ELG 20225: Applied Machine Learning

Assignment 2

Group 18

Part 1: Calculations:

1. **Suppose we have the following training data including 15 training samples.**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Color(x1)	R	R	R	R	R	G	G	G	G	G	Y	Y	Y	Y	Y
Gender(x2)	M	M	F	F	M	M	M	M	F	F	F	F	M	M	F
Price(x3)	H	L	L	H	M	M	H	L	L	M	L	H	M	L	M
TARGET(y)	N	N	Y	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	N

Target (yes) → 9

P (yes) = 9 / 15

Color(x_1): in (yes)	Gender(x_2): in (yes)	Price(x_3): in (yes)
➢ P (R | yes) → 2 / 9 ➢ P (G | yes) → 3 / 9 ➢ P(Yellow | yes) → 4 / 9	➢ P (F | yes) = 6 / 9 ➢ P (M | yes) = 3 / 9	➢ P (H | yes) = 2 / 9 ➢ P (L | yes) = 5 / 9 ➢ P (M | yes) = 2 / 9

Target (no) → 6

P (no) = 6 / 15

Color(x_1): in (no)	Gender(x_2): in (no)	Price(x_3): in (no)
➢ P (R | no) → 3 / 6 ➢ P (G | no) → 2 / 6 ➢ P (Yellow | no) → 1 / 6	➢ P (F | no) = 1 / 6 ➢ P (M | no) = 5 / 6	➢ P (H | no) = 2 / 6 ➢ P (L | no) = 1 / 6 ➢ P (M | no) = 3 / 6

$$P\ (\text{yes} \mid G, F, H) = \frac{P\ (G\ |yes) * P(F\ |yes) * P(H\ |yes) * P(yes)}{P(G,F,H)}$$

$$= \frac{\frac{3}{9} * \frac{6}{9} * \frac{2}{9} * \frac{9}{15}}{P(G,F,H)}$$

$P(G, F, H) =$ P (G, F, H | yes) * P (yes) + P (G, F, H | no) * P (no)

= P (G | yes) * P (F | yes) * P (H | yes) * P (yes) + P (G | no) * P (F | no) * P (H | no) * P (no)

$$= \frac{3}{9} * \frac{6}{9} * \frac{2}{9} * \frac{9}{15} + \frac{2}{6} * \frac{1}{6} * \frac{2}{6} * \frac{6}{15} = \frac{4}{135} + \frac{1}{135} = \frac{5}{135} = \frac{1}{27}$$

$$P\ (\text{yes} \mid G, F, H) = \frac{\frac{3}{9} * \frac{6}{9} * \frac{2}{9} * \frac{9}{15}}{\frac{5}{162}} = \frac{\frac{4}{135}}{\frac{1}{27}} = \frac{4}{135} * \frac{27}{1} = \frac{4}{5}$$

$$P\ (\text{no} \mid G, F, H) = \frac{P\ (G\ |no) * P(F\ |no) * P(H\ |no) * P(no)}{P(G,F,H)}$$

$$P\ (\text{no} \mid G, F, H) = \frac{\frac{2}{6} * \frac{1}{6} * \frac{2}{6} * \frac{6}{15}}{\frac{5}{162}} = \frac{\frac{1}{135}}{\frac{1}{27}} = \frac{1}{135} * \frac{27}{1} = \frac{1}{5}$$

The prediction when Color = G, Gender = F, Price=H is 0.80 %

2. Calculate the expected risk of three actions, and determine the rejection area of P (Class1| x)

Target	Class1	Class2
choose class2	5	2
choose class1	0	5
Reject	4	4

We suppose that a1 is (choose class1) and a2 is (choose class2).

R (a1 | X) = 0 * P (Calss1 | X) +5 * P (Class2 | X)

R (a1 | X) = 0 P (Calss1 | X) + 5 (1 – P (Class1 | X)))

R (a1 | X) = $\boxed{\textbf{- 5P (Calss1 | X) + 5}}$ → **equation 1**

R (a2 | X) = 5 * P (Calss1 | X) + 2 * P (Class2 | X)

R (a2 | X) = 5 * P (Calss1 | X) + 2 * (1 – P (Class1 | X))

= $\boxed{\textbf{2 + 3 P (Class1 | X)}}$ → **equation 2**

R (a3 | X) = 4 * P (Calss1 | X) + 4 * P (Class2 | X)

= 4 * P (Calss1 | X) + 4 * (1 – P (Class1 | X))

= 4 * P (Calss1 | X) - 4 * P (Class1 | X) +4 = $\boxed{\textbf{4}}$ → **equation 3**

We choose a1 if: R (a1 | X) < 4	We choose a2 if: R (a2 | X) < 4
From equation 1 - 5P (Calss1 | X) + 5 < 4 - 5 P (Calss1 | X) < -1 P (Calss1 | X) > $\frac{1}{5}$, or equivalently if P(Class2 | X) < $\frac{4}{5}$	From equation 2 2 + 3 P (Class1 | X) < 4 3 P (Class1 | X) < 2 P (Class1 | X) < $\frac{2}{3}$, or equivalently if P(Class2 | X) > $\frac{1}{3}$

There is no rejection area, because there is no intersection between

P (Calss1 | X) > 1/5 and P (Class1 | X) < 2/3.

Part 2: Programming

1-Naïve Bayes Classifier

-loading the wine dataset from sklearn library

```
#load wine dataset from sklean
data = load_wine()
data
```

```
{'DESCR': '.. _wine_dataset:\n\nWine recognition dataset\n------------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 178 (50 in each of three classes)\n    :Number of Attributes: 13 n
 'data': array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
         1.065e+03],
        [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
         1.050e+03],
        [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
         1.185e+03],
        ...,
        [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
         8.350e+02],
        [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
         8.400e+02],
        [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
         5.600e+02]]),
 'feature_names': ['alcohol',
  'malic_acid',
  'ash',
  'alcalinity_of_ash',
  'magnesium',
  'total_phenols',
  'flavanoids',
  'nonflavanoid_phenols',
  'proanthocyanins',
  'color_intensity',
  'hue',
  'od280/od315_of_diluted_wines',
  'proline'],
 'frame': None,
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2]),
 'target_names': array(['class_0', 'class_1', 'class_2'], dtype='<U7')}
```

-There are 13 features and target consist of 3 classes

```
[175] #extract X from dataset convert X to dataframe
      X = pd.DataFrame(data=data.data, columns=data.feature_names)
      X.head(10)
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0
5	14.20	1.76	2.45	15.2	112.0	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450.0
6	14.39	1.87	2.45	14.6	96.0	2.50	2.52	0.30	1.98	5.25	1.02	3.58	1290.0
7	14.06	2.15	2.61	17.6	121.0	2.60	2.51	0.31	1.25	5.05	1.06	3.58	1295.0
8	14.83	1.64	2.17	14.0	97.0	2.80	2.98	0.29	1.98	5.20	1.08	2.85	1045.0
9	13.86	1.35	2.27	16.0	98.0	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045.0

```
[176] #extract y from dataset
      y=data.target
      y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2])
```

-we checked the nan values and we found there isn't any nan values

```
# checking the number of missing data
X.isnull().sum()
```

```
alcohol                         0
malic_acid                      0
ash                             0
alcalinity_of_ash               0
magnesium                       0
total_phenols                   0
flavanoids                      0
nonflavanoid_phenols            0
proanthocyanins                 0
color_intensity                 0
hue                             0
od280/od315_of_diluted_wines    0
proline                         0
dtype: int64
```

-splitting it into test and train data with 80% for training and 20% for testing

```
#split data into training and testing with 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42)
```

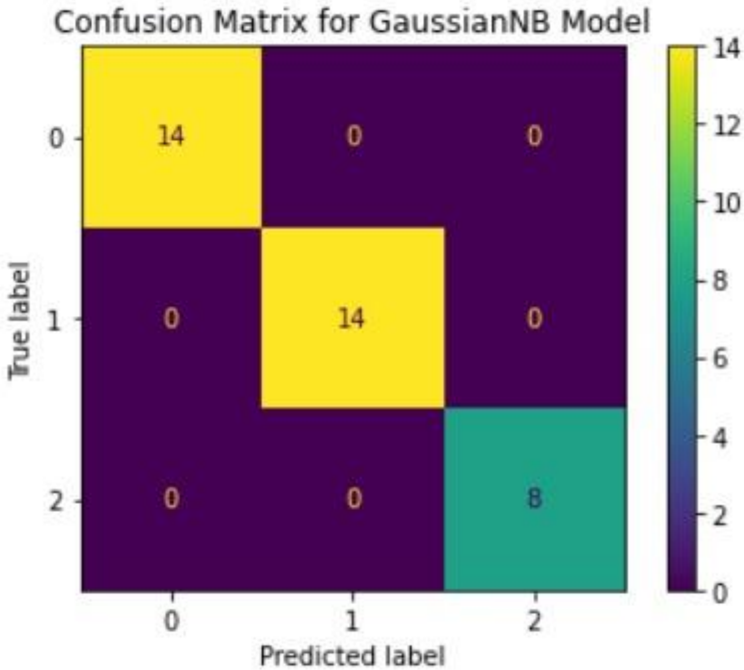
-we applied feature scaling to our data to make all features on the same scale then Gaussian Naïve Bayes Classifier for all features

```
[180] # Feature Scaling
      sc = StandardScaler()
      X_train= sc.fit_transform(X_train)
      X_test = sc.transform(X_test)
```

```
[181] # Train Guassian Naive Model and predit
      nb= GaussianNB()
      NBmodel=nb.fit(X_train,y_train)
      ypred= NBmodel.predict(X_test)
```

- The confusion matrix and classification report for the test set for all features, because the small data the model has an over fitting.

```
[[14  0  0]
 [ 0 14  0]
 [ 0  0  8]]
```




```
Classification Report for all Features
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        14
     class_1       1.00      1.00      1.00        14
     class_2       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```

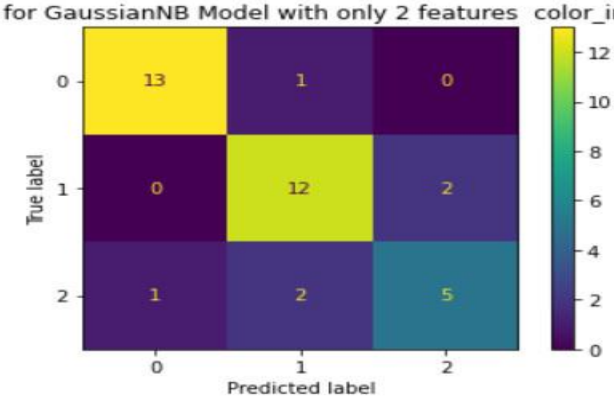
-selecting 2 features to plot the data and its boundaries, here we selected the highest correlation 2 features with the target (color_intensity, proline)

```
best2Features = SelectKBest(chi2, k=2).fit_transform(X, y)
#best 2 features are color_intensity, proline
```

-then we trained new model with only these 2 features and plot our decision boundaries and checked the confusion matrix and classification report to check the result and compare our decision boundaries and our accuracy

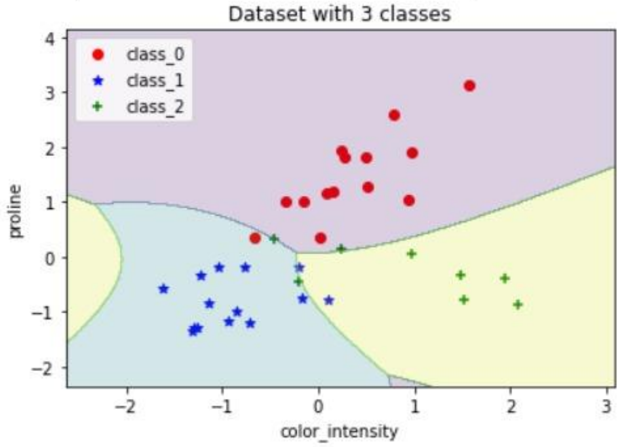
```
[[13  1  0]
 [ 0 12  2]
 [ 1  2  5]]
```

Confusion Matrix for GaussianNB Model with only 2 features color_intensity, proline




```
Classification Report for GaussianNB Model with only 2 features  color_intensity, proline
              precision    recall  f1-score   support

     class_0       0.93      0.93      0.93        14
     class_1       0.80      0.86      0.83        14
     class_2       0.71      0.62      0.67         8

    accuracy                           0.83        36
   macro avg       0.81      0.80      0.81        36
weighted avg       0.83      0.83      0.83        36

Testing for classes 0 , 1 , 2 with only 2 features color_intensity, proline
```

Dataset with 3 classes



part 2: programming - problem 2

a) load the dataset , shuffle it , split it into training , validation

load the dataset

In [17]:
```
dataset2 =pd.read_csv('car_evaluation.csv',header=None)
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
dataset2.columns = col_names
dataset2.head(20)
```

Out[17]:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc
5	vhigh	vhigh	2	2	med	high	unacc
6	vhigh	vhigh	2	2	big	low	unacc
7	vhigh	vhigh	2	2	big	med	unacc
8	vhigh	vhigh	2	2	big	high	unacc
9	vhigh	vhigh	2	4	small	low	unacc
10	vhigh	vhigh	2	4	small	med	unacc
11	vhigh	vhigh	2	4	small	high	unacc
12	vhigh	vhigh	2	4	med	low	unacc
13	vhigh	vhigh	2	4	med	med	unacc
14	vhigh	vhigh	2	4	med	high	unacc
15	vhigh	vhigh	2	4	big	low	unacc

In [18]:
```
dataset2.describe()
```

Out[18]:

	buying	maint	doors	persons	lug_boot	safety	class
count	1728	1728	1728	1728	1728	1728	1728
unique	4	4	4	3	3	3	4
top	vhigh	vhigh	2	2	small	low	unacc
freq	432	432	432	576	576	576	1210

get X -> features , y-> label

In [19]:
```
X = dataset2.drop(['class'], axis=1)
y = dataset2['class']
```

In [20]:
```
X
```

Out[20]:

	buying	maint	doors	persons	lug_boot	safety
0	vhigh	vhigh	2	2	small	low
1	vhigh	vhigh	2	2	small	med
2	vhigh	vhigh	2	2	small	high
3	vhigh	vhigh	2	2	med	low
4	vhigh	vhigh	2	2	med	med
...	...	...	...	...	...	...
1723	low	low	5more	more	med	med
1724	low	low	5more	more	med	high
1725	low	low	5more	more	big	low
1726	low	low	5more	more	big	med
1727	low	low	5more	more	big	high

1728 rows × 6 columns

print the unique value in each coulmn

In [22]:
```
print(f"the unique values in buying    column :{pd.Series.unique(X.buying)}")
print(f"the unique values in maint     column :{pd.Series.unique(X.maint)}")
print(f"the unique values in doors     column :{pd.Series.unique(X.doors)}")
print(f"the unique values in persons   column :{pd.Series.unique(X.persons)}")
print(f"the unique values in lug_boot  column :{pd.Series.unique(X.lug_boot)}")
print(f"the unique values in safety    column :{pd.Series.unique(X.safety)}")
print("========================================================================")
print(f"the unique values in the target label :{pd.Series.unique(y)}")
```

```
the unique values in buying    column :['vhigh' 'high' 'med' 'low']
the unique values in maint     column :['vhigh' 'high' 'med' 'low']
the unique values in doors     column :['2' '3' '4' '5more']
the unique values in persons   column :['2' '4' 'more']
the unique values in lug_boot  column :['small' 'med' 'big']
the unique values in safety    column :['low' 'med' 'high']
========================================================================
the unique values in the target label :['unacc' 'acc' 'vgood' 'good']
```

data preprocessing

In [23]:
```python
# checking the number of missing data
dataset2.isnull().sum()
```

Out[23]:
```
buying      0
maint       0
doors       0
persons     0
lug_boot    0
safety      0
class       0
dtype: int64
```

we used train_test_split and set shuffle = True to shuffle the data, and then we split the dataset to training, validation and testing.

2. a) shuffle the data and get training, validation, testing set

In [24]:
```python
_train, y_test_validation = train_test_split( X, y, test_size=0.4209, random_state=42,shuffle=True)
 y_validation = train_test_split( X_test_validation,y_test_validation, test_size=0.412, random_state=42,shuffle=True)
```

In [25]:
```python
print("length of X_train     :",len(X_train))
print("length of y_train     :",len(y_train))
print("length of X_validation:",len(X_validation))
print("length of y_validation:",len(y_validation))
print("length of X_test      :",len(X_test))
print("length of y_test      :",len(y_test))
```

```
length of X_train     : 1000
length of y_train     : 1000
length of X_validation: 300
length of y_validation: 300
length of X_test      : 428
length of y_test      : 428
```

Parameters:

arrays : *sequence of indexables with same length / shape[0]

Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : *float or int, default=None*

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : *float or int, default=None*

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

random_state : *int, RandomState instance or None, default=None*

Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.

shuffle : *bool, default=True*

Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

b) transform the string values into numeric

```
before encoding
the unique values in buying    column :['high' 'vhigh' 'low' 'med']
the unique values in maint     column :['low' 'med' 'vhigh' 'high']
the unique values in doors     column :['3' '5more' '4' '2']
the unique values in persons   column :['more' '2' '4']
the unique values in lug_boot  column :['big' 'med' 'small']
the unique values in safety    column :['high' 'low' 'med']
==========================================================================
before encoding:
buying          : object
maint           : object
doors           : object
persons         : object
lug_boot        : object
safety          : object
```

In [29]:
```python
# encode the features using ordinalEncoder
import category_encoders as ce

encoder_1= ce.OrdinalEncoder(cols=['buying'],return_df=True,mapping=[{'col':'buying','mapping':{'vhigh':4 ,'high':3 ,
X_train.buying= encoder_1.fit_transform(X_train.buying)

encoder_2= ce.OrdinalEncoder(cols=['maint'],return_df=True,mapping=[{'col':'maint','mapping':{'vhigh':4 ,'high':3 ,'m
X_train.maint= encoder_2.fit_transform(X_train.maint)

encoder_3= ce.OrdinalEncoder(cols=['doors'],return_df=True,mapping=[{'col':'doors','mapping':{'2':2,'3':3,'4':4,'5mor
X_train.doors= encoder_3.fit_transform(X_train.doors)

encoder_4= ce.OrdinalEncoder(cols=['persons'],return_df=True,mapping=[{'col':'persons','mapping':{'2':2,'4':4,'more':
X_train.persons= encoder_4.fit_transform(X_train.persons)

encoder_5= ce.OrdinalEncoder(cols=['lug_boot'],return_df=True,mapping=[{'col':'lug_boot','mapping':{'small':1 ,'med':
X_train.lug_boot= encoder_5.fit_transform(X_train.lug_boot)

encoder_6= ce.OrdinalEncoder(cols=['safety'],return_df=True,mapping=[{'col':'safety','mapping':{'low':1,'med':2,'high
X_train.safety = encoder_6.fit_transform(X_train.safety)
```

In [30]:
```python
# encode the target label using label encoder
labelencoder_y = LabelEncoder()
labelencoder_y.fit(y_train)
y_train  = labelencoder_y.transform(y_train)
```

Encoding the validation data

In [32]:
```
X_validation.buying= encoder_1.transform(X_validation.buying)
X_validation.maint= encoder_2.transform(X_validation.maint)
X_validation.doors= encoder_3.transform(X_validation.doors)
X_validation.persons= encoder_4.transform(X_validation.persons)
X_validation.lug_boot= encoder_5.transform(X_validation.lug_boot)
X_validation.safety = encoder_6.transform(X_validation.safety)
```

In [33]:
```
# encode the target label using label encoder
y_validation  = labelencoder_y.transform(y_validation)
```

Encoding the testing data

In [34]:
```
X_test.buying= encoder_1.transform(X_test.buying)
X_test.maint= encoder_2.transform(X_test.maint)
X_test.doors= encoder_3.transform(X_test.doors)
X_test.persons= encoder_4.transform(X_test.persons)
X_test.lug_boot= encoder_5.transform(X_test.lug_boot)
X_test.safety = encoder_6.transform(X_test.safety)
```

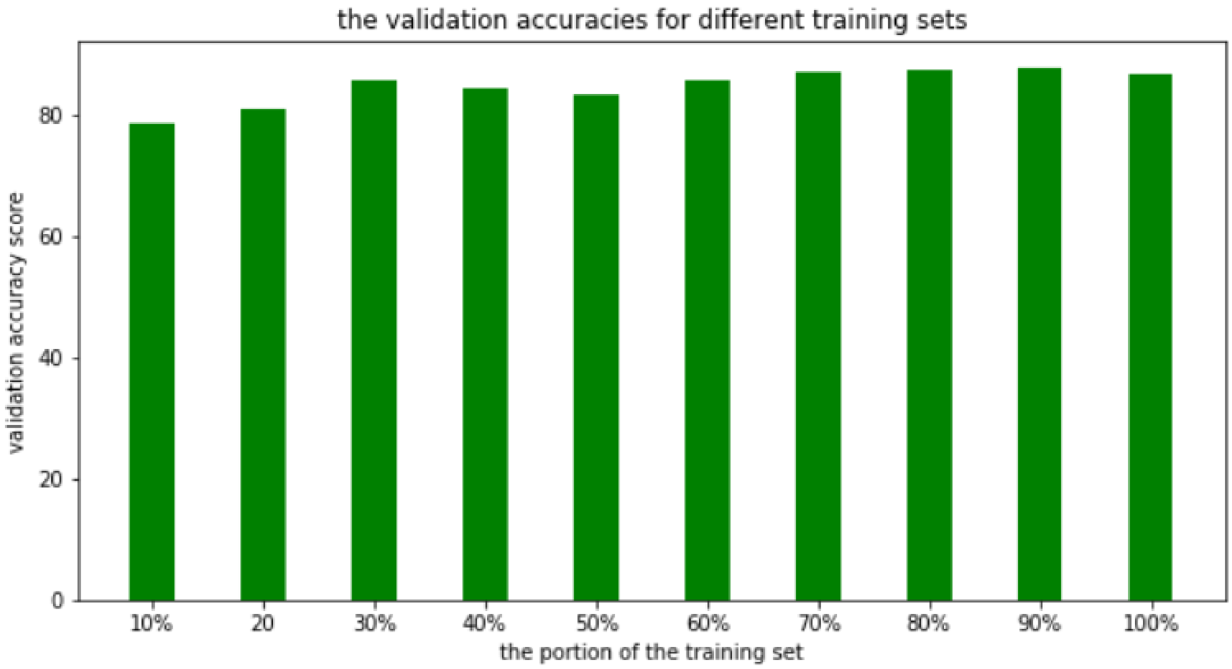
In [35]:
```
# encode the target label using label encoder
y_test  = labelencoder_y.transform(y_test)
```

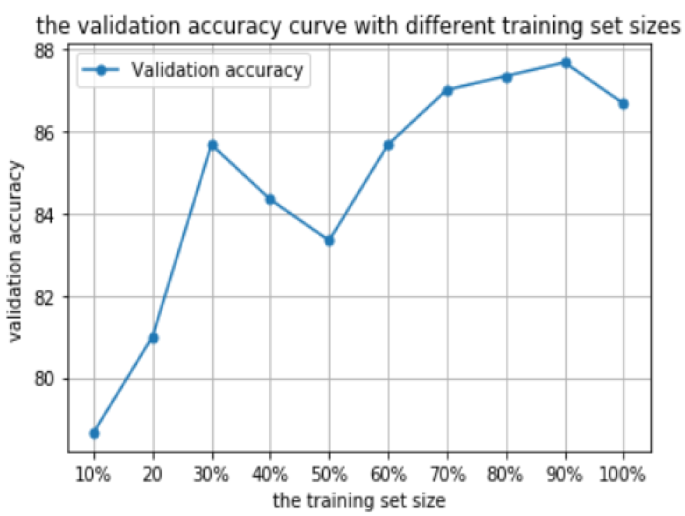
```
after encoding
the unique values in buying    column :[3 4 1 2]
the unique values in maint     column :[1 2 4 3]
the unique values in doors     column :[3 5 4 2]
the unique values in persons   column :[6 2 4]
the unique values in lug_boot  column :[3 2 1]
the unique values in safety    column :[3 1 2]
========================================================================
after encoding:
buying        : int64
maint         : int64
doors         : int64
persons       : int64
lug_boot      : int64
safety        : int64
```

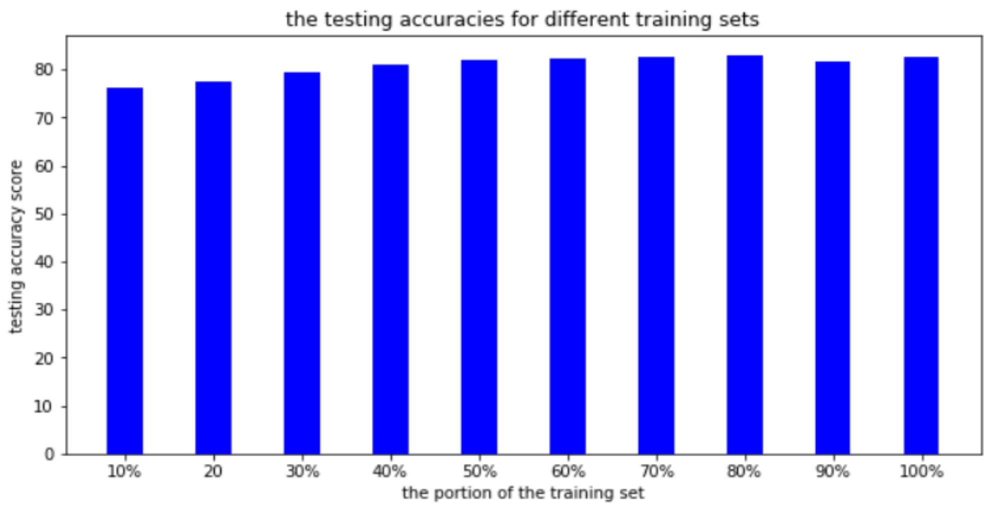
c) the impact of training sample size with a fixed number of K =2 on the accuracy and time.

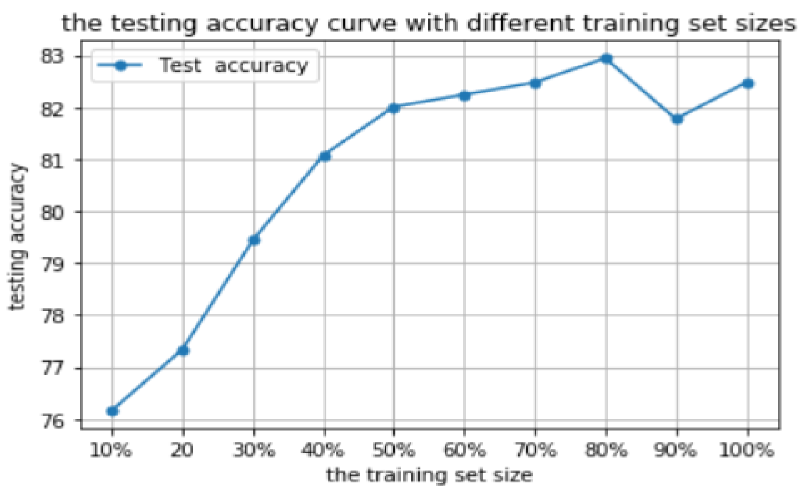
```
percentage of training data:10% , k value : 2
===============================================
training time                           : 0.05447033996460959
prediction time on validation set       : 0.01021883599969442
prediction time on testing set          : 0.012278604001039639
validation accuracy : 78.66666666666666
testing accuracy    : 76.16822429906543
===============================================
percentage of training data:20% , k value : 2
===============================================
training time                           : 0.06995873998675961
prediction time on validation set       : 0.009477341001911554
prediction time on testing set          : 0.013156602002709405
validation accuracy : 81.0
testing accuracy    : 77.33644859813083
===============================================
percentage of training data:30% , k value : 2
===============================================
training time                           : 0.07020816010481212
prediction time on validation set       : 0.008140326997818192
prediction time on testing set          : 0.011374065001291456
validation accuracy : 85.66666666666667
testing accuracy    : 79.43925233644859
===============================================
percentage of training data:40% , k value : 2
===============================================
training time                           : 0.14922869981091935
prediction time on validation set       : 0.01166650399682112
prediction time on testing set          : 0.012788764001015807
validation accuracy : 84.33333333333334
testing accuracy    : 81.07476635514018
===============================================
percentage of training data:50% , k value : 2
===============================================
```

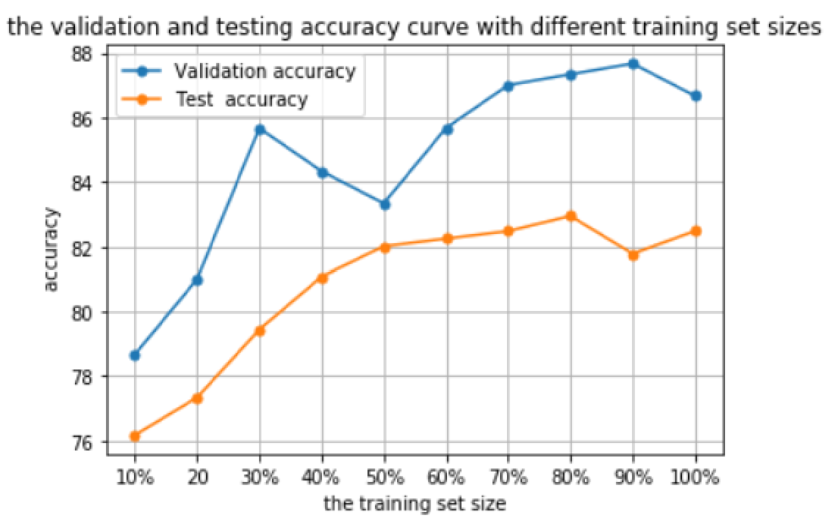
```
training time                          : 0.13971503998618573
prediction time on validation set      : 0.010927513001661282
prediction time on testing set         : 0.011951849999604747
validation accuracy : 83.33333333333334
testing accuracy    : 82.00934579439252
===============================================
percentage of training data:60% , k value : 2
===============================================
training time                          : 0.14794194015848916
prediction time on validation set      : 0.008541500999854179
prediction time on testing set         : 0.011785701997723663
validation accuracy : 85.66666666666667
testing accuracy    : 82.2429906542056
===============================================
percentage of training data:70% , k value : 2
===============================================
training time                          : 0.14783345999603625
prediction time on validation set      : 0.008406997003476135
prediction time on testing set         : 0.011677746999339433
validation accuracy : 87.0
testing accuracy    : 82.4766355140187
===============================================
percentage of training data:80% , k value : 2
===============================================
training time                          : 0.13934982016508002
prediction time on validation set      : 0.008442840000498109
prediction time on testing set         : 0.01161993200003053
validation accuracy : 87.33333333333333
testing accuracy    : 82.94392523364486
===============================================
percentage of training data:90% , k value : 2
===============================================
training time                          : 0.1408605599863222
prediction time on validation set      : 0.009931134998623747
prediction time on testing set         : 0.01340147200244246
validation accuracy : 87.66666666666667
testing accuracy    : 81.77570093457945
===============================================
percentage of training data:100% , k value : 2
===============================================
training time                          : 0.17531628007418476
prediction time on validation set      : 0.009217418999469373
prediction time on testing set         : 0.01421279200076242
validation accuracy : 86.66666666666667
testing accuracy    : 82.4766355140187
===============================================
```


the validation accuracies for different training sets
validation accuracy score
0
20
40
60
80
10%
20
30%
40%
50%
60%
70%
80%
90%
100%
the portion of the training set


the validation accuracy curve with different training set sizes
Validation accuracy
validation accuracy
80
82
84
86
88
10%
20
30%
40%
50%
60%
70%
80%
90%
100%
the training set size


the testing accuracies for different training sets
testing accuracy score
0
10
20
30
40
50
60
70
80
10%
20
30%
40%
50%
60%
70%
80%
90%
100%
the portion of the training set

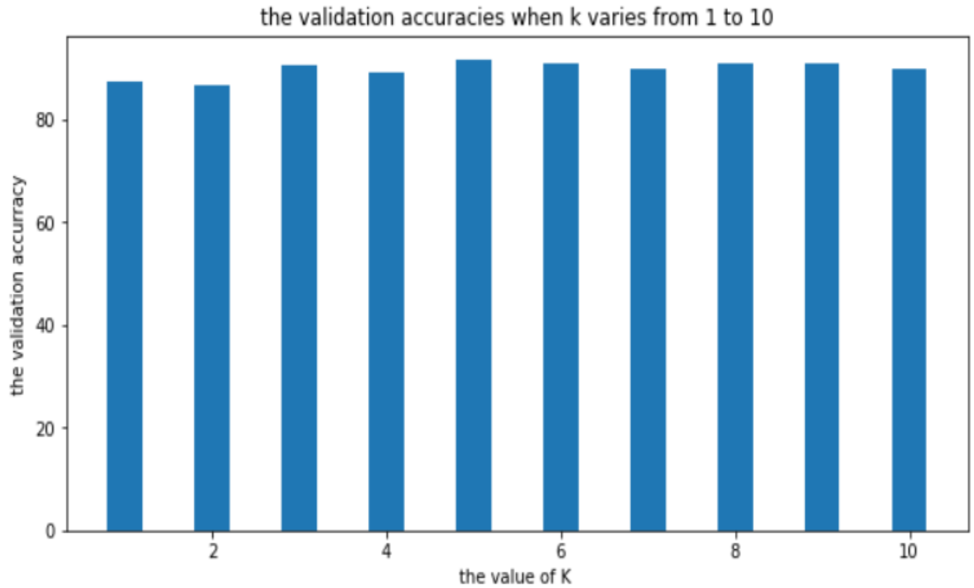

the testing accuracy curve with different training set sizes
Test accuracy
testing accuracy
76
77
78
79
80
81
82
83
10%
20
30%
40%
50%
60%
70%
80%
90%
100%
the training set size

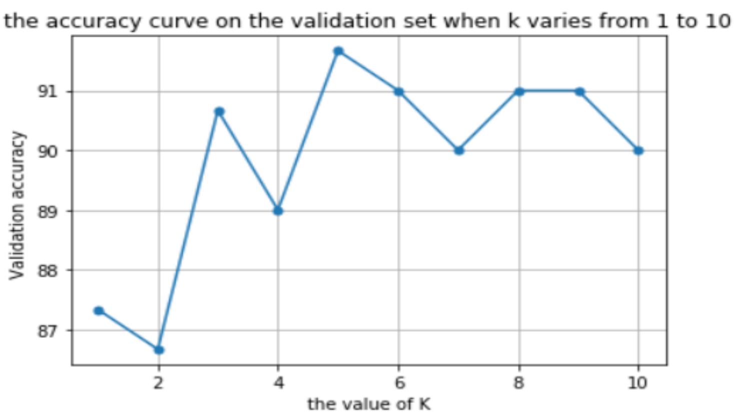

the validation and testing accuracy curve with different training set sizes
Validation accuracy
Test accuracy
accuracy
76
78
80
82
84
86
88
10%
20
30%
40%
50%
60%
70%
80%
90%
100%
the training set size

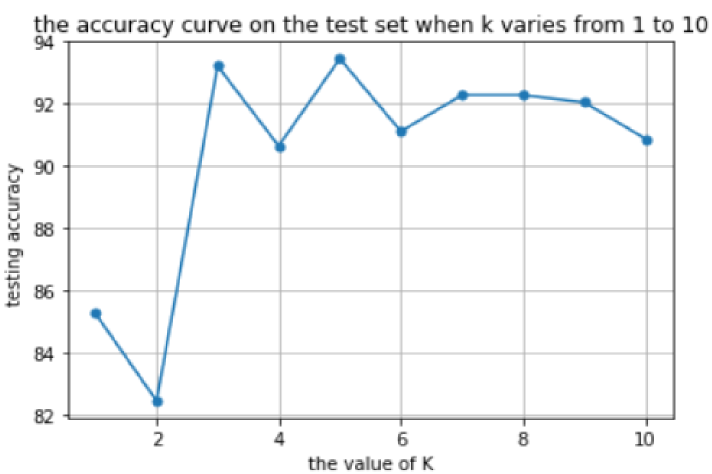
d) the impact of changing the K with fixed number of training sample size = 100%

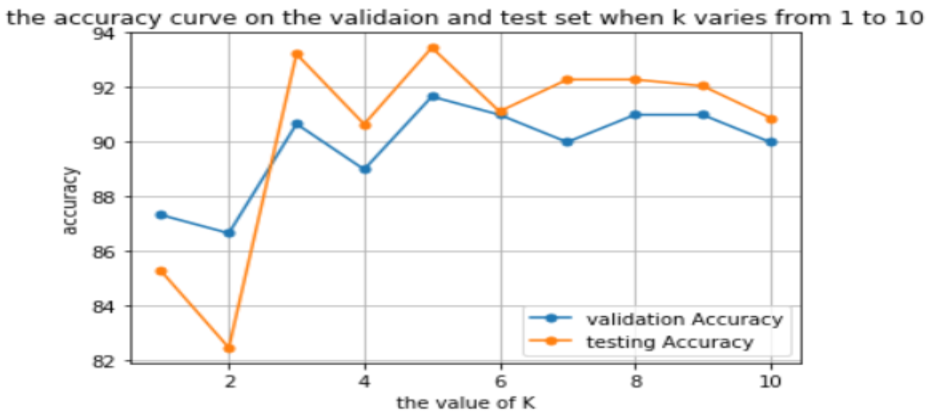
```
percentage of training data:100% , k value : 1
===============================================
training time                         : 0.13052670015895274
prediction time on validation set     : 0.017956153002160136
prediction time on testing set        : 0.0159276869999303
validation accuracy : 87.33333333333333
testing accuracy    : 85.2803738317757
===============================================
percentage of training data:100% , k value : 2
===============================================
training time                         : 0.1830972599418601
prediction time on validation set     : 0.008920855998439947
prediction time on testing set        : 0.011975247001828393
validation accuracy : 86.66666666666667
testing accuracy    : 82.4766355140187
===============================================
percentage of training data:100% , k value : 3
===============================================
training time                         : 0.1508907000243198
prediction time on validation set     : 0.00881323200155748
prediction time on testing set        : 0.011982885000179522
validation accuracy : 90.66666666666666
testing accuracy    : 93.22429906542055
===============================================
percentage of training data:100% , k value : 4
===============================================
training time                         : 0.14795838003919926
prediction time on validation set     : 0.008716636999452021
prediction time on testing set        : 0.01227169100093306
validation accuracy : 89.0
testing accuracy    : 90.65420560747664
===============================================
```

```
percentage of training data:100% , k value : 5
===============================================
training time                        : 0.14870076018269174
prediction time on validation set    : 0.008966050001617987
prediction time on testing set       : 0.012435998000000836
validation accuracy : 91.66666666666666
testing accuracy    : 93.45794392523365
===============================================
percentage of training data:100% , k value : 6
===============================================
training time                        : 0.12431796014425345
prediction time on validation set    : 0.008947110000008252
prediction time on testing set       : 0.012498003998189233
validation accuracy : 91.0
testing accuracy    : 91.1214953271028
===============================================
percentage of training data:100% , k value : 7
===============================================
training time                        : 0.15305856009945273
prediction time on validation set    : 0.009250955998140853
prediction time on testing set       : 0.012747333999868715
validation accuracy : 90.0
testing accuracy    : 92.28971962616822
===============================================
percentage of training data:100% , k value : 8
===============================================
training time                        : 0.14830554013315123
prediction time on validation set    : 0.009151876998657826
prediction time on testing set       : 0.014422870000998955
validation accuracy : 91.0
testing accuracy    : 92.28971962616822
-----------------------------------------------

percentage of training data:100% , k value : 9
===============================================
training time                        : 0.13909781999245752
prediction time on validation set    : 0.013366161998419557
prediction time on testing set       : 0.01611647699974128
validation accuracy : 91.0
testing accuracy    : 92.05607476635514
===============================================
percentage of training data:100% , k value : 10
===============================================
training time                        : 0.15247823990648612
prediction time on validation set    : 0.009386836998601211
prediction time on testing set       : 0.013070525998045923
validation accuracy : 90.0
testing accuracy    : 90.88785046728972
===============================================
```


the validation accuracies when k varies from 1 to 10
the validation accurracy
0
20
40
60
80
2
4
6
8
10
the value of K


the accuracy curve on the validation set when k varies from 1 to 10
Validation accuracy
87
88
89
90
91
2
4
6
8
10
the value of K


the accuracy curve on the test set when k varies from 1 to 10
testing accuracy
82
84
86
88
90
92
94
2
4
6
8
10
the value of K

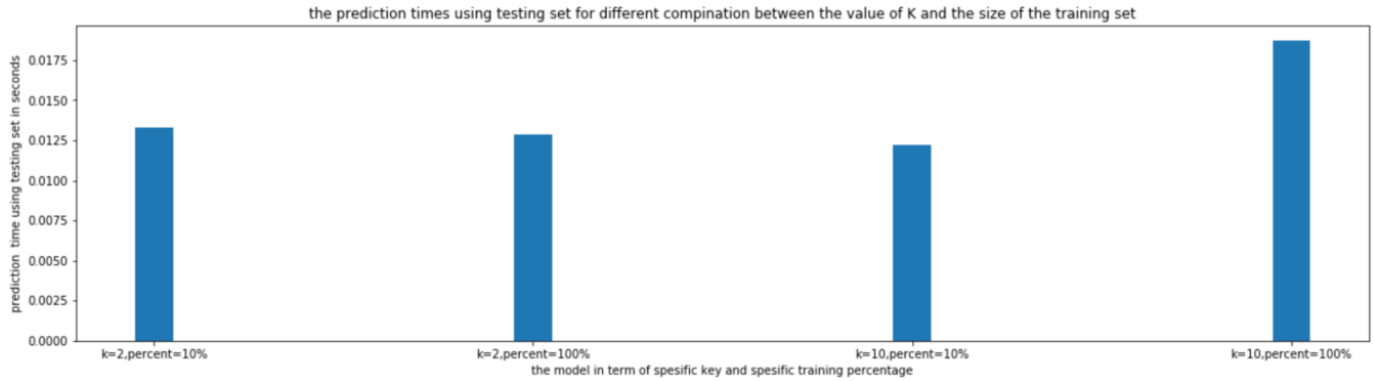


---

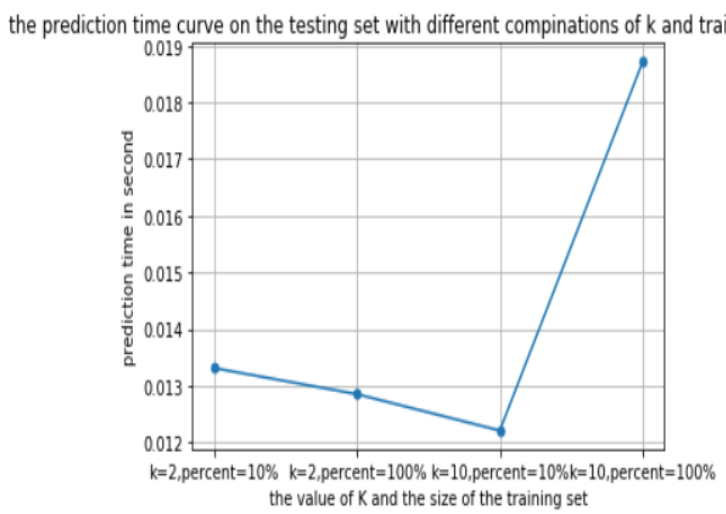
e) the impact of using different combination of k and training samples

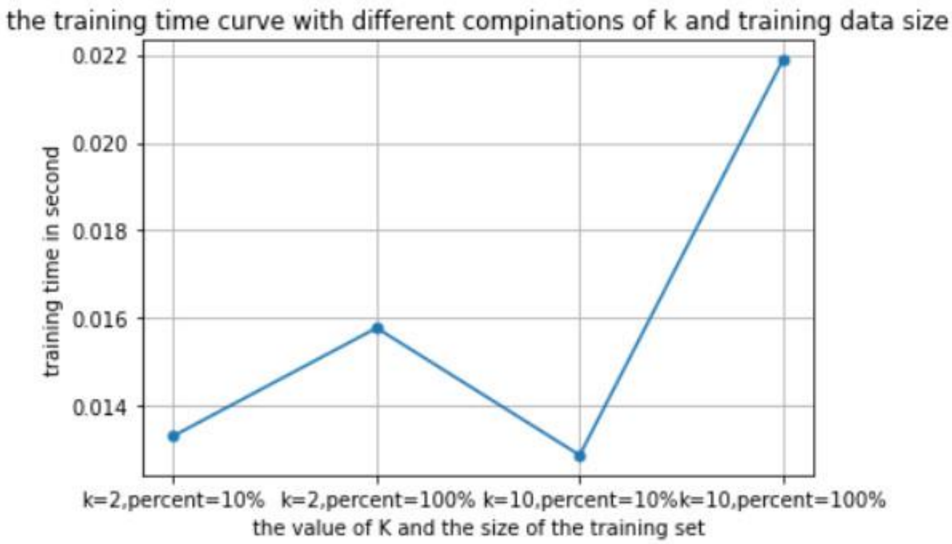
```
percentage of training data:10% , k value : 2
===============================================
training time                          : 0.12100355983420741
prediction time on validation set      : 0.009739812998304842
prediction time on testing set         : 0.014292152001871727
validation accuracy : 78.66666666666666
testing accuracy    : 76.1682242990654З
===============================================
percentage of training data:100% , k value : 2
===============================================
training time                          : 0.15193283994449303
prediction time on validation set      : 0.010103214000992011
prediction time on testing set         : 0.015160610997554613
validation accuracy : 86.66666666666667
testing accuracy    : 82.4766355140187
===============================================
percentage of training data:10% , k value : 10
===============================================
training time                          : 0.12406338013533968
prediction time on validation set      : 0.010961112999211764
prediction time on testing set         : 0.012787009000021499
validation accuracy : 80.0
testing accuracy    : 76.4018691588785
===============================================
percentage of training data:100% , k value : 10
===============================================
training time                          : 0.13260414001706522
prediction time on validation set      : 0.009515468998870347
prediction time on testing set         : 0.013159611000446603
validation accuracy : 90.0
testing accuracy    : 90.88785046728972
===============================================
```

```python
# creating the bar plot
fig = plt.figure(figsize = (20, 5))
labels = ["k=2,percent=10%","k=2,percent=100%","k=10,percent=10%","k=10,percent=100%"]

plt.bar(labels, prediction_testing_times3,width = 0.1)

plt.xlabel("the model in term of spesific key and spesific training percentage")
plt.ylabel("prediction  time using testing set in seconds")
plt.title("the prediction times using testing set for different compination between the value of K and the size of th
plt.show()
```




```python
plt.plot(["k=2,percent=10%","k=2,percent=100%","k=10,percent=10%","k=10,percent=100%"], prediction_testing_times3,mar
plt.xlabel('the value of K and the size of the training set')
plt.ylabel('prediction time in second')
plt.title("the prediction time curve on the testing set with different compinations of k and training data size")
plt.grid()
plt.show()
```





f) the conclusion

for the experiment, c) the size of the training set affects the validation accuracy; the more the training set size the more the validation accuracy.
the size of the training set also affects the test accuracy, the testing accuracy increases by increasing the size of the training set.
for the experiment, d) the value of K affects the accuracy of the model, we tried different numbers of k with 100% of training set, the best model was the model with k = 5.
for the experiment, e) the usage of different combinations between the k value and the size of the training set affects the training time, and also affects the accuracy which is, the more the training set size the more time the model takes to train.