```
#!/usr/bin/env python
# coding: utf-8

# # Question 1
#
# ## Import Data

# In[1]:


# Code to import "Alternative Fuel Vehicles US.csv"

import pandas as pd
data_1 = pd.read_csv('Alternative Fuel Vehicles US.csv')
data = pd.read_csv('Alternative Fuel Vehicles US.csv')


# ## Part 1 - a)

# In[2]:


# Filter dataframe for European and Japanese makes

Japanese = ['Hino','Hyundia','Toyota','Honda','Lexus']
European= ['Audi','BMW','Jaguar','Mercedes-Benze','Land Rover']
fuelmodel = data[['Manufacturer','Fuel','Conventional Fuel Economy Combined']]
fuelmode_final_European =
fuelmodel[(fuelmodel['Manufacturer'].isin(European))&(fuelmodel['Fuel']=='Hybrid
Electric')].dropna()
fuelmode_final_Japanese =
fuelmodel[(fuelmodel['Manufacturer'].isin(Japanese))&(fuelmodel['Fuel']=='Hybrid
Electric')].dropna()


# In[3]:


# Plot histograms side-by-side


import matplotlib.pyplot as plt
plt.figure()
plt.subplot(2,2,1)
plt.hist(fuelmode_final_European['Conventional Fuel Economy Combined'])
plt.title('Fuel economy of European')
plt.xlabel('Fuel economy')
plt.ylabel('Number')


plt.subplot(2,2,2)
plt.hist(fuelmode_final_Japanese['Conventional Fuel Economy Combined'])
plt.title('Fuel economy of Japanese')
plt.xlabel('Fuel economy')
plt.ylabel('Number')


# It is clear from the histogram that European cars are more fuel efficient than
Japanese cars
```

```
#

#

# ## Part 1 - b)

# In[4]:


# Code for plotting histogram
plt.figure()
plt.hist(fuelmode_final_Japanese['Conventional Fuel Economy Combined'])
plt.title('Fuel economy of Japanese Hybird Electric cars')
plt.xlabel('Fuel economy')
plt.ylabel('Number')
plt.show()


# In[5]:


# Code for getting and saving "TopTenElectric.csv"
All_Electric_Range = data[(data['Fuel'] == 'Electric')]
All_Electric_Range = All_Electric_Range.sort_values(by = ['All-Electric
Range'],ascending = False)
num= round(All_Electric_Range.shape[0]/10)
All_Electric_Range_final = All_Electric_Range[0:num]
All_Electric_Range_final
All_Electric_Range_final.to_csv("TopTenElectric.csv")


# ## Part 2 - a)

# Pick out the 'Economy Combined' entries for those vehicles that have a non-empty
'conventional fuel'.
# After setting column A, what are D, K, L, M and Q, fill in the NANs in these
columns according to the mode or average.
# Then convert the feature columns except the target column to numerical variables
through one-hot encoding

# In[6]:


# Code for filtering dataset and other pre-processing applied

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
Economy_Combined = data[~data['Conventional Fuel Economy Combined'].isnull()]
A = 'Category'
D = 'Manufacturer'
K = 'Conventional Fuel Economy Combined'
L = 'Alternative Fuel Economy City'
M = 'Alternative Fuel Economy Highway'
Q = 'Drivetrain'
E = 'Fuel'
Euro_pean = ['Audi','Land Rover','BMW','Mercedes-
Benz','Jaguar','Volvo','Mini','Porsche','Bentley Motors','Ferrari']
```

```
American
=['Chevrolet','GMC','Cadillac','Ford','Jeep','Ram','Lincoln','Chrysler','Karma','Mi
tsubishi','Polestar Automotive USA']
Asian_manufacturers = ['Hyundai','Toyota','Honda','Kia','Lexus','Acura','Subaru']
cols =[A,D,K,L,M,Q,E]
dx = Economy_Combined[cols]
data = dx.copy()
data["Alternative Fuel Economy City"].fillna(dx["Alternative Fuel Economy
City"].median(skipna=True), inplace=True)
data["Alternative Fuel Economy Highway"].fillna(dx["Alternative Fuel Economy
Highway"].median(skipna=True), inplace=True)
data["Drivetrain"].fillna(dx['Drivetrain'].value_counts().idxmax(), inplace=True)
data['Fuel'] = data['Fuel'].astype('category')
data['Fuel'] = data['Fuel'].cat.codes
data_clean = data
data_clean = pd.get_dummies(data_clean[cols])


# ## Part 2 - b)

# In[7]:


# Get average accuracy of logistic regression across 20 random train/test splits
# You can manually change test_frac and present only the best value here

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
new_cols = ['Conventional Fuel Economy Combined','Alternative Fuel Economy
City','Alternative Fuel Economy Highway','Category_Passenger Van/Shuttle
Bus','Category_Pickup','Category_SUV','Category_Sedan/Wagon','Category_Van','Manufa
cturer_Acura','Manufacturer_Porsche','Manufacturer_Ram','Manufacturer_Subaru','Manu
facturer_Toyota','Manufacturer_Volvo','Drivetrain_4WD','Drivetrain_AWD','Drivetrain
_FWD','Drivetrain_Part-Time 4WD','Drivetrain_RWD']
j = 0
a_list = []
list_rate = [0.1,0.15,0.2,0.25,0.3]
while j <=4:
    a = 0
    i = 0
    while i <= 19:
        X = data_clean[new_cols]
        y = data_clean['Fuel']
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=list_rate[j])
        logreg = LogisticRegression()
        logreg.fit(X_train, y_train)
        y_pred = logreg.predict(X_test)
        a += accuracy_score(y_test, y_pred)
        i+=1
    a = a/20
    a_list.append(a)
    j+=1

print('Train/Test split results:')
print("The best accuracy : %2.3f" % max(a_list))
```

```
# ## Part 2 - c)

# In[8]:


# Average accuracy of MLP across 20 random train/test splits
# You can manually change number of neurons in hidden layer and present best result
from sklearn.neural_network import MLPClassifier
i = 0
acc_final = 0
while i<= 19:
    df_train,df_test = train_test_split(data_clean, test_size=0.3)
    X_train = df_train[new_cols].to_numpy()
    y_train = df_train['Fuel'].to_numpy()

    X_test = df_test[new_cols].to_numpy()
    y_test = df_test['Fuel'].to_numpy()

    clf = MLPClassifier(random_state = 1,hidden_layer_sizes =
(100, )).fit(X_train,y_train)

    y_pred = clf.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    acc_final = acc_final + acc
    i+=1

acc_final = acc_final/20

print("Average accuracy %2.3f" % acc_final)



# # Question 2
#
# ## Import Data

# In[9]:


# Code to import "heart.csv"

import pandas as pd
data = pd.read_csv('heart.csv')


# ## Part 1

# "ST_slope" column should be label encoded because the result has direction, the
rest are one hot encoded

# In[10]:


# Code for any variable transformations/pre-processing here
data['ST_Slope'] = data['ST_Slope'].astype('category')
data['ST_Slope'] = data['ST_Slope'].cat.codes
```

```
data_clean = pd.get_dummies(data)


# ## Part 2

# In[11]:


# AUC and accuracies of Gaussian Naive Bayes across 10 random train/test splits
# Make sure to get a table of all AUC and accuracies

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
i = 0
acc = []
auc = []
while i <= 9:

    df_train, df_test = train_test_split(data_clean, test_size=0.25)
    from sklearn.metrics import accuracy_score


    d=
['Age','RestingBP','Cholesterol','FastingBS','MaxHR','Oldpeak','ST_Slope','Sex_F','
Sex_M','ChestPainType_ASY','ChestPainType_ATA','ChestPainType_NAP','ChestPainType_T
A','RestingECG_LVH','RestingECG_Normal','RestingECG_ST','ExerciseAngina_N','Exercis
eAngina_Y']
    X_train = df_train[d].to_numpy()
    y_train = df_train["HeartDisease"].to_numpy()


    X_test = df_test[d].to_numpy()
    y_test = df_test["HeartDisease"].to_numpy()


    gnb_pend_model = GaussianNB()
    gnb_pend_model.fit(X_train, y_train)


    gnb_pend_pred = gnb_pend_model.predict(X_test)
    gnb_pend_acc = accuracy_score(y_test, gnb_pend_pred)
    gau_roc_auc = roc_auc_score(y_test, gnb_pend_model.predict(X_test))
    acc.append(gnb_pend_acc)
    auc.append(gau_roc_auc)
    i+=1

from pandas import DataFrame
data = {'accuracy':acc,
       'auc':auc}
df = DataFrame(data)
print(df)


# ## Part 3

#  When we don't know some of these variables, we can indirectly infer
probabilities by predicting others
```

```
# In[12]:


# Implement your suggested method here.
cols=
['Age','RestingBP','FastingBS','MaxHR','Oldpeak','Sex_F','Sex_M','ChestPainType_ASY
','ChestPainType_NAP','ChestPainType_TA','ExerciseAngina_N','ExerciseAngina_Y','ST_
Slope']
X_train = df_train[cols].to_numpy()
y_train = df_train["HeartDisease"].to_numpy()


X_test = df_test[cols].to_numpy()
y_test = df_test["HeartDisease"].to_numpy()

gnb_pend_model = GaussianNB()
gnb_pend_model.fit(X_train, y_train)


gnb_pend_pred = gnb_pend_model.predict(X_test)
gnb_pend_acc = accuracy_score(y_test, gnb_pend_pred)
print("The accuracy using num_of_pendown* features is: ", gnb_pend_acc)
gau_roc_auc = roc_auc_score(y_test, gnb_pend_model.predict(X_test))
print('The auc:',gau_roc_auc)


# ## Part 4

# In[13]:


# AUC and accuracies of KNN and WNN using different values of K
# Make sure to get a table of all AUC and accuracies

cols=
['Age','RestingBP','FastingBS','MaxHR','Oldpeak','Sex_F','Sex_M','ChestPainType_ASY
','ChestPainType_NAP','ChestPainType_TA','ExerciseAngina_N','ExerciseAngina_Y','ST_
Slope']
df_train, df_test = train_test_split(data_clean, test_size=0.25)
X_train = df_train[cols].to_numpy()
y_train = df_train["HeartDisease"].to_numpy()

X_test = df_test[cols].to_numpy()
y_test = df_test["HeartDisease"].to_numpy()

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

acc_knn = []
auc_knn = []
acc_wnn = []
auc_wnn = []
i= 1
while i <= 30:
    clf_1 = KNeighborsClassifier(n_neighbors=i, weights = 'uniform')
    clf_1 = clf_1.fit(X_train,y_train)
    clf_1 = clf_1.predict(X_test)
    clf_acc_1 = accuracy_score(y_test, clf_1)
    clf_auc_1 = roc_auc_score(y_test, clf_1)
    acc_knn.append(clf_acc_1)
    auc_knn.append(clf_auc_1)
    clf_2 = KNeighborsClassifier(n_neighbors=i,weights = 'distance')
    clf_2 = clf_2.fit(X_train,y_train)
    clf_2 = clf_2.predict(X_test)
    clf_acc_2 = accuracy_score(y_test, clf_2)
    clf_auc_2 = roc_auc_score(y_test, clf_2)
    acc_wnn.append(clf_acc_2)
    auc_wnn.append(clf_auc_2)
    i+=1

from pandas import DataFrame
data = {'k':
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30],
       'Accuracy (KNN)':acc_knn,
       'Accuracy (WNN)':acc_wnn,
       'AUC (KNN)':auc_knn,
       'AUC (WNN)':auc_wnn}
df = DataFrame(data)
print(df)


# In[ ]:




# In[ ]:




# In[ ]:
```