Task 1

1.0 Import Packages

In [28]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn
from sklearn.preprocessing import OneHotEncoder
from mlxtend.frequent_patterns import apriori, association_rules
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

import warnings
warnings.simplefilter("ignore")

path = "C:/Users/zklou/Documents/2022 summer/data mining/"
```

1.1 Data Importing and Cleaning

First, we import the data which were in three seperate CSV files, and merge them into one dataframe.

Then, since some of the numerical data are saved in string format of "74 bhp", we strip off the alphebetic parts and change data into correct data type.

In [29]:
```
#read car data from three csv files and save as dataframe
tata_data = pd.read_csv(path + '/Tata.csv')
ford_data = pd.read_csv(path + '/Ford.csv')
honda_data = pd.read_csv(path + '/Honda.csv')

#merge three datasets into one
car_data = pd.concat([tata_data,ford_data],axis=0)
car_data = pd.concat([car_data,honda_data],axis=0)
print("There are {} samples and {} features in dataset".format(*car_data.shape))

#drop NA values
car_data = car_data.dropna()

#preview
car_data.head()
```

```
There are 398 samples and 13 features in dataset
```

Out[29]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	e
0	Tata Bolt Quadrajet XE	2017	450000	50000	Diesel	Individual	Manual	First Owner	22.95 kmpl	
1	Tata Bolt Quadrajet XE	2017	450000	50000	Diesel	Individual	Manual	First Owner	22.95 kmpl	
2	Tata Bolt Quadrajet XE	2017	600000	27000	Diesel	Individual	Manual	First Owner	22.95 kmpl	
3	Tata Bolt Quadrajet XM	2017	600000	50000	Diesel	Individual	Manual	First Owner	22.95 kmpl	
4	Tata Bolt Revotron XE	2017	350000	35000	Petrol	Individual	Manual	First Owner	17.57 kmpl	

In [30]:
```
#extract numerical values only
car_data.mileage = car_data.mileage.str.extract('(\d+)')
car_data.engine = car_data.engine.str.extract('(\d+)')
car_data.max_power = car_data.max_power.str.extract('(\d+)')

#turn dtype from object into numerical
car_data.mileage = car_data.mileage.astype(float)
car_data.engine = car_data.engine.astype(float)
car_data.max_power = car_data.max_power.astype(float)

#show data cleaning results
print(car_data.dtypes)
```

```
name            object
year             int64
selling_price    int64
km_driven        int64
fuel            object
seller_type     object
transmission    object
owner           object
mileage        float64
engine         float64
max_power      float64
torque          object
seats          float64
dtype: object
```

1.2 Data Selection and Processing

1.2.1 Data selection

Based on the head of data displaied below, we notice that variables "name" and "torque" may contain many nominal instances, so it is not practical to include these two variables in the association analysis.

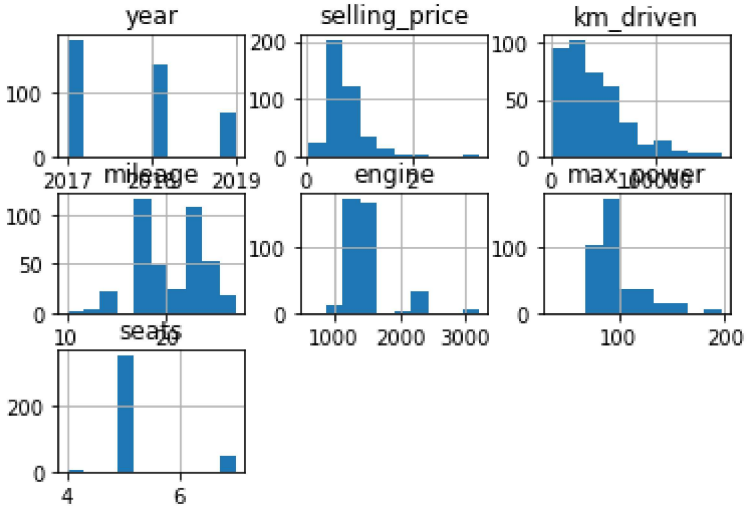
Then, we display the histograms of all rest 11 variables. Notice that the distribution of "seats" is extremely skewed, indicating that including it would be computationally expensive. Therefore, we also exclude this variable

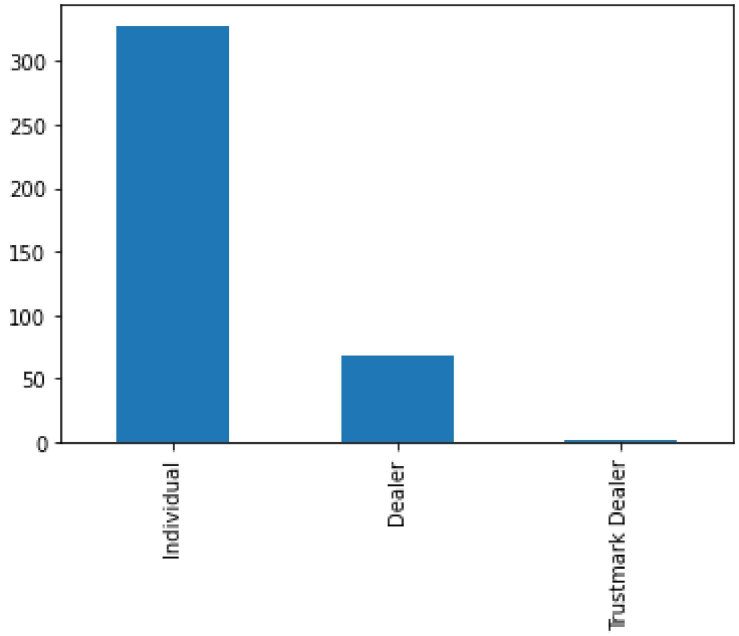
In [31]:
```
car_data.head()
```

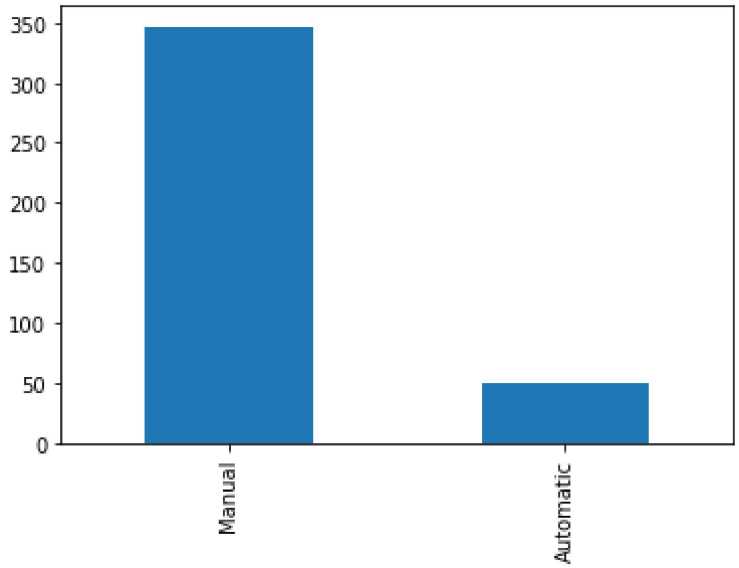
Out[31]:

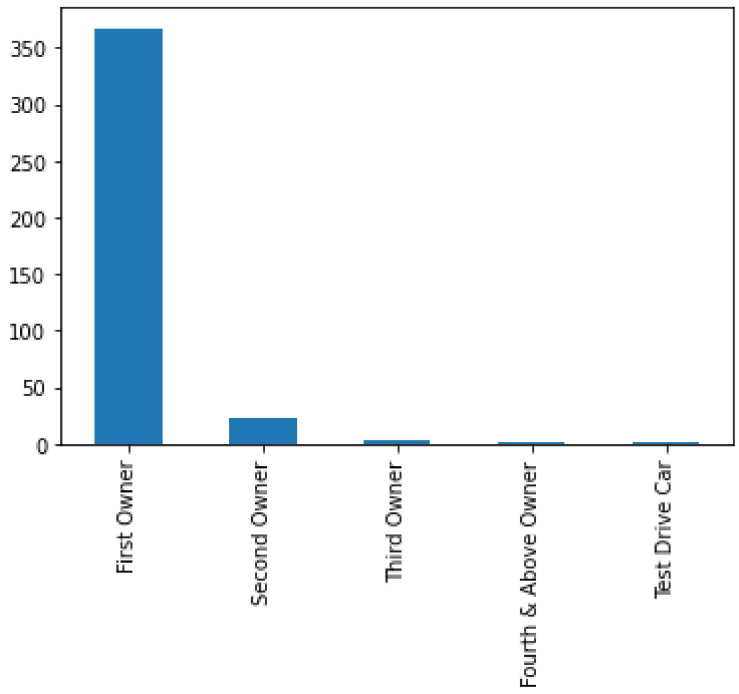
	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	e
0	Tata Bolt Quadrajet XE	2017	450000	50000	Diesel	Individual	Manual	First Owner	22.0	1
1	Tata Bolt Quadrajet XE	2017	450000	50000	Diesel	Individual	Manual	First Owner	22.0	1
2	Tata Bolt Quadrajet XE	2017	600000	27000	Diesel	Individual	Manual	First Owner	22.0	1
3	Tata Bolt Quadrajet XM	2017	600000	50000	Diesel	Individual	Manual	First Owner	22.0	1
4	Tata Bolt Revotron XE	2017	350000	35000	Petrol	Individual	Manual	First Owner	17.0	1

In [32]:
```
#histogram for numerical varibles
numerical_var_names = ['year', 'selling_price', 'km_driven', 'mileage', 'engine', 'm
categorical_var_names = ['seller_type', 'transmission', 'owner']

#plot histograms for the rest variables
car_data.hist(column = numerical_var_names)
plt.show()

car_data[categorical_var_names[0]].value_counts().plot(kind='bar')
plt.show()
car_data[categorical_var_names[1]].value_counts().plot(kind='bar')
plt.show()
car_data[categorical_var_names[2]].value_counts().plot(kind='bar')
plt.show()
```









In [33]:
```
#save to CSV format
all_var_names = list(car_data.columns)
all_var_names.remove('name')
```

```
all_var_names.remove('torque')
all_var_names.remove('owner')

car_select = car_data[all_var_names]
car_select.to_csv(path + " 100131001-100131002—T10ld.csv")
```

1.2.2 Data Discretization

Since there are some continuous variables such as "selling_price" or "mileage", we need to discreteze these variables. We first sort each condinuous data into one of five equal distance intervals, then use one-hot-coding to transform the initial dataset so that there are only 0s and 1s. We end up with a binary dataset with 38 columns.

In [34]:
```
car_clean = car_data
#cont. variable into bins
cont_var_names = ['selling_price', 'km_driven', 'mileage', 'engine', 'max_power']
for var in cont_var_names:
    car_clean[var] = pd.cut(car_data[var], 5)
car_clean.head()
```

Out[34]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	e
0	Tata Bolt Quadrajet XE	2017	(62816.0, 702800.0]	(33200.0, 65400.0]	Diesel	Individual	Manual	First Owner	(20.2, 23.6]	(1 1
1	Tata Bolt Quadrajet XE	2017	(62816.0, 702800.0]	(33200.0, 65400.0]	Diesel	Individual	Manual	First Owner	(20.2, 23.6]	(1 1
2	Tata Bolt Quadrajet XE	2017	(62816.0, 702800.0]	(839.0, 33200.0]	Diesel	Individual	Manual	First Owner	(20.2, 23.6]	(1 1
3	Tata Bolt Quadrajet XM	2017	(62816.0, 702800.0]	(33200.0, 65400.0]	Diesel	Individual	Manual	First Owner	(20.2, 23.6]	(1 1
4	Tata Bolt Revotron XE	2017	(62816.0, 702800.0]	(33200.0, 65400.0]	Petrol	Individual	Manual	First Owner	(16.8, 20.2]	(1 1

In [35]:
```
#one-hot-encoding data
transformer = make_column_transformer(
    (OneHotEncoder(), all_var_names),
    remainder='drop')
transformed = transformer.fit_transform(car_clean)

col_names = transformer.get_feature_names()
for i in range(len(all_var_names)):
        col_names = [sub.replace('onehotencoder__x'+ str(i),
                                 all_var_names[i]) for sub in col_names]
car_binary = pd.DataFrame(transformed.todense(), columns = col_names)
car_binary.head()
```

Out[35]:

	year_2017	year_2018	year_2019	selling_price_(62816.0, 702800.0]	selling_price_(702800.0, 1339600.0]	selling_price_(
0	1.0	0.0	0.0	1.0	0.0	
1	1.0	0.0	0.0	1.0	0.0	
2	1.0	0.0	0.0	1.0	0.0	
3	1.0	0.0	0.0	1.0	0.0	
4	1.0	0.0	0.0	1.0	0.0	

5 rows × 38 columns

In [36]:
```
#save to CSV format
car_binary.to_csv(path + "100131001-100131002—T1Disc.csv")
```

1.3 Data Analysis

1.3.1 Most Frequent Item-Sets

Below are the suppport and contend for the 10 most frequent itemsets. Note that since some itemsets are not displayed completely in the dataframe format, we printed complete itemsets below.

In [37]:
```
#calculate frequent item-sets
freq_items = apriori(car_binary, min_support = 0.05, use_colnames = True)

#delete all itemsets < 2
two_freq_items = freq_items.copy(deep = True)
del_list = []
for i in range(freq_items.shape[0]):
    if len(list(freq_items.iloc[i, 1])) <= 1:
        del_list.append(i)
two_freq_items.drop(del_list, axis = 0, inplace = True)

#sorting and displaying
two_freq_items.sort_values(by = 'support', ascending=False).head(10)
```

Out[37]:

	support	itemsets
207	0.818640	(seats_5.0, engine_(1138.8, 1653.6])
190	0.780856	(seats_5.0, transmission_Manual)
184	0.753149	(transmission_Manual, engine_(1138.8, 1653.6])
166	0.743073	(transmission_Manual, seller_type_Individual)
822	0.732997	(transmission_Manual, engine_(1138.8, 1653.6],...
176	0.727960	(seats_5.0, seller_type_Individual)
170	0.697733	(engine_(1138.8, 1653.6], seller_type_Individual)
800	0.685139	(seats_5.0, engine_(1138.8, 1653.6], seller_ty...
783	0.664987	(seats_5.0, transmission_Manual, seller_type_I...
205	0.644836	(engine_(1138.8, 1653.6], max_power_(69.0, 101...

In [38]:
```
#format for better visualization
for i in range(10):
    print('Itemset' + str(i) + ": " + str(two_freq_items.sort_values(by = 'support',
```

```
Itemset0: frozenset({'seats_5.0', 'engine_(1138.8, 1653.6]'})
Itemset1: frozenset({'seats_5.0', 'transmission_Manual'})
Itemset2: frozenset({'transmission_Manual', 'engine_(1138.8, 1653.6]'})
Itemset3: frozenset({'transmission_Manual', 'seller_type_Individual'})
Itemset4: frozenset({'transmission_Manual', 'engine_(1138.8, 1653.6]', 'seats_5.0'})
Itemset5: frozenset({'seats_5.0', 'seller_type_Individual'})
Itemset6: frozenset({'engine_(1138.8, 1653.6]', 'seller_type_Individual'})
Itemset7: frozenset({'seats_5.0', 'engine_(1138.8, 1653.6]', 'seller_type_Individua
l'})
Itemset8: frozenset({'seats_5.0', 'transmission_Manual', 'seller_type_Individual'})
Itemset9: frozenset({'engine_(1138.8, 1653.6]', 'max_power_(69.0, 101.0]'})
```

1.3.2 Association Rules

Below are the 5 most confident association rules and the 5 least confident rules. Note that since some rules are not displayed completely in the dataframe format, we printed complete itemsets below.

The confidence for the top 5 rules are all 1. The confidence for all bottom 5 rules are 0.057.

In [39]:
```
#calculate association rules
rules = association_rules(freq_items, metric ="lift", min_threshold = 1)

#sort and display rules based on confidence level
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
rules.head()
```

Out[39]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	
14978	(mileage_(16.8, 20.2], engine_(2168.4, 2683.2])	(max_power_(133.0, 165.0], seats_7.0)	0.055416	0.073048	0.055416	1.0	1
45624	(engine_(2168.4, 2683.2], mileage_(16.8, 20.2]...	(max_power_(133.0, 165.0], seats_7.0)	0.055416	0.073048	0.055416	1.0	1
45638	(mileage_(16.8, 20.2], engine_(2168.4, 2683.2])	(max_power_(133.0, 165.0], seats_7.0, fuel_Die...	0.055416	0.073048	0.055416	1.0	1
47078	(seller_type_Individual, mileage_(16.8, 20.2],...	(max_power_(133.0, 165.0], seats_7.0)	0.050378	0.073048	0.050378	1.0	1
97995	(fuel_Diesel, seller_type_Individual, mileage_...	(max_power_(133.0, 165.0], seats_7.0)	0.050378	0.073048	0.050378	1.0	1

In [40]:
```
rules.tail()
```

Out[40]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	li
53983	(seats_5.0)	(transmission_Manual, year_2017, max_power_(69...	0.879093	0.055416	0.050378	0.057307	1.0341
54024	(seats_5.0)	(engine_(1138.8, 1653.6], year_2017, max_power...	0.879093	0.055416	0.050378	0.057307	1.0341
54050	(seats_5.0)	(transmission_Manual, engine_(1138.8, 1653.6],...	0.879093	0.055416	0.050378	0.057307	1.0341
76403	(seats_5.0)	(transmission_Manual, seller_type_Individual, ...	0.879093	0.055416	0.050378	0.057307	1.0341
108164	(seats_5.0)	(transmission_Manual, engine_(1138.8, 1653.6],...	0.879093	0.055416	0.050378	0.057307	1.0341

In [41]:
```
#format for better visualization
print("==========================Five Most Confident Associative Rules:==========
for i in range(5):
    print("Rule" + str(i+1) + ":")
    print(str(rules.iloc[i, 0]) + "->\n" + str(rules.iloc[i, 1]) + "\n")

print("==========================Five Least Confident Associative Rules==========
for i in range(1, 6):
    print("Rule" + str(5-i) + ":")
    print(str(rules.iloc[-i, 0]) + "->\n" + str(rules.iloc[-i, 1]) + "\n")
```

```
==========================Five Most Confident Associative Rules:==================
======
Rule1:
frozenset({'mileage_(16.8, 20.2]', 'engine_(2168.4, 2683.2]'})->
frozenset({'max_power_(133.0, 165.0]', 'seats_7.0'})

Rule2:
frozenset({'engine_(2168.4, 2683.2]', 'mileage_(16.8, 20.2]', 'fuel_Diesel'})->
frozenset({'max_power_(133.0, 165.0]', 'seats_7.0'})

Rule3:
frozenset({'mileage_(16.8, 20.2]', 'engine_(2168.4, 2683.2]'})->
frozenset({'max_power_(133.0, 165.0]', 'seats_7.0', 'fuel_Diesel'})

Rule4:
frozenset({'seller_type_Individual', 'mileage_(16.8, 20.2]', 'engine_(2168.4, 2683.
2]'})->
frozenset({'max_power_(133.0, 165.0]', 'seats_7.0'})

Rule5:
frozenset({'fuel_Diesel', 'seller_type_Individual', 'mileage_(16.8, 20.2]', 'engine_
(2168.4, 2683.2]'})->
frozenset({'max_power_(133.0, 165.0]', 'seats_7.0'})

==========================Five Least Confident Associative Rules==================
======
Rule4:
frozenset({'seats_5.0'})->
frozenset({'transmission_Manual', 'engine_(1138.8, 1653.6]', 'selling_price_(702800.
0, 1339600.0]', 'max_power_(69.0, 101.0]', 'year_2017', 'fuel_Diesel'})

Rule3:
frozenset({'seats_5.0'})->
frozenset({'transmission_Manual', 'seller_type_Individual', 'km_driven_(839.0, 3320
0.0]', 'year_2019', 'mileage_(16.8, 20.2]'})

Rule2:
frozenset({'seats_5.0'})->
frozenset({'transmission_Manual', 'engine_(1138.8, 1653.6]', 'selling_price_(702800.
0, 1339600.0]', 'max_power_(69.0, 101.0]', 'year_2017'})

Rule1:
frozenset({'seats_5.0'})->
frozenset({'engine_(1138.8, 1653.6]', 'year_2017', 'max_power_(69.0, 101.0]', 'selli
ng_price_(702800.0, 1339600.0]', 'fuel_Diesel'})

Rule0:
frozenset({'seats_5.0'})->
frozenset({'transmission_Manual', 'year_2017', 'max_power_(69.0, 101.0]', 'selling_p
rice_(702800.0, 1339600.0]', 'fuel_Diesel'})
```

1.3.3 Measures of Interest

Here we choose Interst, Cosine, Piatesky_Shapiro, Jaccard, All-confidence as our 5 different measures of interest. Results are displayed below in the same order as the section above.

In [42]:
```
#helper function that takes in a rule and calculate 5 MOIs
def get_moi(ante, conse):
    N = car_binary.shape[0]
    #f1+
    res_1p = np.ones(N)
```

```
    for condition in list(ante):
        res_1p = np.logical_and(res_1p, car_binary[condition])
    f_1p = np.sum(res_1p)
    #f+1
    res_p1 = np.ones(car_binary.shape[0])
    for condition in list(conse):
        res_p1 = np.logical_and(res_p1, car_binary[condition])
    f_p1 = np.sum(res_p1)
    #f11
    res_11 = np.logical_and(res_1p, res_p1)
    f_11 = np.sum(res_11)

    moi = [0, 0, 0, 0, 0]
    #interest
    moi[0] = N*f_11/f_1p/f_p1
    #cosine
    moi[1] = f_11/np.sqrt(f_1p * f_p1)
    #Piatetsky_Shapiro
    moi[2] = f_11/N - (f_1p * f_p1/(N^2))
    #Jaccard
    moi[3] = f_11/(f_1p + f_p1 - f_11)
    #all-confidence
    moi[4] = min(f_11/f_1p, f_11/f_p1)

    return moi
```

In [43]:
```
moi_df = pd.DataFrame(columns = ('Interst', 'Cosine', 'Piatesky_Shapiro', 'Jaccard',
for i in range(5):
    moi = get_moi(rules.iloc[i, 0], rules.iloc[i, 1])
    moi_df.loc[len(moi_df.index)] = moi

print(moi_df)
```

	Interst	Cosine	Piatesky_Shapiro	Jaccard	All-confidence
0	13.689655	0.870988	-1.543582	0.758621	0.758621
1	13.689655	0.870988	-1.543582	0.758621	0.758621
2	13.689655	0.870988	-1.543582	0.758621	0.758621
3	13.689655	0.830455	-1.403256	0.689655	0.689655
4	13.689655	0.830455	-1.403256	0.689655	0.689655