MSDS 692 Practicum I Predicting the Price of Used Cars Project

July 5, 2019

0.1 Introduction

Context

The dataset used for this project contains over 370,000 used cars information with 20 attributes scraped with Scrapy from Ebay-Kleinanzeigen. In this dataset, each entry represents an offering of used car in Germany. The used cars dataset was collected between 03-05-2016 and 04-07-2016. The link to this used cars Kaggle dataset is located at https://www.kaggle.com/orgesleka/used-cars-database.

Content

The content of the original dataset is in German. Google Translator was used to make the necessary translations to English.

The following is the description of the dataset 20 attributes:

dateCrawled: The date when this advertisement was first crawled, all field-values were obtained on this date.

name: "name" of the car.

seller: seller type – private or dealer.

offerType: Offer Type - offer or request.

price: the price in Euro on the advertisement to sell the car.

abtest: abtest category - test or control

vehicleType: vehicle body type - limousine, small car, station wagon, bus, cabrio, coupe, suv, other.

yearOfRegistration: At what year the car was first registered - the age of the car.

Transmission: Transmission Type - manual or automatic.

powerPS: Car Engine Power in PS

model: car model.

Kilometer: car mileage in kilometer.

monthOfRegistration: the month of the year the car was first registered.

fuelType: Fuel Type - gas, diesel, autogas, compressed natural gas, hybrid, other, or electric.

Brand: car brand

notRepairedDamage: Unrepaired Damage - yes or no.

dateCreated: The date the ad was created on Ebay-Kleinanzeigen.

nrOfPictures: number of pictures in the ad

postalCode: car seller postal code

lastSeen: when the crawler saw this ad last online

0.2 Required Libraries and dataset

In [1]:
```
# Load the pandas,numpy,matplotlib and other required libraries with an import stateme
import pandas as pd                # importing pandas as pd
import numpy as np                 # importing numpy as np
import matplotlib.pyplot as plt    #importing matplotlib as plt
import seaborn as sns              #importing seaborn as sns
%matplotlib inline
import datetime
from dateutil.relativedelta import relativedelta
from datetime import date
from datetime import time
```

In [2]:
```
# Load the dataset into a pandas dataframe
df = pd.read_csv('autos_english.csv', sep=',',low_memory=False, encoding ='latin-1')
```

In [38]:
```
# Display the first ten lines of the dataset
df.head(5)
```

Out[38]:

	Date_Crawled	Car_Name	Car_Seller_Type	Offer_Type
0	3/24/2016 11:52	Golf_3_1.6	private	offer
1	3/24/2016 10:58	A5_Sportback_2.7_Tdi	private	offer
2	3/14/2016 12:52	Jeep_Grand_Cherokee_"Overland"	private	offer
3	3/17/2016 16:54	GOLF_4_1_4__3TÜRER	private	offer
4	3/31/2016 17:25	Skoda_Fabia_1.4_TDI_PD_Classic	private	offer

	Car_Price	Abtest_Type	Vechicle_Type	Year_of_Car_Registration
0	480.0	test	NaN	1993.0
1	18300.0	test	coupe	2011.0
2	9800.0	test	suv	2004.0
3	1500.0	test	small car	2001.0
4	3600.0	test	small car	2008.0

	Car_Transmission_Type	Car_Engine_Power_PS	Car_Model	Car_Mileage_Kilometer
0	manual	0.0	golf	150000
1	manual	190.0	NaN	125000
2	automatic	163.0	grand	125000
3	manual	75.0	golf	150000
4	manual	69.0	fabia	90000

	Month_of_Car_Registration	Fuel_Type	Car_Brand	UnRepaired_Damage
0	0.0	gas	volkswagen	NaN
1	5.0	diesel	audi	yes
2	8.0	diesel	jeep	NaN
3	6.0	gas	volkswagen	no
4	7.0	diesel	skoda	no

	Date_Created	No_of_Pictures	Seller_Postal_Code	Date_LastSeen_Online
0	3/24/2016 0:00	0.0	70435.0	4/7/2016 3:16

```
1  3/24/2016 0:00            0.0            66954.0       4/7/2016 1:46
2  3/14/2016 0:00            0.0            90480.0       4/5/2016 12:47
3  3/17/2016 0:00            0.0            91074.0      3/17/2016 17:40
4  3/31/2016 0:00            0.0            60437.0       4/6/2016 10:17
```

In [4]:
```
df.keys()
```

Out[4]:
```
Index(['dateCrawled', 'name', 'seller', 'offerType', 'price', 'abtest',
       'vehicleType', 'yearOfRegistration', 'Transmission', 'powerPS', 'model',
       'kilometer', 'monthOfRegistration', 'fuelType', 'brand',
       'notRepairedDamage', 'dateCreated', 'nrOfPictures', 'postalCode',
       'lastSeen'],
      dtype='object')
```

In [5]:
```
# Display the last few lines of the dataset
df.tail(10)
```

Out[5]:

	dateCrawled	name	\
371530	4/2/2016 20:37	Bmw_320_D_DPF_Touring_!!!	
371531	3/9/2016 13:37	Alfa_Romeo_159_Jtdm_1.9_150_ps_13_600_km_top_voll	
371532	3/19/2016 19:53	turbo_defekt	
371533	3/27/2016 20:36	Opel_Zafira_1.6_Elegance_TÜV_12/16	
371534	3/21/2016 9:50	Mitsubishi_Cold	
371535	3/14/2016 17:48	Suche_t4___vito_ab_6_sitze	
371536	3/5/2016 19:56	Smart_smart_leistungssteigerung_100ps	
371537	3/19/2016 18:57	Volkswagen_Multivan_T4_TDI_7DC_UY2	
371538	3/20/2016 19:41	VW_Golf_Kombi_1_9l_TDI	
371539	3/7/2016 19:39	BMW_M135i_vollausgestattet_NP_52.720____Euro	

	seller	offerType	price	abtest	vehicleType	\
371530	private	offer	3999.0	test	station wagon	
371531	private	offer	5250.0	control	NaN	
371532	private	offer	3200.0	control	limousine	
371533	private	offer	1150.0	control	bus	
371534	private	offer	0.0	control	NaN	
371535	private	offer	2200.0	test	NaN	
371536	private	offer	1199.0	test	cabrio	
371537	private	offer	9200.0	test	bus	
371538	private	offer	3400.0	test	station wagon	
371539	private	offer	28990.0	control	limousine	

	yearOfRegistration	Transmission	powerPS	model	kilometer	\
371530	2005.0	manual	3.0	3er	150000	
371531	2016.0	automatic	150.0	159	150000	
371532	2004.0	manual	225.0	leon	150000	
371533	2000.0	manual	0.0	zafira	150000	
371534	2005.0	manual	0.0	colt	150000	
371535	2005.0	NaN	0.0	NaN	20000	
371536	2000.0	automatic	101.0	fortwo	125000	

```
371537              1996.0       manual    102.0  transporter    150000
371538              2002.0       manual    100.0         golf    150000
371539              2013.0       manual    320.0      m_reihe     50000

        monthOfRegistration fuelType           brand notRepairedDamage  \
371530                  5.0   diesel             bmw                no
371531                 12.0      NaN      alfa_romeo                no
371532                  5.0      gas            seat               yes
371533                  3.0      gas            opel                no
371534                  7.0      gas      mitsubishi               yes
371535                  1.0      NaN  sonstige_autos               NaN
371536                  3.0      gas           smart                no
371537                  3.0   diesel      volkswagen                no
371538                  6.0   diesel      volkswagen               NaN
371539                  8.0      gas             bmw                no

           dateCreated  nrOfPictures  postalCode         lastSeen
371530   4/2/2016 0:00           0.0     81825.0   4/6/2016 20:47
371531   3/9/2016 0:00           0.0     51371.0   3/13/2016 1:44
371532  3/19/2016 0:00           0.0     96465.0  3/19/2016 20:44
371533  3/27/2016 0:00           0.0     26624.0  3/29/2016 10:17
371534  3/21/2016 0:00           0.0      2694.0  3/21/2016 10:42
371535  3/14/2016 0:00           0.0     39576.0    4/6/2016 0:46
371536   3/5/2016 0:00           0.0     26135.0  3/11/2016 18:17
371537  3/19/2016 0:00           0.0     87439.0    4/7/2016 7:15
371538  3/20/2016 0:00           0.0     40764.0  3/24/2016 12:45
371539   3/7/2016 0:00           0.0     73326.0   3/22/2016 3:17
```

0.3 Knowing the dataset

In [6]:
```
# Display summary of the dataframe including the data types of each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 371540 entries, 0 to 371539
Data columns (total 20 columns):
dateCrawled            371539 non-null object
name                   371539 non-null object
seller                 371538 non-null object
offerType              371538 non-null object
price                  371538 non-null float64
abtest                 371538 non-null object
vehicleType            333669 non-null object
yearOfRegistration     371537 non-null float64
Transmission           351329 non-null object
powerPS                371538 non-null float64
model                  351054 non-null object
kilometer              371538 non-null object
```

```
monthOfRegistration    371537 non-null float64
fuelType               338151 non-null object
brand                  371537 non-null object
notRepairedDamage      299477 non-null object
dateCreated            371537 non-null object
nrOfPictures           371537 non-null float64
postalCode             371537 non-null float64
lastSeen               371537 non-null object
dtypes: float64(6), object(14)
memory usage: 56.7+ MB
```

```
In [7]: # Checking the unique values
        print(df.nunique())
```

```
dateCrawled             15623
name                   233534
seller                      3
offerType                   3
price                    5597
abtest                      3
vehicleType                 9
yearOfRegistration        155
Transmission                3
powerPS                   794
model                     252
kilometer                  14
monthOfRegistration        13
fuelType                    7
brand                      40
notRepairedDamage           2
dateCreated               114
nrOfPictures                1
postalCode               8150
lastSeen                18705
dtype: int64
```

```
In [8]: #Frequency Counts of offerType
        df['offerType'].value_counts()
```

```
Out[8]: offer      371525
        request        12
        150000          1
        Name: offerType, dtype: int64
```

```
In [9]: # Frequency Counts of seller Types
        df['seller'].value_counts()
```

```
Out[9]: private    371534
        dealer          3
        golf            1
        Name: seller, dtype: int64

In [10]: # Frequency counts of abtest
         df['abtest'].value_counts()

Out[10]: test       192591
         control    178946
         gas             1
         Name: abtest, dtype: int64

In [11]: # Frequency counts of vehicleType
         df['vehicleType'].value_counts()

Out[11]: limousine        95896
         small car        80026
         station wagon    67564
         bus              30202
         cabrio           22899
         coupe            19016
         suv              14708
         other             3357
         volkswagen           1
         Name: vehicleType, dtype: int64

In [12]: # Frequency counts of yearofRegistration
         df['yearOfRegistration'].value_counts().nlargest(20)

Out[12]: 2000.0    24552
         1999.0    22768
         2005.0    22316
         2006.0    20232
         2001.0    20218
         2003.0    19873
         2004.0    19746
         2002.0    19189
         1998.0    17951
         2007.0    17673
         2008.0    16175
         2009.0    15607
         1997.0    14707
         2010.0    12354
         2011.0    12068
         1996.0    10886
         2017.0    10546
         2016.0     9859
         1995.0     9658
```

```
2012.0     9418
Name: yearOfRegistration, dtype: int64
```

```
In [13]: # Frequency counts of Transmission Types
         df['Transmission'].value_counts()
```

```
Out[13]: manual            274219
         automatic          77109
         3/25/2016 0:00         1
         Name: Transmission, dtype: int64
```

```
In [14]: # Frequency counts of fuelType
         df['fuelType'].value_counts()
```

```
Out[14]: gas                       223863
         diesel                    107748
         autogas                     5378
         compressed natural gas       571
         hybrid                       279
         other                        208
         electric                     104
         Name: fuelType, dtype: int64
```

```
In [15]: # Frequency counts of notRepairedDamage
         df['notRepairedDamage'].value_counts()
```

```
Out[15]: no     263189
         yes     36288
         Name: notRepairedDamage, dtype: int64
```

```
In [16]: # Find the number of non-NA/null value across each column
         df.count()
```

```
Out[16]: dateCrawled            371539
         name                   371539
         seller                 371538
         offerType              371538
         price                  371538
         abtest                 371538
         vehicleType            333669
         yearOfRegistration     371537
         Transmission           351329
         powerPS                371538
         model                  351054
         kilometer              371538
         monthOfRegistration    371537
         fuelType               338151
         brand                  371537
         notRepairedDamage      299477
```

```
dateCreated            371537
nrOfPictures           371537
postalCode             371537
lastSeen               371537
dtype: int64
```

```
In [17]: # Frequency Counts of kilometer values
         df['kilometer'].value_counts()
```

```
Out[17]: 150000            240802
         125000             38067
         100000             15920
         90000              12524
         80000              11053
         70000               9773
         60000               8669
         50000               7616
         5000                7070
         40000               6377
         30000               6041
         20000               5676
         10000               1949
         3/30/2016 0:44         1
         Name: kilometer, dtype: int64
```

```
In [18]: # Display the column names
         df.columns
```

```
Out[18]: Index(['dateCrawled', 'name', 'seller', 'offerType', 'price', 'abtest',
                'vehicleType', 'yearOfRegistration', 'Transmission', 'powerPS', 'model',
                'kilometer', 'monthOfRegistration', 'fuelType', 'brand',
                'notRepairedDamage', 'dateCreated', 'nrOfPictures', 'postalCode',
                'lastSeen'],
               dtype='object')
```

```
In [19]: # Find the number of NaN values in each column
         autos_missing_values = df.isnull().sum()
```

```
In [20]: autos_missing_values
```

```
Out[20]: dateCrawled                1
         name                       1
         seller                     2
         offerType                  2
         price                      2
         abtest                     2
         vehicleType            37871
         yearOfRegistration         3
         Transmission           20211
```

```
powerPS                  2
model                20486
kilometer                2
monthOfRegistration      3
fuelType             33389
brand                    3
notRepairedDamage    72063
dateCreated              3
nrOfPictures             3
postalCode               3
lastSeen                 3
dtype: int64
```

In [21]:
```
# Find the number of rows and columns in the dataset
# Check the dataset size
df.shape
```

Out[21]: (371540, 20)

In [22]:
```
# Descriptive or Summary statistics of numeric columns
df.describe()
```

Out[22]:

	price	yearOfRegistration	powerPS	monthOfRegistration	\
count	3.715380e+05	371537.000000	371538.000000	371537.000000	
mean	1.729544e+04	2004.577883	115.548840	5.734473	
std	3.587905e+06	92.865496	192.137238	3.712383	
min	0.000000e+00	1000.000000	0.000000	0.000000	
25%	1.150000e+03	1999.000000	70.000000	3.000000	
50%	2.950000e+03	2003.000000	105.000000	6.000000	
75%	7.200000e+03	2008.000000	150.000000	9.000000	
max	2.147484e+09	9999.000000	20000.000000	12.000000	

	nrOfPictures	postalCode
count	371537.0	371537.000000
mean	0.0	50820.666402
std	0.0	25799.080292
min	0.0	1067.000000
25%	0.0	30459.000000
50%	0.0	49610.000000
75%	0.0	71546.000000
max	0.0	99998.000000

In [23]:
```
# Summary statistics of character or non-numeric columns
df.describe(include=['object'])
```

Out[23]:

	dateCrawled	name	seller	offerType	abtest	vehicleType	\
count	371539	371539	371538	371538	371538	333669	
unique	15623	233534	3	3	3	9	
top	3/5/2016 14:25	Ford_Fiesta	private	offer	test	limousine	

```
freq                68          657    371534    371525  192591       95896

       Transmission   model kilometer fuelType       brand notRepairedDamage  \
count        351329  351054    371538   338151      371537             299477
unique            3     252        14        7          40                  2
top          manual    golf    150000      gas  volkswagen                 no
freq         274219   30070    240802   223863       79640             263189

          dateCreated       lastSeen
count          371537         371537
unique            114          18705
top     4/3/2016 0:00  4/7/2016 6:45
freq            14451            708
```

0.4 Create meaningful Column names

To change the column names using rename function in Pandas, one needs to specify a mapper, a dictionary with old name as keys and new name as values. We will also use inplace=True to change column names in place.

In [24]:
```
# Change column names using rename function

df.rename(columns={'dateCrawled':'Date_Crawled',
                   'name':'Car_Name',
                   'seller':'Car_Seller_Type',
                   'offerType':'Offer_Type',
                   'price':'Car_Price',
                   'abtest':'Abtest_Type',
                   'vehicleType':'Vechicle_Type',
                   'yearOfRegistration':'Year_of_Car_Registration',
                   'Transmission':'Car_Transmission_Type',
                   'powerPS':'Car_Engine_Power_PS',
                   'model':'Car_Model',
                   'kilometer':'Car_Mileage_Kilometer',
                   'monthOfRegistration':'Month_of_Car_Registration',
                   'fuelType':'Fuel_Type',
                   'brand':'Car_Brand',
                   'notRepairedDamage': 'UnRepaired_Damage',
                   'dateCreated': 'Date_Created',
                   'nrOfPictures': 'No_of_Pictures',
                   'postalCode': 'Seller_Postal_Code',
                   'lastSeen': 'Date_LastSeen_Online'},
                   inplace=True)
```

In [25]:
```
# Display the new column names
df.columns
```

```
Out[25]: Index(['Date_Crawled', 'Car_Name', 'Car_Seller_Type', 'Offer_Type',
                'Car_Price', 'Abtest_Type', 'Vechicle_Type', 'Year_of_Car_Registration',
                'Car_Transmission_Type', 'Car_Engine_Power_PS', 'Car_Model',
                'Car_Mileage_Kilometer', 'Month_of_Car_Registration', 'Fuel_Type',
                'Car_Brand', 'UnRepaired_Damage', 'Date_Created', 'No_of_Pictures',
                'Seller_Postal_Code', 'Date_LastSeen_Online'],
               dtype='object')
```

```
In [26]: # Display the first five lines
         df.head()
```

```
Out[26]:       Date_Crawled                         Car_Name Car_Seller_Type Offer_Type  \
         0  3/24/2016 11:52                       Golf_3_1.6         private      offer
         1  3/24/2016 10:58             A5_Sportback_2.7_Tdi         private      offer
         2  3/14/2016 12:52  Jeep_Grand_Cherokee_"Overland"         private      offer
         3  3/17/2016 16:54               GOLF_4_1_4__3TÜRER         private      offer
         4  3/31/2016 17:25  Skoda_Fabia_1.4_TDI_PD_Classic         private      offer

            Car_Price Abtest_Type Vechicle_Type  Year_of_Car_Registration  \
         0      480.0        test           NaN                   1993.0
         1    18300.0        test         coupe                   2011.0
         2     9800.0        test           suv                   2004.0
         3     1500.0        test     small car                   2001.0
         4     3600.0        test     small car                   2008.0

           Car_Transmission_Type  Car_Engine_Power_PS Car_Model Car_Mileage_Kilometer  \
         0                manual                  0.0      golf                150000
         1                manual                190.0       NaN                125000
         2             automatic                163.0     grand                125000
         3                manual                 75.0      golf                150000
         4                manual                 69.0     fabia                 90000

            Month_of_Car_Registration Fuel_Type   Car_Brand UnRepaired_Damage  \
         0                        0.0       gas  volkswagen              NaN
         1                        5.0    diesel        audi              yes
         2                        8.0    diesel        jeep              NaN
         3                        6.0       gas  volkswagen               no
         4                        7.0    diesel       skoda               no

              Date_Created  No_of_Pictures  Seller_Postal_Code Date_LastSeen_Online
         0  3/24/2016 0:00             0.0             70435.0        4/7/2016 3:16
         1  3/24/2016 0:00             0.0             66954.0        4/7/2016 1:46
         2  3/14/2016 0:00             0.0             90480.0       4/5/2016 12:47
         3  3/17/2016 0:00             0.0             91074.0      3/17/2016 17:40
         4  3/31/2016 0:00             0.0             60437.0       4/6/2016 10:17
```

```
In [27]: # Display data type of column
         print(df.dtypes)
```

```
Date_Crawled                 object
Car_Name                     object
Car_Seller_Type              object
Offer_Type                   object
Car_Price                   float64
Abtest_Type                  object
Vechicle_Type                object
Year_of_Car_Registration    float64
Car_Transmission_Type        object
Car_Engine_Power_PS         float64
Car_Model                    object
Car_Mileage_Kilometer        object
Month_of_Car_Registration   float64
Fuel_Type                    object
Car_Brand                    object
UnRepaired_Damage            object
Date_Created                 object
No_of_Pictures              float64
Seller_Postal_Code          float64
Date_LastSeen_Online         object
dtype: object
```

```
In [28]: # Frequency Counts of Date_Crawled
         df['Date_Crawled'].value_counts().nlargest(20)
```

```
Out[28]: 3/5/2016 14:25     68
         3/5/2016 14:26     62
         3/5/2016 17:49     58
         3/5/2016 15:48     58
         3/20/2016 11:50    55
         3/5/2016 14:49     55
         3/16/2016 18:49    55
         3/21/2016 16:50    55
         3/27/2016 15:50    55
         3/29/2016 21:50    55
         3/20/2016 18:50    54
         3/7/2016 16:50     54
         3/26/2016 20:50    54
         3/29/2016 15:49    54
         3/26/2016 21:50    53
         3/23/2016 13:50    53
         3/12/2016 17:49    53
         3/7/2016 22:50     53
         3/5/2016 14:30     53
         3/30/2016 16:49    52
         Name: Date_Crawled, dtype: int64
```

```
In [29]: # Display data type of column
```

```
print(df.dtypes)
```

```
Date_Crawled                 object
Car_Name                     object
Car_Seller_Type              object
Offer_Type                   object
Car_Price                   float64
Abtest_Type                  object
Vechicle_Type                object
Year_of_Car_Registration    float64
Car_Transmission_Type        object
Car_Engine_Power_PS         float64
Car_Model                    object
Car_Mileage_Kilometer        object
Month_of_Car_Registration   float64
Fuel_Type                    object
Car_Brand                    object
UnRepaired_Damage            object
Date_Created                 object
No_of_Pictures              float64
Seller_Postal_Code          float64
Date_LastSeen_Online         object
dtype: object
```

```
In [30]: # Frequency Counts of Car Seller Type
         df['Car_Seller_Type'].value_counts()
```

```
Out[30]: private    371534
         dealer          3
         golf            1
         Name: Car_Seller_Type, dtype: int64
```

```
In [31]: # Make a copy of the dataframe
         autos_df = df.copy()
```

```
In [32]: # Display the first five lines of the dataframe
         autos_df.head()
```

```
Out[32]:       Date_Crawled                        Car_Name Car_Seller_Type Offer_Type  \
         0  3/24/2016 11:52                     Golf_3_1.6         private      offer
         1  3/24/2016 10:58           A5_Sportback_2.7_Tdi         private      offer
         2  3/14/2016 12:52  Jeep_Grand_Cherokee_"Overland"         private      offer
         3  3/17/2016 16:54             GOLF_4_1_4__3TÜRER         private      offer
         4  3/31/2016 17:25  Skoda_Fabia_1.4_TDI_PD_Classic         private      offer

            Car_Price Abtest_Type Vechicle_Type  Year_of_Car_Registration  \
         0      480.0        test           NaN                    1993.0
         1    18300.0        test         coupe                    2011.0
```

```
2     9800.0        test           suv                    2004.0
3     1500.0        test     small car                    2001.0
4     3600.0        test     small car                    2008.0

  Car_Transmission_Type  Car_Engine_Power_PS Car_Model Car_Mileage_Kilometer  \
0                manual                  0.0      golf                150000
1                manual                190.0       NaN                125000
2             automatic                163.0     grand                125000
3                manual                 75.0      golf                150000
4                manual                 69.0     fabia                 90000

   Month_of_Car_Registration Fuel_Type   Car_Brand UnRepaired_Damage  \
0                        0.0       gas  volkswagen               NaN
1                        5.0    diesel        audi               yes
2                        8.0    diesel        jeep               NaN
3                        6.0       gas  volkswagen                no
4                        7.0    diesel       skoda                no

     Date_Created  No_of_Pictures  Seller_Postal_Code Date_LastSeen_Online
0  3/24/2016 0:00             0.0             70435.0        4/7/2016 3:16
1  3/24/2016 0:00             0.0             66954.0        4/7/2016 1:46
2  3/14/2016 0:00             0.0             90480.0       4/5/2016 12:47
3  3/17/2016 0:00             0.0             91074.0      3/17/2016 17:40
4  3/31/2016 0:00             0.0             60437.0       4/6/2016 10:17
```

```
In [33]: # Frequency Counts of Car Seller Type
         autos_df['Car_Seller_Type'].value_counts()

Out[33]: private    371534
         dealer          3
         golf            1
         Name: Car_Seller_Type, dtype: int64
```

The above Car Seller Type output shows almost all the car listings or advertisements in this dataset were from private sellers. I would keep this column for private sellers only, hence, I would remove the (3) dealer and (1)golf entries.

```
In [34]: # Keep the Car_Seller_Type column only for private seller type
         autos_df = autos_df[autos_df.Car_Seller_Type == 'private']

In [35]: # Frequency Counts of Car Seller Type
         autos_df['Car_Seller_Type'].value_counts()

Out[35]: private    371534
         Name: Car_Seller_Type, dtype: int64

In [36]: # Frequency Counts of Offer Type
         autos_df['Offer_Type'].value_counts()
```

```
Out[36]: offer      371522
         request        12
         Name: Offer_Type, dtype: int64
```

The above Offer Type output shows almost all the Offer Type in this dataset are of the type offer. I would retain this column for offer only,hence, I would remove the (12) request offer entries.

```
In [37]: # Keep the Offer_Type column only for offer
         # Remove request offer type from the column
         autos_df = autos_df[autos_df.Offer_Type == 'offer']
```

```
In [39]: # Frequency Counts of Offer Type
         autos_df['Offer_Type'].value_counts()
```

```
Out[39]: offer    371522
         Name: Offer_Type, dtype: int64
```

```
In [40]: # https://stackoverflow.com/questions/35364601/group-by-and-find-top-n-value-counts-p
         # Frequency Counts of Car_Price in Euros
         autos_df['Car_Price'].value_counts().nlargest(20)
```

```
Out[40]: 0.0       10772
         500.0      5670
         1500.0     5394
         1000.0     4649
         1200.0     4594
         2500.0     4438
         600.0      3819
         3500.0     3792
         800.0      3784
         2000.0     3431
         999.0      3364
         750.0      3203
         650.0      3151
         4500.0     3053
         850.0      2946
         2200.0     2936
         700.0      2936
         1800.0     2886
         900.0      2874
         950.0      2793
         Name: Car_Price, dtype: int64
```

```
In [41]: # https://stackoverflow.com/questions/35364601/group-by-and-find-top-n-value-counts-p
         # Frequency Counts of Car_Price in Euros
         autos_df['Car_Price'].value_counts().nsmallest(10)
```

```
Out[41]: 19770.0     1
         69950.0     1
```

```
4158.0      1
1233.0      1
17989.0     1
78964.0     1
2792.0      1
32250.0     1
517895.0    1
20099.0     1
Name: Car_Price, dtype: int64
```

The above results provide the top 20 prices for the cars in this dataset. It is worth noting that 0.0 seems to actually be be missing values for this attribute. I would be dealing with missing values in another section.

```
In [42]: # Frequency Counts of Abtest Type
         autos_df['Abtest_Type'].value_counts()

Out[42]: test       192586
         control    178936
         Name: Abtest_Type, dtype: int64
```

The above output shows the number of test and control Abtest type.

```
In [43]: # https://stackoverflow.com/questions/35364601/group-by-and-find-top-n-value-counts-p
         # Frequency Counts of Vechicle_Type
         autos_df['Vechicle_Type'].value_counts().nlargest(10)

Out[43]: limousine        95895
         small car        80023
         station wagon    67563
         bus              30200
         cabrio           22899
         coupe            19015
         suv              14708
         other             3357
         Name: Vechicle_Type, dtype: int64
```

Limousine,small car, and station wagon are the top three Vechicle Body Types that are being offered for sales.

```
In [44]: autos_df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 371522 entries, 0 to 371539
Data columns (total 20 columns):
Date_Crawled                371522 non-null object
Car_Name                    371522 non-null object
Car_Seller_Type             371522 non-null object
Offer_Type                  371522 non-null object
```

```
Car_Price                    371522 non-null float64
Abtest_Type                  371522 non-null object
Vechicle_Type                333660 non-null object
Year_of_Car_Registration     371522 non-null float64
Car_Transmission_Type        351319 non-null object
Car_Engine_Power_PS          371522 non-null float64
Car_Model                    351041 non-null object
Car_Mileage_Kilometer        371522 non-null object
Month_of_Car_Registration    371522 non-null float64
Fuel_Type                    338143 non-null object
Car_Brand                    371522 non-null object
UnRepaired_Damage            299469 non-null object
Date_Created                 371522 non-null object
No_of_Pictures               371522 non-null float64
Seller_Postal_Code           371522 non-null float64
Date_LastSeen_Online         371522 non-null object
dtypes: float64(6), object(14)
memory usage: 59.5+ MB
```

```
In [45]: # https://stackoverflow.com/questions/35364601/group-by-and-find-top-n-value-counts-p
         # Frequency Counts of Year_of_Car_Registration
         autos_df['Year_of_Car_Registration'].value_counts().nlargest(10)
```

```
Out[45]: 2000.0    24549
         1999.0    22768
         2005.0    22312
         2006.0    20231
         2001.0    20218
         2003.0    19873
         2004.0    19745
         2002.0    19189
         1998.0    17950
         2007.0    17673
         Name: Year_of_Car_Registration, dtype: int64
```

```
In [46]: # https://stackoverflow.com/questions/35364601/group-by-and-find-top-n-value-counts-p
         # Frequency Counts of Year_of_Car_Registration
         autos_df['Year_of_Car_Registration'].value_counts().nsmallest(10)
```

```
Out[46]: 1255.0    1
         1253.0    1
         2066.0    1
         9996.0    1
         1039.0    1
         8455.0    1
         4800.0    1
         1200.0    1
         7800.0    1
```

```
9450.0    1
Name: Year_of_Car_Registration, dtype: int64
```

```
In [47]: # https://stackoverflow.com/questions/35364601/group-by-and-find-top-n-value-counts-p
         # Frequency Counts of Car_Transmission_Type
         autos_df['Car_Transmission_Type'].value_counts()
```

```
Out[47]: manual       274211
         automatic     77108
         Name: Car_Transmission_Type, dtype: int64
```

It is interesting to discover that there is more market for manual transmission cars in Germany or Europe in general. The above result shows the sellers are offering to sell manual transmission car significantly more than automatic transmission cars. The reverse is the case in United States.

```
In [48]: # Frequency Counts of Car_Engine_Power_PS
         autos_df['Car_Engine_Power_PS'].value_counts().nlargest(20)
```

```
Out[48]: 0.0      40812
         75.0     24035
         60.0     15907
         150.0    15442
         140.0    13585
         101.0    13312
         90.0     12748
         116.0    11963
         170.0    10983
         105.0    10429
         125.0     7051
         136.0     6952
         102.0     6500
         163.0     6287
         54.0      5752
         143.0     5547
         122.0     5322
         131.0     5180
         110.0     4862
         109.0     4816
         Name: Car_Engine_Power_PS, dtype: int64
```

The above result highlights the top 20 Car Engine Power PS. It is worth noting that 40812 implies missing values. There is no car with zero Car Engine Power.

```
In [49]: # Frequency Counts of Car_Model
         autos_df['Car_Model'].value_counts().nlargest(20)
```

```
Out[49]: golf           30069
         other          26402
         3er            20566
```

```
polo           13092
corsa          12573
astra          10829
passat         10306
a4             10257
c_klasse        8776
5er             8546
e_klasse        7560
a3              6604
a6              6023
focus           5950
fiesta          5775
transporter     5527
twingo          4953
2_reihe         4816
fortwo          4338
a_klasse        4317
Name: Car_Model, dtype: int64
```

```
In [50]: # Frequency Counts of Car_Mileage_Kilometer
         autos_df['Car_Mileage_Kilometer'].value_counts().nlargest(10)
```

```
Out[50]: 150000    240793
         125000     38067
         100000     15920
         90000      12524
         80000      11053
         70000       9772
         60000       8669
         50000       7615
         5000        7067
         40000       6377
         Name: Car_Mileage_Kilometer, dtype: int64
```

The above result showcases the top 10 Car Mileage in Kilometer. This also implies that older cars are being offered for sales than the newer cars.

```
In [51]: # Frequency Counts of Month_of_Car_Registration
         autos_df['Month_of_Car_Registration'].value_counts().nlargest(10)
```

```
Out[51]: 0.0     37670
         3.0     36168
         6.0     33170
         4.0     30919
         5.0     30631
         7.0     28960
         10.0    27338
         11.0    25489
         12.0    25379
```

```
9.0     25074
Name: Month_of_Car_Registration, dtype: int64
```

The above result showcases the the top 10 months of the year for cars first registrations.

```
In [52]: # Frequency Counts of Fuel_Type
         autos_df['Fuel_Type'].value_counts().nlargest(10)
```

```
Out[52]: gas                       223857
         diesel                    107746
         autogas                     5378
         compressed natural gas       571
         hybrid                       279
         other                        208
         electric                     104
         Name: Fuel_Type, dtype: int64
```

The above result means most vehicles are using either gas or diesel.

```
In [53]: # Frequency Counts of Car_Brand
         autos_df['Car_Brand'].value_counts().nlargest(10)
```

```
Out[53]: volkswagen       79638
         bmw              40271
         opel             40135
         mercedes_benz    35312
         audi             32873
         ford             25572
         renault          17969
         peugeot          11027
         fiat              9676
         seat              7022
         Name: Car_Brand, dtype: int64
```

The above result highlights the top Car Brands being offered for sales.

```
In [54]: # Frequency Counts of UnRepaired_Damage
         autos_df['UnRepaired_Damage'].value_counts()
```

```
Out[54]: no     263183
         yes     36286
         Name: UnRepaired_Damage, dtype: int64
```

The above output depicts cars that required no repairs are being offered for sales significantly more than cars that require repairs.

```
In [55]: # Frequency Counts of No_of_Pictures
         autos_df['No_of_Pictures'].value_counts()
```

```
Out[55]: 0.0    371522
         Name: No_of_Pictures, dtype: int64
```

The above result highlights there are no pictures available with the car listing advertisements collected in this dataset.

```
In [56]: # Frequency Counts of Seller_Postal_Code
         autos_df['Seller_Postal_Code'].value_counts().nlargest(10)
```

```
Out[56]: 10115.0    828
         65428.0    637
         66333.0    349
         38518.0    326
         44145.0    323
         32257.0    323
         52525.0    314
         78224.0    309
         26789.0    301
         48599.0    294
         Name: Seller_Postal_Code, dtype: int64
```

The above result showcases the top 10 postal or zip codes associated with the car offerings collected in this dataset.

0.5 Missing Values

There are missing values identified as NaNs in the dataset. However, these are not the only missing values. The data used '0.0' as missing values as well. Specifically, the data used 0.0 in Car_Price(Euros), Car_Engine_Power_PS, Month_of_Car_Registration and No_of_Pictures as missing values as well. And these have to be cleaned up. Null values would result in bias resulting from differences between missing and complete data.

```
In [57]: # https://stackoverflow.com/questions/49575897/cant-replace-0-to-nan-in-python-using-
         # Convert zeros(0.0) to missing values - Replace 0 with NaNs
         autos_df.replace(0.0, np.nan, inplace=True)
```

```
In [58]: autos_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 371522 entries, 0 to 371539
Data columns (total 20 columns):
Date_Crawled                 371522 non-null object
Car_Name                     371522 non-null object
Car_Seller_Type              371522 non-null object
Offer_Type                   371522 non-null object
Car_Price                    360750 non-null float64
Abtest_Type                  371522 non-null object
Vechicle_Type                333660 non-null object
Year_of_Car_Registration     371522 non-null float64
Car_Transmission_Type        351319 non-null object
Car_Engine_Power_PS          330710 non-null float64
Car_Model                    351041 non-null object
```

```
Car_Mileage_Kilometer        371522 non-null object
Month_of_Car_Registration    333852 non-null float64
Fuel_Type                    338143 non-null object
Car_Brand                    371522 non-null object
UnRepaired_Damage            299469 non-null object
Date_Created                 371522 non-null object
No_of_Pictures               0 non-null float64
Seller_Postal_Code           371522 non-null float64
Date_LastSeen_Online         371522 non-null object
dtypes: float64(6), object(14)
memory usage: 59.5+ MB
```

```
In [59]: autos_df.isnull().sum()
```

```
Out[59]: Date_Crawled                     0
         Car_Name                         0
         Car_Seller_Type                  0
         Offer_Type                       0
         Car_Price                    10772
         Abtest_Type                      0
         Vechicle_Type                37862
         Year_of_Car_Registration         0
         Car_Transmission_Type        20203
         Car_Engine_Power_PS          40812
         Car_Model                    20481
         Car_Mileage_Kilometer            0
         Month_of_Car_Registration    37670
         Fuel_Type                    33379
         Car_Brand                        0
         UnRepaired_Damage            72053
         Date_Created                     0
         No_of_Pictures              371522
         Seller_Postal_Code               0
         Date_LastSeen_Online             0
         dtype: int64
```

```
In [60]: autos_df.head(10)
```

```
Out[60]:       Date_Crawled                                          Car_Name  \
         0  3/24/2016 11:52                                       Golf_3_1.6
         1  3/24/2016 10:58                             A5_Sportback_2.7_Tdi
         2  3/14/2016 12:52                Jeep_Grand_Cherokee_"Overland"
         3  3/17/2016 16:54                                  GOLF_4_1_4__3TÜRER
         4  3/31/2016 17:25                  Skoda_Fabia_1.4_TDI_PD_Classic
         5   4/4/2016 17:36  BMW_316i___e36_Limousine___Bastlerfahrzeug__Ex...
         6   4/1/2016 20:48                    Peugeot_206_CC_110_Platinum
         7  3/21/2016 18:54                    VW_Derby_Bj_80__Scheunenfund
         8   4/4/2016 23:42          Ford_C___Max_Titanium_1_0_L_EcoBoost
```

```
9  3/17/2016 10:53  VW_Golf_4_5_tuerig_zu_verkaufen_mit_Anhaengerk...

  Car_Seller_Type Offer_Type  Car_Price Abtest_Type Vechicle_Type  \
0         private      offer      480.0        test           NaN
1         private      offer    18300.0        test         coupe
2         private      offer     9800.0        test           suv
3         private      offer     1500.0        test     small car
4         private      offer     3600.0        test     small car
5         private      offer      650.0        test     limousine
6         private      offer     2200.0        test        cabrio
7         private      offer        NaN        test     limousine
8         private      offer    14500.0     control           bus
9         private      offer      999.0        test     small car

   Year_of_Car_Registration Car_Transmission_Type  Car_Engine_Power_PS  \
0                    1993.0                manual                  NaN
1                    2011.0                manual                190.0
2                    2004.0             automatic                163.0
3                    2001.0                manual                 75.0
4                    2008.0                manual                 69.0
5                    1995.0                manual                102.0
6                    2004.0                manual                109.0
7                    1980.0                manual                 50.0
8                    2014.0                manual                125.0
9                    1998.0                manual                101.0

  Car_Model Car_Mileage_Kilometer  Month_of_Car_Registration Fuel_Type  \
0      golf                150000                       NaN       gas
1       NaN                125000                       5.0    diesel
2     grand                125000                       8.0    diesel
3      golf                150000                       6.0       gas
4     fabia                 90000                       7.0    diesel
5       3er                150000                      10.0       gas
6   2_reihe                150000                       8.0       gas
7     other                 40000                       7.0       gas
8     c_max                 30000                       8.0       gas
9      golf                150000                       NaN       NaN

    Car_Brand UnRepaired_Damage    Date_Created  No_of_Pictures  \
0  volkswagen              NaN  3/24/2016 0:00             NaN
1        audi              yes  3/24/2016 0:00             NaN
2        jeep              NaN  3/14/2016 0:00             NaN
3  volkswagen               no  3/17/2016 0:00             NaN
4       skoda               no  3/31/2016 0:00             NaN
5         bmw              yes   4/4/2016 0:00             NaN
6     peugeot               no   4/1/2016 0:00             NaN
7  volkswagen               no  3/21/2016 0:00             NaN
8        ford              NaN   4/4/2016 0:00             NaN
```

```
9  volkswagen               NaN  3/17/2016 0:00             NaN

   Seller_Postal_Code Date_LastSeen_Online
0             70435.0        4/7/2016 3:16
1             66954.0        4/7/2016 1:46
2             90480.0       4/5/2016 12:47
3             91074.0      3/17/2016 17:40
4             60437.0       4/6/2016 10:17
5             33775.0       4/6/2016 19:17
6             67112.0       4/5/2016 18:18
7             19348.0      3/25/2016 16:47
8             94505.0       4/4/2016 23:42
9             27472.0      3/31/2016 17:17
```

```
In [61]: autos_df.dtypes
```

```
Out[61]: Date_Crawled                 object
         Car_Name                     object
         Car_Seller_Type              object
         Offer_Type                   object
         Car_Price                   float64
         Abtest_Type                  object
         Vechicle_Type                object
         Year_of_Car_Registration    float64
         Car_Transmission_Type        object
         Car_Engine_Power_PS         float64
         Car_Model                    object
         Car_Mileage_Kilometer        object
         Month_of_Car_Registration   float64
         Fuel_Type                    object
         Car_Brand                    object
         UnRepaired_Damage            object
         Date_Created                 object
         No_of_Pictures              float64
         Seller_Postal_Code          float64
         Date_LastSeen_Online         object
         dtype: object
```

0.6 Change column types to appropriate data types

```
In [62]: autos_df2 = autos_df.copy()
```

```
In [63]: autos_df2.head()
```

```
Out[63]:       Date_Crawled                        Car_Name Car_Seller_Type Offer_Type  \
         0  3/24/2016 11:52                     Golf_3_1.6         private      offer
         1  3/24/2016 10:58           A5_Sportback_2.7_Tdi         private      offer
         2  3/14/2016 12:52  Jeep_Grand_Cherokee_"Overland"         private      offer
         3  3/17/2016 16:54             GOLF_4_1_4__3TÜRER         private      offer
```

```
4  3/31/2016 17:25  Skoda_Fabia_1.4_TDI_PD_Classic         private      offer

   Car_Price Abtest_Type Vechicle_Type  Year_of_Car_Registration  \
0      480.0        test           NaN                   1993.0
1    18300.0        test         coupe                   2011.0
2     9800.0        test           suv                   2004.0
3     1500.0        test     small car                   2001.0
4     3600.0        test     small car                   2008.0

  Car_Transmission_Type  Car_Engine_Power_PS Car_Model Car_Mileage_Kilometer  \
0                manual                  NaN      golf                150000
1                manual                190.0       NaN                125000
2             automatic                163.0     grand                125000
3                manual                 75.0      golf                150000
4                manual                 69.0     fabia                 90000

   Month_of_Car_Registration Fuel_Type   Car_Brand UnRepaired_Damage  \
0                        NaN       gas  volkswagen               NaN
1                        5.0    diesel        audi               yes
2                        8.0    diesel        jeep               NaN
3                        6.0       gas  volkswagen                no
4                        7.0    diesel       skoda                no

     Date_Created  No_of_Pictures  Seller_Postal_Code Date_LastSeen_Online
0  3/24/2016 0:00             NaN             70435.0        4/7/2016 3:16
1  3/24/2016 0:00             NaN             66954.0        4/7/2016 1:46
2  3/14/2016 0:00             NaN             90480.0       4/5/2016 12:47
3  3/17/2016 0:00             NaN             91074.0      3/17/2016 17:40
4  3/31/2016 0:00             NaN             60437.0       4/6/2016 10:17
```

```
In [64]: # Convert Car Seller Type from string to categorical
         autos_df2['Car_Seller_Type'] = autos_df2.Car_Seller_Type.astype('category')

In [65]: autos_df2.dtypes

Out[65]: Date_Crawled                 object
         Car_Name                     object
         Car_Seller_Type            category
         Offer_Type                   object
         Car_Price                   float64
         Abtest_Type                  object
         Vechicle_Type                object
         Year_of_Car_Registration    float64
         Car_Transmission_Type        object
         Car_Engine_Power_PS         float64
         Car_Model                    object
         Car_Mileage_Kilometer        object
         Month_of_Car_Registration   float64
```

```
Fuel_Type                    object
Car_Brand                    object
UnRepaired_Damage            object
Date_Created                 object
No_of_Pictures              float64
Seller_Postal_Code          float64
Date_LastSeen_Online         object
dtype: object
```

In [66]:
```
autos_df2.head()
```

Out[66]:

	Date_Crawled	Car_Name	Car_Seller_Type	Offer_Type	Car_Price	Abtest_Type	Vechicle_Type	Year_of_Car_Registration	Car_Transmission_Type	Car_Engine_Power_PS	Car_Model	Car_Mileage_Kilometer	Month_of_Car_Registration	Fuel_Type	Car_Brand	UnRepaired_Damage	Date_Created	No_of_Pictures	Seller_Postal_Code	Date_LastSeen_Online
0	3/24/2016 11:52	Golf_3_1.6	private	offer	480.0	test	NaN	1993.0	manual	NaN	golf	150000	NaN	gas	volkswagen	NaN	3/24/2016 0:00	NaN	70435.0	4/7/2016 3:16
1	3/24/2016 10:58	A5_Sportback_2.7_Tdi	private	offer	18300.0	test	coupe	2011.0	manual	190.0	NaN	125000	5.0	diesel	audi	yes	3/24/2016 0:00	NaN	66954.0	4/7/2016 1:46
2	3/14/2016 12:52	Jeep_Grand_Cherokee_"Overland"	private	offer	9800.0	test	suv	2004.0	automatic	163.0	grand	125000	8.0	diesel	jeep	NaN	3/14/2016 0:00	NaN	90480.0	4/5/2016 12:47
3	3/17/2016 16:54	GOLF_4_1_4__3TÜRER	private	offer	1500.0	test	small car	2001.0	manual	75.0	golf	150000	6.0	gas	volkswagen	no	3/17/2016 0:00	NaN	91074.0	3/17/2016 17:40
4	3/31/2016 17:25	Skoda_Fabia_1.4_TDI_PD_Classic	private	offer	3600.0	test	small car	2008.0	manual	69.0	fabia	90000	7.0	diesel	skoda	no	3/31/2016 0:00	NaN	60437.0	4/6/2016 10:17

In [67]:
```
# https://stackoverflow.com/questions/17134716/convert-dataframe-column-type-from-str
# Change Date_Crawled, Date_Created, and Date_LastSeen_Online column data type from s
```

```
        # with date specified in American date format
        autos_df2['Date_Crawled'] = pd.to_datetime(autos_df2['Date_Crawled'])
```

```
In [68]: autos_df2.dtypes
```

```
Out[68]: Date_Crawled                 datetime64[ns]
         Car_Name                             object
         Car_Seller_Type                    category
         Offer_Type                           object
         Car_Price                           float64
         Abtest_Type                          object
         Vechicle_Type                        object
         Year_of_Car_Registration            float64
         Car_Transmission_Type                object
         Car_Engine_Power_PS                 float64
         Car_Model                            object
         Car_Mileage_Kilometer                object
         Month_of_Car_Registration           float64
         Fuel_Type                            object
         Car_Brand                            object
         UnRepaired_Damage                    object
         Date_Created                         object
         No_of_Pictures                      float64
         Seller_Postal_Code                  float64
         Date_LastSeen_Online                 object
         dtype: object
```

```
In [69]: autos_df2.head()
```

```
Out[69]:          Date_Crawled                         Car_Name Car_Seller_Type  \
         0 2016-03-24 11:52:00                       Golf_3_1.6         private
         1 2016-03-24 10:58:00             A5_Sportback_2.7_Tdi         private
         2 2016-03-14 12:52:00  Jeep_Grand_Cherokee_"Overland"         private
         3 2016-03-17 16:54:00               GOLF_4_1_4__3TÜRER         private
         4 2016-03-31 17:25:00  Skoda_Fabia_1.4_TDI_PD_Classic         private

           Offer_Type  Car_Price Abtest_Type Vechicle_Type  Year_of_Car_Registration  \
         0      offer      480.0        test           NaN                   1993.0
         1      offer    18300.0        test         coupe                   2011.0
         2      offer     9800.0        test           suv                   2004.0
         3      offer     1500.0        test     small car                   2001.0
         4      offer     3600.0        test     small car                   2008.0

           Car_Transmission_Type  Car_Engine_Power_PS Car_Model Car_Mileage_Kilometer  \
         0                manual                  NaN      golf                150000
         1                manual                190.0       NaN                125000
         2             automatic                163.0     grand                125000
         3                manual                 75.0      golf                150000
         4                manual                 69.0     fabia                 90000
```

```
   Month_of_Car_Registration Fuel_Type   Car_Brand UnRepaired_Damage  \
0                        NaN       gas  volkswagen               NaN
1                        5.0    diesel        audi               yes
2                        8.0    diesel        jeep               NaN
3                        6.0       gas  volkswagen                no
4                        7.0    diesel       skoda                no

     Date_Created  No_of_Pictures  Seller_Postal_Code Date_LastSeen_Online
0  3/24/2016 0:00             NaN             70435.0        4/7/2016 3:16
1  3/24/2016 0:00             NaN             66954.0        4/7/2016 1:46
2  3/14/2016 0:00             NaN             90480.0       4/5/2016 12:47
3  3/17/2016 0:00             NaN             91074.0      3/17/2016 17:40
4  3/31/2016 0:00             NaN             60437.0       4/6/2016 10:17
```

In [70]:
```
# https://stackoverflow.com/questions/17134716/convert-dataframe-column-type-from-str
# Change Date_Crawled, Date_Created, and Date_LastSeen_Online column data type from s
# with date specified in American date format

autos_df2['Date_Created'] = pd.to_datetime(autos_df2['Date_Created'])
```

In [71]:
```
# https://stackoverflow.com/questions/17134716/convert-dataframe-column-type-from-str
# Change Date_Crawled, Date_Created, and Date_LastSeen_Online column data type from s
# with date specified in American date format
autos_df2['Date_LastSeen_Online'] = pd.to_datetime(autos_df2['Date_LastSeen_Online'])
```

In [72]:
```
autos_df2.head()
```

Out[72]:
```
          Date_Crawled                      Car_Name Car_Seller_Type  \
0 2016-03-24 11:52:00                    Golf_3_1.6         private
1 2016-03-24 10:58:00          A5_Sportback_2.7_Tdi         private
2 2016-03-14 12:52:00  Jeep_Grand_Cherokee_"Overland"         private
3 2016-03-17 16:54:00            GOLF_4_1_4__3TÜRER         private
4 2016-03-31 17:25:00  Skoda_Fabia_1.4_TDI_PD_Classic         private

  Offer_Type  Car_Price Abtest_Type Vechicle_Type  Year_of_Car_Registration  \
0      offer      480.0        test           NaN                   1993.0
1      offer    18300.0        test         coupe                   2011.0
2      offer     9800.0        test           suv                   2004.0
3      offer     1500.0        test     small car                   2001.0
4      offer     3600.0        test     small car                   2008.0

  Car_Transmission_Type  Car_Engine_Power_PS Car_Model Car_Mileage_Kilometer  \
0                manual                  NaN      golf                150000
1                manual                190.0       NaN                125000
2             automatic                163.0     grand                125000
3                manual                 75.0      golf                150000
4                manual                 69.0     fabia                 90000
```

```
   Month_of_Car_Registration Fuel_Type   Car_Brand UnRepaired_Damage  \
0                        NaN       gas  volkswagen               NaN
1                        5.0    diesel        audi               yes
2                        8.0    diesel        jeep               NaN
3                        6.0       gas  volkswagen                no
4                        7.0    diesel       skoda                no

  Date_Created  No_of_Pictures  Seller_Postal_Code Date_LastSeen_Online
0   2016-03-24             NaN             70435.0  2016-04-07 03:16:00
1   2016-03-24             NaN             66954.0  2016-04-07 01:46:00
2   2016-03-14             NaN             90480.0  2016-04-05 12:47:00
3   2016-03-17             NaN             91074.0  2016-03-17 17:40:00
4   2016-03-31             NaN             60437.0  2016-04-06 10:17:00
```

In [73]:
```
# Display column data types
autos_df2.dtypes
```

Out[73]:
```
Date_Crawled                datetime64[ns]
Car_Name                            object
Car_Seller_Type                   category
Offer_Type                          object
Car_Price                          float64
Abtest_Type                         object
Vechicle_Type                       object
Year_of_Car_Registration           float64
Car_Transmission_Type               object
Car_Engine_Power_PS                float64
Car_Model                           object
Car_Mileage_Kilometer               object
Month_of_Car_Registration          float64
Fuel_Type                           object
Car_Brand                           object
UnRepaired_Damage                   object
Date_Created                datetime64[ns]
No_of_Pictures                     float64
Seller_Postal_Code                 float64
Date_LastSeen_Online        datetime64[ns]
dtype: object
```

In [74]:
```
autos_df2.isnull().sum()
```

Out[74]:
```
Date_Crawled                    0
Car_Name                        0
Car_Seller_Type                 0
Offer_Type                      0
Car_Price                   10772
Abtest_Type                     0
Vechicle_Type               37862
Year_of_Car_Registration        0
```

```
Car_Transmission_Type         20203
Car_Engine_Power_PS           40812
Car_Model                     20481
Car_Mileage_Kilometer             0
Month_of_Car_Registration     37670
Fuel_Type                     33379
Car_Brand                         0
UnRepaired_Damage             72053
Date_Created                      0
No_of_Pictures               371522
Seller_Postal_Code                0
Date_LastSeen_Online              0
dtype: int64
```

0.7 Dealing with Missing Values

It is important to deal with missing values to get the dataset into a more useful form by deleting columns and rows that are not required for further analysis. Furthermore, decision should be made on how to treat NaNs/missing values.

```
In [75]: #Summarize the number of rows and columns in the dataset
         print(autos_df2.shape)
```

```
(371522, 20)
```

```
In [76]: # Drop the rows where all elements are missing
         autos_clean_df = autos_df2.dropna(how ='all')
```

```
In [77]: #Summarize the number of rows and columns in the dataset
         print(autos_clean_df.shape)
```

```
(371522, 20)
```

```
In [78]: # Drop rows that contain less than five observations
         autos_clean_df = autos_clean_df.dropna(thresh=5)
```

```
In [79]: #Summarize the number of rows and columns in the dataset
         print(autos_clean_df.shape)
```

```
(371522, 20)
```

```
In [80]: # The No_of_Pictures column contains missing values only, we need to remove it.
         # We won't get anything meaningful from a column with all missing values
         autos_clean_df = autos_clean_df.drop('No_of_Pictures', axis=1)
```

```
In [81]: #Summarize the number of rows and columns in the dataset
         print(autos_clean_df.shape)
```

```
(371522, 19)
```

```
In [82]: autos_clean_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 371522 entries, 0 to 371539
Data columns (total 19 columns):
Date_Crawled                 371522 non-null datetime64[ns]
Car_Name                     371522 non-null object
Car_Seller_Type              371522 non-null category
Offer_Type                   371522 non-null object
Car_Price                    360750 non-null float64
Abtest_Type                  371522 non-null object
Vechicle_Type                333660 non-null object
Year_of_Car_Registration     371522 non-null float64
Car_Transmission_Type        351319 non-null object
Car_Engine_Power_PS          330710 non-null float64
Car_Model                    351041 non-null object
Car_Mileage_Kilometer        371522 non-null object
Month_of_Car_Registration    333852 non-null float64
Fuel_Type                    338143 non-null object
Car_Brand                    371522 non-null object
UnRepaired_Damage            299469 non-null object
Date_Created                 371522 non-null datetime64[ns]
Seller_Postal_Code           371522 non-null float64
Date_LastSeen_Online         371522 non-null datetime64[ns]
dtypes: category(1), datetime64[ns](3), float64(5), object(10)
memory usage: 54.2+ MB
```

0.8 Drop columns not required for further data analysis

```
In [83]: # Drop Date_Crawled we don't need it for further data analysis
         # Drop Car_Name, we don't need it for further data analysis
         # Drop Car_Seller_Type it only has one factor level - private
         # Drop Offer_Type it only has one factor level - offer
         # Drop Abtest_Type it is only used for internal control

         autos_clean_df = autos_clean_df.drop(['Date_Crawled','Car_Name', 'Car_Seller_Type', '(
```

```
In [84]: autos_clean_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 371522 entries, 0 to 371539
Data columns (total 14 columns):
Car_Price                    360750 non-null float64
Vechicle_Type                333660 non-null object
Year_of_Car_Registration     371522 non-null float64
```

```
Car_Transmission_Type        351319 non-null object
Car_Engine_Power_PS          330710 non-null float64
Car_Model                    351041 non-null object
Car_Mileage_Kilometer        371522 non-null object
Month_of_Car_Registration    333852 non-null float64
Fuel_Type                    338143 non-null object
Car_Brand                    371522 non-null object
UnRepaired_Damage            299469 non-null object
Date_Created                 371522 non-null datetime64[ns]
Seller_Postal_Code           371522 non-null float64
Date_LastSeen_Online         371522 non-null datetime64[ns]
dtypes: datetime64[ns](2), float64(5), object(7)
memory usage: 42.5+ MB
```

```
In [85]: autos_clean_df.head(10)
```

Out[85]:

	Car_Price	Vechicle_Type	Year_of_Car_Registration	Car_Transmission_Type
0	480.0	NaN	1993.0	manual
1	18300.0	coupe	2011.0	manual
2	9800.0	suv	2004.0	automatic
3	1500.0	small car	2001.0	manual
4	3600.0	small car	2008.0	manual
5	650.0	limousine	1995.0	manual
6	2200.0	cabrio	2004.0	manual
7	NaN	limousine	1980.0	manual
8	14500.0	bus	2014.0	manual
9	999.0	small car	1998.0	manual

	Car_Engine_Power_PS	Car_Model	Car_Mileage_Kilometer
0	NaN	golf	150000
1	190.0	NaN	125000
2	163.0	grand	125000
3	75.0	golf	150000
4	69.0	fabia	90000
5	102.0	3er	150000
6	109.0	2_reihe	150000
7	50.0	other	40000
8	125.0	c_max	30000
9	101.0	golf	150000

	Month_of_Car_Registration	Fuel_Type	Car_Brand	UnRepaired_Damage
0	NaN	gas	volkswagen	NaN
1	5.0	diesel	audi	yes
2	8.0	diesel	jeep	NaN
3	6.0	gas	volkswagen	no
4	7.0	diesel	skoda	no
5	10.0	gas	bmw	yes

```
6                        8.0       gas     peugeot                 no
7                        7.0       gas  volkswagen                 no
8                        8.0       gas        ford                NaN
9                        NaN       NaN  volkswagen                NaN

  Date_Created  Seller_Postal_Code Date_LastSeen_Online
0   2016-03-24             70435.0  2016-04-07 03:16:00
1   2016-03-24             66954.0  2016-04-07 01:46:00
2   2016-03-14             90480.0  2016-04-05 12:47:00
3   2016-03-17             91074.0  2016-03-17 17:40:00
4   2016-03-31             60437.0  2016-04-06 10:17:00
5   2016-04-04             33775.0  2016-04-06 19:17:00
6   2016-04-01             67112.0  2016-04-05 18:18:00
7   2016-03-21             19348.0  2016-03-25 16:47:00
8   2016-04-04             94505.0  2016-04-04 23:42:00
9   2016-03-17             27472.0  2016-03-31 17:17:00
```

0.9 Cleaning up Erronous Values - Outliers Removal

There is a need to examine the remaining 14 attributes/columns and remove errornous values from the columns. For example, there is no way we could have a car with year of registration as 2018,5300,5600 or 7777 in 2016. Extreme values should also be removed from the columns.

```
In [86]: # Summary statistics for Car_Engine_Power_PS
         print(autos_clean_df['Car_Engine_Power_PS'].describe())

count    330710.000000
mean        129.812358
std         199.054614
min           1.000000
25%          80.000000
50%         116.000000
75%         150.000000
max       20000.000000
Name: Car_Engine_Power_PS, dtype: float64


In [87]: # https://www.autotrader.com/car-news/which-car-has-highest-horsepower-range-28147497
         # https://en.wikipedia.org/wiki/Engine_power#Common_power,_(listed_as_weight_to_power
         # Remove rows with erroneous engine power values
         # Limit Car Engine Power PS the range of 1 to 999

         autos_clean_df = autos_clean_df[(autos_clean_df.Car_Engine_Power_PS > 0) & (autos_clea

In [88]: # Summary statistics for Car_Engine_Power_PS
         print(autos_clean_df['Car_Engine_Power_PS'].describe())

count    330396.000000
mean        125.976779
```

```
std          62.834223
min           1.000000
25%          80.000000
50%         116.000000
75%         150.000000
max         999.000000
Name: Car_Engine_Power_PS, dtype: float64
```

```
In [89]: # Summary statistics for Car_Price in Euros
         print(autos_clean_df['Car_Price'].describe())
```

```
count    3.235060e+05
mean     1.705271e+04
std      3.818140e+06
min      1.000000e+00
25%      1.400000e+03
50%      3.490000e+03
75%      7.950000e+03
max      2.147484e+09
Name: Car_Price, dtype: float64
```

```
In [90]: # Let us keep the price within the range of 1 to 100,000
         autos_clean_df = autos_clean_df[(autos_clean_df.Car_Price > 0) & (autos_clean_df.Car_
```

```
In [91]: # Summary statistics for Car_Price in Euros
         print(autos_clean_df['Car_Price'].describe())
```

```
count    323161.000000
mean       6110.203765
std        7696.080783
min           1.000000
25%        1400.000000
50%        3450.000000
75%        7900.000000
max       99999.000000
Name: Car_Price, dtype: float64
```

```
In [92]: autos_clean_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 323161 entries, 1 to 371539
Data columns (total 14 columns):
Car_Price                   323161 non-null float64
Vechicle_Type               301589 non-null object
Year_of_Car_Registration    323161 non-null float64
Car_Transmission_Type       316958 non-null object
```

```
Car_Engine_Power_PS          323161 non-null float64
Car_Model                    310134 non-null object
Car_Mileage_Kilometer        323161 non-null object
Month_of_Car_Registration    302946 non-null float64
Fuel_Type                    303131 non-null object
Car_Brand                    323161 non-null object
UnRepaired_Damage            275406 non-null object
Date_Created                 323161 non-null datetime64[ns]
Seller_Postal_Code           323161 non-null float64
Date_LastSeen_Online         323161 non-null datetime64[ns]
dtypes: datetime64[ns](2), float64(5), object(7)
memory usage: 37.0+ MB
```

```
In [93]: autos_clean_df.head(10)
```

Out[93]:

	Car_Price	Vechicle_Type	Year_of_Car_Registration	Car_Transmission_Type	Car_Engine_Power_PS	Car_Model	Car_Mileage_Kilometer	Month_of_Car_Registration	Fuel_Type	Car_Brand	UnRepaired_Damage
1	18300.0	coupe	2011.0	manual	190.0	NaN	125000	5.0	diesel	audi	yes
2	9800.0	suv	2004.0	automatic	163.0	grand	125000	8.0	diesel	jeep	NaN
3	1500.0	small car	2001.0	manual	75.0	golf	150000	6.0	gas	volkswagen	no
4	3600.0	small car	2008.0	manual	69.0	fabia	90000	7.0	diesel	skoda	no
5	650.0	limousine	1995.0	manual	102.0	3er	150000	10.0	gas	bmw	yes
6	2200.0	cabrio	2004.0	manual	109.0	2_reihe	150000	8.0	gas	peugeot	no
8	14500.0	bus	2014.0	manual	125.0	c_max	30000	8.0	gas	ford	NaN
9	999.0	small car	1998.0	manual	101.0	golf	150000				
10	2000.0	limousine	2004.0	manual	105.0	3_reihe	150000				
11	2799.0	station wagon	2005.0	manual	140.0	passat	150000				

```
9                        NaN       NaN  volkswagen              NaN
10                      12.0       gas       mazda               no
11                      12.0    diesel  volkswagen              yes

   Date_Created  Seller_Postal_Code Date_LastSeen_Online
1    2016-03-24             66954.0  2016-04-07 01:46:00
2    2016-03-14             90480.0  2016-04-05 12:47:00
3    2016-03-17             91074.0  2016-03-17 17:40:00
4    2016-03-31             60437.0  2016-04-06 10:17:00
5    2016-04-04             33775.0  2016-04-06 19:17:00
6    2016-04-01             67112.0  2016-04-05 18:18:00
8    2016-04-04             94505.0  2016-04-04 23:42:00
9    2016-03-17             27472.0  2016-03-31 17:17:00
10   2016-03-26             96224.0  2016-04-06 10:45:00
11   2016-04-07             57290.0  2016-04-07 10:25:00
```

In [94]:
```
autos_clean_df.isnull().sum()
```

Out[94]:
```
Car_Price                       0
Vechicle_Type               21572
Year_of_Car_Registration        0
Car_Transmission_Type        6203
Car_Engine_Power_PS             0
Car_Model                   13027
Car_Mileage_Kilometer           0
Month_of_Car_Registration   20215
Fuel_Type                   20030
Car_Brand                       0
UnRepaired_Damage           47755
Date_Created                    0
Seller_Postal_Code              0
Date_LastSeen_Online            0
dtype: int64
```

In [95]:
```
# https://en.wikipedia.org/wiki/Vehicle_registration_plate#History
# https://www.platehunter.com/car-registration-years-
# Remove rows with errornous year of car registration values
# Let us focus the dataset on car year of registration between 1950 and 2017

autos_clean_df = autos_clean_df[(autos_clean_df.Year_of_Car_Registration > 1950) & (au
```

In [96]:
```
autos_clean_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 312655 entries, 1 to 371539
Data columns (total 14 columns):
Car_Price                   312655 non-null float64
Vechicle_Type               301499 non-null object
Year_of_Car_Registration    312655 non-null float64
```

```
Car_Transmission_Type        307157 non-null object
Car_Engine_Power_PS          312655 non-null float64
Car_Model                    300912 non-null object
Car_Mileage_Kilometer        312655 non-null object
Month_of_Car_Registration    294541 non-null float64
Fuel_Type                    296863 non-null object
Car_Brand                    312655 non-null object
UnRepaired_Damage            269060 non-null object
Date_Created                 312655 non-null datetime64[ns]
Seller_Postal_Code           312655 non-null float64
Date_LastSeen_Online         312655 non-null datetime64[ns]
dtypes: datetime64[ns](2), float64(5), object(7)
memory usage: 35.8+ MB
```

```
In [97]: autos_clean_df.isnull().sum()
```

```
Out[97]: Car_Price                       0
         Vechicle_Type                11156
         Year_of_Car_Registration         0
         Car_Transmission_Type         5498
         Car_Engine_Power_PS              0
         Car_Model                    11743
         Car_Mileage_Kilometer            0
         Month_of_Car_Registration    18114
         Fuel_Type                    15792
         Car_Brand                        0
         UnRepaired_Damage            43595
         Date_Created                     0
         Seller_Postal_Code               0
         Date_LastSeen_Online             0
         dtype: int64
```

```
In [98]: autos_clean_df.head(10)
```

```
Out[98]:     Car_Price  Vechicle_Type  Year_of_Car_Registration Car_Transmission_Type  \
         1     18300.0          coupe                    2011.0                manual
         2      9800.0            suv                    2004.0             automatic
         3      1500.0      small car                    2001.0                manual
         4      3600.0      small car                    2008.0                manual
         5       650.0      limousine                    1995.0                manual
         6      2200.0         cabrio                    2004.0                manual
         8     14500.0            bus                    2014.0                manual
         9       999.0      small car                    1998.0                manual
         10     2000.0      limousine                    2004.0                manual
         11     2799.0  station wagon                    2005.0                manual

             Car_Engine_Power_PS Car_Model Car_Mileage_Kilometer  \
         1                 190.0       NaN                125000
```

```
2                 163.0     grand                125000
3                  75.0      golf                150000
4                  69.0     fabia                 90000
5                 102.0       3er                150000
6                 109.0   2_reihe                150000
8                 125.0     c_max                 30000
9                 101.0      golf                150000
10                105.0   3_reihe                150000
11                140.0    passat                150000

    Month_of_Car_Registration Fuel_Type   Car_Brand UnRepaired_Damage  \
1                         5.0    diesel        audi               yes
2                         8.0    diesel        jeep               NaN
3                         6.0       gas  volkswagen                no
4                         7.0    diesel       skoda                no
5                        10.0       gas         bmw               yes
6                         8.0       gas     peugeot                no
8                         8.0       gas        ford               NaN
9                         NaN       NaN  volkswagen               NaN
10                       12.0       gas       mazda                no
11                       12.0    diesel  volkswagen               yes

   Date_Created  Seller_Postal_Code Date_LastSeen_Online
1    2016-03-24             66954.0  2016-04-07 01:46:00
2    2016-03-14             90480.0  2016-04-05 12:47:00
3    2016-03-17             91074.0  2016-03-17 17:40:00
4    2016-03-31             60437.0  2016-04-06 10:17:00
5    2016-04-04             33775.0  2016-04-06 19:17:00
6    2016-04-01             67112.0  2016-04-05 18:18:00
8    2016-04-04             94505.0  2016-04-04 23:42:00
9    2016-03-17             27472.0  2016-03-31 17:17:00
10   2016-03-26             96224.0  2016-04-06 10:45:00
11   2016-04-07             57290.0  2016-04-07 10:25:00
```

```
In [99]: # https://datascience.stackexchange.com/questions/30249/how-to-delete-entire-row-if-v
         # Remove rows with UnRepaired_Damage Null/Nan values
         autos_cleaned_df = autos_clean_df[pd.notnull(autos_clean_df['UnRepaired_Damage'])]
```

```
In [100]: autos_cleaned_df.isnull().sum()
```

```
Out[100]: Car_Price                      0
          Vechicle_Type               5725
          Year_of_Car_Registration       0
          Car_Transmission_Type       3045
          Car_Engine_Power_PS            0
          Car_Model                   7931
          Car_Mileage_Kilometer          0
          Month_of_Car_Registration   9225
```

```
Fuel_Type                   9718
Car_Brand                      0
UnRepaired_Damage              0
Date_Created                   0
Seller_Postal_Code             0
Date_LastSeen_Online           0
dtype: int64
```

```
In [101]: # https://datascience.stackexchange.com/questions/30249/how-to-delete-entire-row-if-
          # Remove rows with Month_of_Car_Registration Null/Nan values
          autos_cleaned_df = autos_cleaned_df[pd.notnull(autos_cleaned_df['Month_of_Car_Regist
```

```
In [102]: autos_cleaned_df.isnull().sum()
```

```
Out[102]: Car_Price                      0
          Vechicle_Type               4444
          Year_of_Car_Registration       0
          Car_Transmission_Type       2607
          Car_Engine_Power_PS            0
          Car_Model                   6905
          Car_Mileage_Kilometer          0
          Month_of_Car_Registration      0
          Fuel_Type                   7653
          Car_Brand                      0
          UnRepaired_Damage              0
          Date_Created                   0
          Seller_Postal_Code             0
          Date_LastSeen_Online           0
          dtype: int64
```

```
In [103]: # https://datascience.stackexchange.com/questions/30249/how-to-delete-entire-row-if-
          # Remove rows with Fuel_Type Null/Nan values
          autos_cleaned_df = autos_cleaned_df[pd.notnull(autos_cleaned_df['Fuel_Type'])]
```

```
In [104]: autos_cleaned_df.isnull().sum()
```

```
Out[104]: Car_Price                      0
          Vechicle_Type               2387
          Year_of_Car_Registration       0
          Car_Transmission_Type       2350
          Car_Engine_Power_PS            0
          Car_Model                   5702
          Car_Mileage_Kilometer          0
          Month_of_Car_Registration      0
          Fuel_Type                      0
          Car_Brand                      0
          UnRepaired_Damage              0
          Date_Created                   0
          Seller_Postal_Code             0
```

```
Date_LastSeen_Online            0
dtype: int64
```

```
In [105]: autos_cleaned_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 252182 entries, 1 to 371539
Data columns (total 14 columns):
Car_Price                    252182 non-null float64
Vechicle_Type                249795 non-null object
Year_of_Car_Registration     252182 non-null float64
Car_Transmission_Type        249832 non-null object
Car_Engine_Power_PS          252182 non-null float64
Car_Model                    246480 non-null object
Car_Mileage_Kilometer        252182 non-null object
Month_of_Car_Registration    252182 non-null float64
Fuel_Type                    252182 non-null object
Car_Brand                    252182 non-null object
UnRepaired_Damage            252182 non-null object
Date_Created                 252182 non-null datetime64[ns]
Seller_Postal_Code           252182 non-null float64
Date_LastSeen_Online         252182 non-null datetime64[ns]
dtypes: datetime64[ns](2), float64(5), object(7)
memory usage: 28.9+ MB
```

```
In [106]: # https://datascience.stackexchange.com/questions/30249/how-to-delete-entire-row-if-
          # Remove rows with Car_Model Null/Nan values
          autos_cleaned_df = autos_cleaned_df[pd.notnull(autos_cleaned_df['Car_Model'])]
```

```
In [107]: autos_cleaned_df.isnull().sum()
```

```
Out[107]: Car_Price                       0
          Vechicle_Type                2147
          Year_of_Car_Registration        0
          Car_Transmission_Type        2238
          Car_Engine_Power_PS             0
          Car_Model                       0
          Car_Mileage_Kilometer           0
          Month_of_Car_Registration       0
          Fuel_Type                       0
          Car_Brand                       0
          UnRepaired_Damage               0
          Date_Created                    0
          Seller_Postal_Code              0
          Date_LastSeen_Online            0
          dtype: int64
```

```
In [108]: # https://datascience.stackexchange.com/questions/30249/how-to-delete-entire-row-if-
          # Remove rows with Car_Price(Euros) Null/Nan values
          autos_cleaned_df = autos_cleaned_df[pd.notnull(autos_cleaned_df['Car_Transmission_Ty
```

```
In [109]: autos_cleaned_df.isnull().sum()

Out[109]: Car_Price                       0
          Vechicle_Type                2040
          Year_of_Car_Registration        0
          Car_Transmission_Type          0
          Car_Engine_Power_PS            0
          Car_Model                      0
          Car_Mileage_Kilometer          0
          Month_of_Car_Registration      0
          Fuel_Type                      0
          Car_Brand                      0
          UnRepaired_Damage              0
          Date_Created                   0
          Seller_Postal_Code             0
          Date_LastSeen_Online           0
          dtype: int64

In [110]: # https://datascience.stackexchange.com/questions/30249/how-to-delete-entire-row-if-
          # Remove rows with Car_Price(Euros) Null/Nan values
          autos_cleaned_df = autos_cleaned_df[pd.notnull(autos_cleaned_df['Vechicle_Type'])]

In [111]: autos_cleaned_df.isnull().sum()

Out[111]: Car_Price                     0
          Vechicle_Type                 0
          Year_of_Car_Registration      0
          Car_Transmission_Type         0
          Car_Engine_Power_PS           0
          Car_Model                     0
          Car_Mileage_Kilometer         0
          Month_of_Car_Registration     0
          Fuel_Type                     0
          Car_Brand                     0
          UnRepaired_Damage             0
          Date_Created                  0
          Seller_Postal_Code            0
          Date_LastSeen_Online          0
          dtype: int64
```

We finally have a completely cleaned dataset for further data analysis

```
In [112]: autos_cleaned_df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 242202 entries, 3 to 371539
Data columns (total 14 columns):
Car_Price                    242202 non-null float64
Vechicle_Type                242202 non-null object
```

```
Year_of_Car_Registration     242202 non-null float64
Car_Transmission_Type        242202 non-null object
Car_Engine_Power_PS          242202 non-null float64
Car_Model                    242202 non-null object
Car_Mileage_Kilometer        242202 non-null object
Month_of_Car_Registration    242202 non-null float64
Fuel_Type                    242202 non-null object
Car_Brand                    242202 non-null object
UnRepaired_Damage            242202 non-null object
Date_Created                 242202 non-null datetime64[ns]
Seller_Postal_Code           242202 non-null float64
Date_LastSeen_Online         242202 non-null datetime64[ns]
dtypes: datetime64[ns](2), float64(5), object(7)
memory usage: 27.7+ MB
```

```
In [113]: autos_cleaned_df.head(10)
```

Out[113]:

	Car_Price	Vechicle_Type	Year_of_Car_Registration	Car_Transmission_Type
3	1500.0	small car	2001.0	manual
4	3600.0	small car	2008.0	manual
5	650.0	limousine	1995.0	manual
6	2200.0	cabrio	2004.0	manual
10	2000.0	limousine	2004.0	manual
11	2799.0	station wagon	2005.0	manual
14	17999.0	suv	2011.0	manual
17	1750.0	small car	2004.0	automatic
18	7550.0	bus	2007.0	manual
19	1850.0	bus	2004.0	manual

	Car_Engine_Power_PS	Car_Model	Car_Mileage_Kilometer
3	75.0	golf	150000
4	69.0	fabia	90000
5	102.0	3er	150000
6	109.0	2_reihe	150000
10	105.0	3_reihe	150000
11	140.0	passat	150000
14	190.0	navara	70000
17	75.0	twingo	150000
18	136.0	c_max	150000
19	102.0	a_klasse	150000

	Month_of_Car_Registration	Fuel_Type	Car_Brand	UnRepaired_Damage
3	6.0	gas	volkswagen	no
4	7.0	diesel	skoda	no
5	10.0	gas	bmw	yes
6	8.0	gas	peugeot	no
10	12.0	gas	mazda	no

```
11                        12.0    diesel     volkswagen               yes
14                         3.0    diesel         nissan                no
17                         2.0       gas        renault                no
18                         6.0    diesel           ford                no
19                         1.0       gas  mercedes_benz                no

   Date_Created  Seller_Postal_Code Date_LastSeen_Online
3    2016-03-17             91074.0  2016-03-17 17:40:00
4    2016-03-31             60437.0  2016-04-06 10:17:00
5    2016-04-04             33775.0  2016-04-06 19:17:00
6    2016-04-01             67112.0  2016-04-05 18:18:00
10   2016-03-26             96224.0  2016-04-06 10:45:00
11   2016-04-07             57290.0  2016-04-07 10:25:00
14   2016-03-21              4177.0  2016-04-06 07:45:00
17   2016-03-20             65599.0  2016-04-06 13:16:00
18   2016-03-23             88361.0  2016-04-05 18:45:00
19   2016-04-01             49565.0  2016-04-05 22:46:00
```

In [114]:
```
# https://stackoverflow.com/questions/16923281/writing-a-pandas-dataframe-to-csv-fil
# Writing autos_cleaned_df dataframe to csv file
autos_cleaned_df.to_csv("autos_cleaned_data.csv", index=False)
```

0.10 Change columns to appropriate data types

In [115]:
```
#https://stackoverflow.com/questions/43956335/convert-float64-column-to-int64-in-pan
# convert columns from float64 to int64
autos_cleaned_df['Year_of_Car_Registration'] = autos_cleaned_df['Year_of_Car_Registra
autos_cleaned_df['Month_of_Car_Registration'] = autos_cleaned_df['Month_of_Car_Regist
autos_cleaned_df['Car_Mileage_Kilometer'] = autos_cleaned_df['Car_Mileage_Kilometer']
autos_cleaned_df['Seller_Postal_Code'] = autos_cleaned_df['Seller_Postal_Code'].asty
```

In [116]:
```
# Display data types
autos_cleaned_df.dtypes
```

Out[116]:
```
Car_Price                          float64
Vechicle_Type                       object
Year_of_Car_Registration             int64
Car_Transmission_Type               object
Car_Engine_Power_PS                float64
Car_Model                           object
Car_Mileage_Kilometer                int64
Month_of_Car_Registration            int64
Fuel_Type                           object
Car_Brand                           object
UnRepaired_Damage                   object
Date_Created                datetime64[ns]
Seller_Postal_Code                   int64
Date_LastSeen_Online        datetime64[ns]
dtype: object
```

```
In [117]: autos_cleaned_df.head(10)
```

Out[117]:

	Car_Price	Vechicle_Type	Year_of_Car_Registration	Car_Transmission_Type
3	1500.0	small car	2001	manual
4	3600.0	small car	2008	manual
5	650.0	limousine	1995	manual
6	2200.0	cabrio	2004	manual
10	2000.0	limousine	2004	manual
11	2799.0	station wagon	2005	manual
14	17999.0	suv	2011	manual
17	1750.0	small car	2004	automatic
18	7550.0	bus	2007	manual
19	1850.0	bus	2004	manual

	Car_Engine_Power_PS	Car_Model	Car_Mileage_Kilometer
3	75.0	golf	150000
4	69.0	fabia	90000
5	102.0	3er	150000
6	109.0	2_reihe	150000
10	105.0	3_reihe	150000
11	140.0	passat	150000
14	190.0	navara	70000
17	75.0	twingo	150000
18	136.0	c_max	150000
19	102.0	a_klasse	150000

	Month_of_Car_Registration	Fuel_Type	Car_Brand	UnRepaired_Damage
3	6	gas	volkswagen	no
4	7	diesel	skoda	no
5	10	gas	bmw	yes
6	8	gas	peugeot	no
10	12	gas	mazda	no
11	12	diesel	volkswagen	yes
14	3	diesel	nissan	no
17	2	gas	renault	no
18	6	diesel	ford	no
19	1	gas	mercedes_benz	no

	Date_Created	Seller_Postal_Code	Date_LastSeen_Online
3	2016-03-17	91074	2016-03-17 17:40:00
4	2016-03-31	60437	2016-04-06 10:17:00
5	2016-04-04	33775	2016-04-06 19:17:00
6	2016-04-01	67112	2016-04-05 18:18:00
10	2016-03-26	96224	2016-04-06 10:45:00
11	2016-04-07	57290	2016-04-07 10:25:00
14	2016-03-21	4177	2016-04-06 07:45:00
17	2016-03-20	65599	2016-04-06 13:16:00
18	2016-03-23	88361	2016-04-05 18:45:00

```
          19   2016-04-01               49565  2016-04-05 22:46:00
```

```
In [118]: autos_cleaned_df['Car_Mileage_Kilometer'] = autos_cleaned_df['Car_Mileage_Kilometer']
```

```
In [119]: # Frequency Counts of kilometer values
          autos_cleaned_df['Car_Mileage_Kilometer'].value_counts()
```

```
Out[119]: 150000    147385
          125000     27138
          100000     11425
          90000       9466
          80000       8556
          70000       7693
          60000       7027
          50000       6182
          40000       5223
          30000       4757
          20000       4147
          5000        1786
          10000       1417
          Name: Car_Mileage_Kilometer, dtype: int64
```

```
In [120]: # Frequency counts of Vehicle_Type
          autos_cleaned_df['Vechicle_Type'].value_counts()
```

```
Out[120]: limousine        71451
          small car        53480
          station wagon    49857
          bus              23474
          cabrio           17589
          coupe            13488
          suv              11463
          other             1400
          Name: Vechicle_Type, dtype: int64
```

```
In [121]: # https://stackoverflow.com/questions/47052126/add-numeric-column-to-pandas-datafram
          # Create numeric Vehicle_Type_id column based on Vechicle_Type
          autos_cleaned_df['Vechicle_Type_id'] = pd.Categorical(autos_cleaned_df.Vechicle_Type)
```

```
In [122]: autos_cleaned_df.head()
```

```
Out[122]:     Car_Price Vechicle_Type  Year_of_Car_Registration Car_Transmission_Type  \
          3      1500.0     small car                      2001                manual
          4      3600.0     small car                      2008                manual
          5       650.0     limousine                      1995                manual
          6      2200.0        cabrio                      2004                manual
          10     2000.0     limousine                      2004                manual

              Car_Engine_Power_PS Car_Model  Car_Mileage_Kilometer  \
```

```
3                  75.0      golf                150000
4                  69.0     fabia                 90000
5                 102.0       3er                150000
6                 109.0   2_reihe                150000
10                105.0   3_reihe                150000

    Month_of_Car_Registration Fuel_Type   Car_Brand UnRepaired_Damage  \
3                           6       gas  volkswagen                no
4                           7    diesel       skoda                no
5                          10       gas         bmw               yes
6                           8       gas     peugeot                no
10                         12       gas       mazda                no

   Date_Created  Seller_Postal_Code Date_LastSeen_Online  Vechicle_Type_id
3    2016-03-17               91074  2016-03-17 17:40:00                 5
4    2016-03-31               60437  2016-04-06 10:17:00                 5
5    2016-04-04               33775  2016-04-06 19:17:00                 3
6    2016-04-01               67112  2016-04-05 18:18:00                 1
10   2016-03-26               96224  2016-04-06 10:45:00                 3
```

In [123]:
```
# Frequency counts of Vehicle_Type_id
autos_cleaned_df['Vechicle_Type_id'].value_counts()
```

Out[123]:
```
3    71451
5    53480
6    49857
0    23474
1    17589
2    13488
7    11463
4     1400
Name: Vechicle_Type_id, dtype: int64
```

In [124]:
```
# Frequency counts of Car_Transmission_Type
autos_cleaned_df['Car_Transmission_Type'].value_counts()
```

Out[124]:
```
manual       184818
automatic     57384
Name: Car_Transmission_Type, dtype: int64
```

In [125]:
```
# https://stackoverflow.com/questions/47052126/add-numeric-column-to-pandas-datafram
# Create numeric Car_Transmission_Type_id column based on Car_Transmission_Type

autos_cleaned_df['Car_Transmission_Type_id'] = pd.Categorical(autos_cleaned_df.Car_T
```

In [126]:
```
# Display the few lines of the dataframe
autos_cleaned_df.head()
```

Out[126]:
```
    Car_Price Vechicle_Type  Year_of_Car_Registration Car_Transmission_Type  \
3      1500.0     small car                      2001                manual
```

```
4      3600.0     small car                    2008                manual
5       650.0     limousine                    1995                manual
6      2200.0        cabrio                    2004                manual
10     2000.0     limousine                    2004                manual

    Car_Engine_Power_PS Car_Model  Car_Mileage_Kilometer  \
3                  75.0      golf                 150000
4                  69.0     fabia                  90000
5                 102.0       3er                 150000
6                 109.0   2_reihe                 150000
10                105.0   3_reihe                 150000

    Month_of_Car_Registration Fuel_Type   Car_Brand UnRepaired_Damage  \
3                           6       gas  volkswagen                no
4                           7    diesel       skoda                no
5                          10       gas         bmw               yes
6                           8       gas     peugeot                no
10                         12       gas       mazda                no

   Date_Created  Seller_Postal_Code Date_LastSeen_Online  Vechicle_Type_id  \
3    2016-03-17               91074  2016-03-17 17:40:00                 5
4    2016-03-31               60437  2016-04-06 10:17:00                 5
5    2016-04-04               33775  2016-04-06 19:17:00                 3
6    2016-04-01               67112  2016-04-05 18:18:00                 1
10   2016-03-26               96224  2016-04-06 10:45:00                 3

    Car_Transmission_Type_id
3                          1
4                          1
5                          1
6                          1
10                         1
```

```
In [127]:  # Frequency counts of Car_Transmission_Type_id
           autos_cleaned_df['Car_Transmission_Type_id'].value_counts()

Out[127]: 1    184818
          0     57384
          Name: Car_Transmission_Type_id, dtype: int64
```

Car Transmission Type identity codes have been successfully created for manual and automatic Transmission Types. 1 is the categorical code for manual transmission and 0 is the categorical code for automatic transmission.

```
In [128]: # Frequency counts of Car_Model
          autos_cleaned_df['Car_Model'].value_counts().nlargest(20)

Out[128]: golf           19618
          other          18487
```

```
3er            14579
polo            7998
a4              7434
corsa           7403
passat          6999
astra           6991
5er             6487
c_klasse        6476
e_klasse        5543
a3              4782
a6              4483
focus           4136
transporter     3788
2_reihe         3623
fiesta          3612
1er             3303
a_klasse        2891
fortwo          2721
Name: Car_Model, dtype: int64
```

```
In [129]: # https://stackoverflow.com/questions/47052126/add-numeric-column-to-pandas-datafram
          # Create numeric Car_Model_id column based on Car_Model
          autos_cleaned_df['Car_Model_id'] = pd.Categorical(autos_cleaned_df.Car_Model).codes
```

```
In [130]: # Frequency counts of Car_Model_id
          autos_cleaned_df['Car_Model_id'].value_counts().nlargest(20)
```

```
Out[130]: 117    19618
          167    18487
          11     14579
          174     7998
          29      7434
          83      7403
          171     6999
          42      6991
          15      6487
          59      6476
          96      5543
          28      4782
          31      4483
          104     4136
          224     3788
          8       3623
          103     3612
          6       3303
          33      2891
          107     2721
          Name: Car_Model_id, dtype: int64
```

Car Model categorical codes have been successfully created and assigned to the Car Models. 117 is the categorical code for Volkswagen golf.

```
In [131]: # Frequency counts of Fuel_Type
          autos_cleaned_df['Fuel_Type'].value_counts()

Out[131]: gas                       155144
          diesel                     82721
          autogas                     3611
          compressed natural gas       429
          hybrid                       199
          electric                      50
          other                         48
          Name: Fuel_Type, dtype: int64

In [132]: # Create numeric Fuel_Type_id column based on Fuel_Type
          autos_cleaned_df['Fuel_Type_id'] = pd.Categorical(autos_cleaned_df.Fuel_Type).codes

In [133]: # Frequency counts of Fuel_Type_id
          autos_cleaned_df['Fuel_Type_id'].value_counts()

Out[133]: 4    155144
          2     82721
          0      3611
          1       429
          5       199
          3        50
          6        48
          Name: Fuel_Type_id, dtype: int64
```

categorical codes based on Fuel Type string column have been successfully created. 4 is the categorical code for gas while 2 is the categorical code for diesel.

```
In [134]: # Frequency counts of Car_Brand
          autos_cleaned_df['Car_Brand'].value_counts()

Out[134]: volkswagen       50437
          bmw              28647
          mercedes_benz    25143
          opel             23877
          audi             23516
          ford             15884
          renault          10369
          peugeot           7285
          fiat              5785
          seat              4706
          skoda             4355
          mazda             3745
          toyota            3494
```

```
citroen           3425
nissan            3275
smart             3147
mini              2799
hyundai           2635
volvo             2424
mitsubishi        1890
kia               1820
honda             1799
porsche           1677
alfa_romeo        1573
suzuki            1569
chevrolet         1216
chrysler           928
dacia              686
land_rover         602
jeep               575
subaru             503
daihatsu           462
jaguar             458
saab               396
daewoo             296
lancia             294
rover              240
trabant            147
lada               123
Name: Car_Brand, dtype: int64
```

```
In [135]: # Create numeric Car_Brand_id column based on Car_Brand
          autos_cleaned_df['Car_Brand_id'] = pd.Categorical(autos_cleaned_df.Car_Brand).codes
```

```
In [136]: # Frequency counts of Car_Brand_id
          autos_cleaned_df['Car_Brand_id'].value_counts().nlargest(20)
```

```
Out[136]: 37    50437
          2     28647
          20    25143
          24    23877
          1     23516
          10    15884
          27    10369
          25     7285
          9      5785
          30     4706
          31     4355
          19     3745
          35     3494
          5      3425
```

```
23     3275
32     3147
21     2799
12     2635
38     2424
22     1890
Name: Car_Brand_id, dtype: int64
```

The categorical codes for the three leading auto brands according to this dataset are 37 for volkswagen, 2 for BMW, and 20 for Mercedes Benz.

```
In [137]: # Frequency counts of UnRepaired_Damage
          autos_cleaned_df['UnRepaired_Damage'].value_counts()

Out[137]: no     219292
          yes     22910
          Name: UnRepaired_Damage, dtype: int64

In [138]: # Create numeric UnRepaired_Damage_id column based on UnRepaired_Damage
          autos_cleaned_df['UnRepaired_Damage_id'] = pd.Categorical(autos_cleaned_df.UnRepaire

In [139]: # Frequency counts of UnRepaired_Damage_id
          autos_cleaned_df['UnRepaired_Damage_id'].value_counts()

Out[139]: 0    219292
          1     22910
          Name: UnRepaired_Damage_id, dtype: int64
```

Categorical Code 0 has been assigned to no while 1 is assigned to yes.

```
In [140]: # Display data types
          autos_cleaned_df.dtypes

Out[140]: Car_Price                        float64
          Vechicle_Type                     object
          Year_of_Car_Registration           int64
          Car_Transmission_Type             object
          Car_Engine_Power_PS              float64
          Car_Model                         object
          Car_Mileage_Kilometer              int64
          Month_of_Car_Registration          int64
          Fuel_Type                         object
          Car_Brand                         object
          UnRepaired_Damage                 object
          Date_Created                datetime64[ns]
          Seller_Postal_Code                 int64
          Date_LastSeen_Online        datetime64[ns]
          Vechicle_Type_id                    int8
          Car_Transmission_Type_id            int8
```

```
Car_Model_id                        int16
Fuel_Type_id                         int8
Car_Brand_id                         int8
UnRepaired_Damage_id                 int8
dtype: object
```

In [141]:
```
# Display the few lines of the dataframe
autos_cleaned_df.head()
```

Out[141]:

	Car_Price	Vechicle_Type	Year_of_Car_Registration	Car_Transmission_Type
3	1500.0	small car	2001	manual
4	3600.0	small car	2008	manual
5	650.0	limousine	1995	manual
6	2200.0	cabrio	2004	manual
10	2000.0	limousine	2004	manual

	Car_Engine_Power_PS	Car_Model	Car_Mileage_Kilometer
3	75.0	golf	150000
4	69.0	fabia	90000
5	102.0	3er	150000
6	109.0	2_reihe	150000
10	105.0	3_reihe	150000

	Month_of_Car_Registration	Fuel_Type	Car_Brand	UnRepaired_Damage
3	6	gas	volkswagen	no
4	7	diesel	skoda	no
5	10	gas	bmw	yes
6	8	gas	peugeot	no
10	12	gas	mazda	no

	Date_Created	Seller_Postal_Code	Date_LastSeen_Online	Vechicle_Type_id
3	2016-03-17	91074	2016-03-17 17:40:00	5
4	2016-03-31	60437	2016-04-06 10:17:00	5
5	2016-04-04	33775	2016-04-06 19:17:00	3
6	2016-04-01	67112	2016-04-05 18:18:00	1
10	2016-03-26	96224	2016-04-06 10:45:00	3

	Car_Transmission_Type_id	Car_Model_id	Fuel_Type_id	Car_Brand_id
3	1	117	4	37
4	1	102	2	31
5	1	11	4	2
6	1	8	4	25
10	1	10	4	19

	UnRepaired_Damage_id
3	0
4	0
5	1

```
6                     0
10                    0
```

In [142]:
```
# https://stackoverflow.com/questions/20490274/how-to-reset-index-in-a-pandas-data-f
# Reset the index of this dataframe
autos_cleaned_df = autos_cleaned_df.reset_index()
```

In [143]:
```
# Display the few lines of the dataframe
autos_cleaned_df.head()
```

Out[143]:

	index	Car_Price	Vechicle_Type	Year_of_Car_Registration	Car_Transmission_Type	Car_Engine_Power_PS	Car_Model	Car_Mileage_Kilometer	Month_of_Car_Registration	Fuel_Type	...	UnRepaired_Damage	Date_Created	Seller_Postal_Code	Date_LastSeen_Online	Vechicle_Type_id	Car_Transmission_Type_id	Car_Model_id	Fuel_Type_id	Car_Brand_id	UnRepaired_Damage_id
0	3	1500.0	small car	2001	manual	75.0	golf	150000	6	gas	...	no	2016-03-17	91074	2016-03-17 17:40:00	5	1	117	4	37	0
1	4	3600.0	small car	2008	manual	69.0	fabia	90000	7	diesel	...	no	2016-03-31	60437	2016-04-06 10:17:00	5	1	102	2	31	0
2	5	650.0	limousine	1995	manual	102.0	3er	150000	10	gas	...	yes	2016-04-04	33775	2016-04-06 19:17:00	3	1	11	4		
3	6	2200.0	cabrio	2004	manual	109.0	2_reihe	150000	8	gas	...	no	2016-04-01	67112	2016-04-05 18:18:00	1	1	8	4		
4	10	2000.0	limousine	2004	manual	105.0	3_reihe	150000	12	gas	...	no	2016-03-26	96224	2016-04-06 10:45:00	3	1	10	4		

```
2              2                    1
3             25                    0
4             19                    0

[5 rows x 21 columns]
```

In [144]:
```
# Delete the index column
del autos_cleaned_df['index']
```

In [145]:
```
# Display the few lines of the dataframe
autos_cleaned_df.head()
```

Out[145]:

	Car_Price	Vechicle_Type	Year_of_Car_Registration	Car_Transmission_Type
0	1500.0	small car	2001	manual
1	3600.0	small car	2008	manual
2	650.0	limousine	1995	manual
3	2200.0	cabrio	2004	manual
4	2000.0	limousine	2004	manual

	Car_Engine_Power_PS	Car_Model	Car_Mileage_Kilometer
0	75.0	golf	150000
1	69.0	fabia	90000
2	102.0	3er	150000
3	109.0	2_reihe	150000
4	105.0	3_reihe	150000

	Month_of_Car_Registration	Fuel_Type	Car_Brand	UnRepaired_Damage
0	6	gas	volkswagen	no
1	7	diesel	skoda	no
2	10	gas	bmw	yes
3	8	gas	peugeot	no
4	12	gas	mazda	no

	Date_Created	Seller_Postal_Code	Date_LastSeen_Online	Vechicle_Type_id
0	2016-03-17	91074	2016-03-17 17:40:00	5
1	2016-03-31	60437	2016-04-06 10:17:00	5
2	2016-04-04	33775	2016-04-06 19:17:00	3
3	2016-04-01	67112	2016-04-05 18:18:00	1
4	2016-03-26	96224	2016-04-06 10:45:00	3

	Car_Transmission_Type_id	Car_Model_id	Fuel_Type_id	Car_Brand_id
0	1	117	4	37
1	1	102	2	31
2	1	11	4	2
3	1	8	4	25
4	1	10	4	19

	UnRepaired_Damage_id

```
0                    0
1                    0
2                    1
3                    0
4                    0
```

```
In [146]: # https://stackoverflow.com/questions/16923281/writing-a-pandas-dataframe-to-csv-fil
          # Writing autos_cleaned_df dataframe to csv file
          autos_cleaned_df.to_csv("autos_cleaned_data_with_categorical_columns.csv", index=Fals
```

0.11 Drop columns not needed for further data analysis

Before proceeding to the Exploratory Data Analysis, I will drop Date_Created, Seller_Postal_Code, and Date_LastSeen_Online from further consideration for data analysis.

```
In [147]: #Drop Date_Created, Seller_Postal_Code, and Date_LastSeen_Online Columns
          autos_cleaned_data_df = autos_cleaned_df.drop(['Date_Created','Seller_Postal_Code',
```

```
In [148]: # Display the few lines of the dataframe
          autos_cleaned_data_df.head()
```

Out[148]:

	Car_Price	Vechicle_Type	Year_of_Car_Registration	Car_Transmission_Type	Car_Engine_Power_PS	Car_Model	Car_Mileage_Kilometer	Month_of_Car_Registration	Fuel_Type	Car_Brand	UnRepaired_Damage	Vechicle_Type_id	Car_Transmission_Type_id	Car_Model_id	Fuel_Type_id
0	1500.0	small car	2001	manual	75.0	golf	150000	6	gas	volkswagen	no	5	1	117	4
1	3600.0	small car	2008	manual	69.0	fabia	90000	7	diesel	skoda	no	5	1	102	2
2	650.0	limousine	1995	manual	102.0	3er	150000	10	gas	bmw	yes	3	1	11	4
3	2200.0	cabrio	2004	manual	109.0	2_reihe	150000	8	gas	peugeot	no	1	1	8	4
4	2000.0	limousine	2004	manual	105.0	3_reihe	150000	12	gas	mazda	no	3	1	10	4

```
   Car_Brand_id  UnRepaired_Damage_id
0            37                     0
1            31                     0
2             2                     1
3            25                     0
4            19                     0
```

In [149]:
```
# Display data types
autos_cleaned_data_df.dtypes
```

Out[149]:
```
Car_Price                    float64
Vechicle_Type                 object
Year_of_Car_Registration       int64
Car_Transmission_Type         object
Car_Engine_Power_PS          float64
Car_Model                     object
Car_Mileage_Kilometer          int64
Month_of_Car_Registration      int64
Fuel_Type                     object
Car_Brand                     object
UnRepaired_Damage             object
Vechicle_Type_id                int8
Car_Transmission_Type_id        int8
Car_Model_id                   int16
Fuel_Type_id                    int8
Car_Brand_id                    int8
UnRepaired_Damage_id            int8
dtype: object
```

In [150]:
```
# Display DataFrame Summary
autos_cleaned_data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 242202 entries, 0 to 242201
Data columns (total 17 columns):
Car_Price                    242202 non-null float64
Vechicle_Type                242202 non-null object
Year_of_Car_Registration     242202 non-null int64
Car_Transmission_Type        242202 non-null object
Car_Engine_Power_PS          242202 non-null float64
Car_Model                    242202 non-null object
Car_Mileage_Kilometer        242202 non-null int64
Month_of_Car_Registration    242202 non-null int64
Fuel_Type                    242202 non-null object
Car_Brand                    242202 non-null object
UnRepaired_Damage            242202 non-null object
Vechicle_Type_id             242202 non-null int8
Car_Transmission_Type_id     242202 non-null int8
Car_Model_id                 242202 non-null int16
```

```
Fuel_Type_id                242202 non-null int8
Car_Brand_id                242202 non-null int8
UnRepaired_Damage_id        242202 non-null int8
dtypes: float64(2), int16(1), int64(3), int8(5), object(6)
memory usage: 21.9+ MB
```

The dataset is completely clean with no missing value, we can now proceed to the exploratory data analysis(EDA) and visualization part of this project.

0.12 Exploratory Data Analysis

In [151]:
```
# Descriptive or Summary statistics of numeric columns
autos_cleaned_data_df.describe()
```

Out[151]:

	Car_Price	Year_of_Car_Registration	Car_Engine_Power_PS
count	242202.000000	242202.000000	242202.000000
mean	6877.679239	2003.599326	129.682269
std	8073.523840	6.338326	62.153342
min	1.000000	1951.000000	1.000000
25%	1700.000000	2000.000000	86.000000
50%	4000.000000	2004.000000	116.000000
75%	8990.000000	2008.000000	160.000000
max	99999.000000	2016.000000	999.000000

	Car_Mileage_Kilometer	Month_of_Car_Registration	Vechicle_Type_id
count	242202.000000	242202.000000	242202.000000
mean	123474.702934	6.367309	3.762566
std	39931.509539	3.350762	2.033681
min	5000.000000	1.000000	0.000000
25%	100000.000000	3.000000	3.000000
50%	150000.000000	6.000000	3.000000
75%	150000.000000	9.000000	6.000000
max	150000.000000	12.000000	7.000000

	Car_Transmission_Type_id	Car_Model_id	Fuel_Type_id	Car_Brand_id
count	242202.000000	242202.000000	242202.000000	242202.000000
mean	0.763074	108.821339	3.252987	19.883969
std	0.425198	72.091149	1.031713	13.161958
min	0.000000	0.000000	0.000000	0.000000
25%	1.000000	36.000000	2.000000	8.000000
50%	1.000000	107.000000	4.000000	21.000000
75%	1.000000	167.000000	4.000000	32.000000
max	1.000000	249.000000	6.000000	38.000000

	UnRepaired_Damage_id
count	242202.000000
mean	0.094590

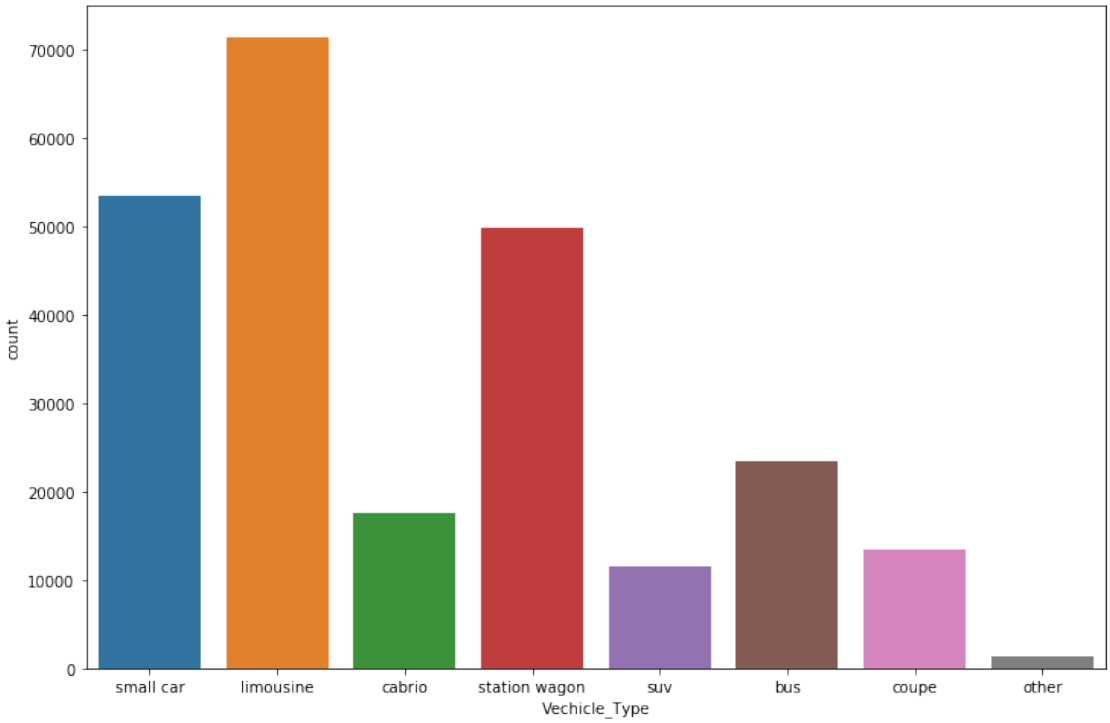
```
std                 0.292649
min                 0.000000
25%                 0.000000
50%                 0.000000
75%                 0.000000
max                 1.000000
```

```
In [152]: # Summary statistics of character or non-numeric columns
          autos_cleaned_data_df.describe(include=['object'])
```

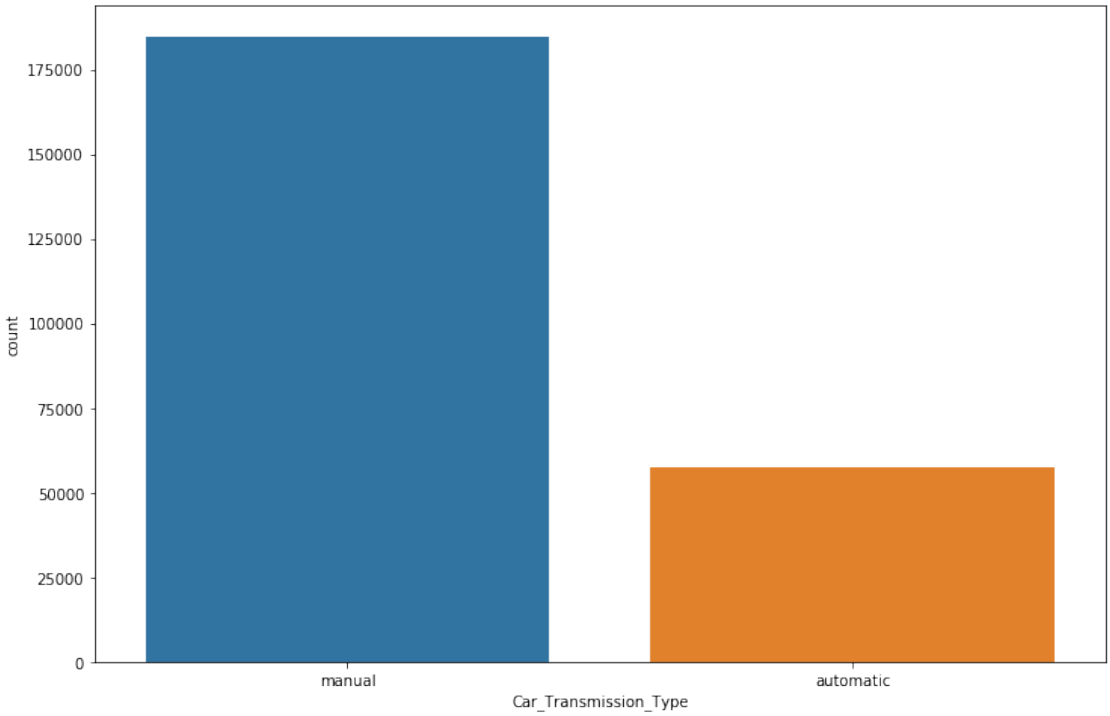
```
Out[152]:        Vechicle_Type Car_Transmission_Type Car_Model Fuel_Type   Car_Brand  \
          count         242202                242202    242202    242202      242202
          unique             8                     2       250         7          39
          top        limousine                manual      golf       gas  volkswagen
          freq           71451                184818     19618    155144       50437

                 UnRepaired_Damage
          count               242202
          unique                   2
          top                     no
          freq                219292
```

The above shows limousine, manual, golf, gas and volkswagen are the top Vehicle Type, Car Transmission Type, Car Model, Fuel Type and Car Brand respectively.

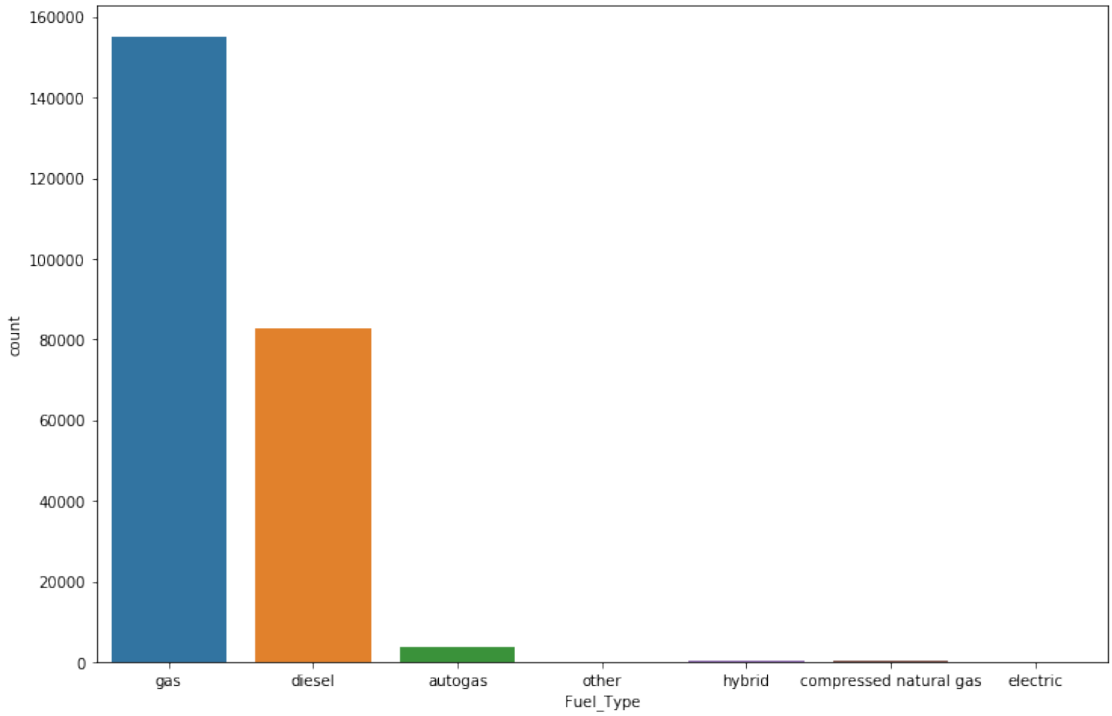
```
In [232]: # Create countplot of Vehicle Type
          # https://stackoverflow.com/questions/42528921/how-to-prevent-overlapping-x-axis-lab
          plt.figure(figsize=(12,8))
          sns.countplot(x= 'Vechicle_Type', data = autos_cleaned_data_df)
          plt.savefig('Vehicle Type Frequency plot.pdf')
```



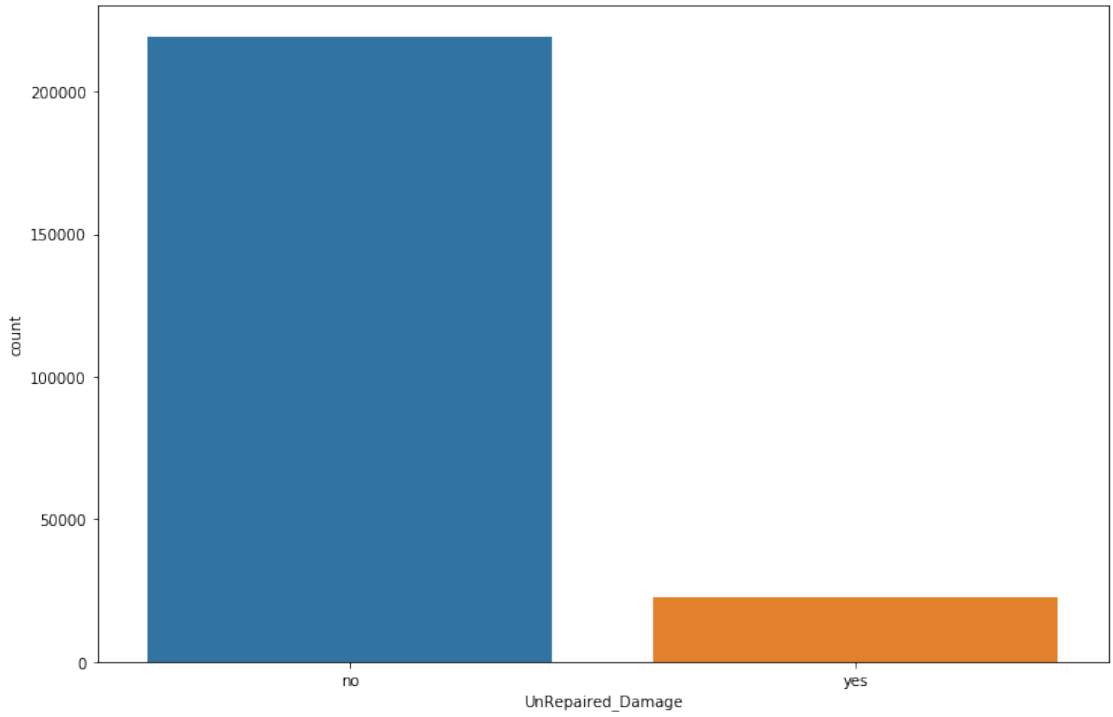
In [233]:
```
# Create countplot of Car_Transmission_Type
# https://stackoverflow.com/questions/42528921/how-to-prevent-overlapping-x-axis-lab
plt.figure(figsize=(12,8))
sns.countplot(x= 'Car_Transmission_Type', data = autos_cleaned_data_df)
plt.savefig('Car Transmission Type plot.pdf')
```




```
In [155]: # Frequency counts of Fuel_Type
          autos_cleaned_data_df['Fuel_Type'].value_counts()

Out[155]: gas                       155144
          diesel                     82721
          autogas                     3611
          compressed natural gas       429
          hybrid                       199
          electric                      50
          other                         48
          Name: Fuel_Type, dtype: int64

In [234]: # Create countplot of Fuel_Type
          # https://stackoverflow.com/questions/42528921/how-to-prevent-overlapping-x-axis-lab
          plt.figure(figsize=(12,8))
          sns.countplot(x= 'Fuel_Type', data = autos_cleaned_data_df)
          plt.savefig('Fuel Type plot.pdf')
```

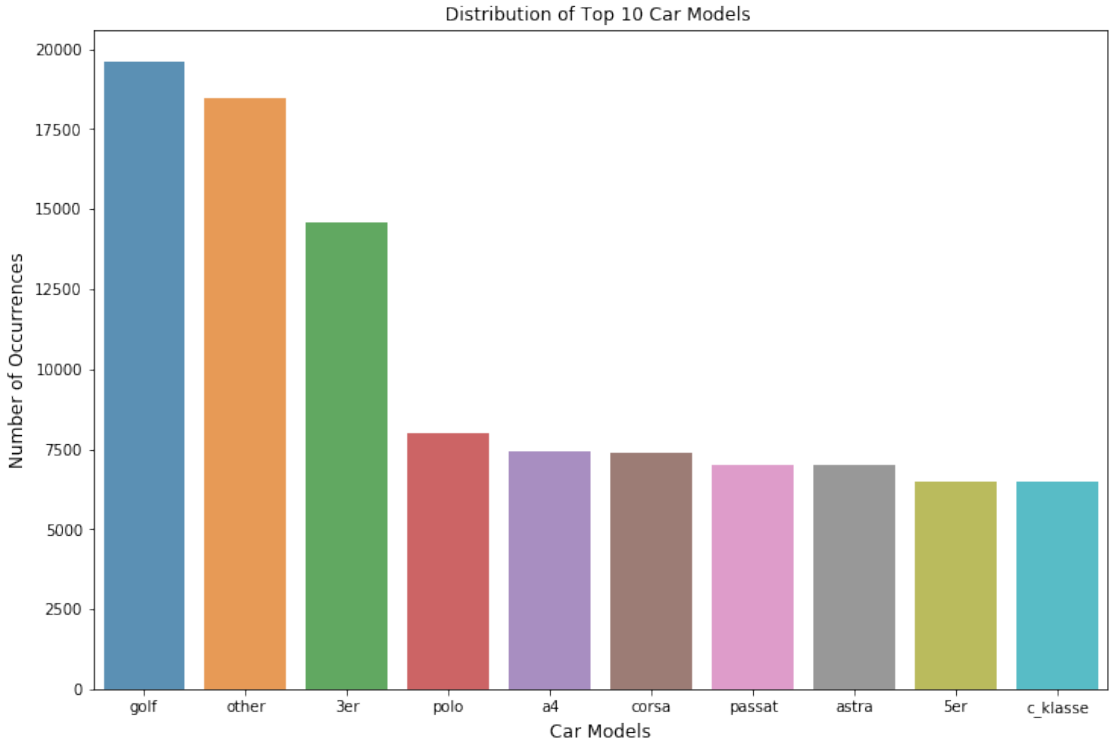



```
In [157]: # Frequency counts of UnRepaired_Damage
          autos_cleaned_data_df['UnRepaired_Damage'].value_counts()

Out[157]: no     219292
          yes     22910
          Name: UnRepaired_Damage, dtype: int64

In [235]: # Create countplot of UnRepaired_Damage
          plt.figure(figsize=(12,8))
          sns.countplot(x= 'UnRepaired_Damage', data = autos_cleaned_data_df)
          plt.savefig('UnRepaired Damage plot.pdf')
```




```
In [159]: # Frequency counts of the top 10 Car Models
          autos_cleaned_data_df['Car_Model'].value_counts().nlargest(10)

Out[159]: golf         19618
          other        18487
          3er          14579
          polo          7998
          a4            7434
          corsa         7403
          passat        6999
          astra         6991
          5er           6487
          c_klasse      6476
          Name: Car_Model, dtype: int64

In [236]: # https://www.kaggle.com/tejainece/seaborn-barplot-and-pandas-value-counts
          # Plotting a bar graph of the top 10 Car Models

          Car_Model_Count  = autos_cleaned_data_df['Car_Model'].value_counts()
          Car_Model_Count = Car_Model_Count[:10,]
          plt.figure(figsize=(12,8))
          sns.barplot(Car_Model_Count.index, Car_Model_Count.values, alpha=0.8)
          plt.title('Distribution of Top 10 Car Models ')
          plt.ylabel('Number of Occurrences', fontsize=12)
```

```
        plt.xlabel('Car Models', fontsize=12)
        plt.show()
        plt.savefig('Top 10 Car Models plot.pdf')
```



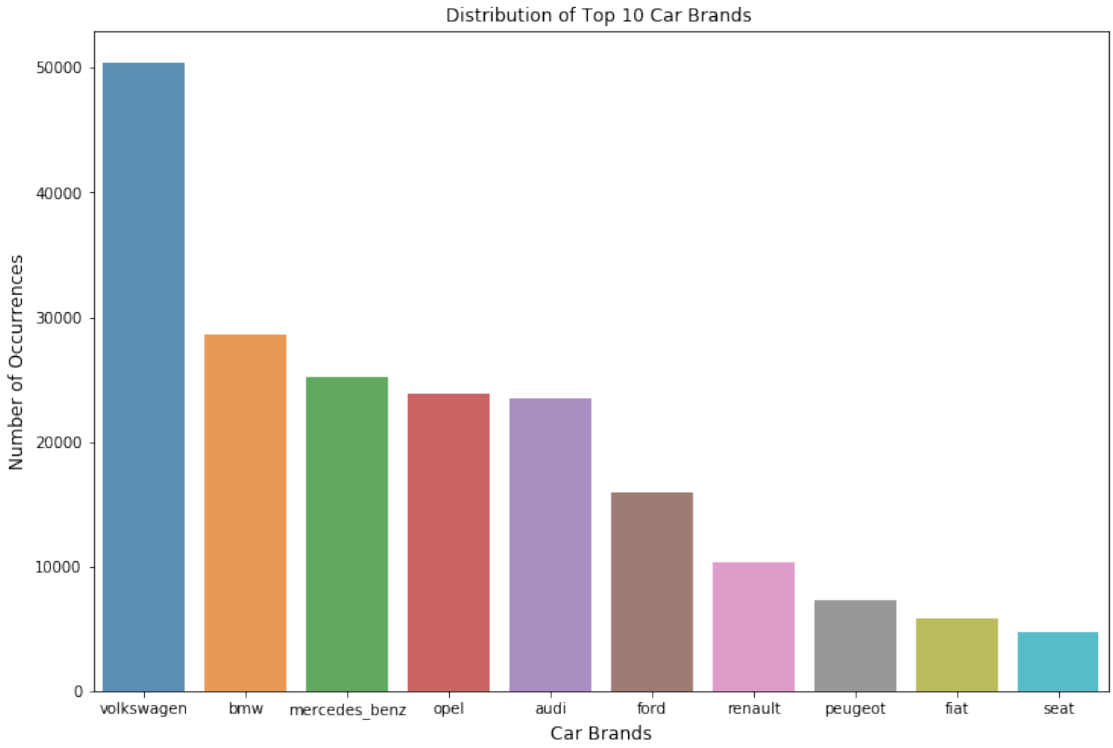

```
<Figure size 432x288 with 0 Axes>
```

```
In [161]: # Frequency counts of Car_Brand
          autos_cleaned_data_df['Car_Brand'].value_counts().nlargest(10)
```

```
Out[161]: volkswagen       50437
          bmw              28647
          mercedes_benz    25143
          opel             23877
          audi             23516
          ford             15884
          renault          10369
          peugeot           7285
          fiat              5785
          seat              4706
          Name: Car_Brand, dtype: int64
```

```
In [237]: # https://www.kaggle.com/tejainece/seaborn-barplot-and-pandas-value-counts
          # Plotting a bar graph of the top 10 Car Brands
```

```
Car_Brand_Count  = autos_cleaned_data_df['Car_Brand'].value_counts()
Car_Brand_Count = Car_Brand_Count[:10,]
plt.figure(figsize=(12,8))
sns.barplot(Car_Brand_Count.index, Car_Brand_Count.values, alpha=0.8)
plt.title('Distribution of Top 10 Car Brands ')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Car Brands', fontsize=12)
plt.show()
plt.savefig('Top 10 Car Brands plot.pdf')
```



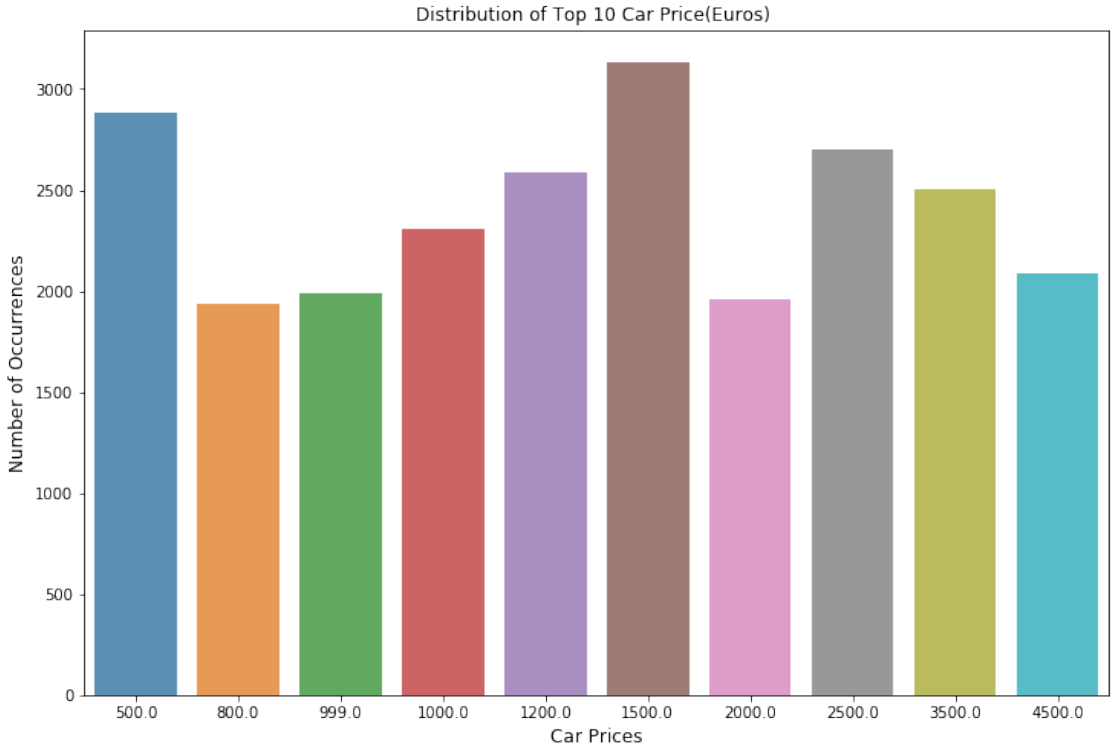

```
<Figure size 432x288 with 0 Axes>
```

In [163]:
```
# Frequency counts of Car_Price in Euros
autos_cleaned_data_df['Car_Price'].value_counts().nlargest(10)
```

Out[163]:
```
1500.0    3135
500.0     2880
2500.0    2700
1200.0    2587
3500.0    2507
1000.0    2306
```

```
4500.0    2089
999.0     1991
2000.0    1960
800.0     1938
Name: Car_Price, dtype: int64
```

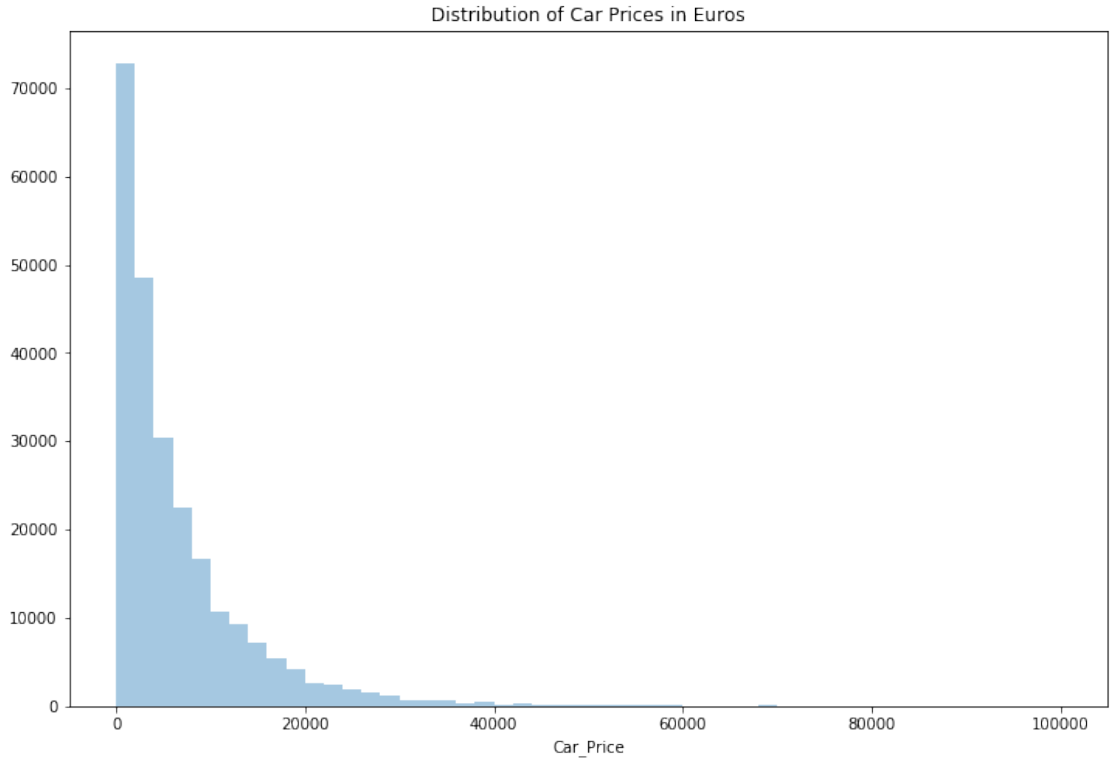
In [241]:
```
# https://www.kaggle.com/tejainece/seaborn-barplot-and-pandas-value-counts
# Plotting a bar graph of the top 10 Car Prices

Car_Price_Count = autos_cleaned_data_df['Car_Price'].value_counts()
Car_Price_Count = Car_Price_Count[:10,]
plt.figure(figsize=(12,8))
sns.barplot(Car_Price_Count.index, Car_Price_Count.values, alpha=0.8)
plt.title('Distribution of Top 10 Car Price(Euros) ')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Car Prices', fontsize=12)
plt.show()
plt.savefig('Top 10 Car Prices plot.png')
```




```
<Figure size 432x288 with 0 Axes>
```

In [167]:
```
# Make default histogram of Car_Price(Euros)
plt.figure(figsize=(12,8))
```

```
sns.distplot( autos_cleaned_data_df['Car_Price'], kde=False )
plt.title('Distribution of Car Prices in Euros')
plt.show()
```

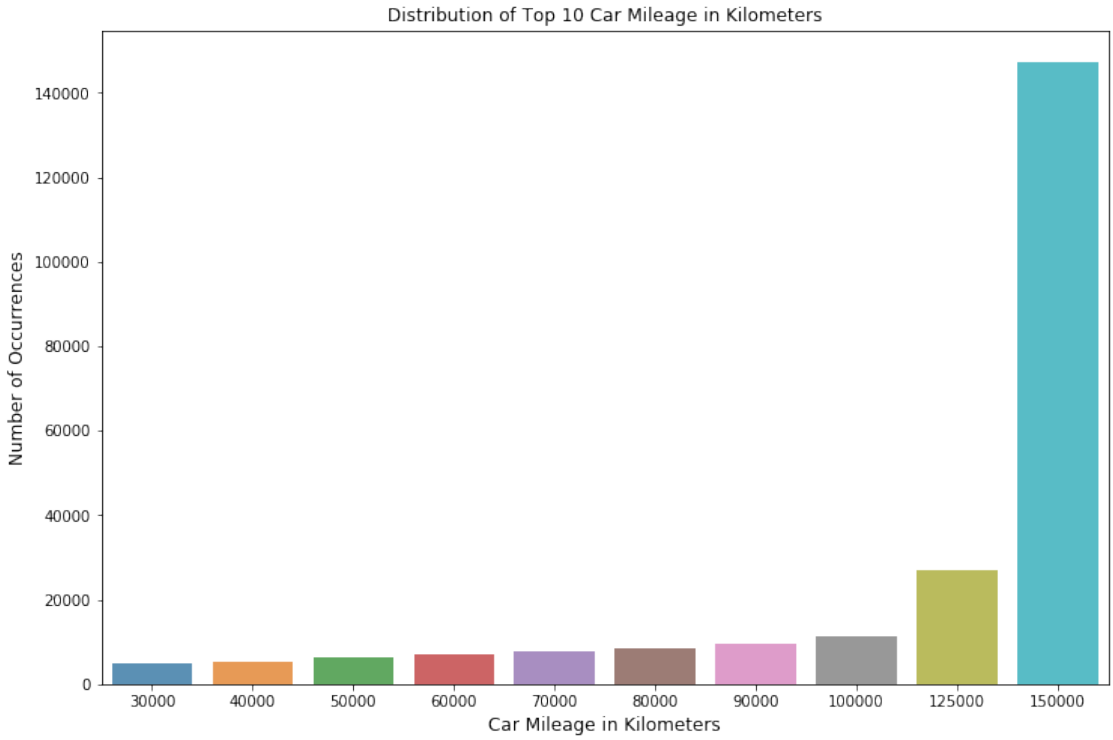


In [168]:
```
# Frequency counts of Car_Mileage_Kilometer
autos_cleaned_data_df['Car_Mileage_Kilometer'].value_counts().nlargest(10)
```

Out[168]:
```
150000    147385
125000     27138
100000     11425
90000       9466
80000       8556
70000       7693
60000       7027
50000       6182
40000       5223
30000       4757
Name: Car_Mileage_Kilometer, dtype: int64
```

In [242]:
```
# https://www.kaggle.com/tejainece/seaborn-barplot-and-pandas-value-counts
# Plotting a bar graph of the top 10 Car Mileage in Kilometers

Car_Mileage_Kilometer_Count = autos_cleaned_data_df['Car_Mileage_Kilometer'].value_co
```

```
Car_Mileage_Kilometer_Count = Car_Mileage_Kilometer_Count[:10,]
plt.figure(figsize=(12,8))
sns.barplot(Car_Mileage_Kilometer_Count.index, Car_Mileage_Kilometer_Count.values, al
plt.title('Distribution of Top 10 Car Mileage in Kilometers')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Car Mileage in Kilometers', fontsize=12)
plt.show()
plt.savefig('Car Mileage plot.pdf')
```



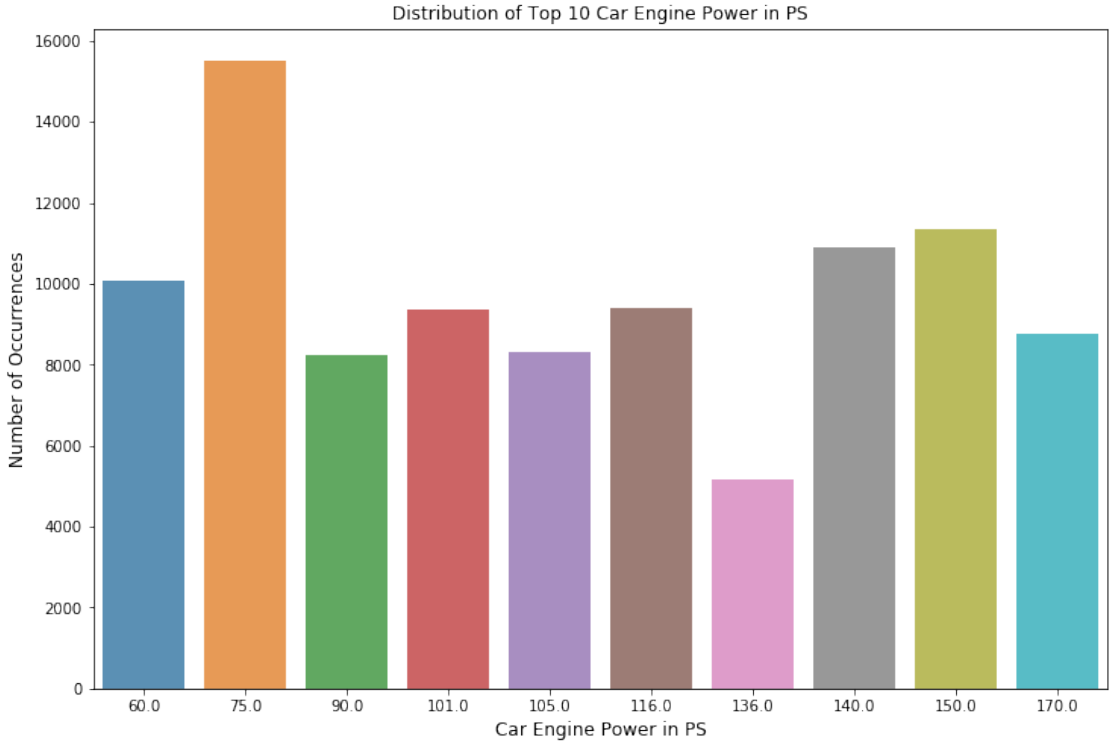

```
<Figure size 432x288 with 0 Axes>
```

In [170]:
```
# Frequency counts of Car_Engine_Power_PS
autos_cleaned_data_df['Car_Engine_Power_PS'].value_counts().nlargest(10)
```

Out[170]:
```
75.0     15519
150.0    11343
140.0    10883
60.0     10087
116.0     9407
101.0     9368
170.0     8773
105.0     8302
90.0      8243
```

```
136.0     5161
Name: Car_Engine_Power_PS, dtype: int64
```

In [171]:
```
# https://www.kaggle.com/tejainece/seaborn-barplot-and-pandas-value-counts
# Plotting a bar graph of the top 10 Car Engine Power in PS

Car_Engine_Power_PS_Count = autos_cleaned_data_df['Car_Engine_Power_PS'].value_counts
Car_Engine_Power_PS_Count = Car_Engine_Power_PS_Count[:10,]
plt.figure(figsize=(12,8))
sns.barplot(Car_Engine_Power_PS_Count.index, Car_Engine_Power_PS_Count.values, alpha=
plt.title('Distribution of Top 10 Car Engine Power in PS')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Car Engine Power in PS', fontsize=12)
plt.show()
```

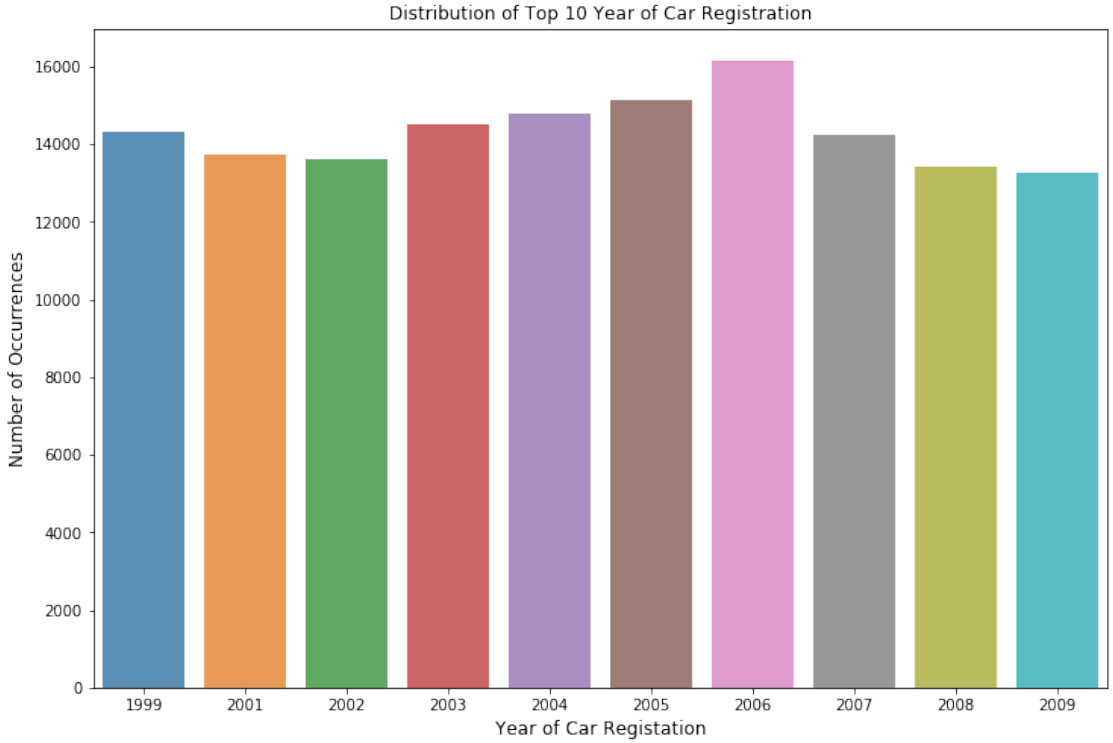


In [172]:
```
# Frequency counts of Year_of_Car_Registration
autos_cleaned_data_df['Year_of_Car_Registration'].value_counts().nlargest(10)
```

Out[172]:
```
2006    16164
2005    15135
2004    14775
2003    14499
1999    14297
2007    14256
```

```
2001    13737
2002    13594
2008    13428
2009    13254
Name: Year_of_Car_Registration, dtype: int64
```

In [244]:
```
# https://www.kaggle.com/tejainece/seaborn-barplot-and-pandas-value-counts
# Plotting a bar graph of the top 10 Year of Car Registration
Year_of_Car_Registration_Count = autos_cleaned_data_df['Year_of_Car_Registration'].va
Year_of_Car_Registration_Count = Year_of_Car_Registration_Count[:10,]
plt.figure(figsize=(12,8))
sns.barplot(Year_of_Car_Registration_Count.index, Year_of_Car_Registration_Count.valu
plt.title('Distribution of Top 10 Year of Car Registration')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Year of Car Registation', fontsize=12)
plt.show()
plt.savefig('Top 10 Years of Car Registration plot.pdf')
```

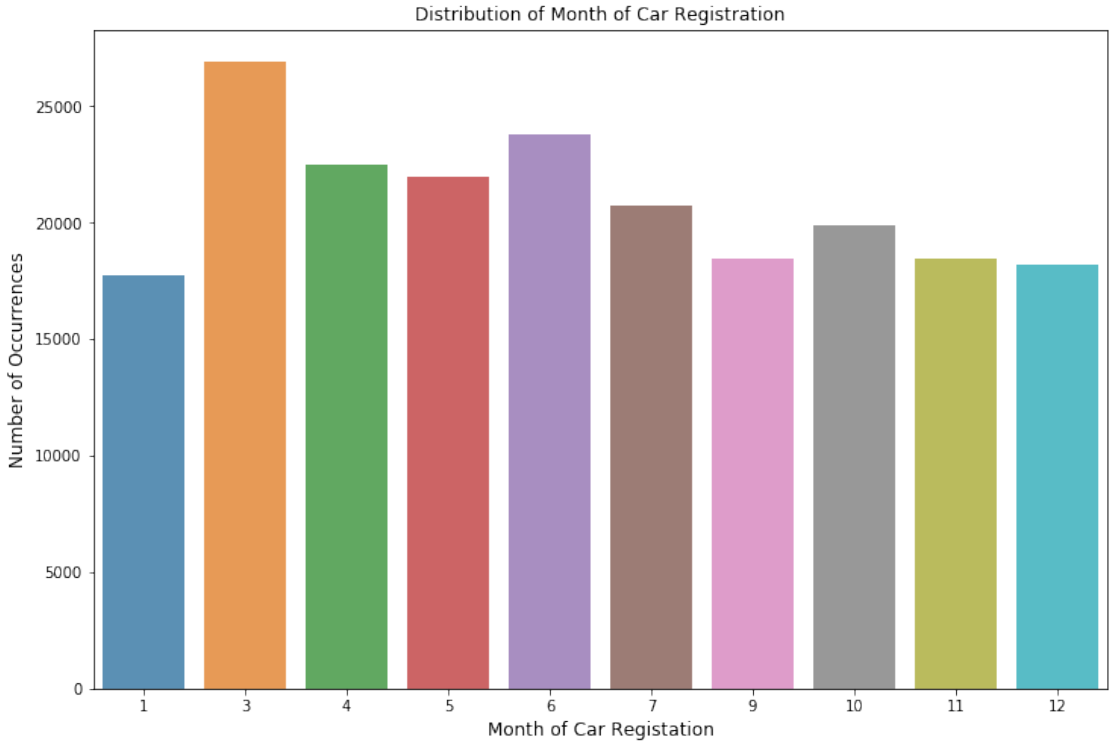



```
<Figure size 432x288 with 0 Axes>
```

In [174]:
```
# Frequency counts of Month_of_Car_Registration
autos_cleaned_data_df['Month_of_Car_Registration'].value_counts()
```

```
Out[174]: 3     26933
          6     23776
          4     22509
          5     21968
          7     20712
          10    19879
          11    18460
          9     18447
          12    18221
          1     17727
          8     17078
          2     16492
          Name: Month_of_Car_Registration, dtype: int64
```

```
In [175]: Month_of_Car_Registration_Count = autos_cleaned_data_df['Month_of_Car_Registration']
          Month_of_Car_Registration_Count = Month_of_Car_Registration_Count[:10,]
          plt.figure(figsize=(12,8))
          sns.barplot(Month_of_Car_Registration_Count.index, Month_of_Car_Registration_Count.va
          plt.title('Distribution of Month of Car Registration')
          plt.ylabel('Number of Occurrences', fontsize=12)
          plt.xlabel('Month of Car Registation', fontsize=12)
          plt.show()
```



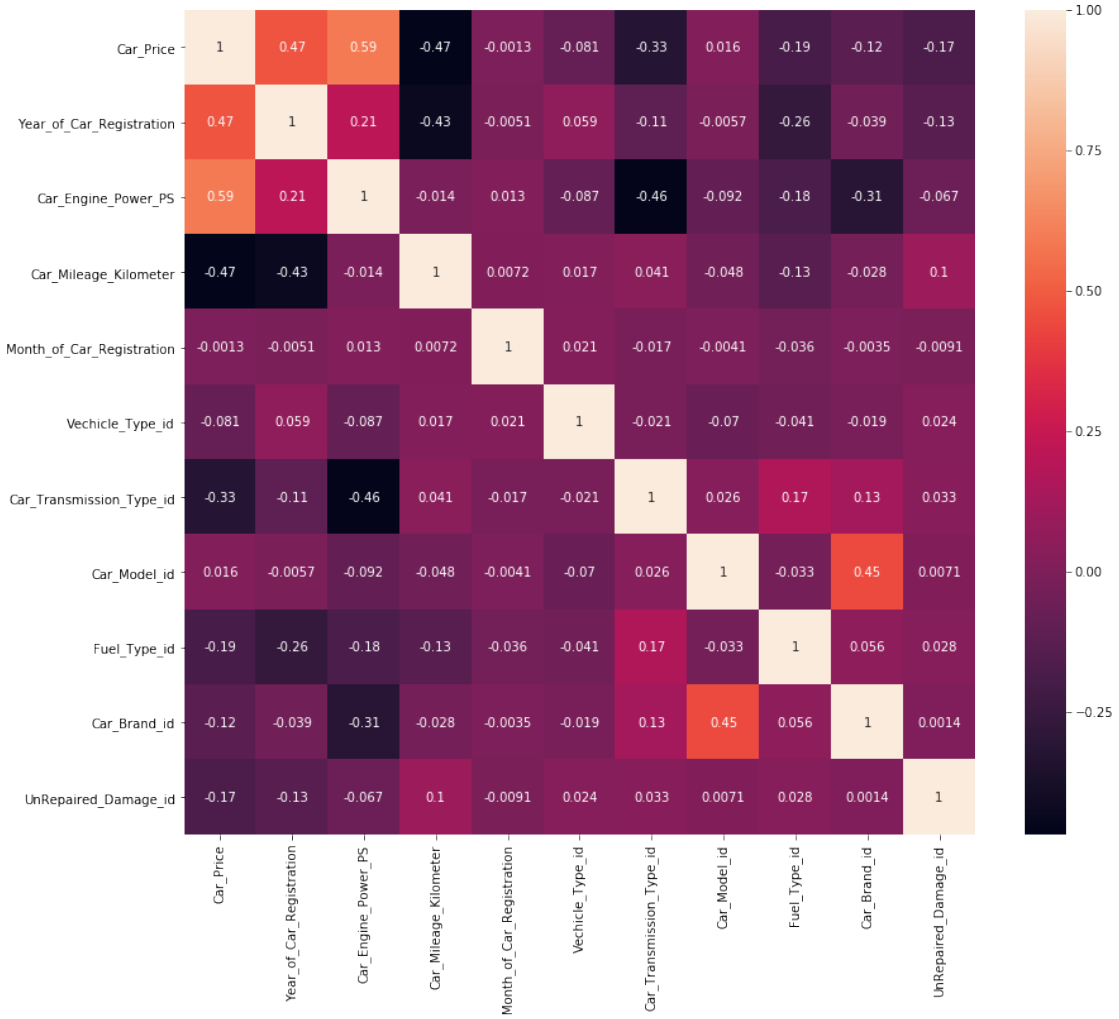
0.13 Correlation

Essentially, correlation would facilitate undertanding of the relationships among the auto attributes, especially with the price of the car.

In [176]:
```
# Drop the string columns
autos_cleaned_data_df2 = autos_cleaned_data_df.drop(['Vechicle_Type','Car_Transmissio
```

In [177]:
```
# Correlation of Data - Visualize potential relationship among the auto attributes
fig, ax = plt.subplots(figsize=(14,12))
sns.heatmap(autos_cleaned_data_df2.corr(), annot=True)
```

Out[177]: <matplotlib.axes._subplots.AxesSubplot at 0x2212f588>



In [178]:
```
# Correlation using .corr() method - Matrix table of relationship among the autos at
autos_cleaned_data_df2.corr()
```

Out[178]:

	Car_Price	Year_of_Car_Registration
Car_Price	1.000000	0.474764
Year_of_Car_Registration	0.474764	1.000000
Car_Engine_Power_PS	0.591456	0.208647
Car_Mileage_Kilometer	-0.469760	-0.426956
Month_of_Car_Registration	-0.001346	-0.005148
Vechicle_Type_id	-0.080578	0.059244
Car_Transmission_Type_id	-0.325680	-0.113401
Car_Model_id	0.015537	-0.005667
Fuel_Type_id	-0.193212	-0.264273
Car_Brand_id	-0.123809	-0.039247
UnRepaired_Damage_id	-0.171799	-0.128884

	Car_Engine_Power_PS	Car_Mileage_Kilometer
Car_Price	0.591456	-0.469760
Year_of_Car_Registration	0.208647	-0.426956
Car_Engine_Power_PS	1.000000	-0.014387
Car_Mileage_Kilometer	-0.014387	1.000000
Month_of_Car_Registration	0.012757	0.007208
Vechicle_Type_id	-0.087092	0.016623
Car_Transmission_Type_id	-0.461485	0.040771
Car_Model_id	-0.092213	-0.047579
Fuel_Type_id	-0.177564	-0.132804
Car_Brand_id	-0.312963	-0.028241
UnRepaired_Damage_id	-0.066848	0.103673

	Month_of_Car_Registration	Vechicle_Type_id
Car_Price	-0.001346	-0.080578
Year_of_Car_Registration	-0.005148	0.059244
Car_Engine_Power_PS	0.012757	-0.087092
Car_Mileage_Kilometer	0.007208	0.016623
Month_of_Car_Registration	1.000000	0.021238
Vechicle_Type_id	0.021238	1.000000
Car_Transmission_Type_id	-0.017470	-0.021085
Car_Model_id	-0.004128	-0.070339
Fuel_Type_id	-0.036249	-0.040644
Car_Brand_id	-0.003471	-0.018853
UnRepaired_Damage_id	-0.009099	0.024021

	Car_Transmission_Type_id	Car_Model_id
Car_Price	-0.325680	0.015537
Year_of_Car_Registration	-0.113401	-0.005667
Car_Engine_Power_PS	-0.461485	-0.092213
Car_Mileage_Kilometer	0.040771	-0.047579
Month_of_Car_Registration	-0.017470	-0.004128
Vechicle_Type_id	-0.021085	-0.070339
Car_Transmission_Type_id	1.000000	0.025971
Car_Model_id	0.025971	1.000000

```
Fuel_Type_id                          0.167968     -0.032975
Car_Brand_id                          0.127871      0.448459
UnRepaired_Damage_id                  0.033147      0.007130

                          Fuel_Type_id  Car_Brand_id  UnRepaired_Damage_id
Car_Price                   -0.193212     -0.123809             -0.171799
Year_of_Car_Registration    -0.264273     -0.039247             -0.128884
Car_Engine_Power_PS         -0.177564     -0.312963             -0.066848
Car_Mileage_Kilometer       -0.132804     -0.028241              0.103673
Month_of_Car_Registration   -0.036249     -0.003471             -0.009099
Vechicle_Type_id            -0.040644     -0.018853              0.024021
Car_Transmission_Type_id     0.167968      0.127871              0.033147
Car_Model_id                -0.032975      0.448459              0.007130
Fuel_Type_id                 1.000000      0.055583              0.027651
Car_Brand_id                 0.055583      1.000000              0.001401
UnRepaired_Damage_id         0.027651      0.001401              1.000000
```

The above two outputs show the numerical and graphical charts that highlight the relationship among the attributes in the data.

0.14 Compute the Age of Car in Months

```
In [179]: #To select rows whose column value equals a scalar, some_value, use ==:
          Count_Car_Registered_in_2016 = autos_cleaned_data_df2.loc[autos_cleaned_data_df2['Yea


In [180]: # Frequency Counts of Month_of_Car_Registration for Cars Registered in 2016
          Count_Car_Registered_in_2016['Month_of_Car_Registration'].value_counts()

Out[180]: 3     132
          2      67
          1      51
          4      21
          12      2
          9       1
          7       1
          6       1
          5       1
          Name: Month_of_Car_Registration, dtype: int64

In [181]: # Calculate the Age_Of_Car_Months based on Year_of_Car_Registration and Month_of_Car_

          def compute_car_age(Registration_Year, Registration_Month):

              return (2016-Registration_Year)*12 +(12-Registration_Month)    #Allow Registrati

          autos_cleaned_data_df2['Age_Of_Car_Months'] = compute_car_age(autos_cleaned_data_df2

In [182]: autos_cleaned_data_df2.head(10)
```

Out[182]:

	Car_Price	Year_of_Car_Registration	Car_Engine_Power_PS
0	1500.0	2001	75.0
1	3600.0	2008	69.0
2	650.0	1995	102.0
3	2200.0	2004	109.0
4	2000.0	2004	105.0
5	2799.0	2005	140.0
6	17999.0	2011	190.0
7	1750.0	2004	75.0
8	7550.0	2007	136.0
9	1850.0	2004	102.0

	Car_Mileage_Kilometer	Month_of_Car_Registration	Vechicle_Type_id
0	150000	6	5
1	90000	7	5
2	150000	10	3
3	150000	8	1
4	150000	12	3
5	150000	12	6
6	70000	3	7
7	150000	2	5
8	150000	6	0
9	150000	1	0

	Car_Transmission_Type_id	Car_Model_id	Fuel_Type_id	Car_Brand_id
0	1	117	4	37
1	1	102	2	31
2	1	11	4	2
3	1	8	4	25
4	1	10	4	19
5	1	171	2	37
6	1	160	2	23
7	0	227	4	27
8	1	60	2	10
9	1	33	4	20

	UnRepaired_Damage_id	Age_Of_Car_Months
0	0	186
1	0	101
2	1	254
3	0	148
4	0	144
5	1	132
6	0	69
7	0	154
8	0	114
9	0	155

In [183]:
```
# Drop Year_of_Car_Registration and Month_of_Car_Registration
autos_cleaned_data_df3 = autos_cleaned_data_df2.drop(['Year_of_Car_Registration','Mo
```

In [184]:
```
autos_cleaned_data_df3.head(10)
```

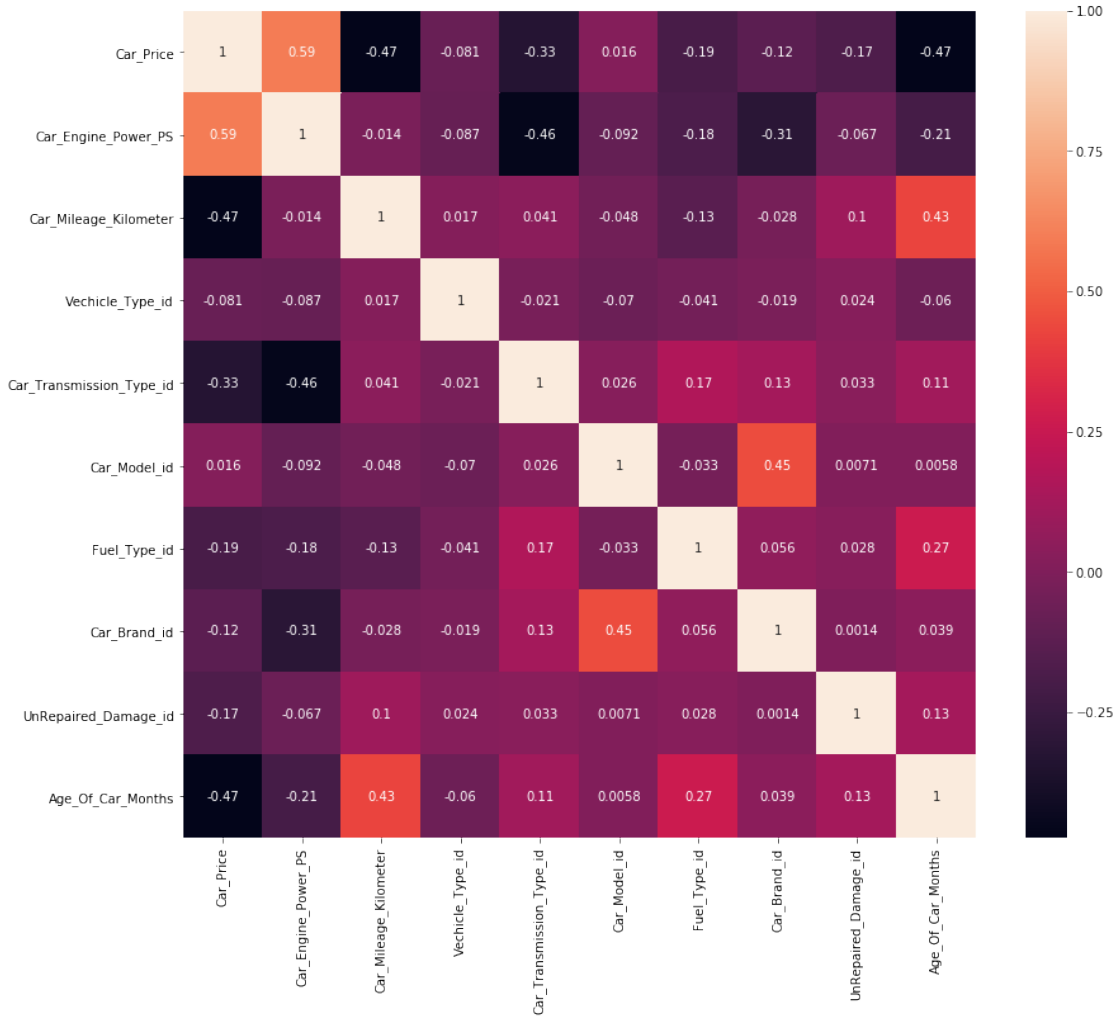
Out[184]:

	Car_Price	Car_Engine_Power_PS	Car_Mileage_Kilometer	Vechicle_Type_id	\
0	1500.0	75.0	150000	5	
1	3600.0	69.0	90000	5	
2	650.0	102.0	150000	3	
3	2200.0	109.0	150000	1	
4	2000.0	105.0	150000	3	
5	2799.0	140.0	150000	6	
6	17999.0	190.0	70000	7	
7	1750.0	75.0	150000	5	
8	7550.0	136.0	150000	0	
9	1850.0	102.0	150000	0	

	Car_Transmission_Type_id	Car_Model_id	Fuel_Type_id	Car_Brand_id	\
0	1	117	4	37	
1	1	102	2	31	
2	1	11	4	2	
3	1	8	4	25	
4	1	10	4	19	
5	1	171	2	37	
6	1	160	2	23	
7	0	227	4	27	
8	1	60	2	10	
9	1	33	4	20	

	UnRepaired_Damage_id	Age_Of_Car_Months
0	0	186
1	0	101
2	1	254
3	0	148
4	0	144
5	1	132
6	0	69
7	0	154
8	0	114
9	0	155

In [243]:
```
# Correlation of Data - Visualize potential relationship among the auto attributes
fig, ax = plt.subplots(figsize=(14,12))
sns.heatmap(autos_cleaned_data_df3.corr(), annot=True)
plt.savefig('Correlation of Data.pdf')
```



In [186]:
```
# Correlation using .corr() method - Matrix table of relationship among the autos at
autos_cleaned_data_df3.corr()
```

Out[186]:

	Car_Price	Car_Engine_Power_PS	\
Car_Price	1.000000	0.591456	
Car_Engine_Power_PS	0.591456	1.000000	
Car_Mileage_Kilometer	-0.469760	-0.014387	
Vechicle_Type_id	-0.080578	-0.087092	
Car_Transmission_Type_id	-0.325680	-0.461485	
Car_Model_id	0.015537	-0.092213	
Fuel_Type_id	-0.193212	-0.177564	
Car_Brand_id	-0.123809	-0.312963	
UnRepaired_Damage_id	-0.171799	-0.066848	
Age_Of_Car_Months	-0.474352	-0.209053	

	Car_Mileage_Kilometer	Vechicle_Type_id	\

Car_Price	-0.469760	-0.080578
Car_Engine_Power_PS	-0.014387	-0.087092
Car_Mileage_Kilometer	1.000000	0.016623
Vechicle_Type_id	0.016623	1.000000
Car_Transmission_Type_id	0.040771	-0.021085
Car_Model_id	-0.047579	-0.070339
Fuel_Type_id	-0.132804	-0.040644
Car_Brand_id	-0.028241	-0.018853
UnRepaired_Damage_id	0.103673	0.024021
Age_Of_Car_Months	0.426321	-0.060135

	Car_Transmission_Type_id	Car_Model_id	\
Car_Price	-0.325680	0.015537	
Car_Engine_Power_PS	-0.461485	-0.092213	
Car_Mileage_Kilometer	0.040771	-0.047579	
Vechicle_Type_id	-0.021085	-0.070339	
Car_Transmission_Type_id	1.000000	0.025971	
Car_Model_id	0.025971	1.000000	
Fuel_Type_id	0.167968	-0.032975	
Car_Brand_id	0.127871	0.448459	
UnRepaired_Damage_id	0.033147	0.007130	
Age_Of_Car_Months	0.114085	0.005844	

	Fuel_Type_id	Car_Brand_id	UnRepaired_Damage_id	\
Car_Price	-0.193212	-0.123809	-0.171799	
Car_Engine_Power_PS	-0.177564	-0.312963	-0.066848	
Car_Mileage_Kilometer	-0.132804	-0.028241	0.103673	
Vechicle_Type_id	-0.040644	-0.018853	0.024021	
Car_Transmission_Type_id	0.167968	0.127871	0.033147	
Car_Model_id	-0.032975	0.448459	0.007130	
Fuel_Type_id	1.000000	0.055583	0.027651	
Car_Brand_id	0.055583	1.000000	0.001401	
UnRepaired_Damage_id	0.027651	0.001401	1.000000	
Age_Of_Car_Months	0.265672	0.039370	0.129189	

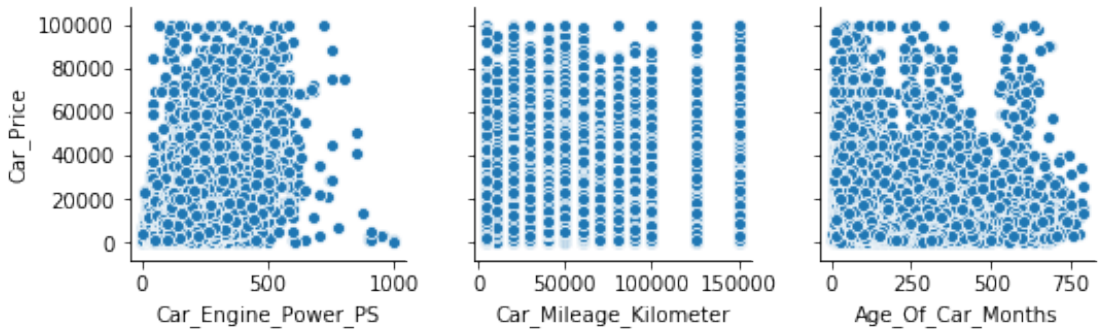
	Age_Of_Car_Months
Car_Price	-0.474352
Car_Engine_Power_PS	-0.209053
Car_Mileage_Kilometer	0.426321
Vechicle_Type_id	-0.060135
Car_Transmission_Type_id	0.114085
Car_Model_id	0.005844
Fuel_Type_id	0.265672
Car_Brand_id	0.039370
UnRepaired_Damage_id	0.129189
Age_Of_Car_Months	1.000000

The correlation results show a positive correlation of (0.59) between Car_Price and

Car_Engine_Power_PS. That means an increase in Car-Engine_Power_PS would have an increase in the Car_Price. Conversely, there is negative correlation between Car_Mileage_Kilometer and Car_Price. Also, there is negative correlation between Age_of_Car_Months and Car_Price. It is a well-known fact the value of a car(Car_Price) decreases as the age of the car goes up.

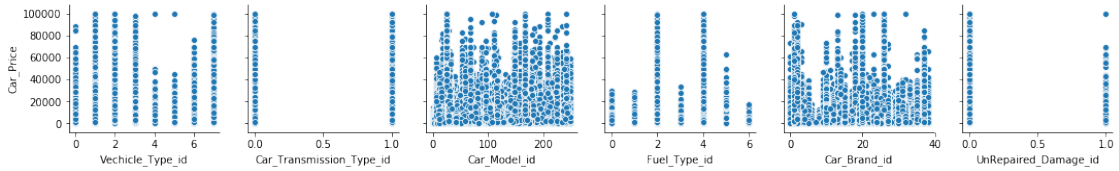
```
In [187]: #Pair Plots between Car_Price and other numerical variables
          plt.figure(figsize=(10,8))
          pp = sns.pairplot(data=autos_cleaned_data_df3,
                            y_vars=['Car_Price'],
                            x_vars=['Car_Engine_Power_PS', 'Car_Mileage_Kilometer', 'Age_Of_Ca
```

```
<Figure size 720x576 with 0 Axes>
```




```
In [188]: #Pair Plots between Car_Price and categeorical variables
          plt.figure(figsize=(10,8))
          pairplot_cat = sns.pairplot(data=autos_cleaned_data_df3,
                            y_vars=['Car_Price'],
                            x_vars=['Vechicle_Type_id', 'Car_Transmission_Type_id', 'Car_Model_
```

```
<Figure size 720x576 with 0 Axes>
```



0.15 Training models

```
In [189]: # Import the required libraries for the splitting the data and for model trainings
          # Import train_test_split
          from sklearn.model_selection import train_test_split
```

```
In [190]: # import metrics for MAE, MSE and RMsE
          from sklearn import metrics

In [191]: # Separate the dataset into Predictors(Features or Inputs) and Target (Response Vari
          X = autos_cleaned_data_df3.drop('Car_Price', axis = 1 )
          y = autos_cleaned_data_df3['Car_Price']
```

From the above, the dataset was split into training data and testing data. The training data is the data that the model would leverage for learning. While testing data is the data that would be leveraged to measure the performance of our models on unseen data.

```
In [192]: X.head(10)
```

Out[192]:

	Car_Engine_Power_PS	Car_Mileage_Kilometer	Vechicle_Type_id	\
0	75.0	150000	5	
1	69.0	90000	5	
2	102.0	150000	3	
3	109.0	150000	1	
4	105.0	150000	3	
5	140.0	150000	6	
6	190.0	70000	7	
7	75.0	150000	5	
8	136.0	150000	0	
9	102.0	150000	0	

	Car_Transmission_Type_id	Car_Model_id	Fuel_Type_id	Car_Brand_id	\
0	1	117	4	37	
1	1	102	2	31	
2	1	11	4	2	
3	1	8	4	25	
4	1	10	4	19	
5	1	171	2	37	
6	1	160	2	23	
7	0	227	4	27	
8	1	60	2	10	
9	1	33	4	20	

	UnRepaired_Damage_id	Age_Of_Car_Months
0	0	186
1	0	101
2	1	254
3	0	148
4	0	144
5	1	132
6	0	69
7	0	154
8	0	114
9	0	155

In [193]:
```
y.head(10)
```

Out[193]:
```
0     1500.0
1     3600.0
2      650.0
3     2200.0
4     2000.0
5     2799.0
6    17999.0
7     1750.0
8     7550.0
9     1850.0
Name: Car_Price, dtype: float64
```

In [194]:
```
# Split the data into training and test data sets using test size of 0.3 and random

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_stat
```

In [195]:
```
# Display five lines of X_train
X_train.head()
```

Out[195]:

	Car_Engine_Power_PS	Car_Mileage_Kilometer	Vechicle_Type_id	\
52212	140.0	5000	7	
122446	239.0	60000	7	
191394	65.0	50000	5	
150793	239.0	125000	7	
210898	150.0	150000	5	

	Car_Transmission_Type_id	Car_Model_id	Fuel_Type_id	Car_Brand_id	\
52212	1	167	4	24	
122446	0	180	2	1	
191394	1	210	0	3	
150793	0	221	2	37	
210898	1	103	4	10	

	UnRepaired_Damage_id	Age_Of_Car_Months
52212	0	14
122446	0	85
191394	0	52
150793	0	65
210898	0	113

In [196]:
```
# Display five lines of Y_train
y_train.head()
```

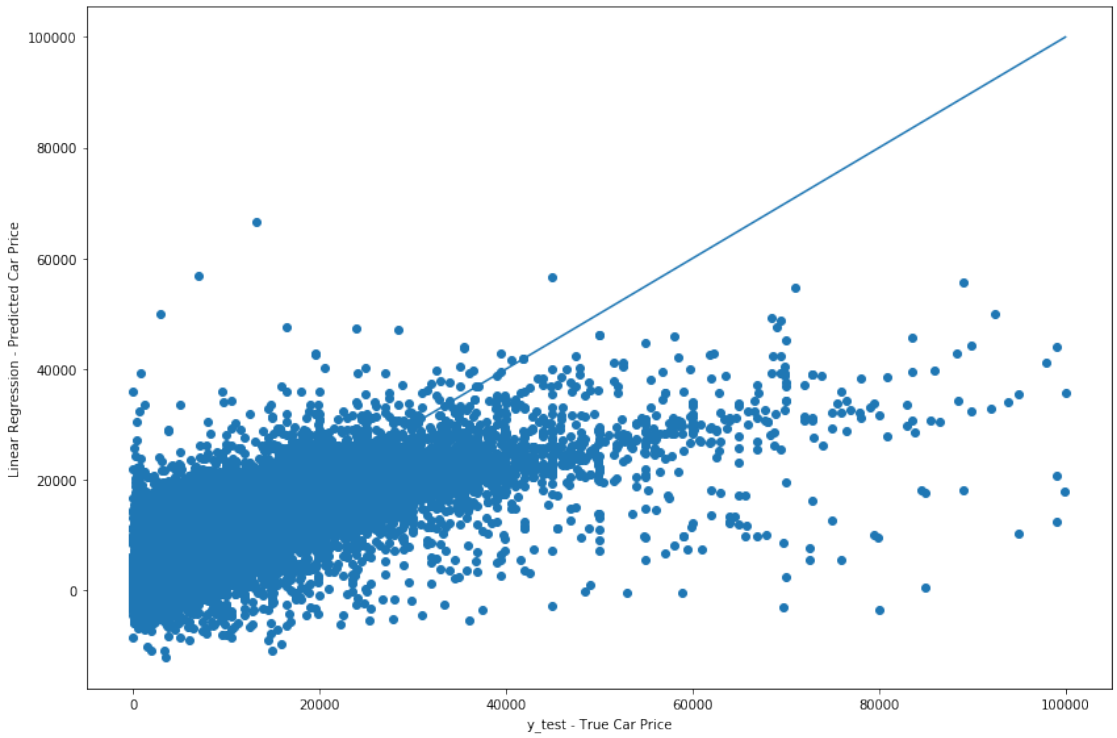
Out[196]:
```
52212    21699.0
122446   12000.0
191394    5899.0
150793   28900.0
210898    5499.0
Name: Car_Price, dtype: float64
```

0.16 Model Training 1 - Linear Regression

```
In [198]: # Import SK Learn Linear Regression
          from sklearn import linear_model

          # Create an instance of Linear Regression model
          Linear_Regressor = linear_model.LinearRegression()

In [199]: # Fit (that is Train) the training data to Linear Regression
          Linear_Regressor.fit(X_train, y_train)

Out[199]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [200]: # Produce Linear Regression Accuracy Score value
          Linear_Regressor.score(X_test,y_test)

Out[200]: 0.6186392679313754
```

Linear Regression model gives prediction accuracy of 62%

```
In [201]: # Let us see what the predictions are by providing the test data to the Linear Regre
          Linear_Regressor_pred = Linear_Regressor.predict(X_test)

In [202]: # Dislay MAE, MSE and RMSE for Linear Regression
          print('MAE:', metrics.mean_absolute_error(y_test, Linear_Regressor_pred))
          print('MSE:', metrics.mean_squared_error(y_test, Linear_Regressor_pred))
          print('RMsE:', np.sqrt(metrics.mean_squared_error(y_test,Linear_Regressor_pred)))

MAE: 3111.6998285111285
MSE: 24598349.6062194
RMsE: 4959.672328513185


In [203]: # create the Scatter plot of Linear Regression model
          plt.figure(figsize=(12,8))
          plt.scatter(y_test, Linear_Regressor_pred)
          plt.xlabel('y_test - True Car Price')
          plt.ylabel('Linear Regression - Predicted Car Price')
          plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)])
          plt.tight_layout()
```



1 Model Training 2 - Decision Tree

```
In [204]: # Import SK Learn Decision Tree Regressor
          from sklearn.tree import DecisionTreeRegressor

          # Create Decision Tree Regressor instance
          Tree_Regressor = DecisionTreeRegressor(random_state=42)
```

```
In [205]: # Fit (that is Train) the training data to Decision Tree Regressor
          Tree_Regressor.fit(X_train, y_train)
```

```
Out[205]: DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                                max_leaf_nodes=None, min_impurity_decrease=0.0,
                                min_impurity_split=None, min_samples_leaf=1,
                                min_samples_split=2, min_weight_fraction_leaf=0.0,
                                presort=False, random_state=42, splitter='best')
```

```
In [206]: # Produce Decision Tree Regressor Accuracy Score value
          Tree_Regressor.score(X_test,y_test)
```

```
Out[206]: 0.8186468753095089
```

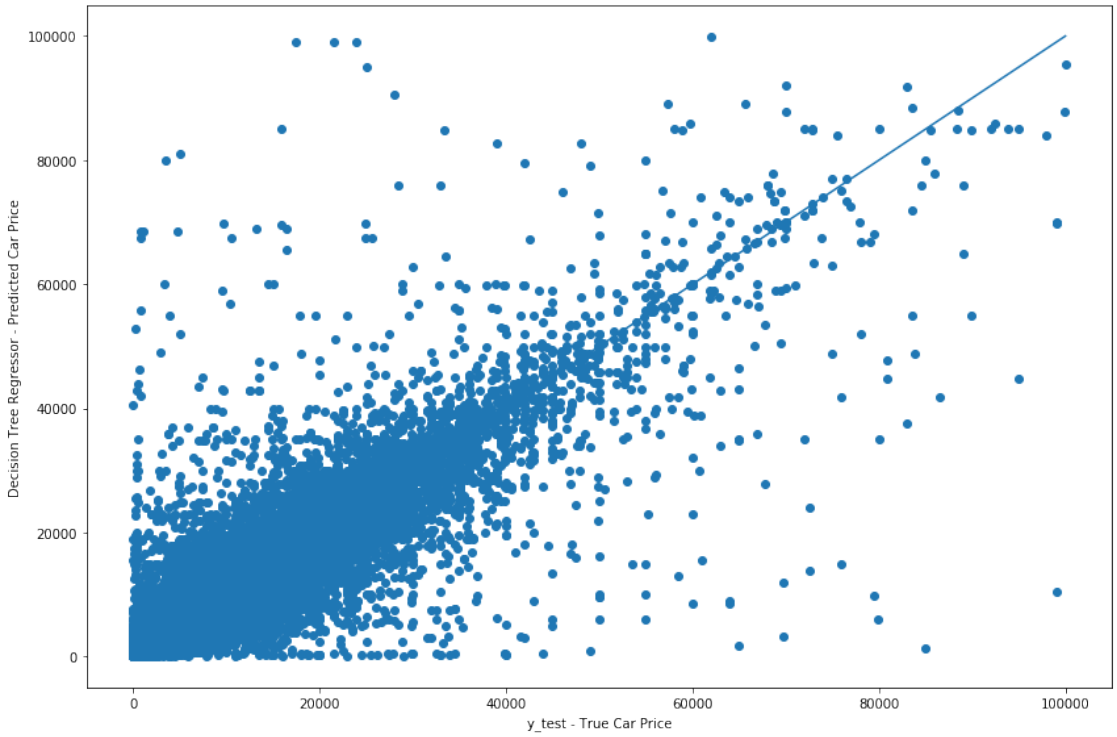
Decision Tree Regression model gives us prediction accuracy of 82%.

In [207]:
```
# Let us see what the predictions are by providing the test data to the Decision Tre
Tree_Regressor_pred = Tree_Regressor.predict(X_test)
```

In [208]:
```
# Dislay MAE, MSE and RMSE for Decision Tree Regression model
print('MAE:', metrics.mean_absolute_error(y_test, Tree_Regressor_pred))
print('MSE:', metrics.mean_squared_error(y_test, Tree_Regressor_pred))
print('RMsE:', np.sqrt(metrics.mean_squared_error(y_test,Tree_Regressor_pred)))
```

```
MAE: 1520.8651744667957
MSE: 11697553.492513902
RMsE: 3420.1686350988457
```

In [209]:
```
# create the Scatter plot of Decision Tree Regression model
plt.figure(figsize=(12,8))
plt.scatter(y_test, Tree_Regressor_pred)
plt.xlabel('y_test - True Car Price')
plt.ylabel('Decision Tree Regressor - Predicted Car Price')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)])
plt.tight_layout()
```



2 Model Training 3 - Random Forest Regressor

In [210]:
```
# Import SK Learn Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
```

```
          # Create Random Forest Regressor instance
          Rd_forest_Regressor = RandomForestRegressor(random_state=42)
```

```
In [211]: # Fit (that is Train) the training data to Random Forest Regressor
          Rd_forest_Regressor.fit(X_train, y_train)
```

```
C:\Users\Babawale Olatunji\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\ensemble
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
Out[211]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                                max_features='auto', max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=10,
                                n_jobs=None, oob_score=False, random_state=42, verbose=0,
                                warm_start=False)
```

```
In [212]: # Produce Random Forest Regressor Accuracy Score value
          Rd_forest_Regressor.score(X_test, y_test)
```

```
Out[212]: 0.8890868687326233
```

Random Forest Regression model gives us prediction accuracy of 89%.

```
In [213]: # Let us see what the predictions are by providing the test data to the
          # Random Forest Regressor model
          Rd_forest_Regressor_pred = Rd_forest_Regressor.predict(X_test)
```

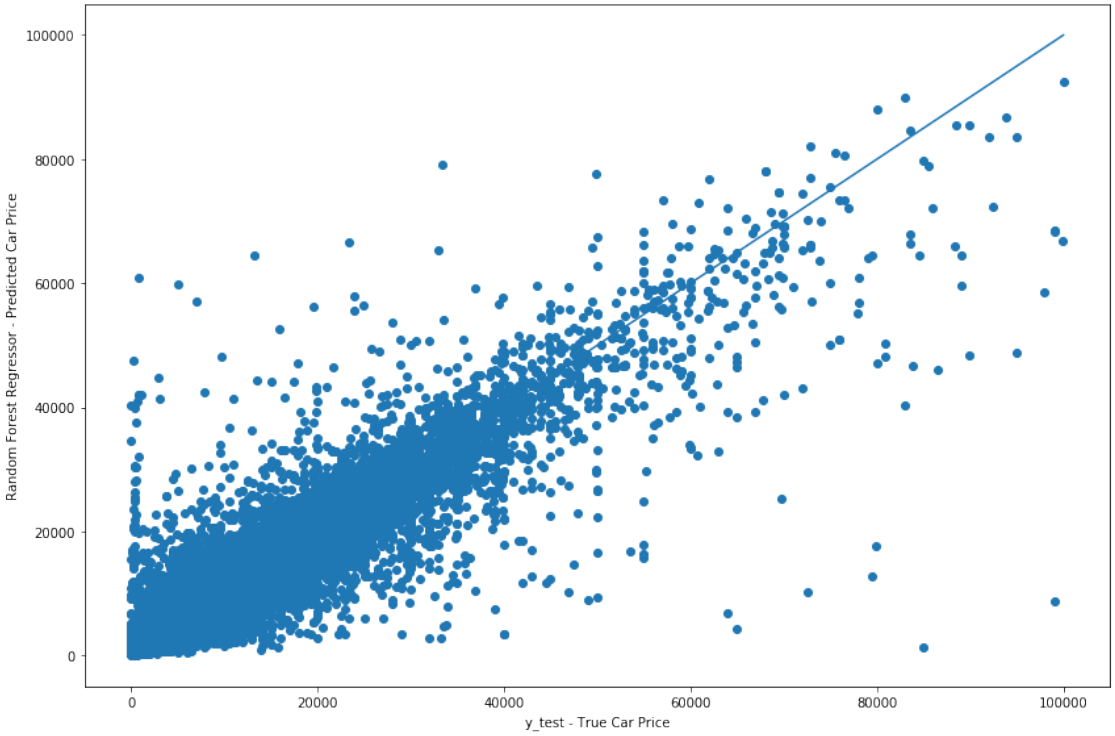
```
In [214]: print(Rd_forest_Regressor_pred)
```

```
[ 8976.66666667  1214.9         2418.96666667 ... 12390.52
   843.          1659.8       ]
```

```
In [215]: # Dislay MAE, MSE and RMSE for Random Forest Regressor
          print('MAE:', metrics.mean_absolute_error(y_test, Rd_forest_Regressor_pred))
          print('MSE:', metrics.mean_squared_error(y_test, Rd_forest_Regressor_pred))
          print('RMsE:', np.sqrt(metrics.mean_squared_error(y_test,Rd_forest_Regressor_pred)))
```

```
MAE: 1288.38336720473
MSE: 7154066.3456259845
RMsE: 2674.7086468671655
```

```
In [216]: # create the Scatter plot of Random Forest Regressor model
          plt.figure(figsize=(12,8))
          plt.scatter(y_test, Rd_forest_Regressor_pred)
          plt.xlabel('y_test - True Car Price')
          plt.ylabel('Random Forest Regressor - Predicted Car Price')
          plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)])
          plt.tight_layout()
```



2.1 Comparing the accurary of the three models

Linear Regression model gives prediction accuracy of 62%.Decision Tree Regression model gives us prediction accuracy of 82%. And Random Forest Regression model gives us prediction accuracy of 89%. Hence, Random Forest Regression Model is the most accurate of the three.

MAE, MSE, and RMSE

Mean Absolute Error(MAE), Mean Squared Error(MSE) and Root Mean Squared Error(RMSE) are the three frequently evaluation metrics for regression problems. These three are loss functions that need to be minimized.

MAE is the average error. While MSE is the mean of the squared errors.Moreover, RMSE is the square root of the mean of the squared errors. RMSE is commonly used measure of the difference between the model predicted values and the true values.

RMSE from Random Forest Regressor, Decision Tree and Linear Regression are 2675, 3420 and 4960 respectively. This is another indication that Random Forest Regressor model outperforms the other two regression models since it has the lowest difference between the model predicted values and the true values for the price of used cars.

2.2 Model Training 4 - Ensemble Learning Model

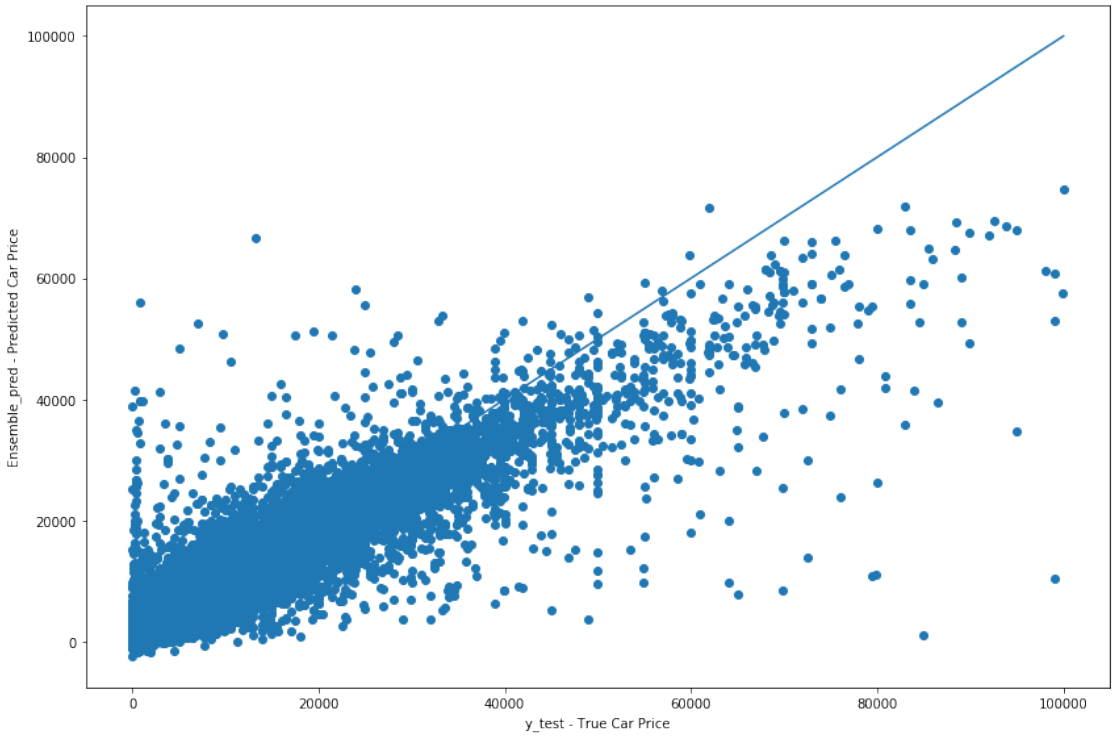
The three individual models including Linear Regression model, Decision Tree Regression model and Random Forest Regression mode have been created above. The next step is to create a voting regressor that would leverage the strength of these regressors.

```
In [217]: # import sk Learn VotingRegressor
          from sklearn.ensemble import VotingRegressor

In [222]: # Create a dictionary of three regression models
          estimators = [('linear_reg', Linear_Regressor), ('tree_reg', Tree_Regressor), ('fores

In [223]: # Create voting regressor, inputting the three regression models
          ensemble = VotingRegressor(estimators)

In [224]: # Fit model to training data
          ensemble.fit(X_train, y_train)

Out[224]: VotingRegressor(estimators=[('linear_reg',
                                       LinearRegression(copy_X=True, fit_intercept=True,
                                                        n_jobs=None, normalize=False)),
                                      ('tree_reg',
                                       DecisionTreeRegressor(criterion='mse',
                                                             max_depth=None,
                                                             max_features=None,
                                                             max_leaf_nodes=None,
                                                             min_impurity_decrease=0.0,
                                                             min_impurity_split=None,
                                                             min_samples_leaf=1,
                                                             min_samples_split=2,
                                                             min_weight_fraction_leaf=0.0,
                                                             presort=F...
                                      ('forest_reg',
                                       RandomForestRegressor(bootstrap=True,
                                                             criterion='mse',
                                                             max_depth=None,
                                                             max_features='auto',
                                                             max_leaf_nodes=None,
                                                             min_impurity_decrease=0.0,
                                                             min_impurity_split=None,
                                                             min_samples_leaf=1,
                                                             min_samples_split=2,
                                                             min_weight_fraction_leaf=0.0,
                                                             n_estimators=10, n_jobs=None,
                                                             oob_score=False,
                                                             random_state=42, verbose=0,
                                                             warm_start=False))],
                                  n_jobs=None, weights=None)

In [225]: #Test Ensemble model on the test data
          ensemble.score(X_test,y_test)

Out[225]: 0.8618224688754966

In [226]: # Let us see what the predictions are by providing the test data to the Ensemble mod
          ensemble_pred = ensemble.predict(X_test)
```

```
In [227]: # Dislay MAE, MSE and RMSE for ensemble model
          print('MAE:', metrics.mean_absolute_error(y_test, ensemble_pred))
          print('MSE:', metrics.mean_squared_error(y_test, ensemble_pred))
          print('RMsE:', np.sqrt(metrics.mean_squared_error(y_test,ensemble_pred)))
```

```
MAE: 1626.8086093107863
MSE: 8912661.772720661
RMsE: 2985.4081417321586
```

```
In [228]: # create the Scatter plot of Ensemble model
          plt.figure(figsize=(12,8))
          plt.scatter(y_test, ensemble_pred)
          plt.xlabel('y_test - True Car Price')
          plt.ylabel('Ensemble_pred - Predicted Car Price')
          plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)])
          plt.tight_layout()
```



Ensemble model gives us prediction accuracy of 89%

That means, Random Forest Regression model which is also an ensemble method gives us better accuracy than Ensemble model. The RMSE (2650) from Random Regression Model is lower than RMSE (2985) produced by Ensemble VotingRegressor method based on the results above.

```
In [229]: # https://stackoverflow.com/questions/16923281/writing-a-pandas-dataframe-to-csv-fil
          # Writing autos_cleaned_data_df3 dataframe to csv file
          autos_cleaned_data_df3.to_csv("autos_cleaned_data3.csv", index=False)
```

2.3 Conclusion

Linear Regression , Decision Tree Regressor, Random Forest Regressor and Ensemble (Voting Regressor) are four popular strategies for machine learning and regression.

For this project, Linear Regression yielded 62% accuracy for predicting the price of used cars. Moreover, Decision Tree Regressor produced 82% accuracy and Ensemble (Voting Regressor) produced 87% accuracy for predicting the price of used cars. Per Random Forest Regressor, it yielded the highest accuracy of 89% accuracy for this used car price predicting project.Before completing this project, I was expecting Ensemble(Voting Regressor) to outperform all the individual Regression models. It is interesting to discover that Random Forest Regressor, also an ensemble regression method would produce the highest accuracy for this project.

Further more, the project showcases that Car Power Engine, Car Mileage, and the Age of the car are the three most important factors in predicting the price of a used car.

Future work on this project would include predicting the price of each brand and model of the car collections in this dataset. It would be interesting to learn how the inclusion of pictures into the dataset can influence predicting the price of a used car.

In [ ]: