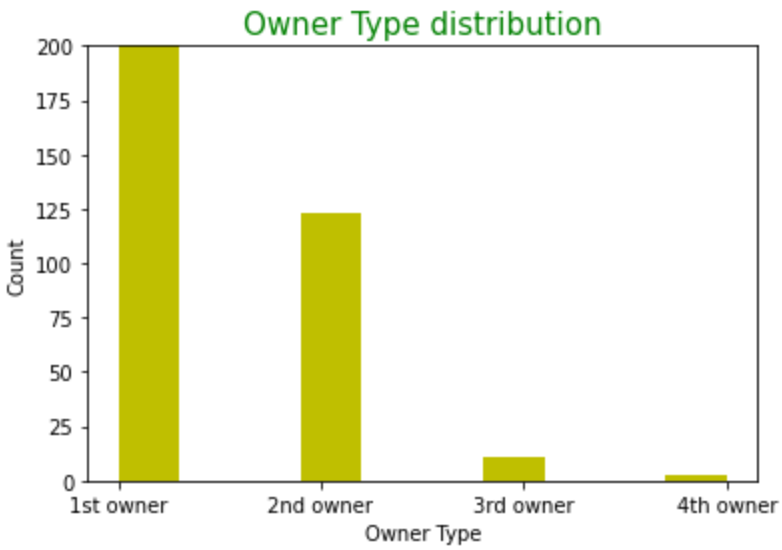
```
ler_type'], bins=3),plt.title('Seller Type Distribution', f
```

```
plt.xlabel('Seller Type',color= 'r'),plt.ylabel('Count',color= 'r')
plt.ylim(ymax=70)
plt.show()
```



In [25]:

```
plt.figure()
plt.hist(bike['owner'],color='y')
plt.title('Owner Type distribution', fontsize= 15,color='g')
plt.xlabel('Owner Type')
plt.ylabel('Count')
plt.ylim(ymax=200)
plt.show()
```



DATA PREPROCESSING

In [26]:

```
bike.shape
```

Out[26]:

```
(1061, 7)
```

HANDLING REDUNDANCY

In [27]:

```
if bike.duplicated().sum() > 0:
  print('There is redundancy')
```

```
else:
  print('No redundancy')
```

```
There is redundancy
```

In [28]:
```
print('Duplicates', bike.duplicated().sum())
```

```
Duplicates 6
```

Remove the redundancy from data

In [29]:
```
data = bike.drop_duplicates()
print('Duplicates : ', data.duplicated().sum())
print('Data :', data.shape[0])
```

```
Duplicates :  0
Data : 1055
```

Handling Missing Values

In [30]:
```
data.isnull().sum()
```

Out[30]:
```
name                   0
selling_price          0
year                   0
seller_type            0
owner                  0
km_driven              0
ex_showroom_price    433
dtype: int64
```

In [31]:
```
def cek_null_percentage(data):
    col = data.isnull().sum().sort_values(ascending=False)
    percent = col / len(data)

    missing_data = pd.concat([col, percent], axis=1, keys=['Total', 'Percent'])
    print(missing_data[missing_data['Total'] > 0])
```

Missing values filled by MEAN

In [32]:
```
cek_null_percentage(data)
```

```
                   Total   Percent
ex_showroom_price    433  0.410427
```

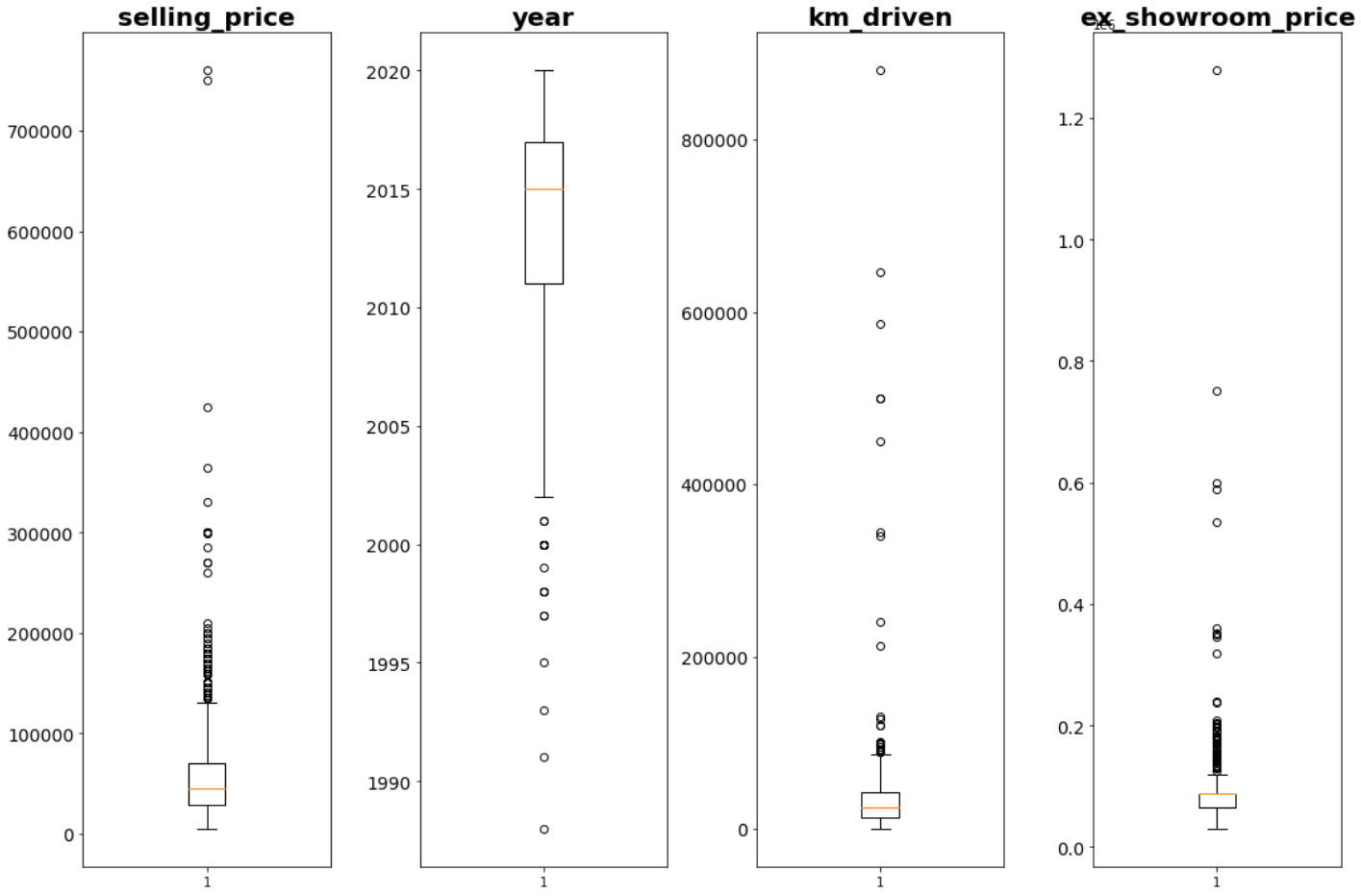
In [33]:
```
data['ex_showroom_price'] = data['ex_showroom_price'].fillna(data['ex_showroom_price'].mea
print('Missing values :', data.isnull().sum().sum())
```

```
Missing values : 0
C:\Users\Shubh\AppData\Local\Temp/ipykernel_11228/4146330614.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  data['ex_showroom_price'] = data['ex_showroom_price'].fillna(data['ex_showroom_price'].m
ean())
```

Outlier Handling

In [34]:
```
data_numeric = data.select_dtypes(include=np.number)
```

In [39]:
```
red_circle = dict(markerfacecolor='pink', marker='g', markeredgecolor='yellow')
fig, axs = plt.subplots(1, len(data_numeric.columns), figsize=(15,10))

for i, ax in enumerate(axs.flat):
    ax.boxplot(data_numeric.iloc[:,i])
    ax.set_title(data_numeric.columns[i], fontsize=20, fontweight='bold')
    ax.tick_params(axis='y', labelsize=14)
plt.tight_layout()
```



In [40]:
```
def outliers(column):
  q1 = data[column].quantile(0.25)
  q3 = data[column].quantile(0.75)
  iqr =  q3 - q1
  upper_range = q3 + (1.5 * iqr)
  lower_range = q1 - (1.5 * iqr)
  count_outlier = 0
  for i in data[column]:
    if i < lower_range or i > upper_range:
      count_outlier += 1
  return count_outlier

for col in data_numeric:
  print('Outlier from', col,':',outliers(col))
```

```
Outlier from selling_price : 87
Outlier from year : 18
```

```
Outlier from km_driven : 38
Outlier from ex_showroom_price : 83
```

Outlier from selling_price

In [41]:
```
q = data['selling_price'].quantile(0.97)
data = data[data['selling_price'] < q]
```

Outlier from km_driven

In [42]:
```
data = data[data['km_driven'] < 100000]
```

Outlier from year

In [43]:
```
q = data['year'].quantile(0.01)
data = data[data['year'] > q]
```

Handling Categorical Features (Encode the label)

In [44]:
```
data = pd.get_dummies(data, columns = ['owner', 'seller_type'], drop_first = True)
```

In [45]:
```
data.drop(['name'], axis=1, inplace=True)
```

In [46]:
```
data.head()
```

Out[46]:

	selling_price	year	km_driven	ex_showroom_price	owner_2nd owner	owner_3rd owner	seller_type_Individual
1	45000	2017	5650	88060.794212	0	0	1
2	150000	2018	12000	148114.000000	0	0	1
3	65000	2015	23000	89643.000000	0	0	1
4	20000	2011	21000	88060.794212	1	0	1
5	18000	2010	60000	53857.000000	0	0	1

DATA MODELING

In [47]:
```
X = data.drop('selling_price', axis=1)
y = data['selling_price']
```

TRAIN TEST SPLIT

In [48]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

MODELING

In [49]:
```
from sklearn.linear_model import RidgeCV, LassoCV, LinearRegression
from sklearn.metrics import mean_absolute_error,mean_squared_error
```

In [50]:
```
# Model 1: Linear Regression
model_lr = LinearRegression().fit(X_train, y_train)
y_pred_lr = model_lr.predict(X_test)
```

In [51]:
```
model_lr.score(X_test, y_test)*100
```

Out[51]:
```
68.11094445961639
```

In [52]:
```
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred_lr))
print('Mean Squared Error:', mean_squared_error(y_test, y_pred_lr))
```

```
Mean Absolute Error: 14106.256829107982
Mean Squared Error: 388954384.8292543
```

In [53]:
```
# Model 2: Ridge Regression CV
model_rr = RidgeCV(alphas=[1, 0.1, 0.01, 0.0005]).fit(X_train, y_train)
y_pred_rr = model_rr.predict(X_test)
```

In [54]:
```
model_rr.score(X_test, y_test)*100
```

Out[54]:
```
68.12270012749212
```

In [55]:
```
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred_rr))
print('Mean Squared Error:', mean_squared_error(y_test, y_pred_rr))
```

```
Mean Absolute Error: 14102.16062238862
Mean Squared Error: 388810999.6304964
```

In [ ]:
```

```