CIND 820 Initial Results and Code

November 6, 2021

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.image as mpimg
import math
import dask.dataframe as dd

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from string import ascii_letters
```

[2]:
```
# Load data
df = pd.read_csv("ca-dealers-used.csv",dtype={'fuel_type': 'object',␣
↪'engine_block': 'object'})
```

[3]:
```
df.dtypes
```

[3]:
```
id              object
vin             object
price          float64
miles          float64
stock_no        object
year           float64
make            object
model           object
trim            object
body_type       object
vehicle_type    object
drivetrain      object
transmission    object
fuel_type       object
engine_size    float64
engine_block    object
seller_name     object
street          object
```

```
city             object
state            object
zip              object
dtype: object
```

[4]:
```
len(df)
```

[4]:
```
393603
```

[5]:
```
df.head(5)
```

[5]:
```
              id                vin     price    miles   stock_no    year  \
0  b39ea795-eca9  19UNC1B01HY800062  179999.0   9966.0    V-P4139  2017.0
1  026cb5b1-6e3e  19UNC1B02HY800023  179995.0   5988.0  PPAP70374  2017.0
2  5cd5d5b2-5cc2  19UNC1B02HY800071  168528.0  24242.0     B21085  2017.0
3  b32473ed-5922  19UNC1B02LY800001  220000.0   6637.0     AP5333  2020.0
4  ac40c9fc-0676  19UNC1B02LY800001  220000.0   6637.0     AP5333  2020.0

    make model  trim body_type  … drivetrain transmission  \
0  Acura   NSX  Base     Coupe  …        4WD    Automatic
1  Acura   NSX  Base     Coupe  …        4WD    Automatic
2  Acura   NSX  Base     Coupe  …        4WD    Automatic
3  Acura   NSX  Base     Coupe  …        4WD    Automatic
4  Acura   NSX  Base     Coupe  …        4WD    Automatic

                    fuel_type engine_size  engine_block  \
0  Electric / Premium Unleaded         3.5             V
1  Electric / Premium Unleaded         3.5             V
2  Electric / Premium Unleaded         3.5             V
3  Electric / Premium Unleaded         3.5             V
4  Electric / Premium Unleaded         3.5             V

             seller_name                          street                 city  \
0       edmundston honda                475 Rue Victoria           Edmundston
1  garage daniel lessard  2795 Route-du-prsident-kennedy  Notre-dame-des-pins
2         lougheed acura           1388 Lougheed Highway            Coquitlam
3        drive autogroup         1305 Parkway Suite 600            Pickering
4        acura pickering              575 Kingston Road            Pickering

  state      zip
0    NB  E3V 2K7
1    QC  G0M 1K0
2    BC  V3K 6S4
3    ON  L1V 3P2
4    ON  L1V 3N7

[5 rows x 21 columns]
```

[6]:
```
# Only select records from Toronto
df=df[df.state == "ON"]
```

[7]:
```
df=df[df.city == "Toronto"]
```

[8]:
```
# drop irrelevant columns for this research
drop_columns = ['id', 'vin', 'stock_no',␣
 ↪'seller_name','street','city','state','zip']
```

[9]:
```
df = df.drop(columns = drop_columns)
```

[10]:
```
# EDA analysis
# check data types
df.dtypes
```

[10]:
```
price           float64
miles           float64
year            float64
make             object
model            object
trim             object
body_type        object
vehicle_type     object
drivetrain       object
transmission     object
fuel_type        object
engine_size     float64
engine_block     object
dtype: object
```

[11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14998 entries, 39 to 393573
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   price         13811 non-null  float64
 1   miles         13229 non-null  float64
 2   year          14998 non-null  float64
 3   make          14998 non-null  object
 4   model         14807 non-null  object
 5   trim          13954 non-null  object
 6   body_type     13442 non-null  object
 7   vehicle_type  13278 non-null  object
 8   drivetrain    14262 non-null  object
 9   transmission  14276 non-null  object
```

```
 10  fuel_type     12615 non-null  object
 11  engine_size   12607 non-null  float64
 12  engine_block  12554 non-null  object
dtypes: float64(4), object(9)
memory usage: 1.6+ MB
```

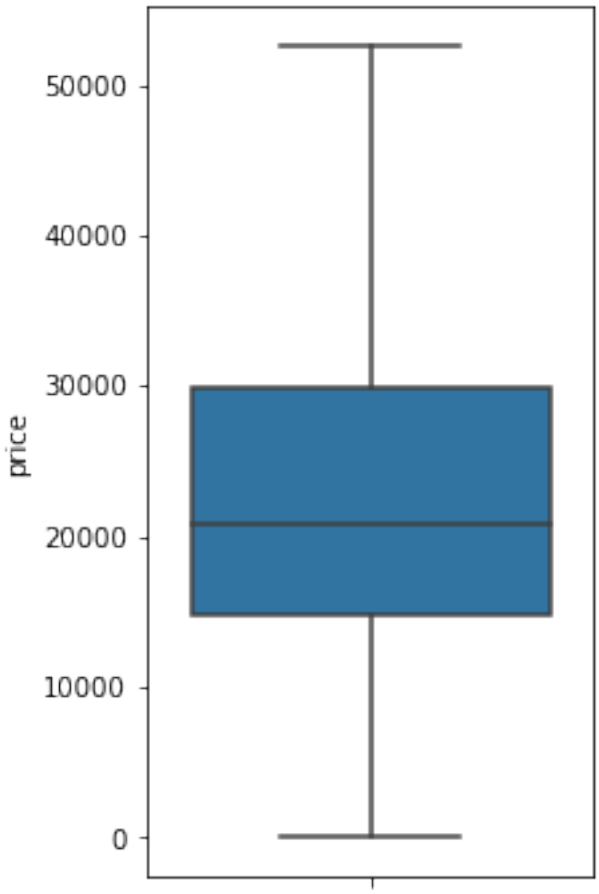
[12]:
```
# Check missing values
missing_values_count = df.isnull().sum()
missing_values_count
```

[12]:
```
price           1187
miles           1769
year               0
make               0
model            191
trim            1044
body_type       1556
vehicle_type    1720
drivetrain       736
transmission     722
fuel_type       2383
engine_size     2391
engine_block    2444
dtype: int64
```

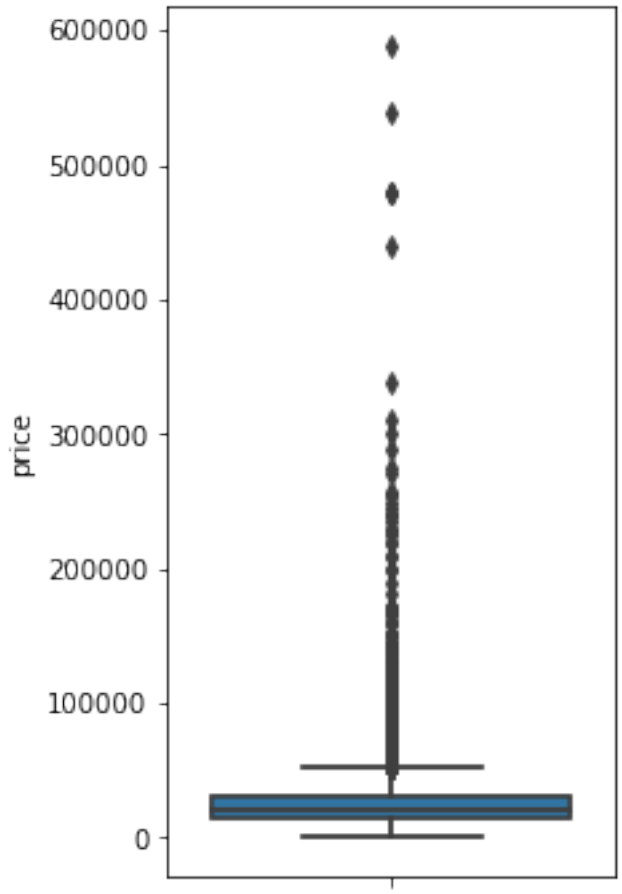
[13]:
```
df['price'].describe()
```

[13]:
```
count     13811.000000
mean      25069.024835
std       21359.905212
min           0.000000
25%       14855.500000
50%       20900.000000
75%       29950.000000
max      589000.000000
Name: price, dtype: float64
```

[14]:
```
# Boxplot for price
plt.figure(figsize=(3,6))
sns.boxplot(y='price', data=df, showfliers=False);
```

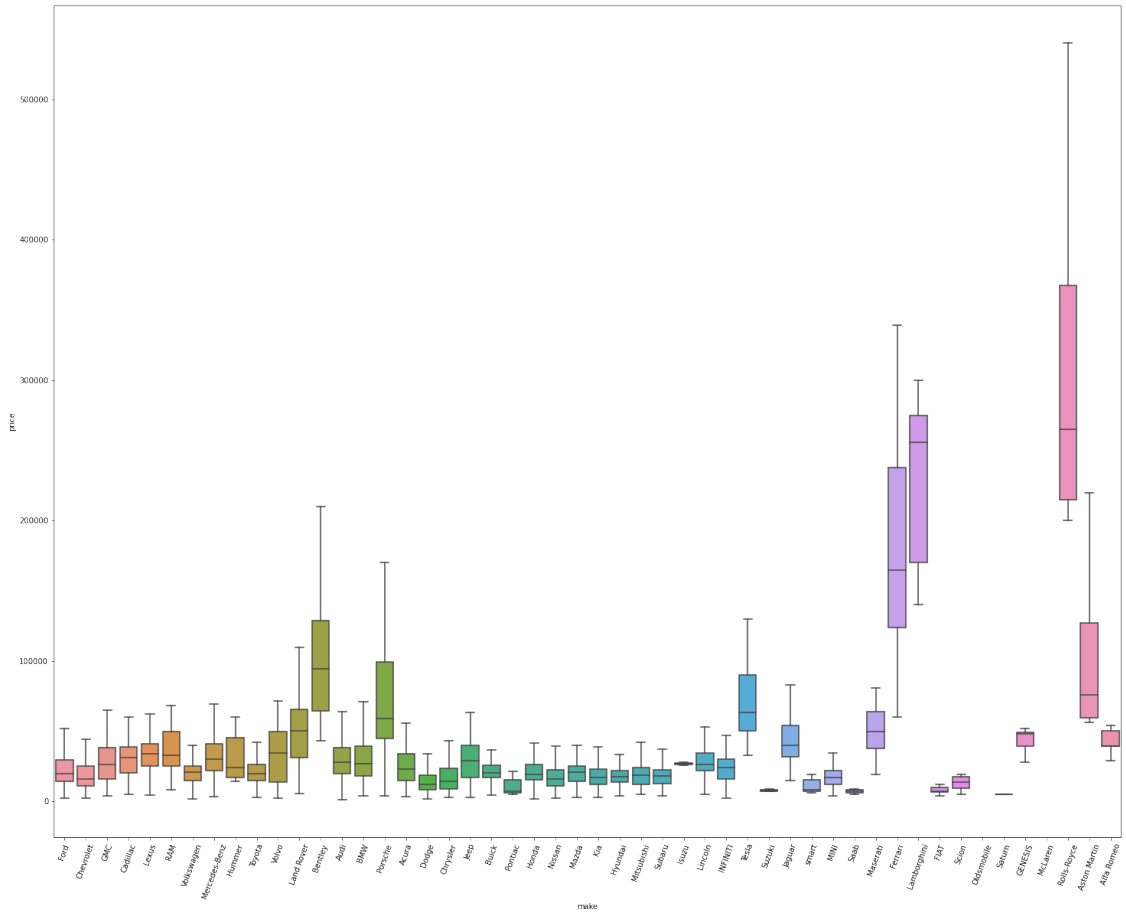


[15]:
```
# Box plot for price with outliers, we can see that there are a lot extreme␣
 ↪high prices
plt.figure(figsize=(3,6))
sns.boxplot(y='price', data=df, showfliers=False)
sns.boxplot(y='price', data=df);
```

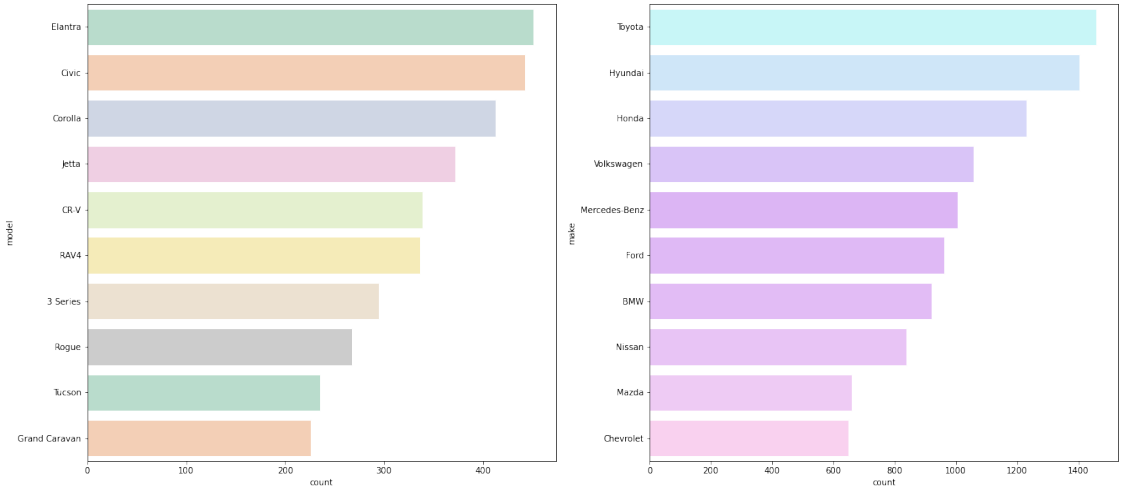


[16]:
```
# Prices by different brands, some brands have high average prices and max␣
↪prices
plt.figure(figsize=(25,20))
plt.xticks(rotation=70)
sns.boxplot(y='price', x='make', data=df, showfliers=False)
```

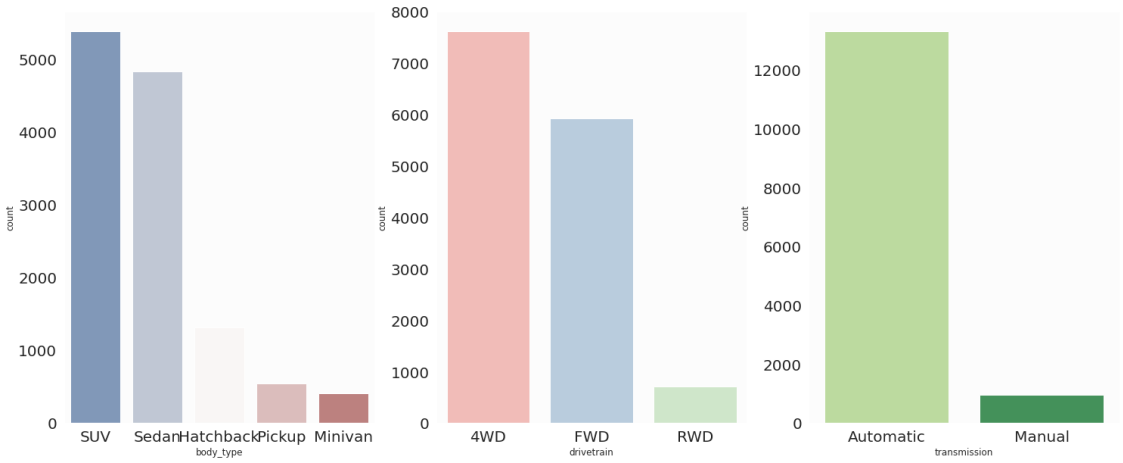
[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7f397f526b90>



[17]:
```
# Count by top 10 most popular brand and model
colors = ["#C0FDFF",␣
 ↪"#C8E7FF","#D0D1FF","#D8BBFF","#DEAAFF","#E2AFFF","#E5B3FE","#ECBCFD","#F3C4FB","#FFCBF2"]
sns.set_palette(sns.color_palette(colors))


fig, ax =plt.subplots(1,2,figsize=(22, 10))
sns.set(rc={"axes.facecolor":"#fcfcfc", "axes.grid":False,'xtick.labelsize':
 ↪20,'ytick.labelsize':20})
sns.countplot(y ='make',data = df,order=df.make.value_counts().iloc[:10].index,␣
 ↪ax = ax[1],palette = colors)
sns.countplot(y ='model',data = df,order=df.model.value_counts().iloc[:10].
 ↪index, ax= ax[0],palette = "Pastel2")
fig.show()
```

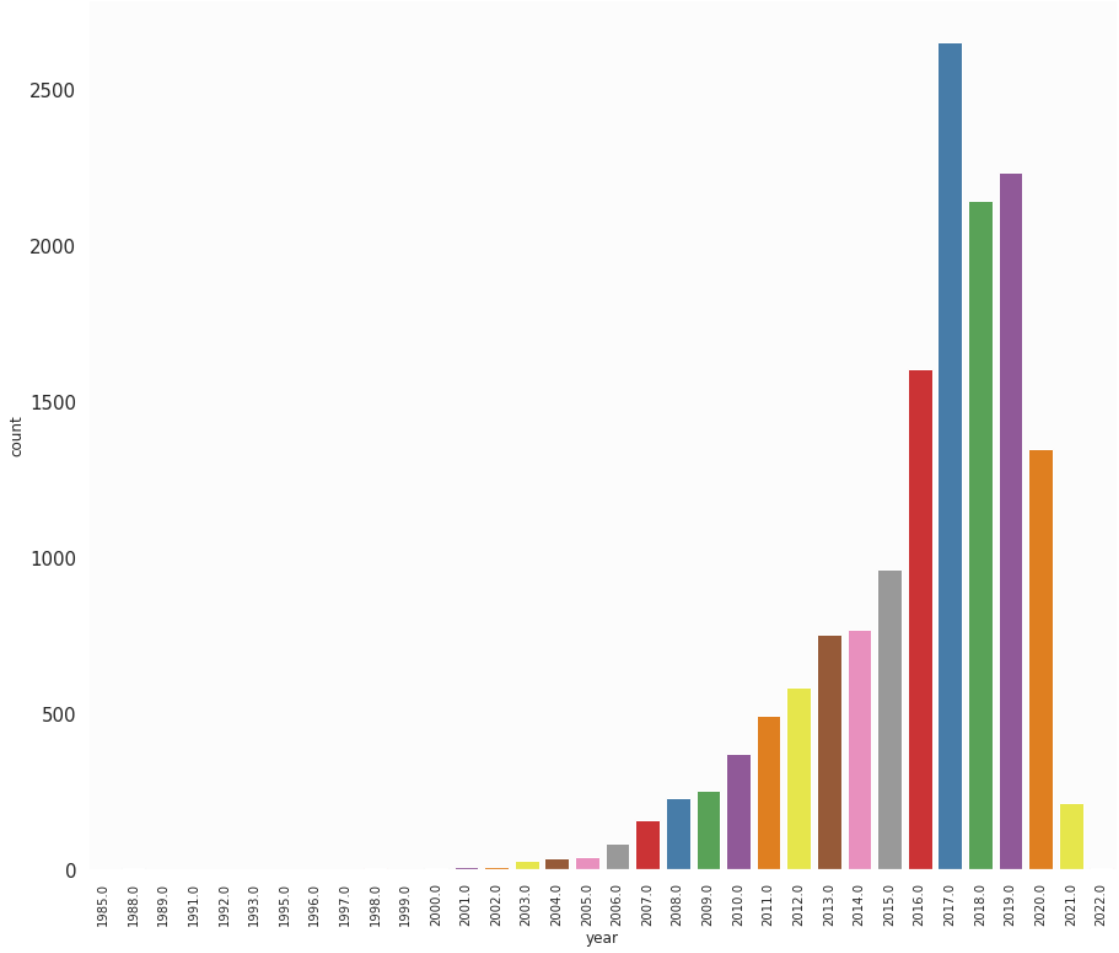


[18]:
```
# Countplot by body_type, drivetrain and transmission
fig, ax =plt.subplots(1,3,figsize=(25, 10))
sns.set(rc={"axes.facecolor":"#fcfcfc", "axes.grid":False,'xtick.labelsize':
 ↪15,'ytick.labelsize':15})
sns.countplot(x ='body_type',data = df,order=df.body_type.value_counts().iloc[:
 ↪5].index, ax = ax[0],palette="vlag")
sns.countplot(x ='drivetrain',data = df,order=df.drivetrain.value_counts().
 ↪iloc[:5].index, ax = ax[1],palette="Pastel1")
sns.countplot(x ='transmission',data = df,order=df.transmission.value_counts().
 ↪iloc[:5].index, ax = ax[2],palette="YlGn")
fig.show()
```



[19]:
```
# Count by year
plt.figure(figsize=(15, 13))
```

```
ax = sns.countplot(x = 'year', data=df, palette='Set1')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90,fontsize=10);
```



[20]:
```
# Sample data:
df.tail(20)
```

[20]:

	price	miles	year	make	model	\
393008	9990.0	96588.0	2013.0	Hyundai	Sonata Hybrid	
393022	NaN	154300.0	2013.0	Hyundai	Sonata Hybrid	
393040	38998.0	15365.0	2020.0	Hyundai	Sonata Hybrid	
393067	8995.0	112000.0	2012.0	Kia	Optima	
393134	16999.0	103590.0	2016.0	Kia	Optima	
393144	14950.0	90000.0	2016.0	Kia	Optima	
393177	13888.0	97123.0	2015.0	Kia	Optima	
393232	18990.0	64476.0	2017.0	Kia	Optima	
393235	18990.0	64476.0	2017.0	Kia	Optima	
393236	18990.0	64476.0	2017.0	Kia	Optima	

```
393237  18990.0   64476.0  2017.0           Kia              Optima
393241  27654.0      99.0  2019.0           Kia                Niro
393279  16647.0  122138.0  2017.0           Kia                Niro
393454  38140.0    8500.0  2020.0           Kia                Niro
393545  62988.0   22352.0  2020.0    Land Rover  Range Rover Evoque
393546  62988.0   22352.0  2020.0    Land Rover  Range Rover Evoque
393561  26450.0   56000.0  2017.0           BMW             3 Series
393568  45880.0   69000.0  2018.0  Mercedes-Benz           GLC-Class
393570  43880.0   65000.0  2018.0  Mercedes-Benz           GLC-Class
393573  46800.0   19833.0  2018.0  Mercedes-Benz           GLC-Class

                    trim  body_type vehicle_type drivetrain transmission  \
393008            Hybrid      Sedan          Car        FWD    Automatic
393022            Hybrid      Sedan          Car        FWD    Automatic
393040               SEL      Sedan          Car        FWD    Automatic
393067            Hybrid      Sedan          Car        FWD    Automatic
393134         EX Hybrid      Sedan          Car        FWD    Automatic
393144         EX Hybrid      Sedan          Car        FWD    Automatic
393177         EX Hybrid      Sedan          Car        FWD    Automatic
393232  EX Plug-In Hybrid      Sedan          Car        FWD    Automatic
393235  EX Plug-In Hybrid      Sedan          Car        FWD    Automatic
393236  EX Plug-In Hybrid      Sedan          Car        FWD    Automatic
393237  EX Plug-In Hybrid      Sedan          Car        FWD    Automatic
393241                 LX  Hatchback          Car        FWD    Automatic
393279                 EX  Hatchback          Car        FWD    Automatic
393454                 EX  Hatchback          Car        FWD    Automatic
393545                HSE        SUV        Truck        4WD    Automatic
393546                HSE        SUV        Truck        4WD    Automatic
393561               330e      Sedan          Car        RWD    Automatic
393568            GLC350e        SUV        Truck        4WD    Automatic
393570            GLC350e        SUV        Truck        4WD    Automatic
393573            GLC350e        SUV        Truck        4WD    Automatic

                fuel_type  engine_size engine_block
393008  Electric / Unleaded          2.4            I
393022  Electric / Unleaded          2.4            I
393040  Electric / Unleaded          2.0            I
393067  Electric / Unleaded          2.4            I
393134  Electric / Unleaded          2.4            I
393144  Electric / Unleaded          2.4            I
393177  Electric / Unleaded          2.4            I
393232  Electric / Unleaded          2.0            I
393235  Electric / Unleaded          2.0            I
393236  Electric / Unleaded          2.0            I
393237  Electric / Unleaded          2.0            I
393241  Electric / Unleaded          1.6            I
393279  Electric / Unleaded          1.6            I
```

```
393454  Electric / Unleaded          1.6            I
393545  Electric / Unleaded          2.0            I
393546  Electric / Unleaded          2.0            I
393561  Electric / Unleaded          2.0            I
393568  Electric / Unleaded          2.0            I
393570  Electric / Unleaded          2.0            I
393573  Electric / Unleaded          2.0            I
```

[21]:
```
# Data cleaning and preprocessing:
# Lets drop records without prices, since these records will be useless for our␣
 ↪analysis
df_tor=df.dropna(subset=['price'])
```

[22]:
```
pip install pandas-profiling
```

```
Collecting pandas-profiling
  Using cached pandas_profiling-3.1.0-py2.py3-none-any.whl (261 kB)
Collecting multimethod>=1.4
  Using cached multimethod-1.6-py3-none-any.whl (9.4 kB)
Collecting tqdm>=4.48.2
  Using cached tqdm-4.62.3-py2.py3-none-any.whl (76 kB)
Collecting visions[type_image_path]==0.7.4
  Using cached visions-0.7.4-py3-none-any.whl (102 kB)
Collecting phik>=0.11.1
  Using cached phik-0.12.0-cp37-cp37m-manylinux2010_x86_64.whl (675 kB)
Requirement already satisfied: pandas!=1.0.0,!=1.0.1,!=1.0.2,!=1.1.0,>=0.25.3 in
/opt/conda/lib/python3.7/site-packages (from pandas-profiling) (1.0.3)
Collecting tangled-up-in-unicode==0.1.0
  Using cached tangled_up_in_unicode-0.1.0-py3-none-any.whl (3.1 MB)
Requirement already satisfied: jinja2>=2.11.1 in /opt/conda/lib/python3.7/site-
packages (from pandas-profiling) (2.11.2)
Requirement already satisfied: seaborn>=0.10.1 in /opt/conda/lib/python3.7/site-
packages (from pandas-profiling) (0.10.1)
Collecting pydantic>=1.8.1
  Using cached pydantic-1.8.2-cp37-cp37m-manylinux2014_x86_64.whl (10.1 MB)
Requirement already satisfied: numpy>=1.16.0 in /opt/conda/lib/python3.7/site-
packages (from pandas-profiling) (1.18.4)
Collecting requests>=2.24.0
  Using cached requests-2.26.0-py2.py3-none-any.whl (62 kB)
Requirement already satisfied: matplotlib>=3.2.0 in
/opt/conda/lib/python3.7/site-packages (from pandas-profiling) (3.2.1)
Processing ./.cache/pip/wheels/70/e1/52/5b14d250ba868768823940c3229e9950d201a26d
0bd3ee8655/htmlmin-0.1.12-py3-none-any.whl
Collecting missingno>=0.4.2
  Using cached missingno-0.5.0-py3-none-any.whl (8.8 kB)
Collecting joblib~=1.0.1
  Using cached joblib-1.0.1-py3-none-any.whl (303 kB)
```

```
Requirement already satisfied: scipy>=1.4.1 in /opt/conda/lib/python3.7/site-
packages (from pandas-profiling) (1.4.1)
Collecting markupsafe~=2.0.1
  Using cached MarkupSafe-2.0.1-cp37-cp37m-manylinux2010_x86_64.whl (31 kB)
Requirement already satisfied: PyYAML>=5.0.0 in /opt/conda/lib/python3.7/site-
packages (from pandas-profiling) (5.3.1)
Requirement already satisfied: networkx>=2.4 in /opt/conda/lib/python3.7/site-
packages (from visions[type_image_path]==0.7.4->pandas-profiling) (2.4)
Requirement already satisfied: attrs>=19.3.0 in /opt/conda/lib/python3.7/site-
packages (from visions[type_image_path]==0.7.4->pandas-profiling) (19.3.0)
Requirement already satisfied: Pillow; extra == "type_image_path" in
/opt/conda/lib/python3.7/site-packages (from
visions[type_image_path]==0.7.4->pandas-profiling) (7.1.2)
Processing ./.cache/pip/wheels/4c/d5/59/5e3e297533ddb09407769762985d134135064c68
31e29a914e/ImageHash-4.2.1-py2.py3-none-any.whl
Requirement already satisfied: pytz>=2017.2 in /opt/conda/lib/python3.7/site-
packages (from pandas!=1.0.0,!=1.0.1,!=1.0.2,!=1.1.0,>=0.25.3->pandas-profiling)
(2020.1)
Requirement already satisfied: python-dateutil>=2.6.1 in
/opt/conda/lib/python3.7/site-packages (from
pandas!=1.0.0,!=1.0.1,!=1.0.2,!=1.1.0,>=0.25.3->pandas-profiling) (2.8.1)
Collecting typing-extensions>=3.7.4.3
  Using cached typing_extensions-3.10.0.2-py3-none-any.whl (26 kB)
Requirement already satisfied: idna<4,>=2.5; python_version >= "3" in
/opt/conda/lib/python3.7/site-packages (from requests>=2.24.0->pandas-profiling)
(2.9)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.24.0->pandas-profiling)
(2020.4.5.2)
Collecting charset-normalizer~=2.0.0; python_version >= "3"
  Using cached charset_normalizer-2.0.7-py3-none-any.whl (38 kB)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.24.0->pandas-profiling)
(1.25.9)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-
packages (from matplotlib>=3.2.0->pandas-profiling) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.2.0->pandas-
profiling) (1.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.2.0->pandas-
profiling) (2.4.7)
Requirement already satisfied: decorator>=4.3.0 in
/opt/conda/lib/python3.7/site-packages (from
networkx>=2.4->visions[type_image_path]==0.7.4->pandas-profiling) (4.4.2)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages
(from imagehash; extra ==
"type_image_path"->visions[type_image_path]==0.7.4->pandas-profiling) (1.14.0)
```

```
Requirement already satisfied: PyWavelets in /opt/conda/lib/python3.7/site-
packages (from imagehash; extra ==
"type_image_path"->visions[type_image_path]==0.7.4->pandas-profiling) (1.1.1)
ERROR: phik 0.12.0 has requirement scipy>=1.5.2, but you'll have scipy
1.4.1 which is incompatible.
Installing collected packages: multimethod, tqdm, tangled-up-in-unicode,
imagehash, visions, joblib, phik, typing-extensions, pydantic, charset-
normalizer, requests, htmlmin, missingno, markupsafe, pandas-profiling
  Attempting uninstall: tqdm
    Found existing installation: tqdm 4.45.0
    Uninstalling tqdm-4.45.0:
      Successfully uninstalled tqdm-4.45.0
  Attempting uninstall: joblib
    Found existing installation: joblib 0.15.1
    Uninstalling joblib-0.15.1:
      Successfully uninstalled joblib-0.15.1
  Attempting uninstall: typing-extensions
    Found existing installation: typing-extensions 3.7.4.2
    Uninstalling typing-extensions-3.7.4.2:
      Successfully uninstalled typing-extensions-3.7.4.2
  Attempting uninstall: requests
    Found existing installation: requests 2.23.0
    Uninstalling requests-2.23.0:
      Successfully uninstalled requests-2.23.0
  Attempting uninstall: markupsafe
    Found existing installation: MarkupSafe 1.1.1
    Uninstalling MarkupSafe-1.1.1:
      Successfully uninstalled MarkupSafe-1.1.1
Successfully installed charset-normalizer-2.0.7 htmlmin-0.1.12 imagehash-4.2.1
joblib-1.0.1 markupsafe-2.0.1 missingno-0.5.0 multimethod-1.6 pandas-
profiling-3.1.0 phik-0.12.0 pydantic-1.8.2 requests-2.26.0 tangled-up-in-
unicode-0.1.0 tqdm-4.62.3 typing-extensions-3.10.0.2 visions-0.7.4
Note: you may need to restart the kernel to use updated packages.
```

[23]:
```
# Using pandas profiling to inspect our dataset. Pandas profiling gives you␣
↪very detailed information on variables and correlations
from pandas_profiling import ProfileReport as pp
```

[24]:
```
profile = pp(df_tor)
profile.to_notebook_iframe()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
<IPython.core.display.HTML object>
```

[25]:
```
profile.to_file("profile_bf_cleaning_tor.html")
```

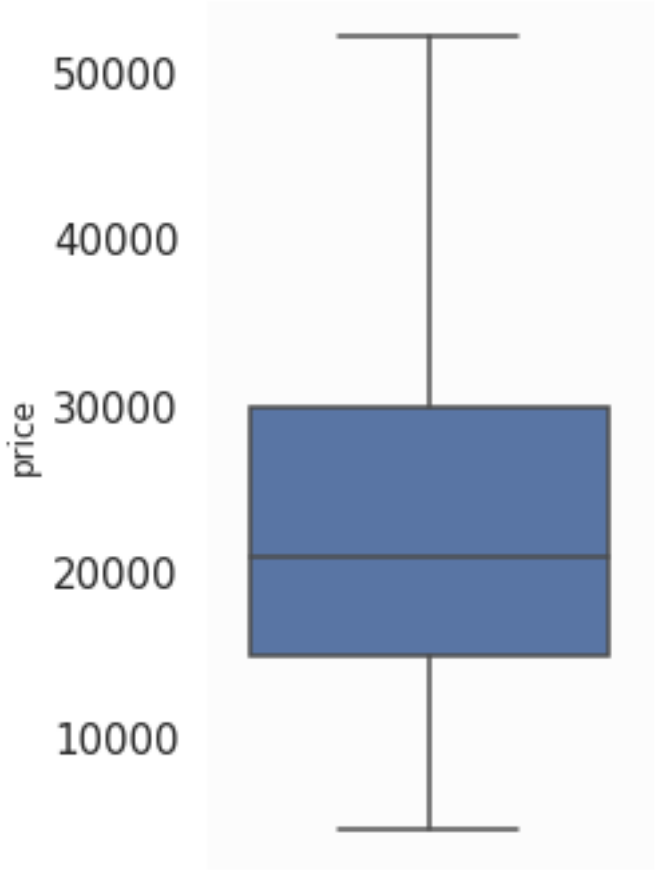
```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

[26]:
```
rr=sorted(df_tor["price"])
quantile1, quantile3= np.percentile(rr,[1,99])
print(quantile1,quantile3)
```

```
4490.0 93999.0
```

[27]:
```
# There are extreame values exist on both side of the distribution. Also the␣
 ↪difference between 75% value and max value is too large so lets leave 1%␣
 ↪values at both ends of a distribution.These values either do not make sense␣
 ↪or will significantly impact our models.
# Remove records with prices <4490 and price > 93999.
df_tor=df_tor[(df_tor.price < 93999) & (df_tor.price > 4490 )]
```

[28]:
```
plt.figure(figsize=(3,6))
sns.boxplot(y='price', data=df_tor, showfliers=False);
```



[29]:
```
# Any car that was made before 1990 should be considered as vintage or classic
 ↪car. There is a special group of people who colloect these type
 ↪cars,however, these cars should not be inlcuded in this study. cars made in
 ↪2022 are probably entered by mistake
df_tor=df_tor[(df_tor.year > 1990) & (df_tor.year < 2022 )]
```

[30]:
```
# Below attributes have missing values, they need to be cleaned up:
missing_values_count = df_tor.isnull().sum()
missing_values_count
```

[30]:
```
price             0
miles          1557
year              0
make              0
model           168
trim            947
body_type      1418
vehicle_type   1572
drivetrain      665
```

```
transmission     654
fuel_type       2171
engine_size     2177
engine_block    2217
dtype: int64
```

[31]:
```
len(df_tor)
```

[31]:
```
13523
```

[32]:
```
# The trim attributes has too much inconsistent data with high cardinality, so␣
 ↪we will drop this column.
df_tor=df_tor.drop(["trim"],axis=1)
```

[33]:
```
# There are records with missing values in 'model'.These records have to be␣
 ↪removed as model can't be replaced easily without changing the reality of␣
 ↪the instances.
df_tor=df_tor.dropna(subset=['model'])
```

[34]:
```
missing_values_count = df_tor.isnull().sum()
missing_values_count
```

[34]:
```
price             0
miles          1544
year              0
make              0
model             0
body_type      1256
vehicle_type   1405
drivetrain      652
transmission    641
fuel_type      2003
engine_size    2009
engine_block   2049
dtype: int64
```

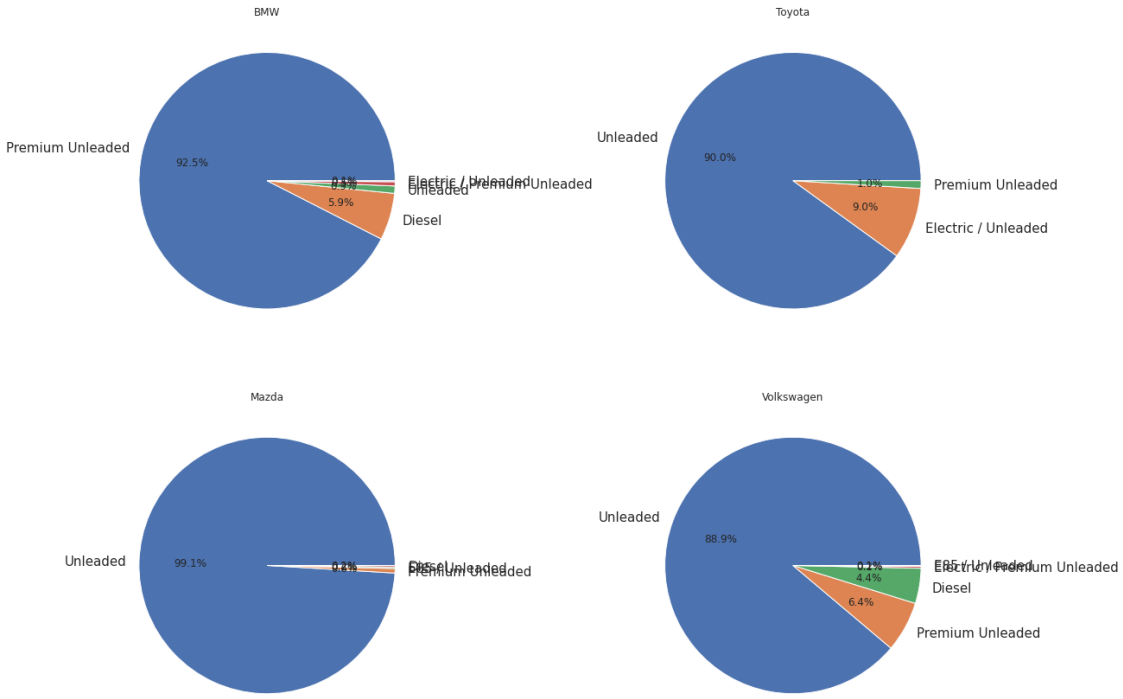
[35]:
```
# I am droping records with missnig drivetrain and transmission as well since␣
 ↪they are only account for less than 1% of total records.
df_tor=df_tor.dropna(subset=['drivetrain','transmission'])
```

[36]:
```
missing_values_count = df_tor.isnull().sum()
missing_values_count
```

[36]:
```
price             0
miles          1439
year              0
make              0
```

```
model              0
body_type       1169
vehicle_type    1313
drivetrain         0
transmission       0
fuel_type       1338
engine_size     1375
engine_block    1394
dtype: int64
```

[37]:
```
# Fuel_type can be an important feature, instead of simply dropping all the␣
 ↪records and also for the purpose of learning, lets try to replace these␣
 ↪missing values.
# Panadas profiling indicates that fuel_type has a relative high correlation␣
 ↪with make. We will select 4 random car brands and compare its fuel_type in␣
 ↪pie charts.
# Pie charts show that these brands prefer to produce cars with certain␣
 ↪fuel_type
plt.figure(figsize=(20,15))
plt.subplot(221)
df_tor["fuel_type"][df_tor["make"]=='BMW'].value_counts().plot.pie(autopct = '%.
 ↪1f%%', title="BMW")
plt.axis('off')

plt.subplot(222)
df_tor["fuel_type"][df_tor["make"]=='Toyota'].value_counts().plot.pie(autopct =␣
 ↪'%.1f%%', title="Toyota")
plt.axis('off')

plt.subplot(223)
df_tor["fuel_type"][df_tor["make"]=='Mazda'].value_counts().plot.pie(autopct =␣
 ↪'%.1f%%', title="Mazda")
plt.axis('off')

plt.subplot(224)
df_tor["fuel_type"][df_tor["make"]=='Volkswagen'].value_counts().plot.
 ↪pie(autopct = '%.1f%%', title="Volkswagen")
plt.axis('off')

plt.show()
```



[38]:
```
# The approach here is to replace missing fuel_type with the most common value␣
↪for each brand. We first try to create a list with each brand and its most␣
↪commonly used fuel_type
fuel_count = df_tor.groupby(['fuel_type','make'], sort=True).size().
↪reset_index(name='Count').sort_values(['make','Count'],ascending=False).
↪groupby(['make']).first()
```

[39]:
```
fuel_count.reset_index(inplace=True)
```

[40]:
```
fuel_pair = fuel_count[['make','fuel_type']]
```

[41]:
```
fuel_replace = pd.Series(fuel_pair.fuel_type.values,index = fuel_pair.make)
```

[42]:
```
# Replace with missing fuel_type:
df_tor['fuel_type'] = df_tor['fuel_type'].fillna(df_tor['make'].apply(lambda x:␣
↪fuel_replace.get(x)))
```

[43]:
```
# Miles
# miles has a strong negative correlation with year as shown below:
df_tor.corr()
```

[43]:

	price	miles	year	engine_size
price	1.000000	-0.476104	0.474156	0.311024
miles	-0.476104	1.000000	-0.717641	0.273222

```
year         0.474156 -0.717641  1.000000    -0.291305
engine_size  0.311024  0.273222 -0.291305     1.000000
```

[44]:
```
# Replacing missing values in miles with median value of each year.
miles_median =  dict(df_tor.groupby('year')['miles'].median())
```

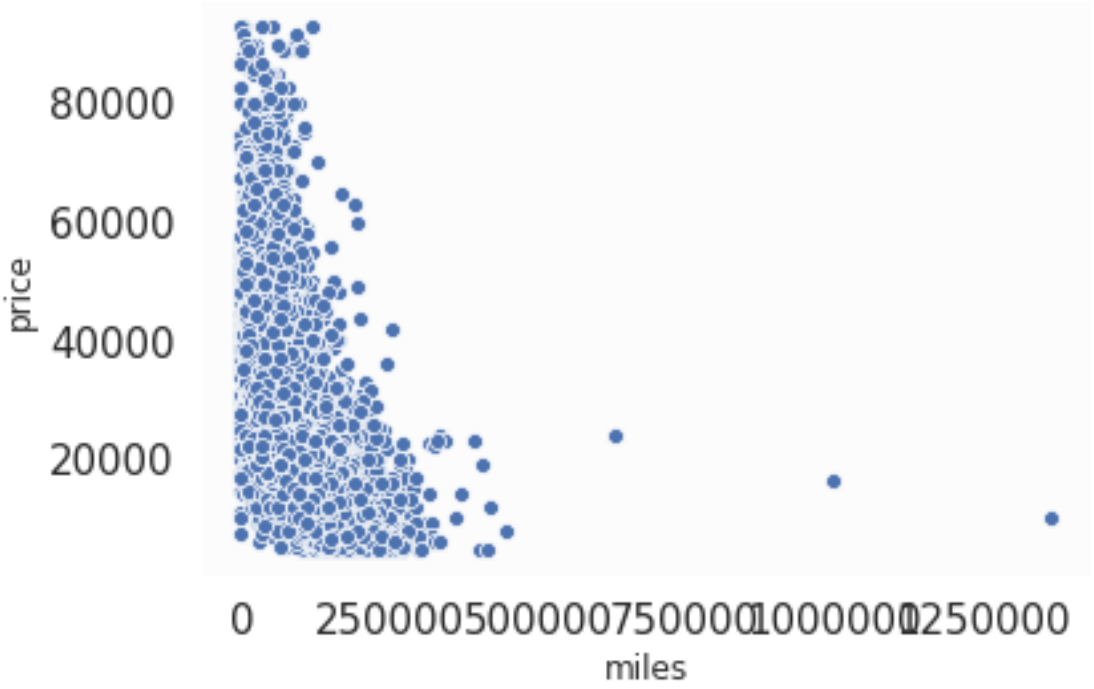
[45]:
```
df_tor['miles'] = df_tor['miles'].fillna(df_tor['year'].apply(lambda x:␣
↪miles_median.get(x)))
```

[46]:
```
# Removing the left missing values in miles because no information available␣
↪for that year.
df_tor=df_tor.dropna(subset=['miles'])
```

[47]:
```
missing_values_count = df_tor.isnull().sum()
missing_values_count
```

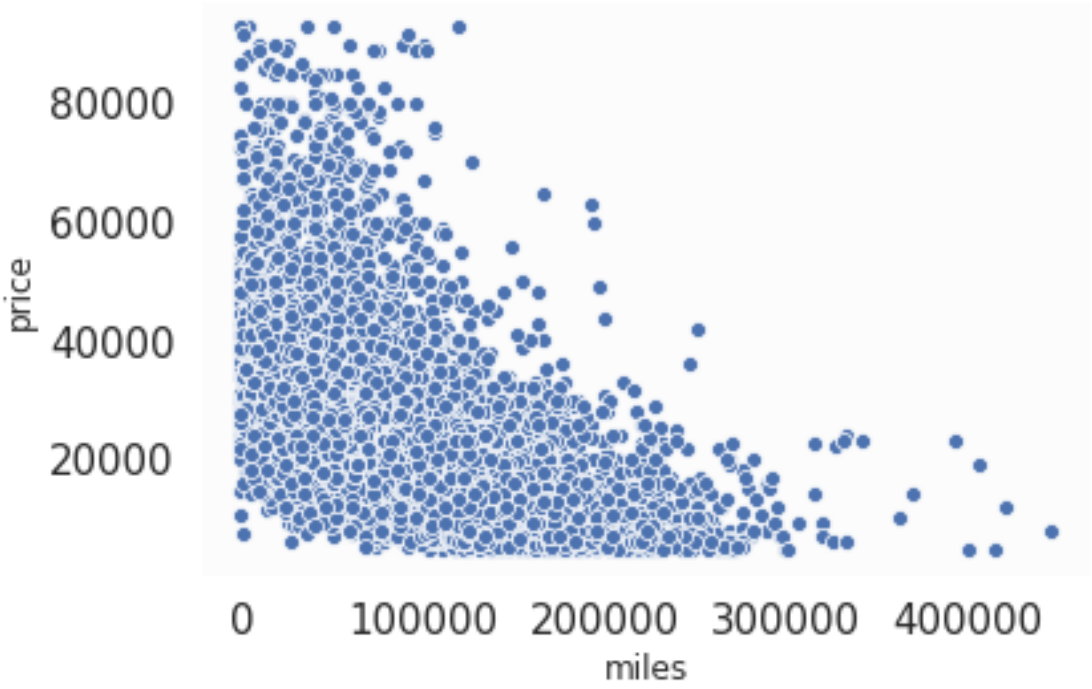
[47]:
```
price              0
miles              0
year               0
make               0
model              0
body_type       1169
vehicle_type    1313
drivetrain         0
transmission       0
fuel_type          0
engine_size     1375
engine_block    1394
dtype: int64
```

[48]:
```
# Lets see if miles has any outliers
ax = sns.scatterplot(x="miles", y="price", data=df_tor)
ax.get_xaxis().get_major_formatter().set_scientific(False)
ax.get_yaxis().get_major_formatter().set_scientific(False)
```



[49]:
```
# From the scatterplot above we can easily see some outliers, let's first only␣
↪keep cars between 1 miles and 500000 miles.
df_tor=df_tor[(df_tor.miles < 500000) & (df_tor.miles > 1 )]
```

[50]:
```
#Miles is another important feature, lets see if it has any outliers:
ax = sns.scatterplot(x="miles", y="price", data=df_tor)
ax.get_xaxis().get_major_formatter().set_scientific(False)
ax.get_yaxis().get_major_formatter().set_scientific(False)
```



[51]:
```
# There are still several instances where miles > 300000, lets remove those as well
df_tor=df_tor[(df_tor.miles < 300000) & (df_tor.miles > 1 )]
```

[52]:
```
# Body_type
# Pandas profiling indicates that body type has a high correltion with vehicle type, we will replace missing body_type based on vehicle type. Lets take a look at these two attributes together.
df_bt = df_tor.groupby(['vehicle_type','body_type'], sort=True).size().reset_index(name='Count')
df_bt
```

[52]:

	vehicle_type	body_type	Count
0	Car	Car Van	1
1	Car	Convertible	84
2	Car	Coupe	293
3	Car	Hatchback	1008
4	Car	Micro Car	6
5	Car	Mini Mpv	20
6	Car	SUV	1
7	Car	Sedan	4127
8	Car	Targa	4
9	Car	Wagon	107
10	Truck	Car Van	8

```
11        Truck      Cargo Van    111
12        Truck    Chassis Cab     10
13        Truck        Cutaway     22
14        Truck       Mini Mpv      4
15        Truck        Minivan    336
16        Truck  Passenger Van     13
17        Truck         Pickup    438
18        Truck            SUV   4730
```

[53]:
```
# We find out that there are records missing both attributes
df_tor.loc[(df_tor['vehicle_type'].isna()) & (df_tor['body_type'].isna())].
↪count()
```

[53]:
```
price           1154
miles           1154
year            1154
make            1154
model           1154
body_type          0
vehicle_type       0
drivetrain      1154
transmission    1154
fuel_type       1154
engine_size        0
engine_block       0
dtype: int64
```

[54]:
```
# Lets drop these records because the values cant be replaced by the approach␣
↪we are using.
df_tor=df_tor.dropna(subset=['vehicle_type','body_type'],how = 'all')
```

[55]:
```
missing_values_count = df_tor.isnull().sum()
missing_values_count
```

[55]:
```
price             0
miles             0
year              0
make              0
model             0
body_type        11
vehicle_type    155
drivetrain        0
transmission      0
fuel_type         0
engine_size     217
engine_block    236
dtype: int64
```

[56]:
```
len(df_tor)
```

[56]:
```
11489
```

[57]:
```
# Replacing missing body_type based on vehicle type
df_bodytype = df_tor.groupby(['vehicle_type','body_type'], sort=True).size().
 ↪reset_index(name='Count')
df_bodytype
```

[57]:

	vehicle_type	body_type	Count
0	Car	Car Van	1
1	Car	Convertible	84
2	Car	Coupe	293
3	Car	Hatchback	1008
4	Car	Micro Car	6
5	Car	Mini Mpv	20
6	Car	SUV	1
7	Car	Sedan	4127
8	Car	Targa	4
9	Car	Wagon	107
10	Truck	Car Van	8
11	Truck	Cargo Van	111
12	Truck	Chassis Cab	10
13	Truck	Cutaway	22
14	Truck	Mini Mpv	4
15	Truck	Minivan	336
16	Truck	Passenger Van	13
17	Truck	Pickup	438
18	Truck	SUV	4730

[58]:
```
# The approach is to replace the missing value with most common body type for␣
 ↪each vehicle type
df_tor.loc[(df_tor['vehicle_type'] == 'Car') & (df_tor['body_type'].
 ↪isna()),'body_type'] = 'Sedan'
df_tor.loc[(df_tor['vehicle_type'] == 'Truck') & (df_tor['body_type'].
 ↪isna()),'body_type'] = 'SUV'
```

[59]:
```
# Now lets deal with missing vehicle type, similarly we will replace missing␣
 ↪vehicle type based on body_type:
df_vechile = df_tor.groupby(['body_type','vehicle_type'], sort=True).size().
 ↪reset_index(name='Count')
df_vechile
# We can see that in most cases, each body_type corresponding to one vehicle␣
 ↪type. Some incosistent data points are probably entered by mistake.
```

[59]:

	body_type	vehicle_type	Count
0	Car Van	Car	1
1	Car Van	Truck	8
2	Cargo Van	Truck	111
3	Chassis Cab	Truck	10
4	Convertible	Car	84
5	Coupe	Car	293
6	Cutaway	Truck	22
7	Hatchback	Car	1008
8	Micro Car	Car	6
9	Mini Mpv	Car	20
10	Mini Mpv	Truck	4
11	Minivan	Truck	336
12	Passenger Van	Truck	13
13	Pickup	Truck	438
14	SUV	Car	1
15	SUV	Truck	4730
16	Sedan	Car	4138
17	Targa	Car	4
18	Wagon	Car	107

[60]:
```
# Lets first drop these inconsistent records:
df_tor = df_tor.drop(df_tor[(df_tor['body_type'] == 'Car Van') &␣
 ↪(df_tor['vehicle_type'] == 'Car')].index)
```

[61]:
```
df_tor = df_tor.drop(df_tor[(df_tor['body_type'] == 'Mini Mpv') &␣
 ↪(df_tor['vehicle_type'] == 'Truck')].index)
```

[62]:
```
df_tor = df_tor.drop(df_tor[(df_tor['body_type'] == 'SUV') &␣
 ↪(df_tor['vehicle_type'] == 'Car')].index)
```

[63]:
```
# Replacing missing vehicle type based on its body type
df_vechile = df_tor.groupby(['body_type','vehicle_type'], sort=True).size().
 ↪reset_index(name='Count')
df_vechile
```

[63]:

	body_type	vehicle_type	Count
0	Car Van	Truck	8
1	Cargo Van	Truck	111
2	Chassis Cab	Truck	10
3	Convertible	Car	84
4	Coupe	Car	293
5	Cutaway	Truck	22
6	Hatchback	Car	1008
7	Micro Car	Car	6
8	Mini Mpv	Car	20
9	Minivan	Truck	336

```
10  Passenger Van        Truck     13
11         Pickup        Truck    438
12            SUV        Truck   4730
13          Sedan          Car   4138
14          Targa          Car      4
15          Wagon          Car    107
```

[64]:
```
vechile_pair = df_vechile[['body_type','vehicle_type']]
vechile_replace = pd.Series(vechile_pair.vehicle_type.values,index =␣
↪vechile_pair.body_type)
df_tor['vehicle_type'] = df_tor['vehicle_type'].fillna(df_tor['body_type'].
↪apply(lambda x: vechile_replace.get(x)))
```

[65]:
```
missing_values_count = df_tor.isnull().sum()
missing_values_count
```

[65]:
```
price             0
miles             0
year              0
make              0
model             0
body_type         0
vehicle_type      1
drivetrain        0
transmission      0
fuel_type         0
engine_size     216
engine_block    235
dtype: int64
```

[66]:
```
# dropping that one record since no information is available
df_tor=df_tor.dropna(subset=['vehicle_type'])
```

[67]:
```
# Engine_block
# Lets first change engine block of all electric cars to 'N/A'
df_tor.loc[df_tor.fuel_type == 'Electric', 'engine_block'] = 'N/A'
```

[68]:
```
# Pandas Profiling report indicates engine_block is highly correlated with car␣
↪makes. We will replace the missing values with most commonly used engine␣
↪block type
```

[69]:
```
block_count = df_tor.groupby(['engine_block','make'], sort=True).size().
↪reset_index(name='Count').sort_values(['make','Count'],ascending=False).
↪groupby(['make']).first()
```

[70]:
```
block_count.reset_index(inplace=True)
```

[71]:
```
block_pair = block_count[['make','engine_block']]
```

[72]:
```
block_replace = pd.Series(block_pair.engine_block.values,index = block_pair.
 ↪make)
df_tor['engine_block'] = df_tor['engine_block'].fillna(df_tor['make'].
 ↪apply(lambda x: block_replace.get(x)))
```

[73]:
```
missing_values_count = df_tor.isnull().sum()
missing_values_count
```

[73]:
```
price             0
miles             0
year              0
make              0
model             0
body_type         0
vehicle_type      0
drivetrain        0
transmission      0
fuel_type         0
engine_size     215
engine_block      6
dtype: int64
```

[74]:
```
# dropping those 6 records since no information is available
df_tor=df_tor.dropna(subset=['engine_block'])
```

[75]:
```
# Engine_size
# Lets first change engine size of all electric cars to 0
df_tor.loc[df_tor.fuel_type == 'Electric', 'engine_size'] = 0
```

[76]:
```
# Pandas Profiling report indicates engine_block is highly correlated with␣
 ↪engine_block . We will replace the missing values with most commonly used␣
 ↪engine size for each engine block
size_count = df_tor.groupby(['engine_size','engine_block'], sort=True).size().
 ↪reset_index(name='Count').
 ↪sort_values(['engine_block','Count'],ascending=False).
 ↪groupby(['engine_block']).first()
```

[77]:
```
size_count.reset_index(inplace=True)
```

[78]:
```
size_pair = size_count[['engine_block','engine_size']]
```

[79]:
```
size_replace = pd.Series(size_pair.engine_size.values,index = size_pair.
 ↪engine_block)
```

[80]:
```
df_tor['engine_size'] = df_tor['engine_size'].fillna(df_tor['engine_block'].
↪apply(lambda x: size_replace.get(x)))
```

[81]:
```
missing_values_count = df_tor.isnull().sum()
missing_values_count
```

[81]:
```
price           0
miles           0
year            0
make            0
model           0
body_type       0
vehicle_type    0
drivetrain      0
transmission    0
fuel_type       0
engine_size     0
engine_block    0
dtype: int64
```

[82]:
```
len(df_tor)
```

[82]:
```
11476
```

[83]:
```
df_tor.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11476 entries, 446 to 393573
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   price         11476 non-null  float64
 1   miles         11476 non-null  float64
 2   year          11476 non-null  float64
 3   make          11476 non-null  object
 4   model         11476 non-null  object
 5   body_type     11476 non-null  object
 6   vehicle_type  11476 non-null  object
 7   drivetrain    11476 non-null  object
 8   transmission  11476 non-null  object
 9   fuel_type     11476 non-null  object
 10  engine_size   11476 non-null  float64
 11  engine_block  11476 non-null  object
dtypes: float64(4), object(8)
memory usage: 1.1+ MB
```

[84]:
```
profile1 = pp(df_tor)
profile1.to_notebook_iframe()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[85]:
```
profile1.to_file("profile_clead_tor.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

[86]:
```
df_tor.make.value_counts().to_frame()
```

[86]:

	make
Hyundai	1243
Toyota	1133
Volkswagen	890
Honda	834
Ford	787
BMW	760
Mercedes-Benz	707
Nissan	656
Chevrolet	526
Mazda	526
Kia	387
Lexus	367
Audi	352
Dodge	308
Jeep	229
Acura	203
Land Rover	174
GMC	135
Subaru	133
INFINITI	130
Cadillac	121
RAM	107
Buick	106
MINI	92
Jaguar	80
Lincoln	79

```
Porsche          76
Chrysler         71
Volvo            70
Mitsubishi       66
GENESIS          25
Tesla            22
Maserati         19
FIAT             15
Scion            13
smart             7
Pontiac           7
Hummer            6
Alfa Romeo        6
Aston Martin      5
Saturn            1
Suzuki            1
Saab              1
```

[87]:
```
# Some brands have too few samples. For the precision of our future model, I␣
 ↪choose to remove the car brands which have less than 20 samples. It will␣
 ↪narrow the capability of our model, but in return lower the bias and␣
 ↪variance.
rm_brands = ['Maserati', 'FIAT', 'Scion', 'smart', 'Pontiac', 'Alfa Romeo',␣
 ↪'Hummer', 'Aston Martin', 'Saturn', 'Saab','Suzuki']
for brand in rm_brands:
    df_tor = df_tor[~(df_tor['make'] == brand)]
```

[88]:
```
# The end of data cleaning and processing for toronto data
```

[89]:
```
# Another dataset is selected for comparison for our reserch purpose. The city␣
 ↪selected is Boston, since it is comparable to Toronto.
# The same data cleaning and processing approach will be applied to the new␣
 ↪dataset.
```

[90]:
```
# Load data
import dask.dataframe as dd
df2 = dd.read_csv("us-dealers-used.csv",dtype={'fuel_type': 'object',␣
 ↪'engine_block': 'object','zip': 'object', 'year': 'float64'})
```

[91]:
```
# drop irrelevant columns for this research
drop_column = ['id', 'vin', 'stock_no', 'seller_name','street','zip']
```

[92]:
```
df2 = df2.drop(columns = drop_column)
```

[93]:
```
df2 =df2[df2.state == "MA"]
```

[94]:
```
# onley select city as Boston
df2=df2[df2.city == "Boston"]
```

[95]:
```
df_bs = df2.compute()
```

[96]:
```
len(df_bs)
```

[96]: 12780

[97]:
```
# drop state, city and trim
df_bs=df_bs.drop(["state","city","trim"],axis=1)
```

[98]:
```
df_bs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12780 entries, 141 to 9367
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   price         12664 non-null  float64
 1   miles         12760 non-null  float64
 2   year          12780 non-null  float64
 3   make          12780 non-null  object
 4   model         12775 non-null  object
 5   body_type     12761 non-null  object
 6   vehicle_type  12757 non-null  object
 7   drivetrain    12772 non-null  object
 8   transmission  12767 non-null  object
 9   fuel_type     12749 non-null  object
 10  engine_size   12655 non-null  float64
 11  engine_block  12629 non-null  object
dtypes: float64(4), object(8)
memory usage: 1.3+ MB
```

[99]:
```
missing_values_count = df_bs.isnull().sum()
missing_values_count
```

[99]:
```
price           116
miles            20
year              0
make              0
model             5
body_type        19
vehicle_type     23
drivetrain        8
transmission     13
fuel_type        31
```

```
engine_size     125
engine_block    151
dtype: int64
```

[100]:
```
# sample data:
df_bs.head(20)
```

[100]:

	price	miles	year	make	model	body_type	vehicle_type	drivetrain	transmission	fuel_type	engine_size	engine_block
141	22998.0	87571.0	2013.0	Mercedes-Benz	CLS-Class	Coupe	Car	4WD	Automatic	Premium Unleaded	4.7	125
512	10998.0	34998.0	2015.0	Chevrolet	Spark	Hatchback	Car	FWD	Automatic	Unleaded	1.2	125
882	58998.0	30819.0	2018.0	Mercedes-Benz	GLS-Class	SUV	Truck	4WD	Automatic	Premium Unleaded	4.7	125
3534	79998.0	10492.0	2019.0	Porsche	Cayenne	SUV	Truck	4WD	Automatic	Premium Unleaded	2.9	125
4772	58498.0	21363.0	2017.0	Mercedes-Benz	GLS-Class	SUV	Truck	4WD	Automatic	Premium Unleaded	4.7	125
5289	58998.0	45391.0	2017.0	Mercedes-Benz	GLE-Class	SUV	Truck	4WD	Automatic	Premium Unleaded	5.5	125
6160	NaN	8.0	2021.0	BMW	8 Series	Sedan	Car	4WD	Automatic	Premium Unleaded	4.4	125
6700	23498.0	58970.0	2017.0	GMC	Savana Cargo	Cargo Van	Truck	RWD	Automatic	Unleaded	4.8	125
8004	53998.0	27262.0	2014.0	BMW	6 Series	Sedan	Car	RWD	Automatic	Premium Unleaded	4.4	125
8339	43900.0	37212.0	2012.0	Porsche	Boxster	Roadster	Car	RWD	Manual	Premium Unleaded	3.4	125
8657	22498.0	87571.0	2013.0	Mercedes-Benz	CLS-Class	Coupe	Car	4WD	Automatic	Premium Unleaded	4.7	125
9003	189998.0	31385.0	2014.0	Ferrari	458 Spider	Convertible	Car	RWD	Automatic	Premium Unleaded	4.5	125
9491	40998.0	62303.0	2015.0	Mercedes-Benz	S-Class	Sedan	Car	4WD	Automatic	Premium Unleaded	4.7	125
9643	299998.0	559.0	2020.0	Bentley	Mulsanne	Sedan	Car	RWD	Automatic	Premium Unleaded	6.8	125
10280	84998.0	18663.0	2018.0	Porsche	Panamera	Hatchback	Car	4WD	Automatic	Premium Unleaded	2.9	125
10650	122998.0	28419.0	2018.0	Mercedes-Benz	G-Class	SUV	Truck	4WD	Automatic	Premium Unleaded	5.5	125
11316	83998.0	17534.0	2018.0	BMW	M5	Sedan	Car	4WD	Automatic	Premium Unleaded	4.4	125
11771	122998.0	28419.0	2018.0	Mercedes-Benz	G-Class	SUV						
12567	122998.0	28419.0	2018.0	Mercedes-Benz	G-Class	SUV						
13906	49998.0	24160.0	2017.0	Mercedes-Benz	CLS-Class	Coupe						

```
11771        Truck        4WD    Automatic  Premium Unleaded          5.5
12567        Truck        4WD    Automatic  Premium Unleaded          5.5
13906          Car        4WD    Automatic  Premium Unleaded          4.7

      engine_block
141              V
512              I
882              V
3534             V
4772             V
5289             V
6160             V
6700             V
8004             V
8339             H
8657             V
9003             V
9491             V
9643             V
10280            V
10650            V
11316            V
11771            V
12567            V
13906            V
```

[101]:
```
# Lets drop records without prices, since these records will be useless for our␣
 ↪analysis
df_bs=df_bs.dropna(subset=['price'])
```

[102]:
```
profile2 = pp(df_bs)
profile2.to_notebook_iframe()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[103]:
```
profile2.to_file("profile_bf_cleaing_bos.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

[104]:
```
# We will simply drop records with missing values, since they only account for␣
 ↪a very small percentage of the whole dataset.
df_bs=df_bs.
 ↪dropna(subset=['miles','model','body_type','vehicle_type','drivetrain','transmission','fuel
```

[105]:
```
missing_values_count = df_bs.isnull().sum()
missing_values_count
```

[105]:
```
price           0
miles           0
year            0
make            0
model           0
body_type       0
vehicle_type    0
drivetrain      0
transmission    0
fuel_type       0
engine_size     0
engine_block    0
dtype: int64
```

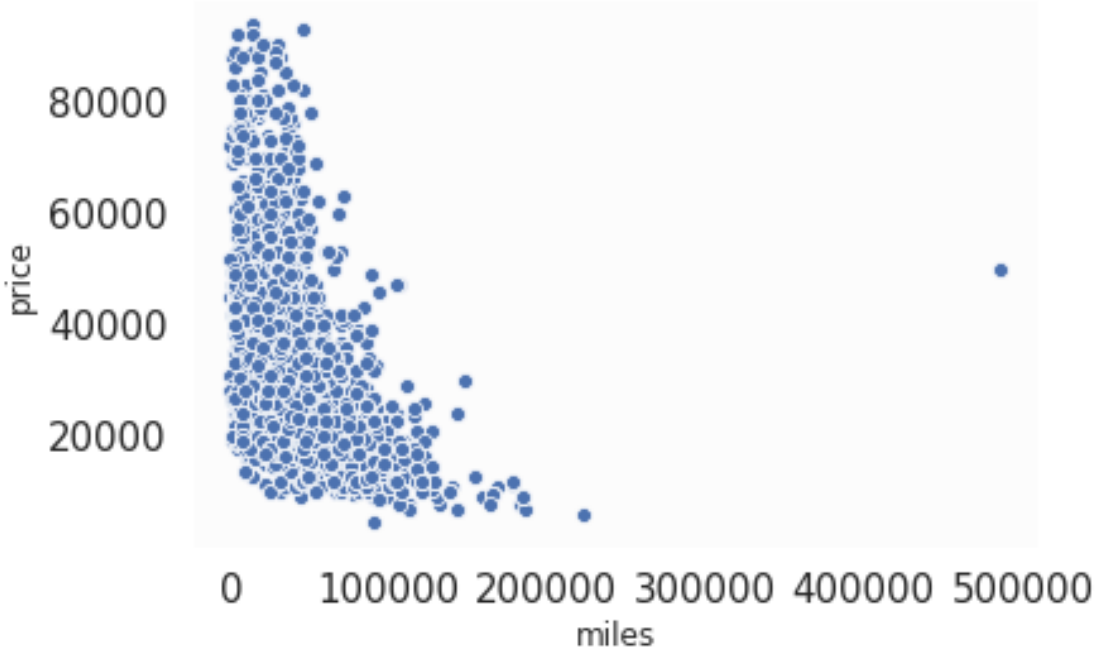
[106]:
```
len(df_bs)
```

[106]:
```
12495
```

[107]:
```
# Price:
# Keep the same price range as toronto
df_bs=df_bs[(df_bs.price < 93999) & (df_bs.price > 4490 )]
```

[108]:
```
# Year
# Any car that was made before 1990 should be considered as vintage or classic␣
 ↪car. There is a special group of people who colloect these cars,however,␣
 ↪these cars should not be inlcuded in this study.
df_bs=df_bs[(df_bs.year > 1990)]
```

[109]:
```
# Miles
# Lets see if miles has any outliers
ax = sns.scatterplot(x="miles", y="price", data=df_bs)
ax.get_xaxis().get_major_formatter().set_scientific(False)
ax.get_yaxis().get_major_formatter().set_scientific(False)
```

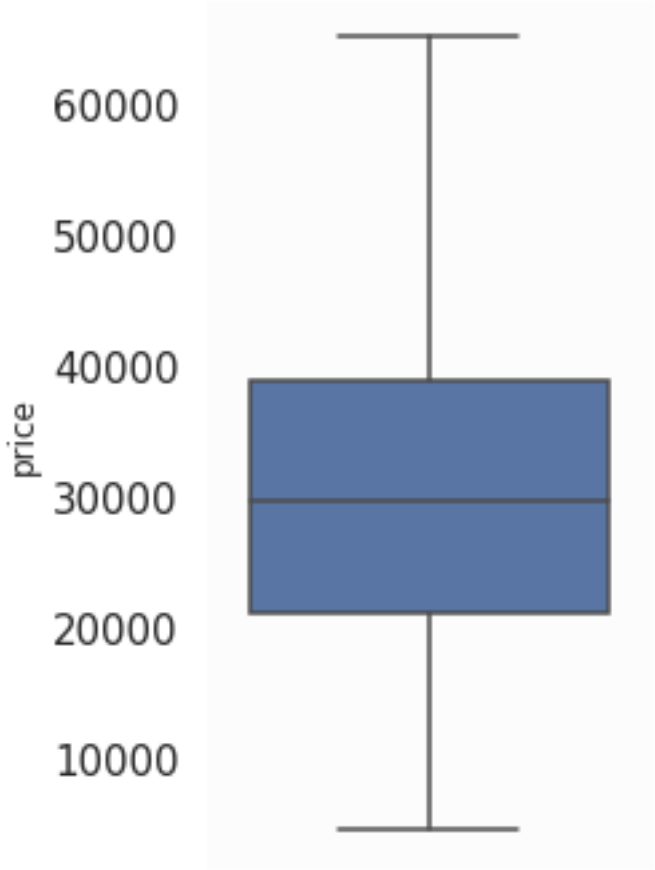


[110]:
```
# Remove outliers
df_bs=df_bs[(df_bs.miles < 300000) & (df_bs.miles > 1 )]
```

[111]:
```
# Engine_block
# Change engine block of all electric cars to 'N/A'
df_bs.loc[df_bs.fuel_type == 'Electric', 'engine_block'] = 'N/A'
```

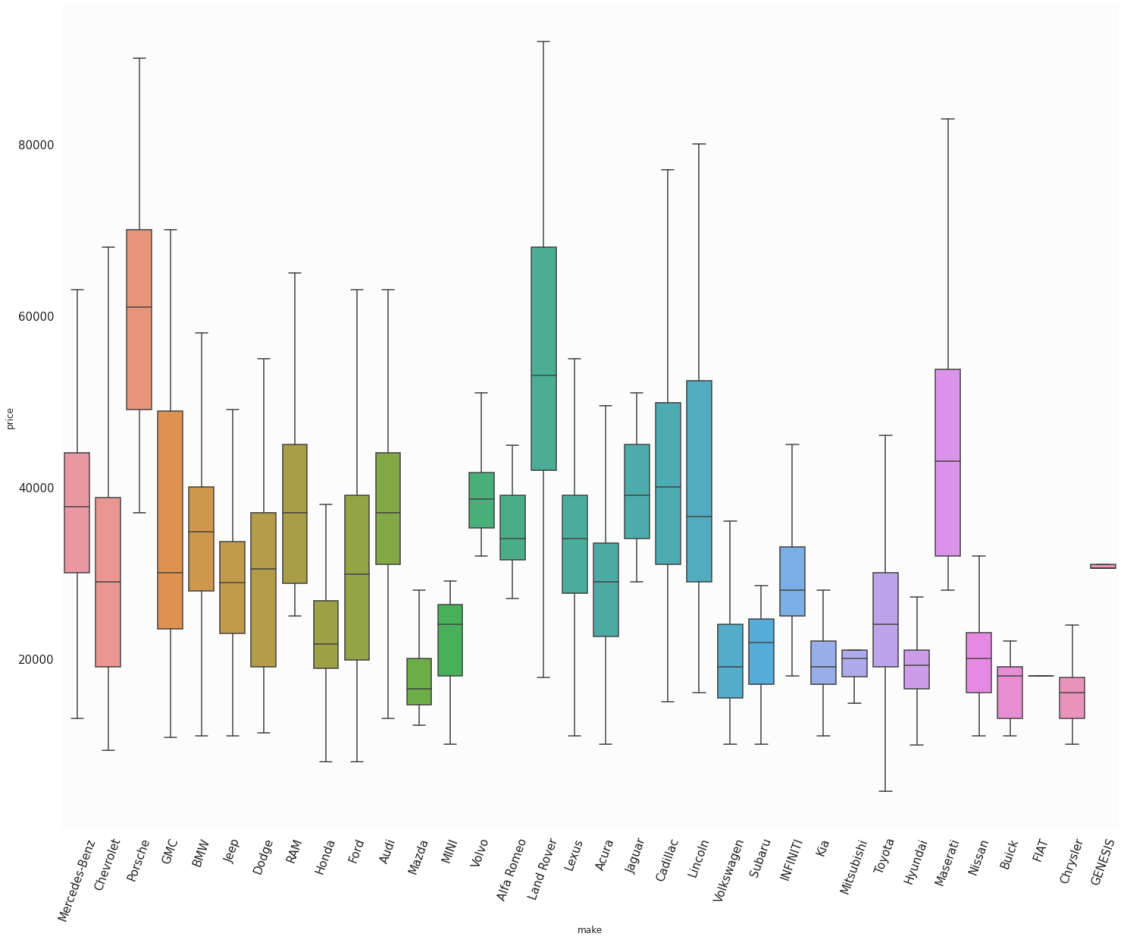
[112]:
```
# Engine_size
# Change engine size of all electric cars to 0
df_bs.loc[df_bs.fuel_type == 'Electric', 'engine_size'] = 0
```

[113]:
```
plt.figure(figsize=(3,6))
sns.boxplot(y='price', data=df_bs, showfliers=False);
```

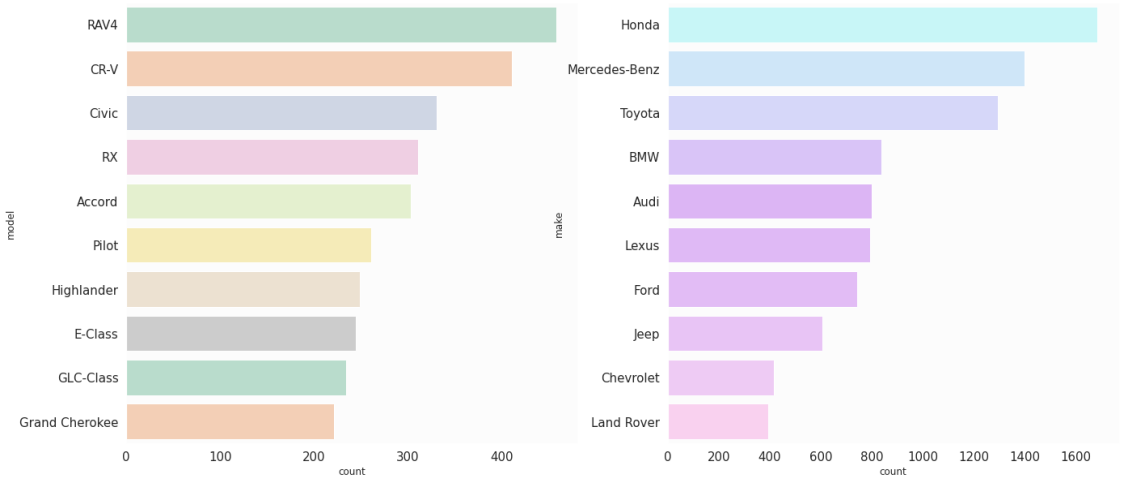


[114]:
```
plt.figure(figsize=(25,20))
plt.xticks(rotation=70)
sns.boxplot(y='price', x='make', data=df_bs, showfliers=False)
```

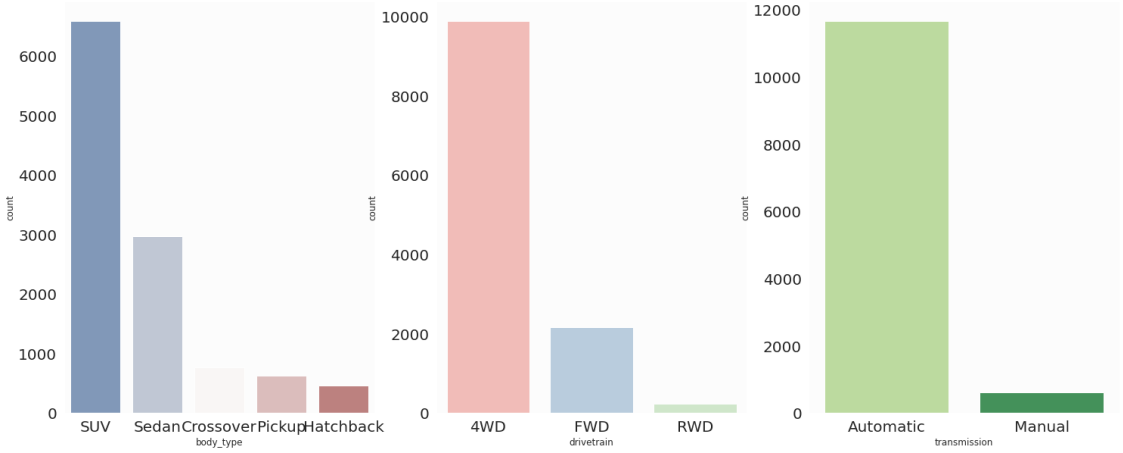
[114]: <matplotlib.axes._subplots.AxesSubplot at 0x7f38f9a8bf90>



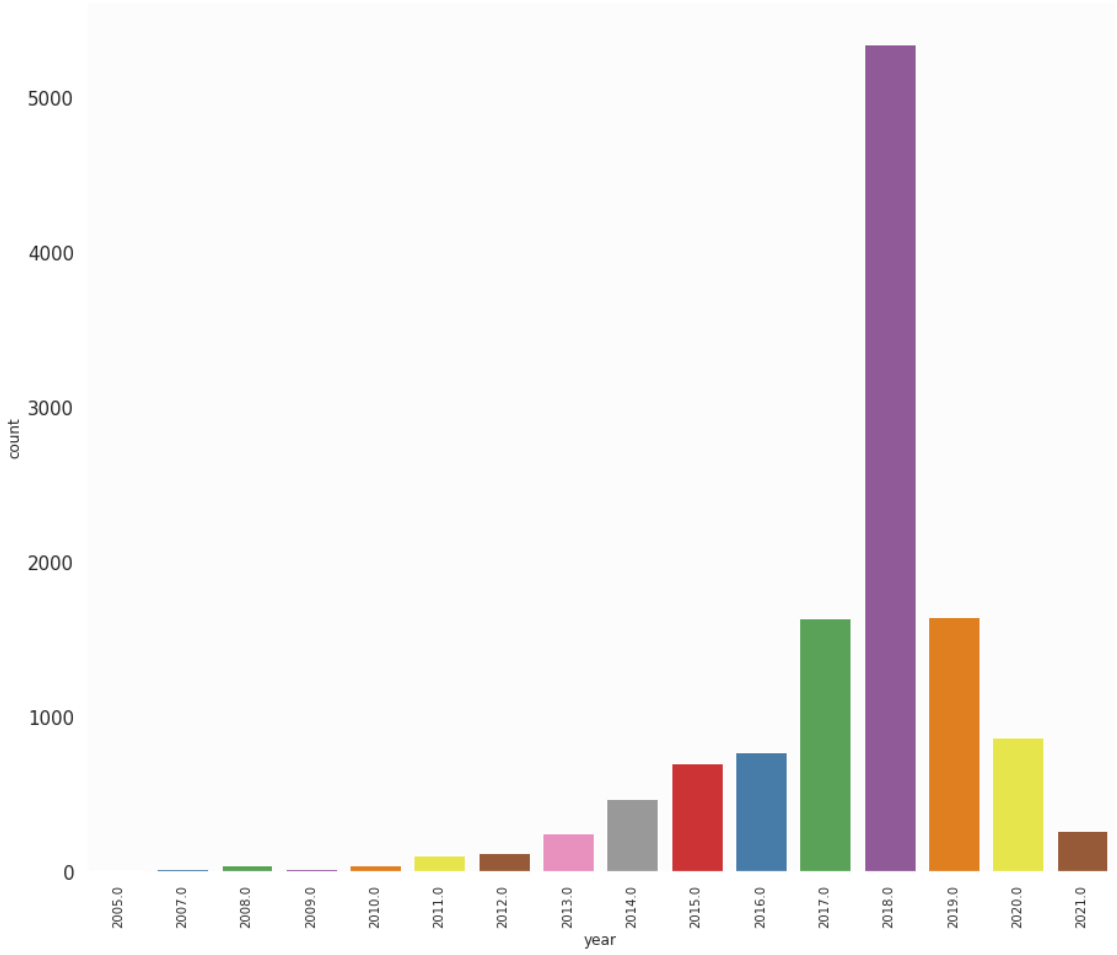
[115]:
```
colors = ["#C0FDFF",␣
 ↪"#C8E7FF","#D0D1FF","#D8BBFF","#DEAAFF","#E2AFFF","#E5B3FE","#ECBCFD","#F3C4FB","#FFCBF2"]
sns.set_palette(sns.color_palette(colors))


fig, ax =plt.subplots(1,2,figsize=(22, 10))
sns.set(rc={"axes.facecolor":"#fcfcfc", "axes.grid":False,'xtick.labelsize':
 ↪20,'ytick.labelsize':20})
sns.countplot(y ='make',data = df_bs,order=df_bs.make.value_counts().iloc[:10].
 ↪index, ax = ax[1],palette = colors)
sns.countplot(y ='model',data = df_bs,order=df_bs.model.value_counts().iloc[:
 ↪10].index, ax= ax[0],palette = "Pastel2")
fig.show()
```



[116]:
```
fig, ax =plt.subplots(1,3,figsize=(25, 10))
sns.set(rc={"axes.facecolor":"#fcfcfc", "axes.grid":False,'xtick.labelsize':
 ↪15,'ytick.labelsize':15})
sns.countplot(x ='body_type',data = df_bs,order=df_bs.body_type.value_counts().
 ↪iloc[:5].index, ax = ax[0],palette="vlag")
sns.countplot(x ='drivetrain',data = df_bs,order=df_bs.drivetrain.
 ↪value_counts().iloc[:5].index, ax = ax[1],palette="Pastel1")
sns.countplot(x ='transmission',data = df_bs,order=df_bs.transmission.
 ↪value_counts().iloc[:5].index, ax = ax[2],palette="YlGn")
fig.show()
```



[117]:
```
plt.figure(figsize=(15, 13))
ax = sns.countplot(x = 'year', data=df_bs, palette='Set1')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90,fontsize=10);
```



[118]:
```
# Now lets deal with missing vehicle type, similarly we will replace missing␣
 ↪vehicle type based on body_type:
df_bv = df_bs.groupby(['body_type','vehicle_type'], sort=True).size().
 ↪reset_index(name='Count')
df_bv
# We can see that in most cases, each body_type corresponding to one vehicle␣
 ↪type. Some outliers are probably entered by mistake.
```

[118]:

	body_type	vehicle_type	Count
0	Cargo Van	Truck	63
1	Chassis Cab	Truck	10
2	Convertible	Car	64
3	Coupe	Car	468
4	Crossover	Car	9
5	Crossover	Truck	765
6	Hatchback	Car	468
7	Mini Mpv	Truck	7

```
8          Minivan        Truck    114
9    Passenger Van        Truck     16
10          Pickup        Truck    623
11        Roadster          Car      5
12             SUV          Car     16
13             SUV        Truck   6576
14           Sedan          Car   2965
15           Wagon          Car    118
```

[119]:
```
df_bs = df_bs.drop(df_bs[(df_bs['body_type'] == 'Crossover') &␣
↪(df_bs['vehicle_type'] == 'Car')].index)
df_bs = df_bs.drop(df_bs[(df_bs['body_type'] == 'SUV') & (df_bs['vehicle_type']␣
↪== 'Car')].index)
```

[120]:
```
df_bs.make.value_counts().to_frame()
```

[120]:
```
                make
Honda           1684
Mercedes-Benz   1397
Toyota          1284
BMW              823
Audi             798
Lexus            794
Ford             744
Jeep             606
Chevrolet        417
Land Rover       393
Hyundai          344
Acura            334
Cadillac         294
Nissan           275
Porsche          232
Lincoln          186
Volvo            185
INFINITI         170
Subaru           156
Kia              145
Alfa Romeo       137
MINI             137
Volkswagen       120
Dodge            112
RAM               94
Jaguar            93
GMC               89
Mazda             74
Buick             50
Mitsubishi        30
```

```
Chrysler         29
Maserati         26
FIAT              5
GENESIS           5
```

[121]:
```
# Remove the car brands which have less than 20 samples. It will narrow the␣
 ↪capability of our model, but in return lower the bias and variance.
rm_brands = ['FIAT', 'GENESIS']
for brand in rm_brands:
    df_bs = df_bs[~(df_bs['make'] == brand)]
```

[122]:
```
profile3 = pp(df_bs)
profile3.to_notebook_iframe()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[123]:
```
profile2.to_file("profile_cleaned_bos.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

[124]:
```
# Before we start to train the models, We also need to normalize the values in␣
 ↪the numerical features ("year" ,"engine_size" "miles"), as they do not have␣
 ↪the same scale.
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
num_vars= ['year', 'miles', 'engine_size']
df_tor[num_vars] = scaler.fit_transform(df_tor[num_vars])
```

[125]:
```
# The first model we will implement is Linear Regression, however, linear␣
 ↪regression can not handle categorical data directly. Thus, we need to apply␣
 ↪encoding first.
# Most of our categorical features have more than two values. If we use␣
 ↪LabelEncoder then these values will be treated as ordinal ones by the␣
 ↪machine learning model.
# In this case, we will select One-hot encoder, also called as dummy encoding,␣
 ↪however, model feature has a high cardinality, we need to avoid curse of␣
 ↪dimensionality
df_tor_lr = df_tor.copy()
```

[126]:
```
!pip install feature_engine
from feature_engine.encoding import RareLabelEncoder as␣
 ↪RareLabelCategoricalEncoder
```

```
Collecting feature_engine
  Using cached feature_engine-1.1.2-py2.py3-none-any.whl (180 kB)
Requirement already satisfied: numpy>=1.18.2 in /opt/conda/lib/python3.7/site-
packages (from feature_engine) (1.18.4)
Requirement already satisfied: pandas>=1.0.3 in /opt/conda/lib/python3.7/site-
packages (from feature_engine) (1.0.3)
Requirement already satisfied: scipy>=1.4.1 in /opt/conda/lib/python3.7/site-
packages (from feature_engine) (1.4.1)
Requirement already satisfied: statsmodels>=0.11.1 in
/opt/conda/lib/python3.7/site-packages (from feature_engine) (0.11.1)
Requirement already satisfied: scikit-learn>=0.22.2 in
/opt/conda/lib/python3.7/site-packages (from feature_engine) (0.22.2.post1)
Requirement already satisfied: python-dateutil>=2.6.1 in
/opt/conda/lib/python3.7/site-packages (from pandas>=1.0.3->feature_engine)
(2.8.1)
Requirement already satisfied: pytz>=2017.2 in /opt/conda/lib/python3.7/site-
packages (from pandas>=1.0.3->feature_engine) (2020.1)
Requirement already satisfied: patsy>=0.5 in /opt/conda/lib/python3.7/site-
packages (from statsmodels>=0.11.1->feature_engine) (0.5.1)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.7/site-
packages (from scikit-learn>=0.22.2->feature_engine) (1.0.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-
packages (from python-dateutil>=2.6.1->pandas>=1.0.3->feature_engine) (1.14.0)
Installing collected packages: feature-engine
Successfully installed feature-engine-1.1.2
```

[127]:
```
# Upon further checking, a lot of the model only appear once in our dataset! So␣
 ↪the approach we use here first is to use rarelabel encoder. All models that␣
 ↪appear less than 20 times are labelled as 'Rare'.
rare_encoder = RareLabelCategoricalEncoder(
    tol=0.0018,
    n_categories=10, variables=["model"])
```

[128]:
```
rare_encoder.fit(df_tor_lr)
```

[128]:
```
RareLabelEncoder(ignore_format=False, max_n_categories=None, n_categories=10,
                 replace_with='Rare', tol=0.0018, variables=['model'])
```

[129]:
```
df_tor_lr = rare_encoder.transform(df_tor_lr)
```

[130]:
```
# Define a function to generate dummy variables and mergek it with data frame

def dummies(x,df):
```

```
    temp = pd.get_dummies(df[[x]], drop_first=True)
    df = pd.concat([df,temp], axis=1)
    df.drop([x], axis=1, inplace=True)
    return df

# Apply function to the cars_new df
df_tor_lr = dummies('make', df_tor_lr)
df_tor_lr = dummies('model', df_tor_lr)
df_tor_lr = dummies('body_type', df_tor_lr)
df_tor_lr = dummies('vehicle_type', df_tor_lr)
df_tor_lr = dummies('drivetrain', df_tor_lr)
df_tor_lr = dummies('transmission', df_tor_lr)
df_tor_lr = dummies('fuel_type', df_tor_lr)
df_tor_lr = dummies('engine_block', df_tor_lr)
```

[131]:
```
df_tor_lr.head()
```

[131]:

	price	miles	year	engine_size	make_Audi	make_BMW	\
446	39455.0	0.290890	0.869565	0.684932	0	0	
732	6450.0	0.736667	0.391304	0.739726	0	0	
992	24962.0	0.177380	0.869565	0.205479	0	0	
993	24962.0	0.172075	0.869565	0.205479	0	0	
996	24962.0	0.172075	0.869565	0.205479	0	0	

	make_Buick	make_Cadillac	make_Chevrolet	make_Chrysler	…	\
446	0	0	0	0	…	
732	0	0	0	0	…	
992	0	0	1	0	…	
993	0	0	1	0	…	
996	0	0	1	0	…	

	fuel_type_Electric / E85	fuel_type_Electric / Premium Unleaded	\
446	0	0	
732	0	0	
992	0	1	
993	0	1	
996	0	1	

	fuel_type_Electric / Unleaded	fuel_type_Premium Unleaded	\
446	0	0	
732	0	0	
992	0	0	
993	0	0	
996	0	0	

	fuel_type_Premium Unleaded / Unleaded	fuel_type_Unleaded	\
446	0	0	

```
732                                     0                  0
992                                     0                  0
993                                     0                  0
996                                     0                  0

     fuel_type_Unleaded / Unleaded  engine_block_I  engine_block_N/A  \
446                              1               0                 0
732                              0               0                 0
992                              0               1                 0
993                              0               1                 0
996                              0               1                 0

     engine_block_V
446               1
732               1
992               0
993               0
996               0

[5 rows x 192 columns]
```

[132]:
```
# Split train and test data using 8:2 ratio:
X_train, X_test, Y_train, Y_test = train_test_split(df_tor_lr.
 ↪drop('price',axis=1), df_tor_lr['price'], test_size=0.20, random_state=141)
```

[133]:
```
model_evaluation  = pd.DataFrame(columns=('r2', 'rmse'))
```

[134]:
```
from sklearn.linear_model import LinearRegression
lrm = LinearRegression()
lrm.fit(X_train,Y_train)
```

[134]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[135]:
```
# Overall the performence and accuracy is just mediocre with R2 score around 82%
from sklearn import metrics
lrm_predict = lrm.predict(X_test)

lrm_r2 = metrics.r2_score(Y_test, lrm_predict)
lrm_rmse = math.sqrt(metrics.mean_squared_error(Y_test, lrm_predict))

model_evaluation = model_evaluation.append(pd.DataFrame({'r2':[lrm_r2], 'rmse':
 ↪[lrm_rmse]}, index = ['Linear Regression']))

print('For the linear regressor, the root mean square error for the testing set␣
 ↪is:', lrm_rmse)
print('The r2 score for the testing set is:', lrm_r2)
```

```
For the linear regressor, the root mean square error for the testing set is:
5576.616035914864
The r2 score for the testing set is: 0.82619340703072
```

[136]:
```
# To test whether it is overfitting, we calculate the score for the training␣
 ↪set as well:
lrm_predict_train = lrm.predict(X_train)

lrm_r2_train = metrics.r2_score(Y_train, lrm_predict_train)
lrm_rmse_train = math.sqrt(metrics.mean_squared_error(Y_train,␣
 ↪lrm_predict_train))

print('For the linear regressor, the root mean square error for the training␣
 ↪set is:', lrm_rmse_train)
print('The r2 score for the testing set is:', lrm_r2_train)
# With similar rmse and r2 score, it seems that overfitting is not a probelm.
```

```
For the linear regressor, the root mean square error for the training set is:
5725.808257497235
The r2 score for the testing set is: 0.8217534983075683
```

[137]:
```
# We will apply linear regression on Boston dataset as well and compare the␣
 ↪difference:
scaler = MinMaxScaler()
num_vars= ['year', 'miles', 'engine_size']
df_bs[num_vars] = scaler.fit_transform(df_bs[num_vars])
```

[138]:
```
df_bs_lr = df_bs.copy()
```

[139]:
```
rare_encoder.fit(df_bs_lr)
```

[139]:
```
RareLabelEncoder(ignore_format=False, max_n_categories=None, n_categories=10,
                 replace_with='Rare', tol=0.0018, variables=['model'])
```

[140]:
```
df_bs_lr = rare_encoder.transform(df_bs_lr)
```

[141]:
```
def dummies(x,df):
    temp = pd.get_dummies(df[[x]], drop_first=True)
    df = pd.concat([df,temp], axis=1)
    df.drop([x], axis=1, inplace=True)
    return df

# Apply function to the cars_new df
df_bs_lr = dummies('make', df_bs_lr)
df_bs_lr = dummies('model', df_bs_lr)
df_bs_lr = dummies('body_type', df_bs_lr)
df_bs_lr = dummies('vehicle_type', df_bs_lr)
```

```
df_bs_lr = dummies('drivetrain', df_bs_lr)
df_bs_lr = dummies('transmission', df_bs_lr)
df_bs_lr = dummies('fuel_type', df_bs_lr)
df_bs_lr = dummies('engine_block', df_bs_lr)
```

[142]:
```
df_bs_lr.head()
```

[142]:

	price	miles	year	engine_size	make_Alfa Romeo	make_Audi	\
141	22998.0	0.394358	0.5000	0.611940	0	0	
512	10998.0	0.157576	0.6250	0.089552	0	0	
882	58998.0	0.138755	0.8125	0.611940	0	0	
3534	79998.0	0.047205	0.8750	0.343284	0	0	
4772	58498.0	0.096166	0.7500	0.611940	0	0	

	make_BMW	make_Buick	make_Cadillac	make_Chevrolet	…	\
141	0	0	0	0	…	
512	0	0	0	1	…	
882	0	0	0	0	…	
3534	0	0	0	0	…	
4772	0	0	0	0	…	

	fuel_type_E85 / Premium Unleaded	fuel_type_E85 / Unleaded	\
141	0	0	
512	0	0	
882	0	0	
3534	0	0	
4772	0	0	

	fuel_type_E85 / Unleaded; Unleaded / Unleaded	\
141	0	
512	0	
882	0	
3534	0	
4772	0	

	fuel_type_Electric / Premium Unleaded	\
141	0	
512	0	
882	0	
3534	0	
4772	0	

	fuel_type_Electric / Premium Unleaded; Premium Unleaded	\
141	0	
512	0	
882	0	
3534	0	

```
4772                                                  0

      fuel_type_Electric / Unleaded  fuel_type_Premium Unleaded  \
141                               0                           1
512                               0                           0
882                               0                           1
3534                              0                           1
4772                              0                           1

      fuel_type_Unleaded  engine_block_I  engine_block_V
141                    0               0               1
512                    1               1               0
882                    0               0               1
3534                   0               0               1
4772                   0               0               1

[5 rows x 179 columns]
```

[143]:
```
# Split train and test data using 8:2 ratio:
x_train, x_test, y_train, y_test = train_test_split(df_bs_lr.
 ↪drop('price',axis=1), df_bs_lr['price'], test_size=0.20, random_state=141)
```

[144]:
```
lrm2 = LinearRegression()
lrm2.fit(x_train,y_train)
```

[144]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[145]:
```
# The r2 score is higher and RMSE is somewhat similiar to toronto.
lrm2_predict = lrm2.predict(x_test)

lrm2_r2 = metrics.r2_score(y_test, lrm2_predict)
lrm2_rmse = math.sqrt(metrics.mean_squared_error(y_test, lrm2_predict))

model_evaluation_bs = model_evaluation.append(pd.DataFrame({'r2':[lrm2_r2],␣
 ↪'rmse':[lrm2_rmse]}, index = ['Linear Regression']))

print('For the linear regressor, the root mean square error for the testing set␣
 ↪is:', lrm2_rmse)
print('The r2 score for the testing set is:', lrm2_r2)
```

```
For the linear regressor, the root mean square error for the testing set is:
5122.596901444411
The r2 score for the testing set is: 0.8736260197642974
```

[146]:
```
# Again, to test whether it is overfitting, we calculate the score for the␣
 ↪training set as well:
lrm2_predict_train = lrm2.predict(x_train)
```

```
lrm2_r2_train = metrics.r2_score(y_train, lrm2_predict_train)
lrm2_rmse_train = math.sqrt(metrics.mean_squared_error(y_train,␣
 ↪lrm2_predict_train))

print('For the linear regressor, the root mean square error for the training␣
 ↪set is:', lrm2_rmse_train)
print('The r2 score for the testing set is:', lrm2_r2_train)
# With similar rmse and r2 score, it seems that overfitting is not a probelm.
```

```
For the linear regressor, the root mean square error for the training set is:
5304.935843152996
The r2 score for the testing set is: 0.8639902539603114
```

[147]:
```
# Next we will apply more advanced methods on these two datasets: Random Forest␣
 ↪and XGB
```