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.

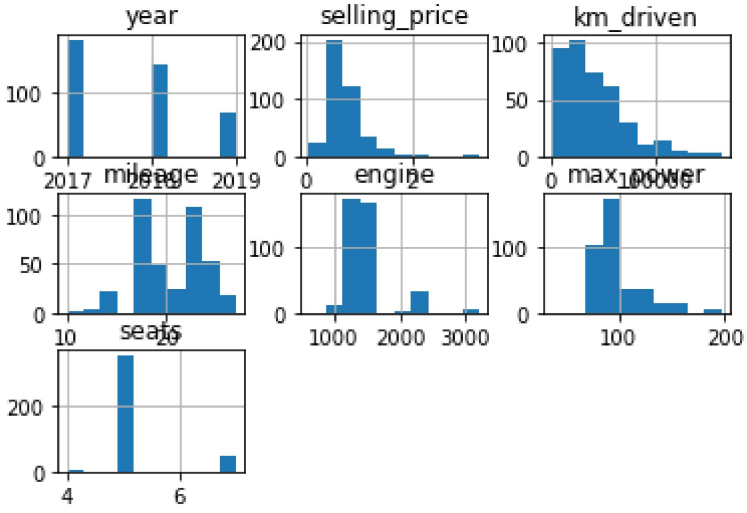
Then, we display the histograms of all rest 11 variables. Notice that the distribution of "seats" is extremely skewed, indicating that including it would be computationally expensive. Therefore, we also exclude this variable

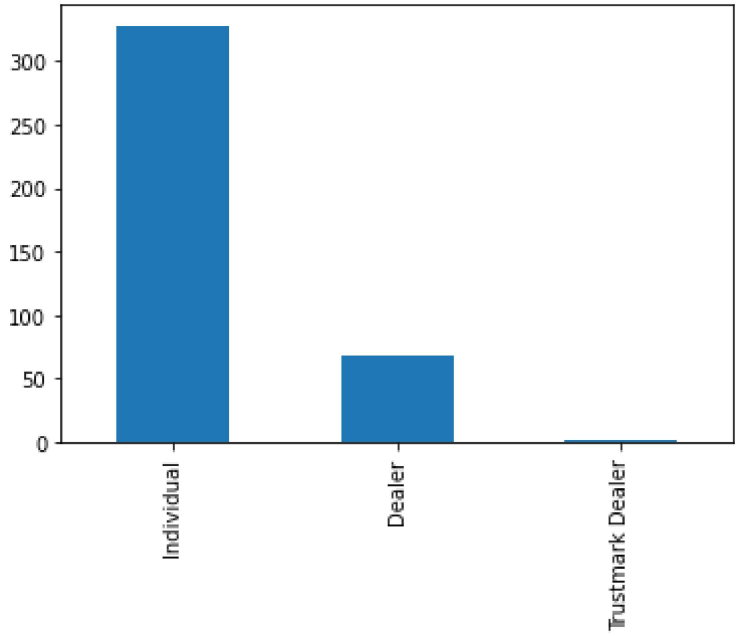
In [31]:
```
car_data.head()
```

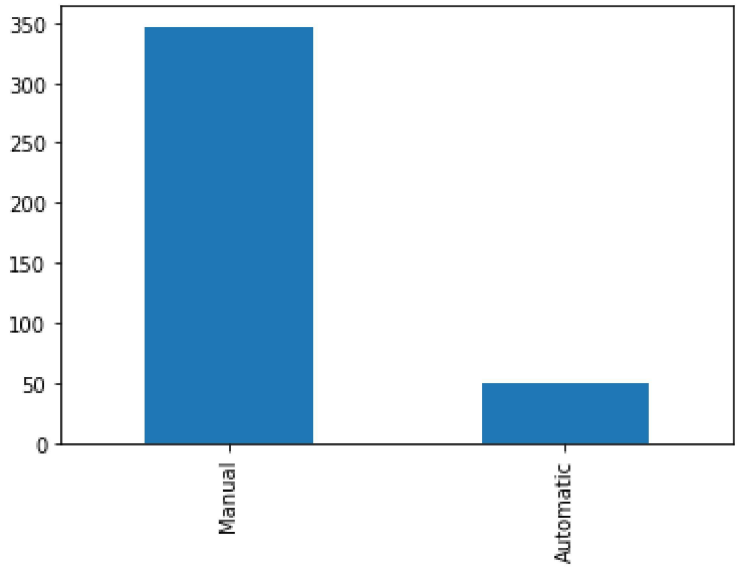
Out[31]:

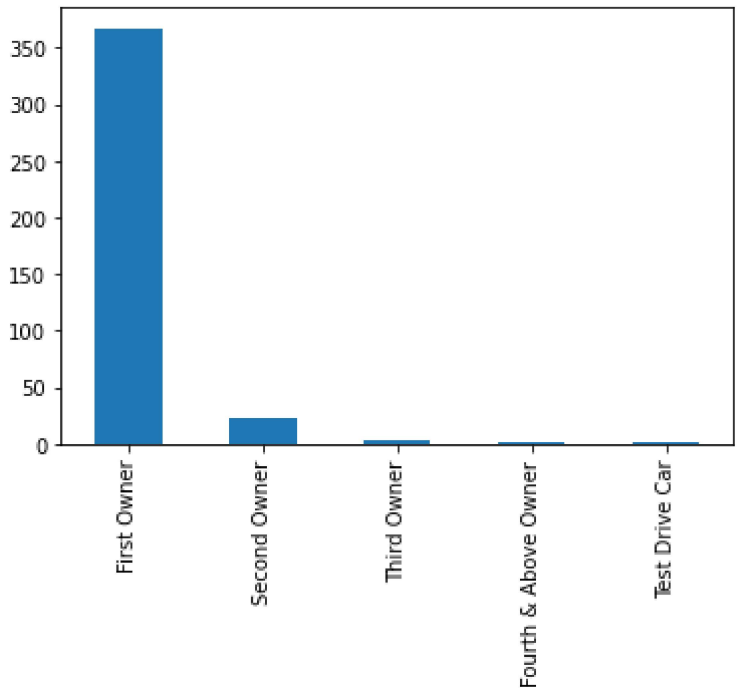
	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	e
0	Tata Bolt Quadrajet XE	2017	450000	50000	Diesel	Individual	Manual	First Owner	22.0	1
1	Tata Bolt Quadrajet XE	2017	450000	50000	Diesel	Individual	Manual	First Owner	22.0	1
2	Tata Bolt Quadrajet XE	2017	600000	27000	Diesel	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

Task 2: Clustering Analysis

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
```

In [2]:
```
data = pd.read_csv("Honda.csv")
data.to_csv('100131001-100131002_T2Org.csv', index=False)
data.head()
```

Out[2]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Honda Amaze E i-DTEC	2017	500000	70000	Diesel	Individual	Manual	First Owner	25.8 kmpl	1498 CC	98.6 bhp	200Nm@ 1750rpm	5
1	Honda Amaze E i-DTEC	2017	515000	60000	Diesel	Individual	Manual	Second Owner	25.8 kmpl	1498 CC	98.6 bhp	200Nm@ 1750rpm	5
2	Honda Amaze E i-VTEC	2017	475000	57000	Petrol	Individual	Manual	First Owner	17.8 kmpl	1198 CC	86.7 bhp	109Nm@ 4500rpm	5
3	Honda Amaze E Option i-DTEC	2017	550000	120000	Diesel	Individual	Manual	First Owner	25.8 kmpl	1498 CC	98.6 bhp	200Nm@ 1750rpm	5
4	Honda Amaze i-VTEC Privilege Edition	2017	490000	80000	Petrol	Individual	Manual	First Owner	17.8 kmpl	1198 CC	86.7 bhp	109Nm@ 4500rpm	5

In [3]:
```
edit_data = data.drop(['name', 'fuel','seller_type', 'transmission', 'owner', 'mileage', 'engine', 'max_power', 'torque'],axis=1)
edit_data.to_csv('100131001-100131002_T2Mod.csv', index=False)
```

In [4]:
```
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(edit_data)
```

Perform any necessary preprocessing steps. Add explanation to your report.

Ans: I dropped the unused features and only keep the features I plan to use, such as year, selling_price, km_driven, seats.

In [5]:
```
from sklearn.cluster import KMeans
kmeans_3 = KMeans(n_clusters=3)
kmeans_3.fit(X)
kmeans_4 = KMeans(n_clusters=4)
kmeans_4.fit(X)
kmeans_5 = KMeans(n_clusters=5)
kmeans_5.fit(X)
print("SSE for k=3: {}\nSSE for k=4: {}\nSSE for k=5: {}".
      format(round(kmeans_3.inertia_, 4), round(kmeans_4.inertia_,4), round(kmeans_5.inertia_, 4)))
print("The least SSE is k=5 with SSE={}".format(round(kmeans_5.inertia_, 4)))
```

```
SSE for k=3: 189.2816
SSE for k=4: 143.4199
SSE for k=5: 112.2917
The least SSE is k=5 with SSE=112.2917
```

In [6]:
```
edit_data['class'] = kmeans_5.labels_
edit_data.to_csv('100131001-100131002_T2Class.csv', index=False)
edit_data
```

Out[6]:

	year	selling_price	km_driven	seats	class
0	2017	500000	70000	5	2
1	2017	515000	60000	5	2
2	2017	475000	57000	5	2
3	2017	550000	120000	5	2
4	2017	490000	80000	5	2
...	...	...	...	...	...
90	2019	1300000	20000	5	3
91	2019	2000000	24857	5	4
92	2019	840000	1500	5	3
93	2019	750000	3100	5	3
94	2019	840000	1500	5	3

95 rows × 5 columns

In [ ]:
```

```

In [1]:
```
import numpy as np
from sklearn.model_selection import KFold
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

from sklearn import datasets, metrics, model_selection, svm


data = pd.read_csv("C:/Users/76380/Desktop/Honda.csv")
```

In [ ]:
```
import numpy as np
from sklearn.model_selection import KFold
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, auc
data = pd.read_csv("C:/Users/76380/Desktop/Honda.csv")


X = data.iloc[:,:4]
Y = data.iloc[:,4:5]
X = np.array(X)
Y = np.array(Y)
data = np.array(data)

clf = GaussianNB()
clf.fit(X, Y)
GaussianNB()
clf.predict(X)
#print(clf.predict([[-0.8, -1,50000,7000]]))
```

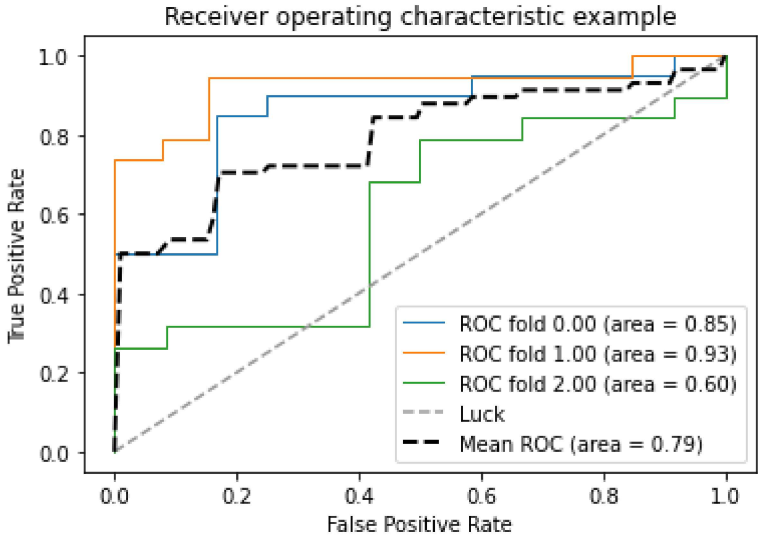
In [ ]:
```
import numpy as np
from sklearn.model_selection import KFold
import pandas as pd

from sklearn.metrics import roc_curve, auc
from sklearn.dummy import DummyClassifier
data = pd.read_csv("C:/Users/76380/Desktop/Honda.csv")


X = data.iloc[:,:4]
Y = data.iloc[:,4:5]
X = np.array(X)
Y = np.array(Y)
data = np.array(data)

dummy_clf = DummyClassifier(strategy="stratified")
dummy_clf.fit(X, Y)
DummyClassifier(strategy='stratified')
dummy_clf.predict(X)
#print(clf.predict([[-0.8, -1,50000,7000]]))
```

In [24]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc
```

```
from sklearn.model_selection import StratifiedKFold

data = pd.read_csv("C:/Users/76380/Desktop/Honda.csv")
#data = np.array(data)
X = data.iloc[:,:4]
Y = data.iloc[:,4:5]
X = np.array(X)
y = []



a = np.array(Y)
y =[]
for m in range(0,95):
    for i in a[m]:
        y.append(i)
y = np.array(y)


X, y = X[y != 2], y[y != 2]
n_samples, n_features = X.shape

random_state = np.random.RandomState(0)
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

cv = StratifiedKFold(n_splits=3)

classifier = GaussianNB()

mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)
cnt = 0
for i, (train, test) in enumerate(cv.split(X,y)):
    cnt +=1
    probas_ = classifier.fit(X[train], y[train]).predict_proba(X[test])
    y_pred = classifier.fit(X[train], y[train]).predict(X[test])
    print("Number of mislabeled points out of a total %d points : %d" % (X[test].sha
    print("Accuracy: %d percent" %(X[test].shape[0]/y[test] != y_pred).sum())
    fpr, tpr, thresholds = roc_curve(y[test], probas_[:, 1])

    mean_tpr += np.interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=1, label='ROC fold {0:.2f} (area = {1:.2f})'.format(i, roc

plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')

mean_tpr /= cnt
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)

plt.plot(mean_fpr, mean_tpr, 'k--',label='Mean ROC (area = {0:.2f})'.format(mean_auc

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
Number of mislabeled points out of a total 32 points : 10
Accuracy: 32 percent
Number of mislabeled points out of a total 32 points : 4
Accuracy: 32 percent
Number of mislabeled points out of a total 31 points : 15
Accuracy: 31 percent
```

```
C:\Users\76380\AppData\Local\Temp\ipykernel_2276\1398343822.py:42: RuntimeWarning: d
ivide by zero encountered in divide
  print("Accuracy: %d percent" %(X[test].shape[0]/y[test] != y_pred).sum())
C:\Users\76380\AppData\Local\Temp\ipykernel_2276\1398343822.py:42: RuntimeWarning: d
ivide by zero encountered in divide
  print("Accuracy: %d percent" %(X[test].shape[0]/y[test] != y_pred).sum())
C:\Users\76380\AppData\Local\Temp\ipykernel_2276\1398343822.py:42: RuntimeWarning: d
ivide by zero encountered in divide
  print("Accuracy: %d percent" %(X[test].shape[0]/y[test] != y_pred).sum())
```



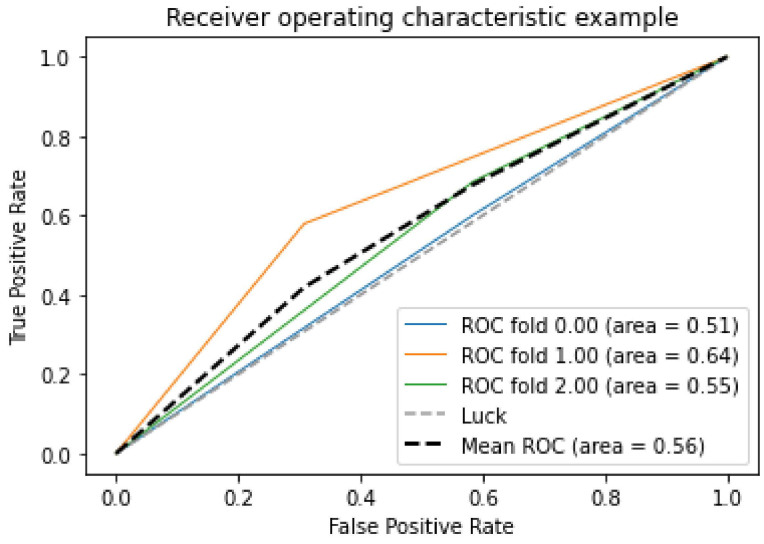
In [23]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import StratifiedKFold

data = pd.read_csv("C:/Users/76380/Desktop/Honda.csv")
#data = np.array(data)
X = data.iloc[:,:4]
Y = data.iloc[:,4:5]
X = np.array(X)
y = []


a = np.array(Y)
y =[]
for m in range(0,95):
    for i in a[m]:
        y.append(i)
y = np.array(y)



X, y = X[y != 2], y[y != 2]
n_samples, n_features = X.shape

random_state = np.random.RandomState(0)
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]
```

```
cv = StratifiedKFold(n_splits=3)

classifier = DummyClassifier(strategy="stratified")

mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)
cnt = 0
for i, (train, test) in enumerate(cv.split(X,y)):
    cnt +=1
    probas_ = classifier.fit(X[train], y[train]).predict_proba(X[test])
    y_pred = classifier.fit(X[train], y[train]).predict(X[test])
    print("Number of mislabeled points out of a total %d points : %d" % (X[test].sha
    print("Accuracy: %d percent" %(X[test].shape[0]/y[test] != y_pred).sum())
    fpr, tpr, thresholds = roc_curve(y[test], probas_[:, 1])

    mean_tpr += np.interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=1, label='ROC fold {0:.2f} (area = {1:.2f})'.format(i, roc

plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')

mean_tpr /= cnt
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)

plt.plot(mean_fpr, mean_tpr, 'k--',label='Mean ROC (area = {0:.2f})'.format(mean_auc

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
Number of mislabeled points out of a total 32 points : 16
Accuracy: 32 percent
Number of mislabeled points out of a total 32 points : 14
Accuracy: 32 percent
Number of mislabeled points out of a total 31 points : 15
Accuracy: 31 percent
```

```
C:\Users\76380\AppData\Local\Temp\ipykernel_2276\3442226410.py:43: RuntimeWarning: d
ivide by zero encountered in divide
  print("Accuracy: %d percent" %(X[test].shape[0]/y[test] != y_pred).sum())
C:\Users\76380\AppData\Local\Temp\ipykernel_2276\3442226410.py:43: RuntimeWarning: d
ivide by zero encountered in divide
  print("Accuracy: %d percent" %(X[test].shape[0]/y[test] != y_pred).sum())
C:\Users\76380\AppData\Local\Temp\ipykernel_2276\3442226410.py:43: RuntimeWarning: d
ivide by zero encountered in divide
  print("Accuracy: %d percent" %(X[test].shape[0]/y[test] != y_pred).sum())
```


Receiver operating characteristic example
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC fold 0.00 (area = 0.51)
ROC fold 1.00 (area = 0.64)
ROC fold 2.00 (area = 0.55)
Luck
Mean ROC (area = 0.56)

In [ ]:

In [ ]:

Answer to Q2: There is an association rule with confidence 1 for {'mileage_(16.8, 20.2]', 'max_power_(69.0, 101.0]', 'km_driven_(839.0, 33200.0]', 'transmission_Manual', 'year_2018'} -> 'fuel_Petrol', 'seats_5.0', 'selling_price_(62816.0, 702800.0]'}, indicating that cars fullfilling the antecedents above are likely to be sold for a price between 62816 and 702800. This tells us that the mileage of a car may affect its price.

Answer to Q5: Based on the result of itemsets, cars with mileage within the 16.8kpmi to 20.2 kpmi range sold by individual seller is favored by dealers, with a support of 36%.