VERZEO FINAL PROJECT

May 29, 2020

1 IMPORTS

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

%matplotlib inline
```

2 DATA ANALYSIS

[2]:
```
data=pd.read_csv('Data_Train.csv')
print(set(data['Location']))
print("\n")
print(set(data['Year']))
print("\n")
print(set(data['Fuel_Type']))
print("\n")
print(set(data['Transmission']))
print("\n")
print(set(data['Owner_Type']))
print("\n")
print(set(data['Seats']))
```

```
{'Delhi', 'Chennai', 'Hyderabad', 'Kolkata', 'Coimbatore', 'Pune', 'Ahmedabad',
'Mumbai', 'Bangalore', 'Jaipur', 'Kochi'}


{1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019}


{'CNG', 'Electric', 'Petrol', 'LPG', 'Diesel'}


{'Automatic', 'Manual'}
```

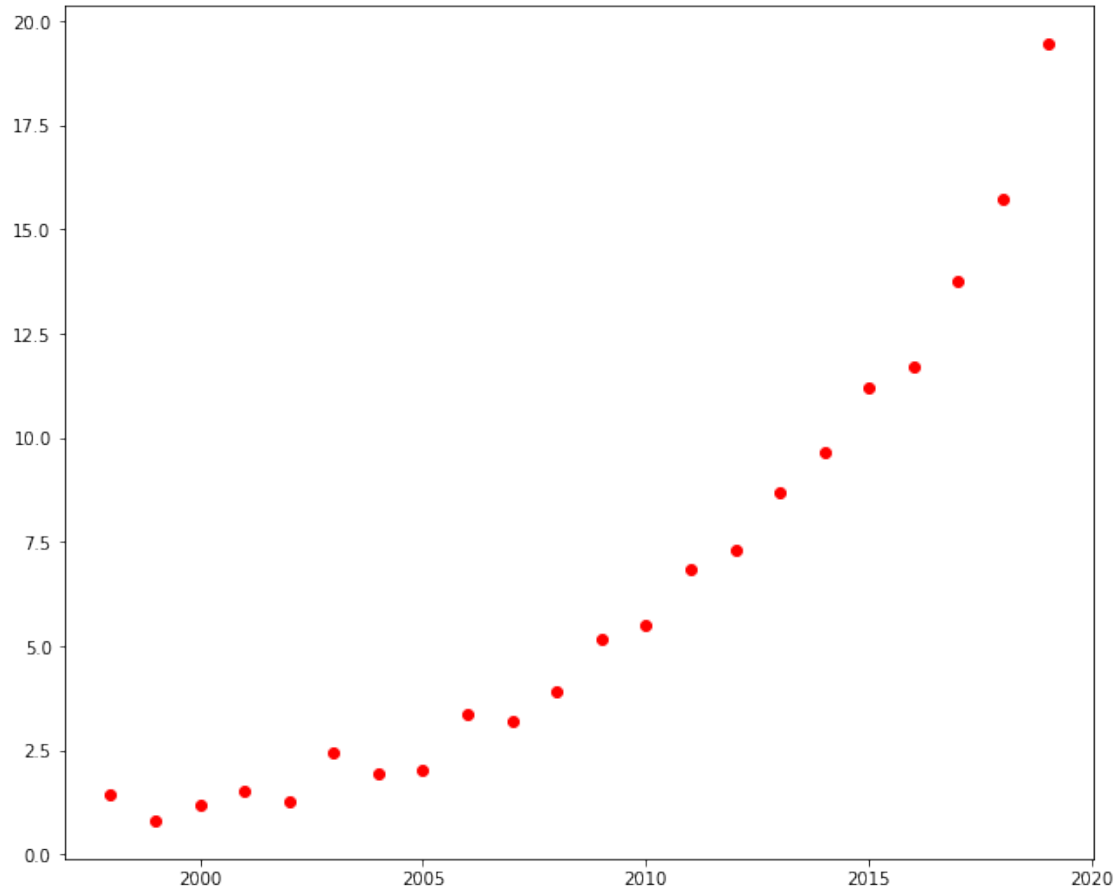
```
{'Third', 'First', 'Fourth & Above', 'Second'}
```

```
{nan, nan, 2.0, nan, 4.0, 5.0, 6.0, 7.0, 8.0, nan, nan, 10.0, nan, nan, 9.0,
nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, 0.0, nan, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan}
```

2.1 Price comparision on the basis of year.

[3]:
```
data=pd.read_csv('Data_Train.csv')
a=list(set(data['Year']))
b=[]
for i in range(len(a)):
    year=data.loc[data['Year']==a[i]]
    b.append(year.mean()[3])
arr=np.array(b)
    #bar graph

data = {"":[b[i] for i in range(len(a))]
       }

index     = [a[i] for i in range(len(a))]


dataFrame = pd.DataFrame(data=data, index=index)
dataFrame.plot.bar(rot=15, title="year wise price comparision")
plt.show()
```

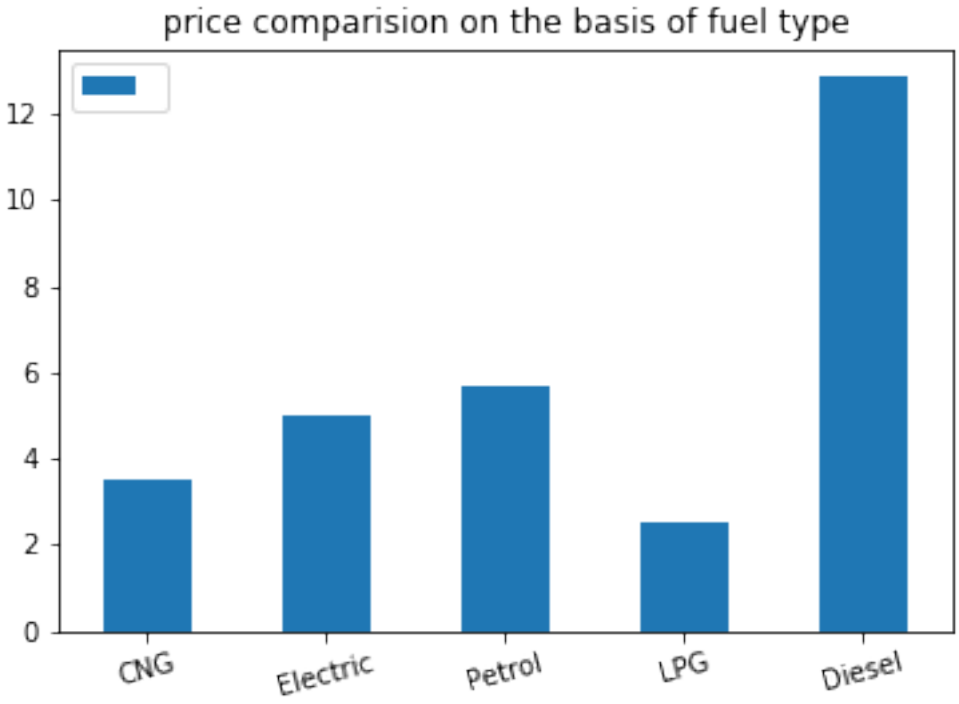
[4]:
```
# point dot graph
plt.figure(figsize=(36, 9))
plt.subplot(132)
plt.plot(range(1998,2020),arr,'ro')
plt.show()
```



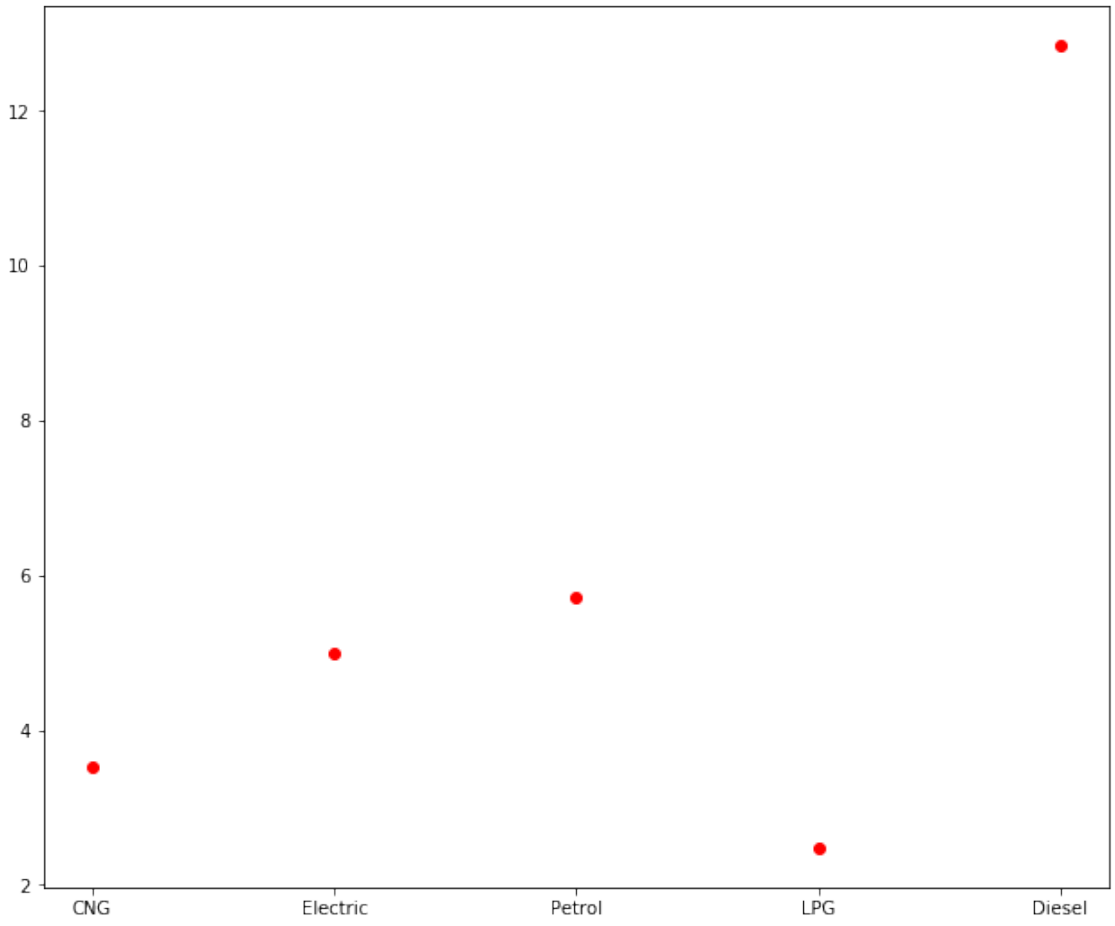
2.2 Price comparision on the basis of fuel type.

[5]:
```
data=pd.read_csv('Data_Train.csv')
a=list(set(data['Fuel_Type']))
b=[]
for i in range(len(a)):
    year=data.loc[data['Fuel_Type']==a[i]]
    b.append(year.mean()[3])
arr=np.array(b)
    #bar graph

data = {"":[b[i] for i in range(len(a))]
       }

index     = [a[i] for i in range(len(a))]


dataFrame = pd.DataFrame(data=data, index=index)
dataFrame.plot.bar(rot=15, title="price comparision on the basis of fuel type")
```

```
plt.show()
```



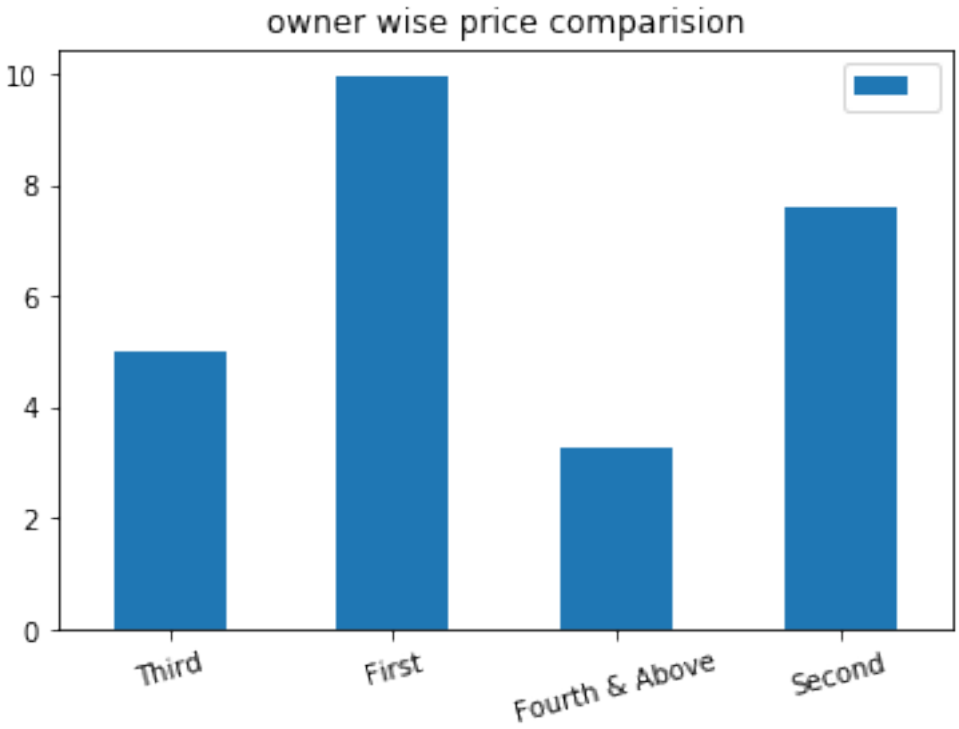
[6]:
```
# point dot graph
plt.figure(figsize=(36, 9))
plt.subplot(132)
plt.xticks(range(len(a)),a)
plt.plot(range(len(a)),arr,'ro')
plt.show()
```



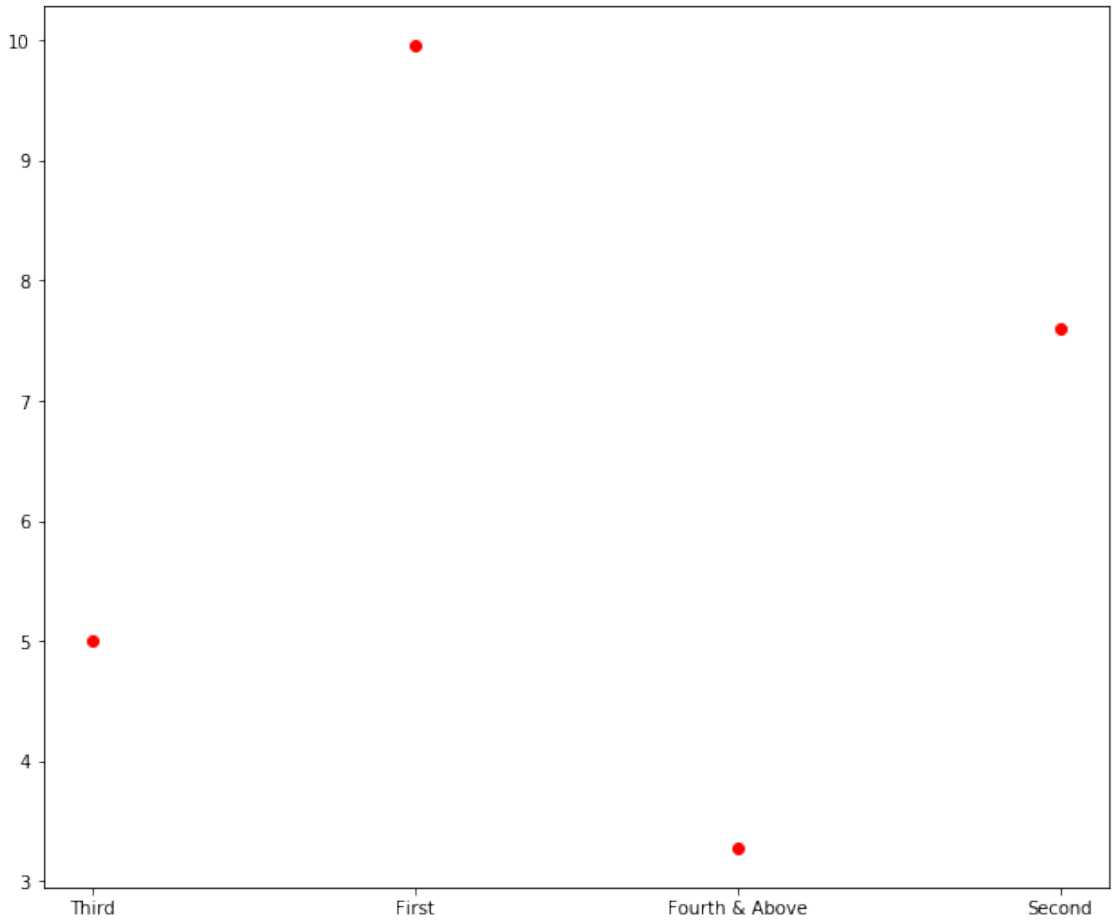
2.3 Price comparision on the basis of Owner_Type

[7]:
```
data=pd.read_csv('Data_Train.csv')
a=list(set(data['Owner_Type']))
b=[]
for i in range(len(a)):
    year=data.loc[data['Owner_Type']==a[i]]
    b.append(year.mean()[3])
arr=np.array(b)
    #bar graph

data = {"":[b[i] for i in range(len(a))]
       }

index     = [a[i] for i in range(len(a))]


dataFrame = pd.DataFrame(data=data, index=index)
```

```
dataFrame.plot.bar(rot=15, title="owner wise price comparision")
plt.show()
```



[8]:
```
# point dot graph
plt.figure(figsize=(36, 9))
plt.subplot(132)
plt.xticks(range(len(a)),a)
plt.plot(range(len(a)),arr,'ro')
plt.show()
```

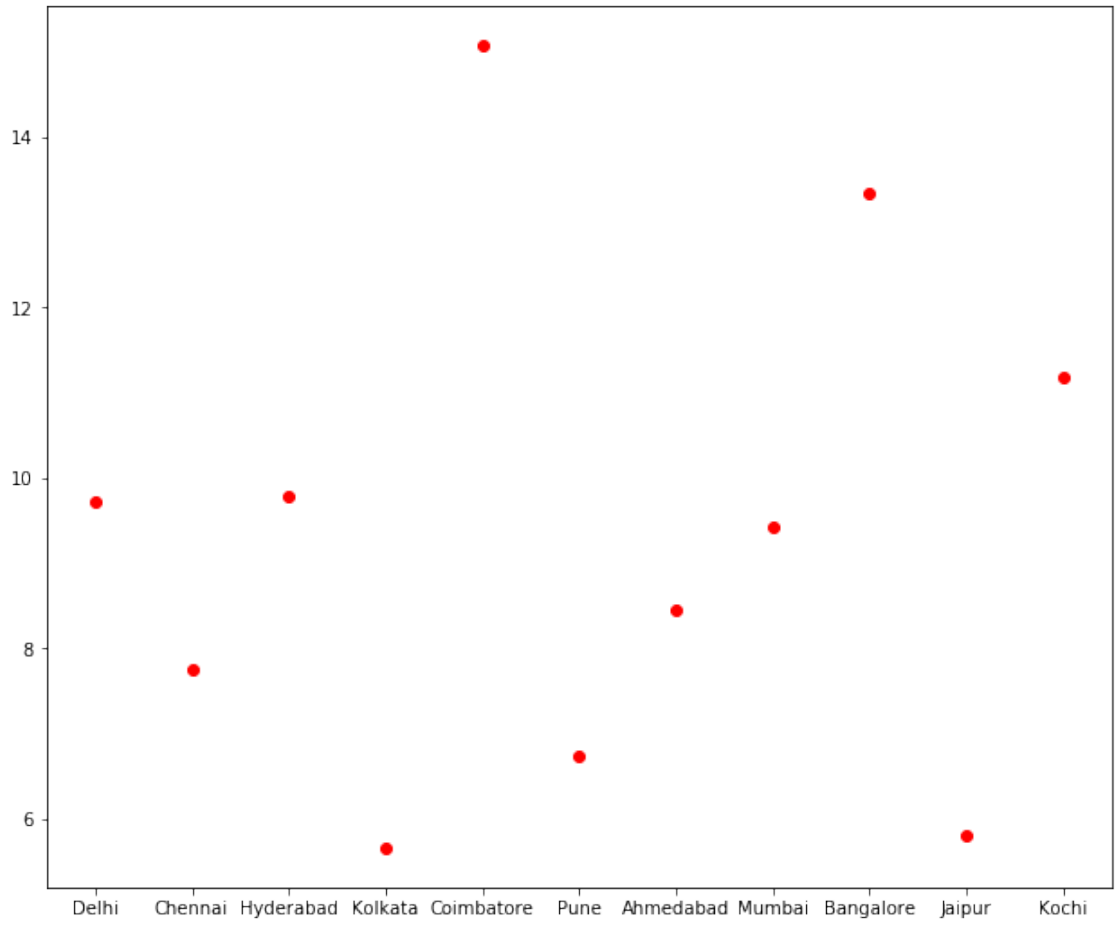


2.4 Comparision on the basis of location

[9]:
```
data=pd.read_csv('Data_Train.csv')
a=list(set(data['Location']))
b=[]
for i in range(len(a)):
    year=data.loc[data['Location']==a[i]]
    b.append(year.mean()[3])
arr=np.array(b)
    #bar graph

data = {"":[b[i] for i in range(len(a))]
       }

index     = [a[i] for i in range(len(a))]


dataFrame = pd.DataFrame(data=data, index=index)
```

```
dataFrame.plot.bar(rot=15, title="price comparision on the basis of location")
plt.show()
```

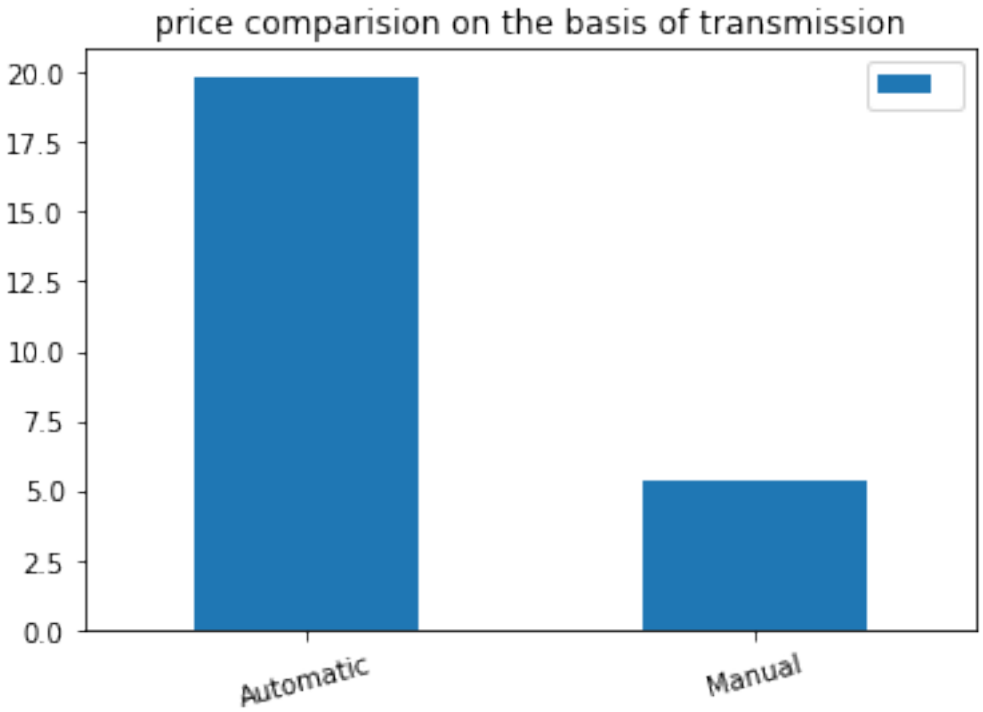
[10]:
```
# point dot graph
plt.figure(figsize=(36, 9))
plt.subplot(132)
plt.xticks(range(len(a)),a)
plt.plot(range(len(a)),arr,'ro')
plt.show()
```



2.5 Comparision on basis of Transmission

[11]:
```
data=pd.read_csv('Data_Train.csv')
a=list(set(data['Transmission']))
b=[]
for i in range(len(a)):
    year=data.loc[data['Transmission']==a[i]]
    b.append(year.mean()[3])
arr=np.array(b)
    #bar graph

data = {"":[b[i] for i in range(len(a))]
       }

index     = [a[i] for i in range(len(a))]


dataFrame = pd.DataFrame(data=data, index=index)
```

```
dataFrame.plot.bar(rot=15, title="price comparision on the basis of␣
 ↪transmission")
plt.show()
```

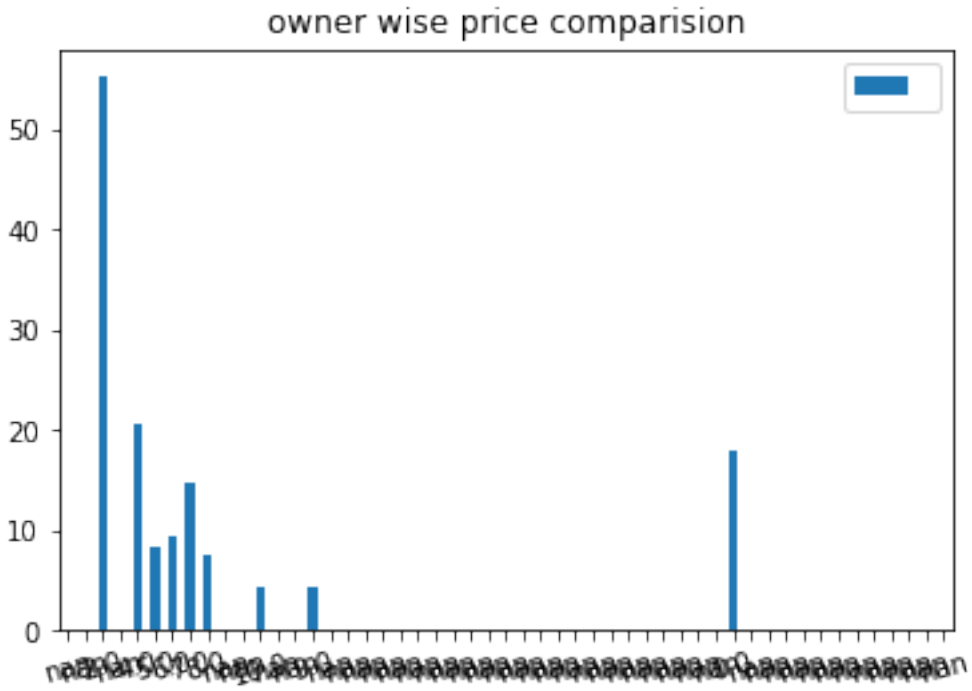


[12]:
```
# point dot graph
plt.figure(figsize=(36, 9))
plt.subplot(132)
plt.xticks(range(len(a)),a)
plt.plot(range(len(a)),arr,'ro')
plt.show()
```

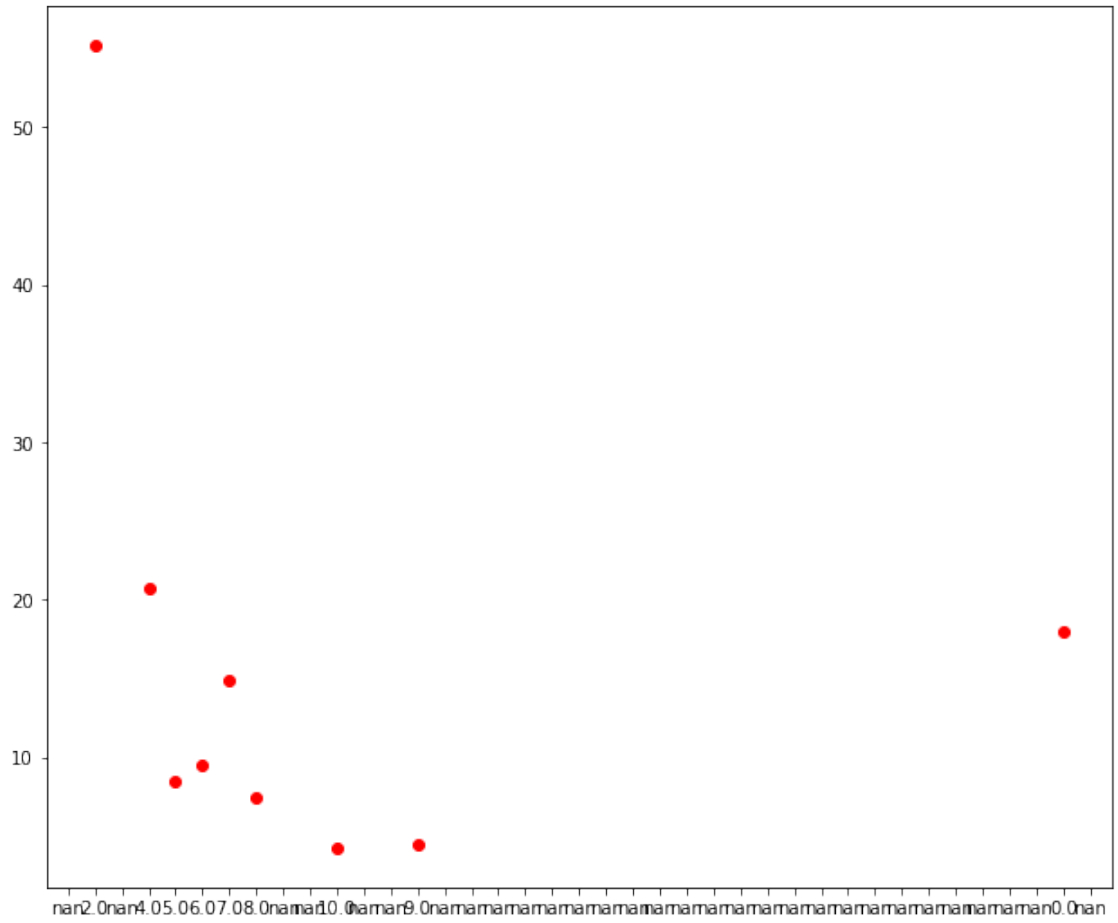
2.6 Comparision on basis of seat

[13]:
```
data=pd.read_csv('Data_Train.csv')
a=list(set(data['Seats']))
b=[]
for i in range(len(a)):
    year=data.loc[data['Seats']==a[i]]
    b.append(year.mean()[3])
arr=np.array(b)
    #bar graph

data = {"":[b[i] for i in range(len(a))]
       }

index     = [a[i] for i in range(len(a))]


dataFrame = pd.DataFrame(data=data, index=index)
```

```
dataFrame.plot.bar(rot=15, title="owner wise price comparision")
plt.show()
```



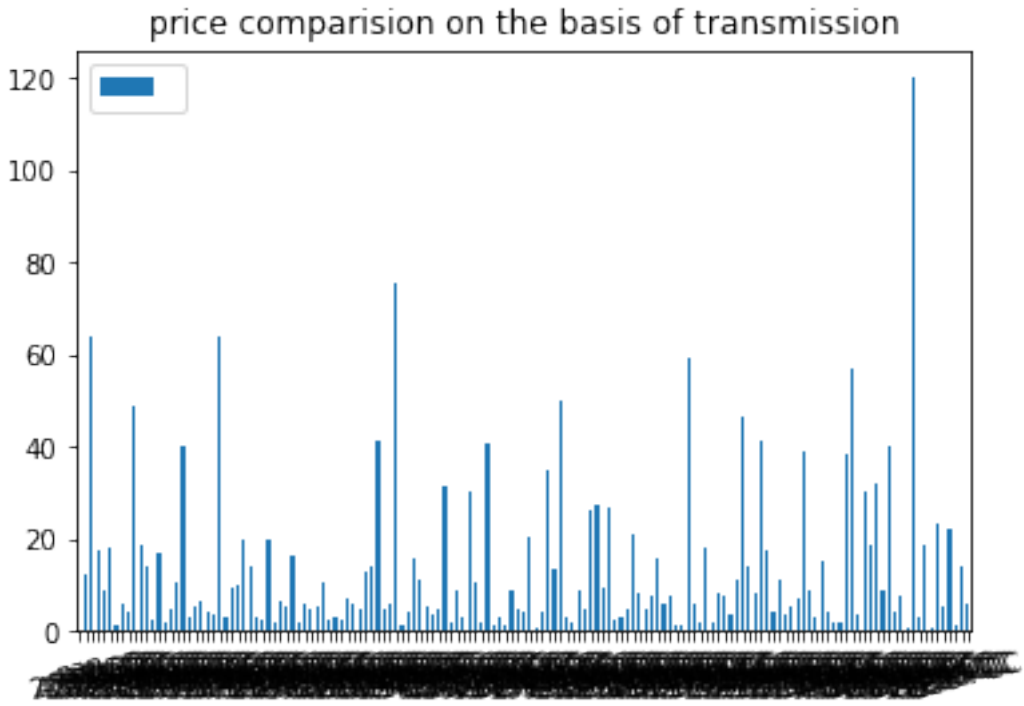
[14]:
```
# point dot graph
plt.figure(figsize=(36, 9))
plt.subplot(132)
plt.xticks(range(len(a)),a)
plt.plot(range(len(a)),arr,'ro')
plt.show()
```



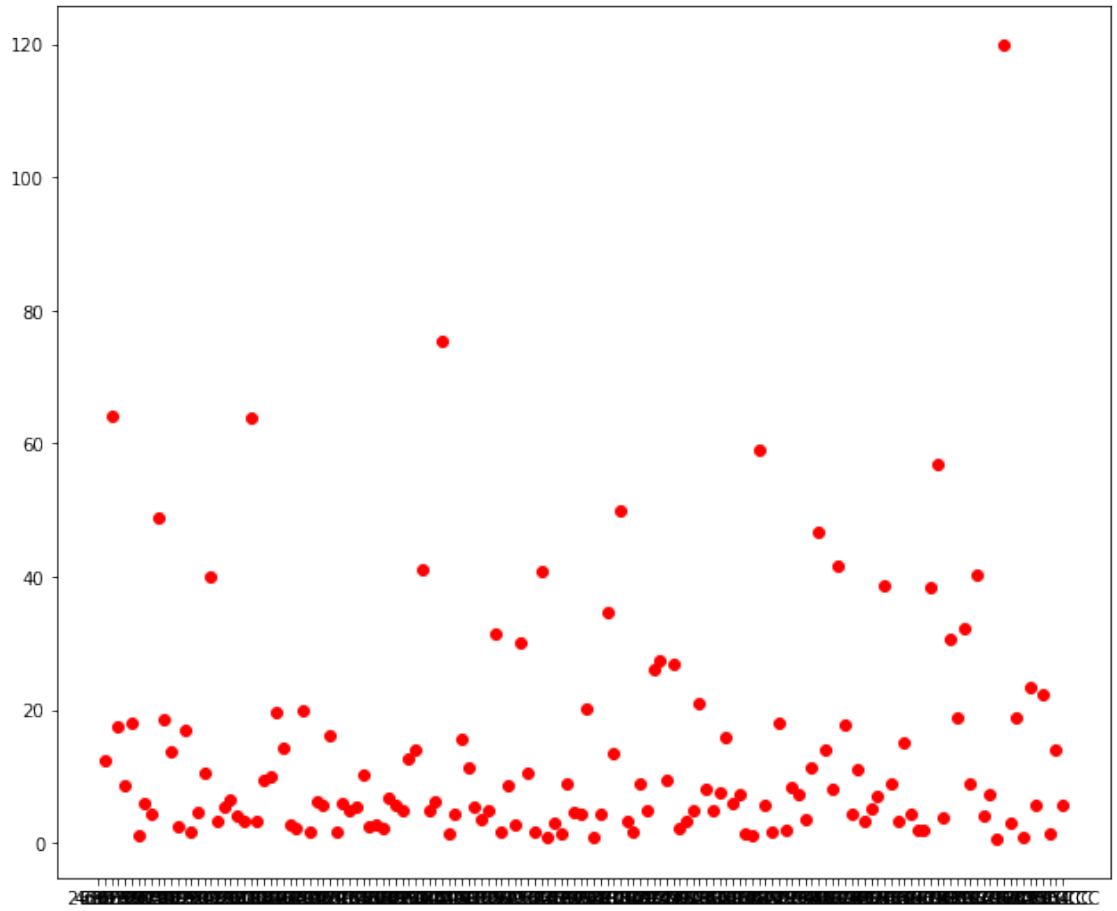
on basis of this analysis, I'm removing 'seats' column because in many cases data is not given in this column.

[15]:
```
data=pd.read_csv('Data_Train.csv')
a=list(set(data['Engine']))
b=[]
for i in range(len(a)):
    year=data.loc[data['Engine']==a[i]]
    b.append(year.mean()[3])
arr=np.array(b)
    #bar graph

data = {"":[b[i] for i in range(len(a))]
       }

index     = [a[i] for i in range(len(a))]
```

```
dataFrame = pd.DataFrame(data=data, index=index)
dataFrame.plot.bar(rot=15, title="price comparision on the basis of␣
 ↪transmission")
plt.show()
```



[16]:
```
# point dot graph
plt.figure(figsize=(36, 9))
plt.subplot(132)
plt.xticks(range(len(a)),a)
plt.plot(range(len(a)),arr,'ro')
plt.show()
```



3 PREDICTION

[17]:
```
df_train = pd.read_csv('Data_Train.csv')
df_test = pd.read_csv('Data_Test.csv')
```

[18]:
```
df_train_orig = df_train.copy()
df_test_orig = df_test.copy()
```

[19]:
```
print("Skew ", df_train['Price'].skew())
print("kurt ", df_train['Price'].kurt())
```

```
Skew  3.3352319876668415
kurt  17.09220197043644
```

[20]:
```
#A trial to check log of target label to avoid skew & kurt
df_test1 = np.log1p(df_train['Price'].values)
```

[21]:
```
df_test1 = df_test1.reshape(-1,1)
df_test1 = pd.DataFrame(df_test1, columns=['PriceNew'])
```

[22]:
```
print("Skew ", df_test1['PriceNew'].skew())
print("kurt ", df_test1['PriceNew'].kurt())
```

```
Skew  0.7543716000992179
kurt  0.31018039291429167
```

[23]:
```
df_train.head()
```

[23]:

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	1.75
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	12.50
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	4.50
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	6.00
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	17.74

[24]:
```
df_test.sample(5)
```

[24]:

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats
186	Hyundai Santro Xing GL	Ahmedabad	2007	78000	Petrol	Manual	First	0.0 kmpl	1086 CC	62 bhp	5.0
793	Maruti Swift VXI BSIII	Hyderabad	2008	81814	Petrol	Manual	First	16.1 kmpl	1298 CC	88.2 bhp	5.0
1099	Honda City 1.5 S MT	Hyderabad	2010	60268	Petrol	Manual	First	17.0 kmpl	1497 CC	118 bhp	5.0
1127	Hyundai Creta 1.6 CRDi SX	Jaipur	2015	65000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0
1219	Audi A4 2.0 TDI	Hyderabad	2011	64000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0

[25]:
```
print(df_train.shape)
print(df_test.shape)
```

```
(6019, 12)
(1234, 11)
```

[26]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1234 entries, 0 to 1233
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               1234 non-null   object
 1   Location           1234 non-null   object
 2   Year               1234 non-null   int64
 3   Kilometers_Driven  1234 non-null   int64
 4   Fuel_Type          1234 non-null   object
 5   Transmission       1234 non-null   object
 6   Owner_Type         1234 non-null   object
 7   Mileage            1234 non-null   object
 8   Engine             1224 non-null   object
 9   Power              1224 non-null   object
 10  Seats              1223 non-null   float64
dtypes: float64(1), int64(2), object(8)
memory usage: 106.2+ KB
```

[27]:
```
df_train.describe()
```

[27]:

	Year	Kilometers_Driven	Seats	Price
count	6019.000000	6.019000e+03	5977.000000	6019.000000
mean	2013.358199	5.873838e+04	5.278735	9.479468
std	3.269742	9.126884e+04	0.808840	11.187917
min	1998.000000	1.710000e+02	0.000000	0.440000
25%	2011.000000	3.400000e+04	5.000000	3.500000
50%	2014.000000	5.300000e+04	5.000000	5.640000
75%	2016.000000	7.300000e+04	5.000000	9.950000
max	2019.000000	6.500000e+06	10.000000	160.000000

1. Price column have outliers

[28]:
```
miss_percent = (df_train.isnull().sum() / len(df_train)) * 100
missing = pd.DataFrame({"percent":miss_percent, 'count':df_train.isnull().
 ↪sum()}).sort_values(by="percent", ascending=False)
missing.loc[missing['percent'] > 0]
```

[28]:

	percent	count
Seats	0.697790	42

```
Engine    0.598106     36
Power     0.598106     36
Mileage   0.033228      2
```

1. New_Price have more than 86% missing need to address, this column seems important to compare price between new car & used car price.
2. Mileage, Engine, Power, Seats have very few missing this can be addressed by filling mean, median or mode to avoid lossing data

[29]:
```
miss_percent = (df_test.isnull().sum() / len(df_test)) * 100
missing = pd.DataFrame({"percent":miss_percent, 'count':df_test.isnull().
 ↪sum()}).sort_values(by="percent", ascending=False)
missing.loc[missing['percent'] > 0]
```

[29]:
```
         percent  count
Seats   0.891410     11
Engine  0.810373     10
Power   0.810373     10
```

[30]:
```
df_train['brand_name'] = df_train['Name'].apply(lambda x: str(x).split(" ")[0])
df_test['brand_name'] = df_test['Name'].apply(lambda x: str(x).split(" ")[0])
```

1. created a new column as 'brand_name'

[31]:
```
df_train.drop(columns=["Name"], axis=1, inplace=True)
df_test.drop(columns=["Name"], axis=1, inplace=True)
```

1. Dropped the 'Name' column from both train & test data

[32]:
```
#df_train.loc[df_train['brand_name'] == 'Maruti']['Seats'].mode()[0]
def fill_na_with_mode(ds, brandname):
  fill_value = ds.loc[ds['brand_name'] == brandname]['Seats'].mode()[0]
  condit = ((ds['brand_name'] == brandname) & (ds['Seats'].isnull()))
  ds.loc[condit, 'Seats'] = ds.loc[condit, 'Seats'].fillna(fill_value)
```

[33]:
```
car_brand =␣
 ↪['Maruti','Hyundai','BMW','Fiat','Land','Ford','Toyota','Honda','Skoda','Mahindra']
for c in car_brand:
  fill_na_with_mode(df_train, c)
  fill_na_with_mode(df_test, c)
```

1. Replaced all missing values in seats with mode of the specified brand name

[34]:
```
import re

df_train['Mileage_upd'] = df_train['Mileage'].apply(lambda x: re.sub(r'(\d+\.
 ↪\d+)\s(kmpl|km\/kg)', r'\1', str(x)))
```

```
df_train['Engine_upd'] = df_train['Engine'].apply(lambda x: re.
 ↪sub(r'(\d+)\s(CC)', r'\1', str(x)))
df_train['Power_upd'] = df_train['Power'].apply(lambda x: re.sub(r'(\d+\.?\d+?
 ↪)\s(bhp)', r'\1', str(x)))

df_test['Mileage_upd'] = df_test['Mileage'].apply(lambda x: re.sub(r'(\d+\.
 ↪\d+)\s(kmpl|km\/kg)', r'\1', str(x)))
df_test['Engine_upd'] = df_test['Engine'].apply(lambda x: re.
 ↪sub(r'(\d+)\s(CC)', r'\1', str(x)))
df_test['Power_upd'] = df_test['Power'].apply(lambda x: re.sub(r'(\d+\.?\d+?
 ↪)\s(bhp)', r'\1', str(x)))
```

1. Removed the km/kg & km/l from mileage to make as numeric column
2. removed the 'CC' and 'bhp' from engine & power columns to change as numeric

[35]:
```
df_train['Mileage_upd'] = pd.to_numeric(df_train['Mileage_upd'],␣
 ↪errors='coerce')
df_train['Engine_upd'] = pd.to_numeric(df_train['Engine_upd'], errors='coerce')
df_train['Power_upd'] = pd.to_numeric(df_train['Power_upd'], errors='coerce')

df_test['Mileage_upd'] = pd.to_numeric(df_test['Mileage_upd'], errors='coerce')
df_test['Engine_upd'] = pd.to_numeric(df_test['Engine_upd'], errors='coerce')
df_test['Power_upd'] = pd.to_numeric(df_test['Power_upd'], errors='coerce')
```

1. converted the 3 columns to float

[36]:
```
df_train.drop(columns=['Mileage', 'Engine', 'Power'], inplace=True)
df_test.drop(columns=['Mileage', 'Engine', 'Power'], inplace=True)
```

1. Removed the mileage,engine,power columns with updated columns

[37]:
```
df_train.drop(df_train[df_train['brand_name'] == 'Smart'].index, axis=0,␣
 ↪inplace=True)
df_test.drop(df_test[df_test['brand_name'] == 'Hindustan'].index, axis=0,␣
 ↪inplace=True)
```

1. Removed 1 row with unique brand having null value for Power.

[38]:
```
#Function to replace na value with mode of that specific brand
def fill_na_with_mode(ds, brandname, colname):
  fill_value = ds.loc[ds['brand_name'] == brandname][colname].mode()[0]
  condit = ((ds['brand_name'] == brandname) & (ds[colname].isnull()))
  ds.loc[condit, colname] = ds.loc[condit, colname].fillna(fill_value)
```

[39]:
```
miss_Mileage_col = df_train.loc[df_train['Mileage_upd'].isnull()]['brand_name'].
 ↪unique()
miss_Engine_col = df_train.loc[df_train['Engine_upd'].isnull()]['brand_name'].
 ↪unique()
```

```
miss_Power_col = df_train.loc[df_train['Power_upd'].isnull()]['brand_name'].
 ↪unique()

for x in miss_Mileage_col:
  fill_na_with_mode(df_train, x, 'Mileage_upd')
for y in miss_Engine_col:
  fill_na_with_mode(df_train, y, 'Engine_upd')
for z in miss_Power_col:
  fill_na_with_mode(df_train, z, 'Power_upd')
```

[40]:
```
miss_ts_Mileage_col = df_test.loc[df_test['Mileage_upd'].
 ↪isnull()]['brand_name'].unique()
miss_ts_Engine_col = df_test.loc[df_test['Engine_upd'].isnull()]['brand_name'].
 ↪unique()
miss_ts_Power_col = df_test.loc[df_test['Power_upd'].isnull()]['brand_name'].
 ↪unique()

for x in miss_ts_Mileage_col:
  fill_na_with_mode(df_test, x, 'Mileage_upd')
for y in miss_ts_Engine_col:
  fill_na_with_mode(df_test, y, 'Engine_upd')
for z in miss_ts_Power_col:
  fill_na_with_mode(df_test, z, 'Power_upd')
```

[41]:
```
zero_mileage_col = df_train.loc[df_train['Mileage_upd'] == 0.0]['brand_name'].
 ↪unique()

for m in zero_mileage_col:
  fill_zero = df_train.loc[df_train['brand_name'] == m]['Mileage_upd'].mode()[0]
  m1 = ((df_train['brand_name'] == m) & (df_train['Mileage_upd'] == 0.0))
  df_train.loc[m1, 'Mileage_upd'] = fill_zero
```

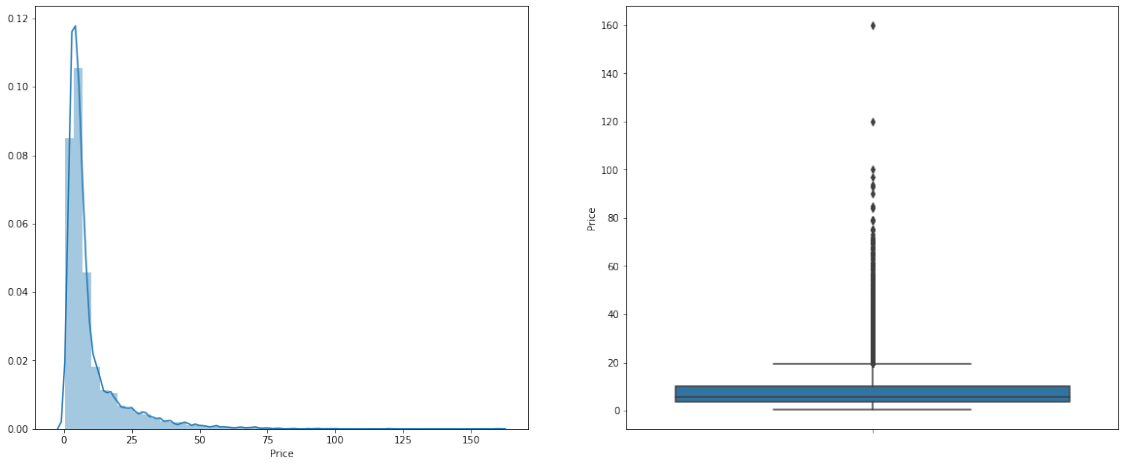
[42]:
```
zero_mileage_col2 = df_test.loc[df_test['Mileage_upd'] == 0.0]['brand_name'].
 ↪unique()

for m in zero_mileage_col2:
  fill_zero = df_test.loc[df_test['brand_name'] == m]['Mileage_upd'].mode()[0]
  m1 = ((df_test['brand_name'] == m) & (df_test['Mileage_upd'] == 0.0))
  df_test.loc[m1, 'Mileage_upd'] = fill_zero
```

1. Replaced 0.0 values with mode for column Mileage_upd

[43]:
```
m1 = (df_train['Seats'] == 0.0)
df_train.loc[m1, 'Seats'] = 5.0
```

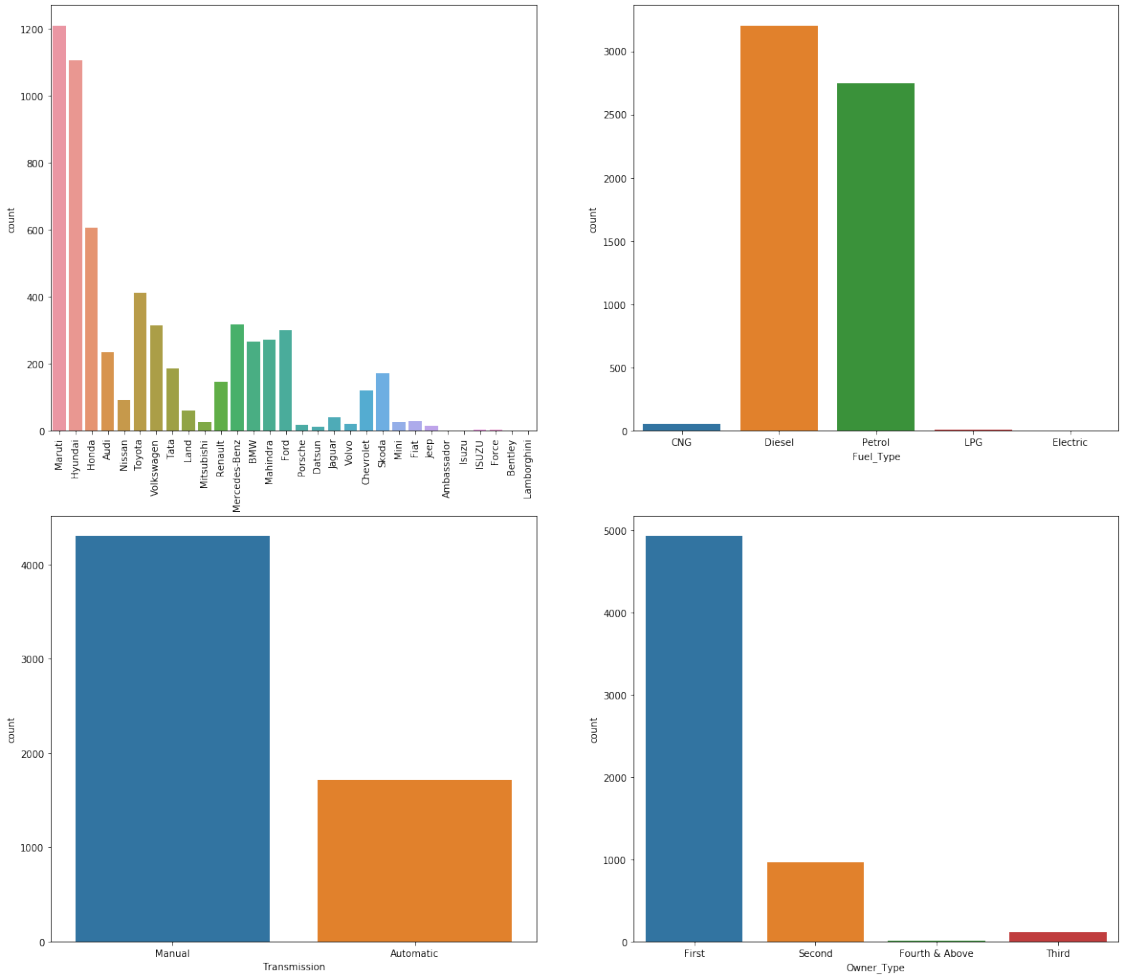
1. Replaced 1 zero value of seats with 5.0

[44]:
```
plt.figure(figsize=(20,8))
plt.subplot(1,2,1)
sns.distplot(df_train['Price'])

plt.subplot(1,2,2)
sns.boxplot(y=df_train['Price'])
plt.show()
```



1. Price column squweed in right, so label is not properly distributed

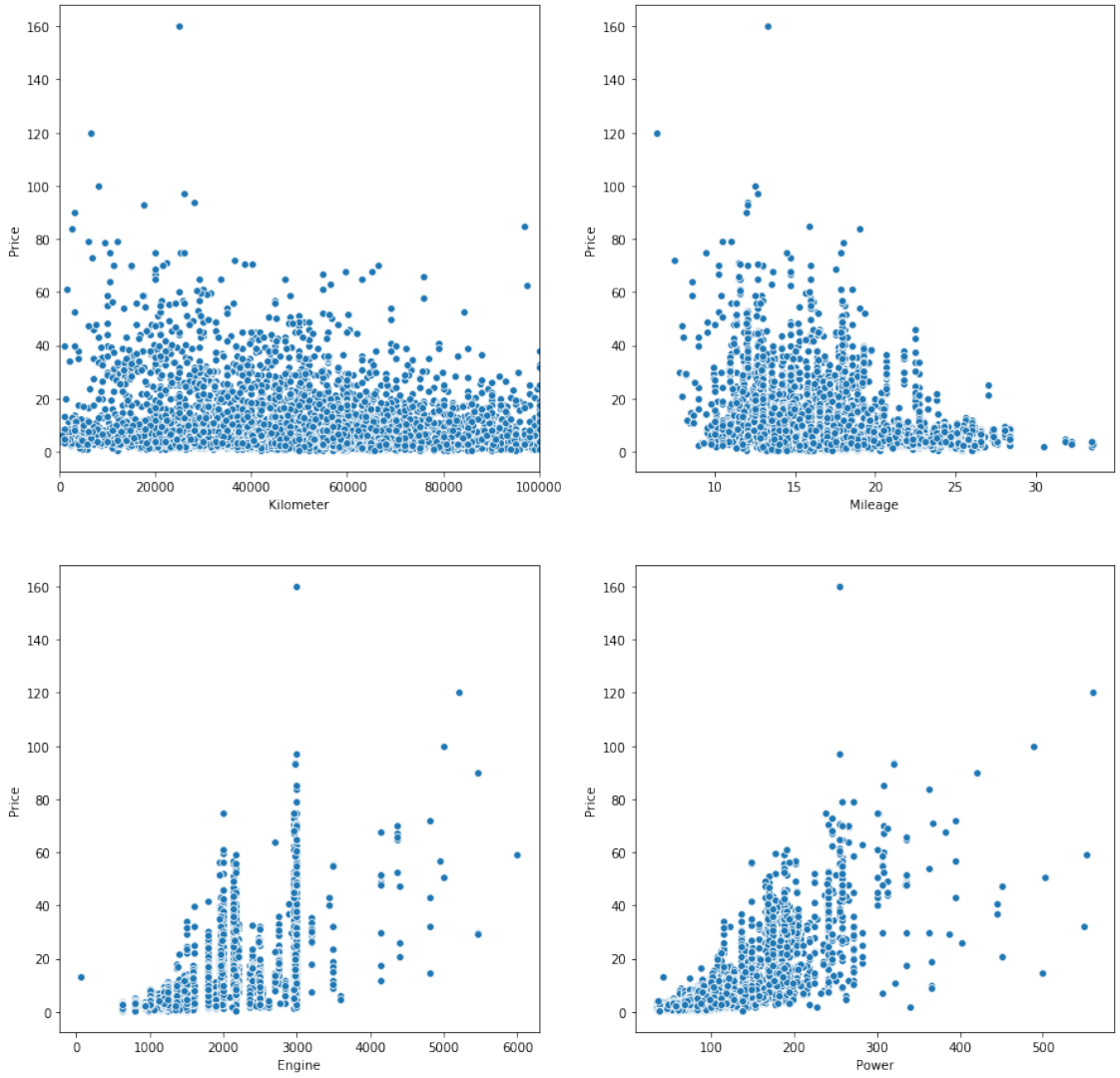
[45]:
```
fig = plt.figure(figsize=(20,18))
fig.subplots_adjust(hspace=0.2, wspace=0.2)
fig.add_subplot(2,2,1)
g1 = sns.countplot(x='brand_name', data=df_train)
loc,labels = plt.xticks()
g1.set_xticklabels(labels,rotation=90)
fig.add_subplot(2,2,2)
g2 = sns.countplot(x='Fuel_Type', data=df_train)
loc,labels = plt.xticks()
g2.set_xticklabels(labels,rotation=0)
fig.add_subplot(2,2,3)
g3 = sns.countplot(x='Transmission', data=df_train)
loc,labels = plt.xticks()
g3.set_xticklabels(labels,rotation=0)
fig.add_subplot(2,2,4)
g4 = sns.countplot(x='Owner_Type', data=df_train)
loc,labels = plt.xticks()
g4.set_xticklabels(labels,rotation=0)
plt.show()
```



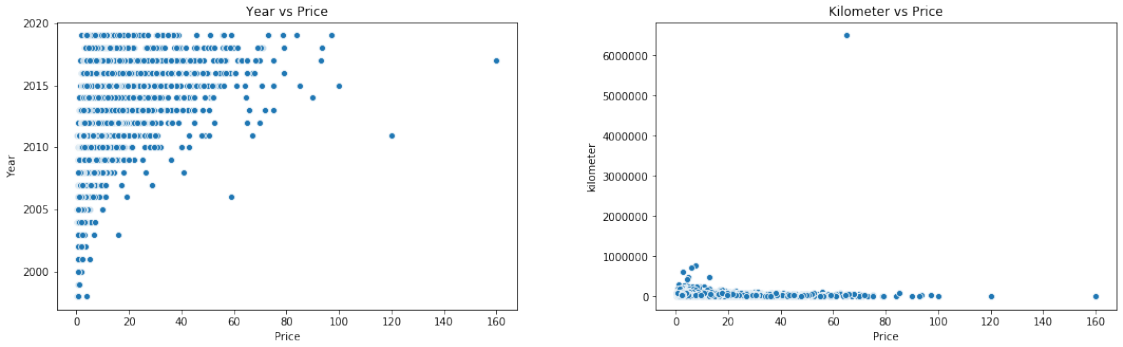
1. Maruti is leading car brand, fueltype both diesel & petrol are almost equal
2. Manual gear transmission is high, First owernership is high, also have second

[46]:
```
fig = plt.figure(figsize=(15,15))
fig.subplots_adjust(hspace=0.2, wspace=0.2)
ax1 = fig.add_subplot(2,2,1)
plt.xlim([0, 100000])
p1 = sns.scatterplot(x="Kilometers_Driven", y="Price", data=df_train)
loc, labels = plt.xticks()
ax1.set_xlabel('Kilometer')

ax2 = fig.add_subplot(2,2,2)
#plt.xlim([0, 100000])
p2 = sns.scatterplot(x="Mileage_upd", y="Price", data=df_train)
loc, labels = plt.xticks()
ax2.set_xlabel('Mileage')
```

```
ax3 = fig.add_subplot(2,2,3)
#plt.xlim([0, 100000])
p3 = sns.scatterplot(x="Engine_upd", y="Price", data=df_train)
loc, labels = plt.xticks()
ax3.set_xlabel('Engine')

ax4 = fig.add_subplot(2,2,4)
#plt.xlim([0, 100000])
p4 = sns.scatterplot(x="Power_upd", y="Price", data=df_train)
loc, labels = plt.xticks()
ax4.set_xlabel('Power')

plt.show()
```



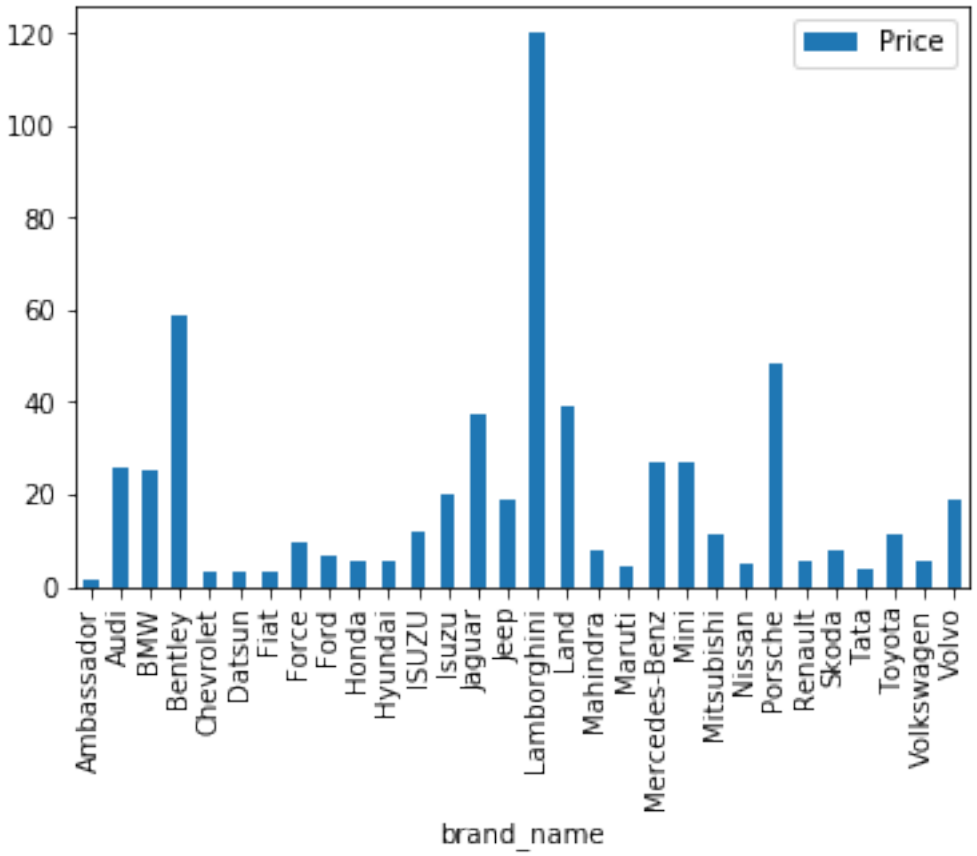
[47]:
```
fig = plt.figure(figsize=(18,5))
fig.subplots_adjust(hspace=0.3, wspace=0.3)

ax1 = fig.add_subplot(1,2,1)
sns.scatterplot(x='Price', y="Year", data=df_train)
ax1.set_xlabel('Price')
ax1.set_ylabel('Year')
ax1.set_title('Year vs Price')

ax2 = fig.add_subplot(1,2,2)
sns.scatterplot(x='Price', y='Kilometers_Driven', data=df_train)
ax2.set_ylabel('kilometer')
ax2.set_xlabel('Price')
ax2.set_title('Kilometer vs Price')
plt.show()
```



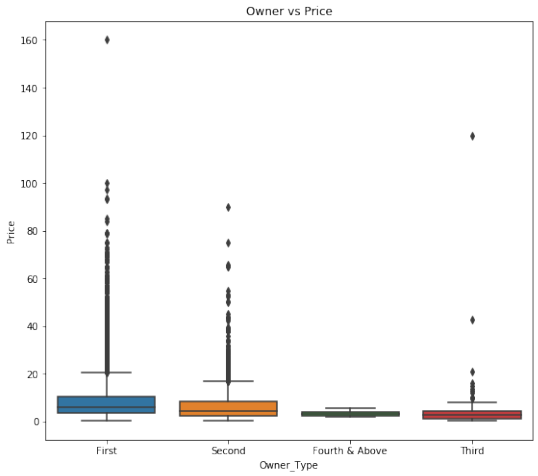
[48]:
```
df_train.drop(df_train[df_train['Kilometers_Driven'] >= 6500000].index, axis=0,␣
↪inplace=True)
```

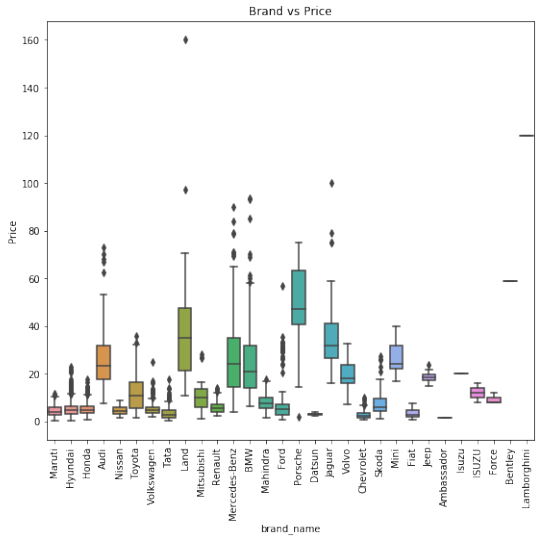
1. deleted a outlier row from training data.

[49]:
```
df_vis_1 = pd.DataFrame(df_train.groupby('brand_name')['Price'].mean())
df_vis_1.plot.bar()
plt.show()
```

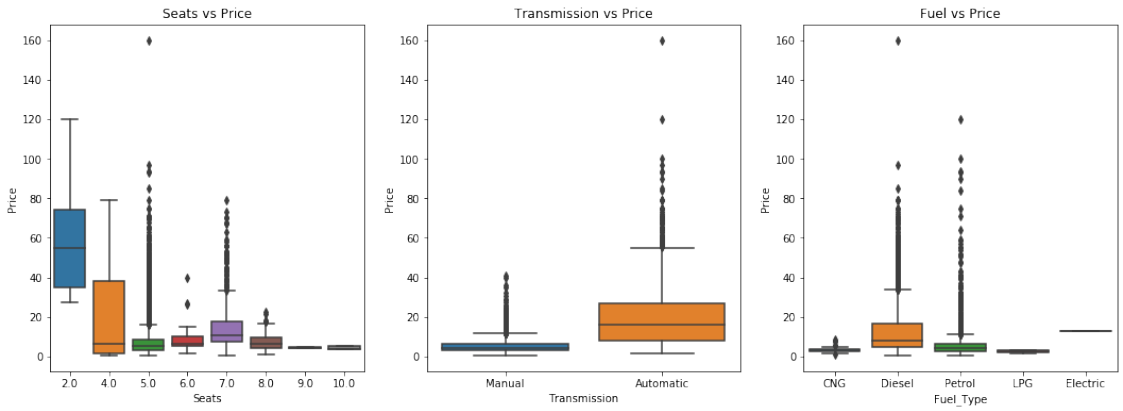


[50]:
```
fig = plt.figure(figsize=(20,8))
ax1 = fig.add_subplot(1,2,1)
sns.boxplot(x='Owner_Type', y='Price', data=df_train)
ax1.set_title('Owner vs Price')

ax2 = fig.add_subplot(1,2,2)
sns.boxplot(x='brand_name', y='Price', data=df_train)
loc,labels = plt.xticks()
ax2.set_xticklabels(labels, rotation=90)
ax2.set_title('Brand vs Price')
plt.show()
```





[51]:
```
fig = plt.figure(figsize=(18,6))
ax1 = fig.add_subplot(1,3,1)
sns.boxplot(x='Seats', y='Price', data=df_train)
ax1.set_title('Seats vs Price')

ax2 = fig.add_subplot(1,3,2)
sns.boxplot(x='Transmission', y='Price', data=df_train)
ax2.set_title('Transmission vs Price')

ax3 = fig.add_subplot(1,3,3)
sns.boxplot(x='Fuel_Type', y='Price', data=df_train)
ax3.set_title('Fuel vs Price')

plt.show()
```



[52]:
```
import datetime
now = datetime.datetime.now()
df_train['Year_upd'] = df_train['Year'].apply(lambda x : now.year - x)
df_test['Year_upd'] = df_test['Year'].apply(lambda x : now.year - x)
```

1. Added new column by getting the year count when it is bought

[53]:
```
df_train.drop(columns=['Year'], axis=1, inplace=True)
df_test.drop(columns=['Year'], axis=1, inplace=True)
```

1. dropped the 'year' column

[54]:
```
df_train.drop(columns=['Location'], axis=1, inplace=True)
df_test.drop(columns=['Location'], axis=1, inplace=True)
```

1. 'Location' column not needed for price prediction.

[55]:
```
df_train_norm = pd.get_dummies(df_train, drop_first=True)
df_test_norm = pd.get_dummies(df_test, drop_first=True)
```

1. Changed categorical variables to numerical data the both training and test set

[56]:
```
df_train_norm['Price_upd'] = np.log1p(df_train_norm['Price'].values)
```

1. add new column after taking logarithm for the dependent variable to avoid high skewness & kurtosis

[57]:
```
df_train_norm.drop(columns=['Price'], axis=1, inplace=True)
```

[58]:
```
df_train_X = df_train_norm.drop(columns=['Price_upd'], axis=1)
df_train_y = df_train_norm[['Price_upd']]
```

1. Seperated X & y values

[59]:
```
df_train_X = (df_train_X - df_train_X.mean())/df_train_X.std()
df_test_norm = (df_test_norm - df_test_norm.mean())/df_test_norm.std()
```

1. Normalized the train and test data

[60]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split(df_train_X, df_train_y,␣
 ↪test_size=0.22, random_state=1)
reg = lm.fit(X_train, y_train)
```

1. splitted train test split because test set dont have labels to verify accuracy

[61]:
```
y_predict = reg.predict(X_test)
y_predict
```

[61]:
```
array([[1.26552093],
       [1.60052778],
       [0.93832768],
       …,
       [1.74871724],
       [2.08694159],
       [3.47966153]])
```

[62]:
```
from sklearn.metrics import r2_score

r2_score(y_predict, y_test)
```

[62]:
```
0.90559821108742
```

[63]:
```
reg.score(X_test,y_test)
```

[63]:
```
0.9157487125391294
```

Thanks!