Report Assignment 2

(Calculations & Programming)

Applied Machine Learning ELG5255[EG]

Group Number: G_26

Prepared by:

Sawsan Awad (300327224)

Sondos Ali (300327219)

Toka Mostafa (300327284)

Part 1: Calculations

1. Suppose we have some data collected from a cloth shop, and the dataset contains three features. The first feature is the cloth color (x1), the second feature is the consumer's gender(x2), and the third feature is the price (x3) (we simplify the problem and use high, medium, and low to present different prices). The label TARGET (y) is whether the consumer buys the cloth. Suppose we have the following training data including 15 training samples. Using Bayesian Rule-Based Classifier to make a prediction when Color = G, Gender = F, Price=H. Please include the detailed calculation process.

   Notes: In the color row, R, G, and Y are short for Red, Green, and Yellow; in the Gender row, M and F mean Male and Female, respectively; in the Price row, H, M, and L stand for High Prices, Medium Price and Low Prices, respectively and in the Target row, N and Y are No and Yes

	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

Solution

$$P(Y|G,F,H) = \frac{P(G|Y) * P(F|Y) * P(H|Y) * P(Y)}{P(G,F,H)}$$

$$P(Y|G,F,H) = \frac{{}^{3}/_{9} * {}^{6}/_{9} * {}^{2}/_{9} * {}^{9}/_{15}}{P(G,F,H)} = \frac{{}^{4}/_{135}}{P(G,F,H)}$$

$$P(N|G,F,H) = \frac{P(G|N) * P(F|N) * P(H|N) * P(N)}{P(G,F,H)}$$

$$P(N|G,F,H) = \frac{{}^{2}/_{6} * {}^{1}/_{6} * {}^{2}/_{6} * {}^{6}/_{15}}{P(G,F,H)} = \frac{{}^{1}/_{135}}{P(G,F,H)}$$

P(G,F,H) = $P(G|Y) * P(F|Y) * P(H|Y) * P(Y) + P(G|N) * P(F|N) * P(H|N) * P(N)$

$= {}^{3}/_{9} * {}^{6}/_{9} * {}^{2}/_{9} * {}^{9}/_{15} + {}^{2}/_{6} * {}^{1}/_{6} * {}^{2}/_{6} * {}^{6}/_{15} = {}^{1}/_{27}$

$$P(Y|G,F,H) = \frac{{}^{4}/_{135}}{{}^{1}/_{27}} = {}^{4}/_{5} = 0.8$$

$P(N|G,F,H) = \frac{1/_{135}}{1/_{27}} = 1/_5 = 0.2$

$\therefore P(N|G,F,H) < P(Y|G,F,H)$

$\therefore \textit{The prediction will be Yes}$

2. Consider the following loss table, which contains three actions and two classes. Calculate the expected risk of three actions, and determine the rejection area of P(Class1| x).

Target	Class1	Class2
a1(Choose Class1)	5	2
a2(Choose Class2)	0	5
a3(Rejection)	4	4

Solution

$R(\alpha_1|x) = \lambda_{11}P(c_1|x) + \lambda_{12}P(c_2|x)$

$R(\alpha_1|x) = 0\ P(c_1|x) + 5P(c_2|x)$

$R(\alpha_1|x) = 5[1 - P(c_1|x)]$

$R(\alpha_1|x) = 5 - 5P(c_1|x)$ ----------------(1)

$R(\alpha_2|x) = \lambda_{21}P(c_1|x) + \lambda_{22}P(c_2|x)$

$R(\alpha_2|x) = 5P(c_1|x) + 2P(c_2|x)$

$R(\alpha_2|x) = 5P(c_1|x) + 2[1 - P(c_1|x)]$

$R(\alpha_2|x) = 3P(c_1|x) + 2$ --------------------(2)

$R(\alpha_3|x) = \lambda_{31}P(c_1|x) + \lambda_{32}P(c_2|x)$

$R(\alpha_2|x) = 5P(c_1|x) + 2P(c_2|x)$

$R(\alpha_2|x) = 5P(c_1|x) + 2[1 - P(c_1|x)]$

$R(\alpha_2|x) = 4P(c_1|x) + 4 - 4P(c_1|x)$

$R(\alpha_2|x) = 4$ ----------------------------------(3)

We choose α_1 if:

$R(\alpha_1|x) < 4 \longrightarrow 5 - 5P(c_1|x) < 4 \longrightarrow -5P(c_1|x) < -1 \longrightarrow P(c_1|x) > \frac{1}{5}$

We choose α_2 if:

$R(\alpha_2|x) < 4 \longrightarrow 3P(c_1|x) + 2 < 4 \longrightarrow 3P(c_1|x) < 2 \longrightarrow P(c_1|x) > \frac{2}{3}$

The rejection area of P (Class1| x) = $\frac{1}{5} < P(c_1|x) < \frac{2}{3}$

Part 2: Programming

1. Naïve Bayesian Classifier

1.1.Importing the libraries & datasets:

- We import all libraries we need in our problem.
- For Preprocessing:
  - Pandas
  - NumPy
- For Visualization:
  - Matplotlib

1.2. Load datasets:

- We load wine dataset. There are 3 classes in this dataset, and each sample in this dataset has 13 features.

```
wi=pd.DataFrame(wine['data'],columns=wine['feature_names'])
wi
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740.0
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750.0
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835.0
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840.0
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560.0

178 rows × 13 columns

1.3.Preprocessing:

- We split the data into X, y
- We split X and y into training and testing sets

1.4.Modeling

Naïve Bayes Model:

✓ We train the model on training set. We use Naïve Bayes Model for solving classification problems

```
# training the model on training set
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
GaussianNB()
```

✓ We make predictions on the testing set.

```
# making predictions on the testing set
y_pred = gnb.predict(X_test)
y_pred
```

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

1.5.Classification report:

We use classification report function to help us calculate precision, recall and f1.

Model accuracy: 99%

```
#classification report to help us calculate precision,recall and f1 on the 13 feature
from sklearn.metrics import accuracy_score,classification_report
cr_NB = classification_report(y_test,y_pred)
acc_NB=accuracy_score(y_test,y_pred)
print(cr_NB)
print("Accuracy : {:.2f}%".format(acc_NB))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        28
           1       1.00      0.96      0.98        27
           2       1.00      1.00      1.00        17

    accuracy                           0.99        72
   macro avg       0.99      0.99      0.99        72
weighted avg       0.99      0.99      0.99        72

Accuracy : 0.99%
```

1.6. Feature Selection:

We use SelectKBest () method to select the best two features. And the best two features were (color_intensity , proline).

```
#select two best features
from sklearn.feature_selection import SelectKBest, chi2

x1=SelectKBest(chi2, k=2).fit_transform(X_train, y_train)
x1
```

```
array([[8.660e+00, 7.500e+02],
       [1.020e+01, 8.350e+02],
       [3.400e+00, 3.720e+02],
       [3.800e+00, 4.280e+02],
       [4.500e+00, 7.700e+02],
       [7.700e+00, 7.400e+02],
       [5.680e+00, 1.185e+03],
       [3.050e+00, 8.700e+02],
       [5.280e+00, 6.750e+02],
       [5.250e+00, 1.290e+03],
       [2.620e+00, 4.500e+02],
       [5.640e+00, 1.065e+03],
       [3.850e+00, 7.200e+02],
       [2.650e+00, 5.000e+02],
       [4.600e+00, 6.780e+02],
       [6.250e+00, 1.120e+03],
       [2.300e+00, 4.060e+02],
       [2.800e+00, 6.800e+02],
       [2.800e+00, 4.380e+02],
       [4.900e+00, 1.065e+03],
       [3.800e+00, 6.300e+02],
       [3.210e+00, 8.860e+02],
       [3.050e+00, 4.950e+02],
       [9.300e+00, 8.400e+02],
       [1.950e+00, 4.950e+02],
       [1.280e+00, 5.640e+02],
       [2.760e+00, 3.780e+02],
       [2.900e+00, 5.620e+02],
       [5.750e+00, 1.510e+03],
       [4.200e+00, 1.095e+03],
       [3.840e+00, 9.900e+02],
       [4.000e+00, 8.300e+02],
       [3.700e+00, 1.020e+03],
       [4.800e+00, 5.150e+02],
       [7.600e+00, 6.400e+02],
       [9.200e+00, 5.600e+02],
       [3.950e+00, 1.285e+03],
```

We used the two features to train the model.

```
[26] #train the model on the two features
     nb1= GaussianNB()
     nb1.fit(x1 ,y_train)
     y1_Pred= nb1.predict(x_test_new)
     y1_Pred
```

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

- We calculated accuracy, precision, recall and f1 on the two features

  Model accuracy: 90%

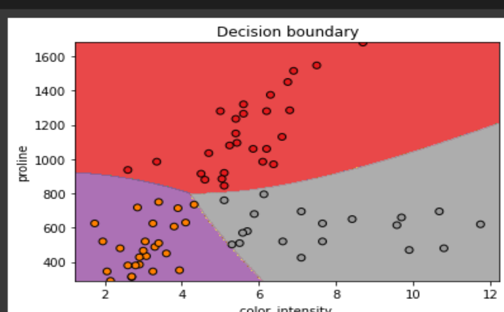
```
#classification report to help us calculate precision,recall and f1 on the only two features
cr1_NB = classification_report(y_test,y1_Pred)
acc1_NB=accuracy_score(y_test,y1_Pred)
print(cr1_NB)
print("Accuracy : {:.2f}%".format(acc1_NB))
```

```
              precision    recall  f1-score   support

           0       0.93      0.89      0.91        28
           1       0.92      0.89      0.91        27
           2       0.84      0.94      0.89        17

    accuracy                           0.90        72
   macro avg       0.90      0.91      0.90        72
weighted avg       0.91      0.90      0.90        72

Accuracy : 0.90%
```

- The decision boundary on the test set on the two features:

```
# this function can be used to plot the decision boundary
def plotDecisionBoundary( X1_test, y1_test, model, title=''):
    plt.close('all')
    plt.figure()
    cm = plt.cm.Set1
    x_min, x_max = X1_test[:, 0].min() - .5, X1_test[:, 0].max() + .5
    y_min, y_max = X1_test[:, 1].min() - .5, X1_test[:, 1].max() + .5
    h = 0.02
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=cm, alpha=.8)
    plt.scatter(
        X1_test[:, 0],
        X1_test[:, 1],
        c=y1_test,
        cmap=cm,
        edgecolors='k',
        alpha=1,
    )
    plt.title(title)
    plt.xlabel("color_intensity ")
    plt.ylabel("proline")
```

```
[21] # plot the decision boundary
plotDecisionBoundary(x_test_new, y1_Pred,model=nb1,title='Decision boundary')
```



2. KNN Classifier

2.1. Importing the libraries & datasets:

- We import all libraries we need in our problem.

2.2. Load datasets:

- First: we installed dataset from Kaggle as csv file.
- Second: we loaded it in google colab.
- Third: we used .head() to show first 5 rows.

Importing the data

```
[2] #Read the data
    Car_Evaluation = pd.read_csv("/content/car_evaluation.csv")
    Car_Evaluation.head()
```

	Price	Maint	Doors	Ppl	Lug_Boot	Safety	Acceptable
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

2.3. Manipulating the dataset

```
[3] Car_Evaluation.columns = ['Price', 'Maint', 'Doors', 'Ppl', 'Lug_Boot', 'Safety', 'Acceptable']
    Car_Evaluation
```

	Price	Maint	Doors	Ppl	Lug_Boot	Safety	Acceptable
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
...	...	...	...	...	...	...	...
1723	low	low	5more	more	med	med	good
1724	low	low	5more	more	med	high	vgood
1725	low	low	5more	more	big	low	unacc
1726	low	low	5more	more	big	med	good
1727	low	low	5more	more	big	high	vgood

1728 rows × 7 columns

```
[4] D = Car_Evaluation.describe()
    print(D)
```

```
        Price  Maint  Doors   Ppl  Lug_Boot  Safety  Acceptable
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
```

```
[5] I = Car_Evaluation.info()
    print(I)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Price       1728 non-null   object
 1   Maint       1728 non-null   object
 2   Doors       1728 non-null   object
 3   Ppl         1728 non-null   object
 4   Lug_Boot    1728 non-null   object
 5   Safety      1728 non-null   object
 6   Acceptable  1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
None
```

```
[6] # Show the cloumns as string to can assign it in another cell
    cloumns = Car_Evaluation.columns
    cloumns
```

```
Index(['Price', 'Maint', 'Doors', 'Ppl', 'Lug_Boot', 'Safety', 'Acceptable'], dtype='object')
```

2.4. Shuffle the data

```
[7] Car_Evaluation = Car_Evaluation.sample(frac = 1).reset_index(drop=True)
    Car_Evaluation
```

	Price	Maint	Doors	Ppl	Lug_Boot	Safety	Acceptable
0	high	med	4	more	big	high	acc
1	low	vhigh	4	more	small	low	unacc
2	high	low	4	4	med	low	unacc
3	med	high	5more	2	med	high	unacc
4	vhigh	med	5more	2	big	med	unacc
...	...	...	...	...	...	...	...
1723	high	vhigh	4	2	small	high	unacc
1724	high	med	3	4	small	high	acc
1725	vhigh	vhigh	5more	more	small	low	unacc
1726	low	med	5more	4	small	med	acc
1727	med	med	4	4	med	med	acc

1728 rows × 7 columns

2.5. Label Encoder

- To represent the attributes by string values.

```
[8] from sklearn import preprocessing
    le = preprocessing.LabelEncoder()
    cloumns = Car_Evaluation.columns
    for i in cloumns:
      Car_Evaluation[i] = le.fit_transform(Car_Evaluation[i])
    Car_Evaluation.head()
```

	Price	Maint	Doors	Ppl	Lug_Boot	Safety	Acceptable
0	0	2	2	2	0	0	0
1	1	3	2	2	2	1	2
2	0	1	2	1	1	1	2
3	2	0	3	0	1	0	2
4	3	2	3	0	0	2	2

2.6. Training and Splitting the data to 3 sets

- First: we split the data into training and remaining dataset, we assigned training dataset size as 1000 that means the remaining dataset will be 728.
- Second: We split the remaining dataset to test and validation sets, we defined test size as 428.

```
x = Car_Evaluation.iloc[:,:-1]
y = Car_Evaluation.iloc[:,-1]
```

```
[10] from sklearn.model_selection import train_test_split
     #In the this step we will split the data into training and remaining dataset, we will assign training dataset size 1000 that means the remaining dataset will be 727
     x_train, x_rem, y_train, y_rem = train_test_split(x,y, train_size=1000)

     #We have to define test_size=427 (that is 58.7% of remaining data)
     x_valid, x_test, y_valid, y_test = train_test_split(x_rem,y_rem, test_size=428)

     print(len(x_train))
     print(len(x_valid))
     print(len(x_test))

1000
300
428
```

2.7. Feature Scaling

- We scaled the features so that all of them can be uniformly evaluated.

```
[11] from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     scaler.fit(x)

     x = scaler.transform(x)
     X_test = scaler.transform(x_test)
```

2.8. Training and Predictions (KNN Classifier)

- First: we used different number of training samples to show the impact of number of training samples, and used 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100% of the training set for 10 separate KNN classifiers and specified a fixed K=2 value.
- Second: model made predicting to x_test and x_validation.
- Third: we calculated the accuracy to test and validation.

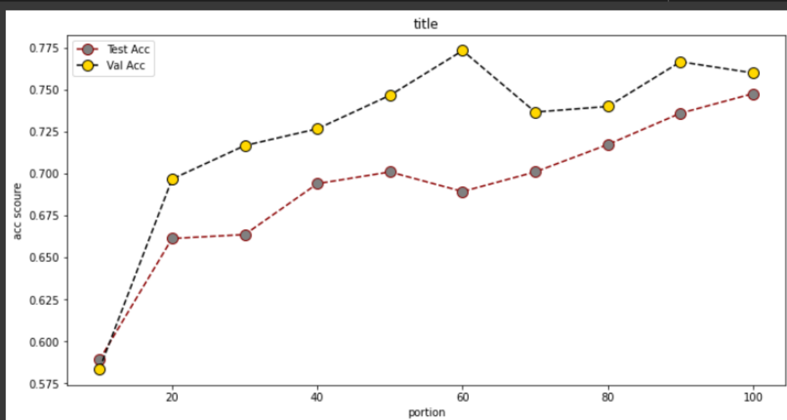
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

Acc = []

for i in np.arange(0.1, 1.1, 0.1):
  model = KNeighborsClassifier(n_neighbors =2)
  #Fiting
  X_train_sample = x_train.sample(frac=i,random_state=2)
  y_train_sample = y_train.sample(frac=i,random_state=2)
  model.fit(X_train_sample, y_train_sample)
  #Predicting
  predict_test = model.predict(x_test)
  predict_val = model.predict(x_valid)
  #Accuracy
  Percent_Accuracy_Test = accuracy_score(y_test,predict_test)
  Percent_Accuracy_Validation =accuracy_score(y_valid,predict_val)


  Acc.append({
      'Samples' : int(i),
      'Test Acc' : Percent_Accuracy_Test,
      'Val Acc' : Percent_Accuracy_Validation
  })
Accu = pd.DataFrame.from_dict(Acc)
print(Accu)
```

```
   Samples  Test Acc   Val Acc
0        0  0.588785  0.583333
1        0  0.661215  0.696667
2        0  0.663551  0.716667
3        0  0.693925  0.726667
4        0  0.700935  0.746667
5        0  0.689252  0.773333
6        0  0.700935  0.736667
7        0  0.717290  0.740000
8        0  0.735981  0.766667
9        1  0.747664  0.760000
```

2.9. Performance of the Validation set and Testing set

- We showed the performance (accuracy score) of the validation set and testing set and made X axis the portion of the training set, Y axis the accuracy score. There are two lines in total, one is for the validation set and another is for the testing set.

```
[14] plt.figure(figsize=(12, 6))
     portion = list(np.arange(10,101,10))
     plt.plot(portion, Accu['Test Acc'], label = 'Test Acc' ,color='darkred', linestyle='dashed', marker='o', markerfacecolor='grey', markersize=10)
     plt.plot(portion, Accu['Val Acc'], label = 'Val Acc' ,color='black', linestyle='dashed', marker='o', markerfacecolor='gold', markersize=10)
     plt.title('Validation & Testing Accuracy')
     plt.xlabel('portion')
     plt.ylabel('acc scoure')
     plt.legend()
     plt.show()
```

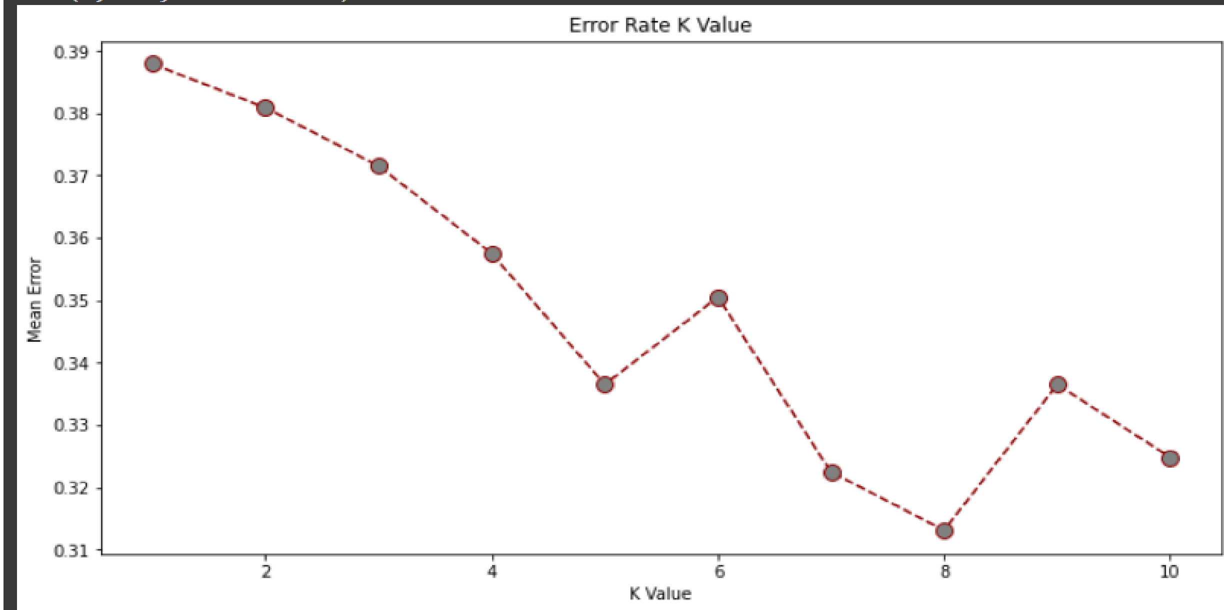


2.10. Comparing Error Rate with the K Value

- We use Error Rate way because it is a one way to help us find the best value of K by plotting the graph of K value and the corresponding error rate for the dataset.
- We noticed in the output that the mean error is zero when the value of the K is 8, but it would be change cause the random sets which model would use it in the next run.

```
[13] Error = []
     #Calculating error for K values between 1 and 10
     for s in range(1, 11):
         knn = KNeighborsClassifier(n_neighbors=s)
         knn.fit(x_train, y_train)
         pred_s = knn.predict(X_test)
         Error.append(np.mean(pred_s != y_test))
     print(Error)
     plt.figure(figsize=(12, 6))
     plt.plot(range(1, 11), Error, color='darkred', linestyle='dashed', marker='o', markerfacecolor='grey', markersize=10)
     plt.title('Error Rate K Value')
     plt.xlabel('K Value')
     plt.ylabel('Mean Error')
```

```
[0.3878504672897196, 0.3808411214953271, 0.37149532710280375, 0.3574766355140187, 0.3364485981308411,
Text(0, 0.5, 'Mean Error')
```



2.11. Use 100% of training samples & Find the best K value

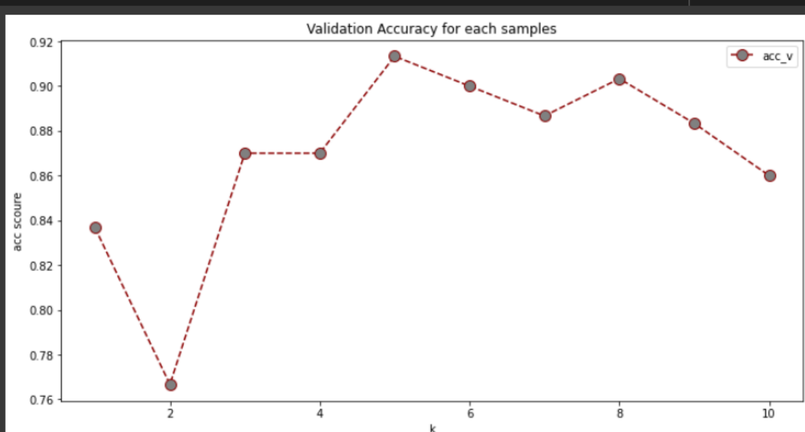
- We selected the best K value with the highest validation and test accuracy by using all of the training data
- **The best K value is: 5**
  Validation Accuracy is: 91%

```
[15] l = []
     for i in range(1, 11):
       model = KNeighborsClassifier(n_neighbors=i)
       model.fit(x_train, y_train)
       pred_t = model.predict(x_test)
       pred_v = model.predict(x_valid)

       acc_t = accuracy_score(y_test,pred_t)
       acc_v = accuracy_score(y_valid,pred_v)
       l.append({
           'n_neighbors' : i,
           'acc_t' : acc_t,
           'acc_v' : acc_v
       })
     k_v = pd.DataFrame.from_dict(l)
     k_v.sort_values(by = ['acc_v','acc_t'], ascending=False)
```

	n_neighbors	acc_t	acc_v
4	5	0.897196	0.913333
7	8	0.866822	0.903333
5	6	0.869159	0.900000
6	7	0.871495	0.886667
8	9	0.871495	0.883333
3	4	0.862150	0.870000
2	3	0.850467	0.870000
9	10	0.862150	0.860000
0	1	0.813084	0.836667
1	2	0.740654	0.766667

2.11.1 . Plotting to Validation Accuracy for each sample

```
[22] plt.figure(figsize=(12, 6))
     portion = list(np.arange(1,11,1))
     plt.plot(portion, k_v['acc_v'], label = 'acc_v' ,color='darkred', linestyle='dashed', marker='o', markerfacecolor='grey', markersize=10)
     plt.xlabel('k')
     plt.ylabel('acc scoure')
     plt.title('Validation Accuracy for each samples')
     plt.legend()
     plt.show()
     type(x_train)
```




```
pandas.core.frame.DataFrame
```

2.12. Analysis the training time when use different number of training samples

- We got 4 cases:
  - 10% of the whole training set and K = 2
    Training_Time = 0.010582208633422852
    Prediction_Time = 0.0411374568939209
  - 100% of the whole training set and K = 2
    Training_Time = 0.002919435501098633
    Prediction_Time = 0.051938533782958984
  - 10% of the whole training set and K = 10
    Training_Time = 0.0032396316528320312
    Prediction_Time = 0.04671621322631836
  - 100% of the whole training set and K = 10.
    Training_Time = 0.002959728240966797
    Prediction_Time = 0.05570387840270996

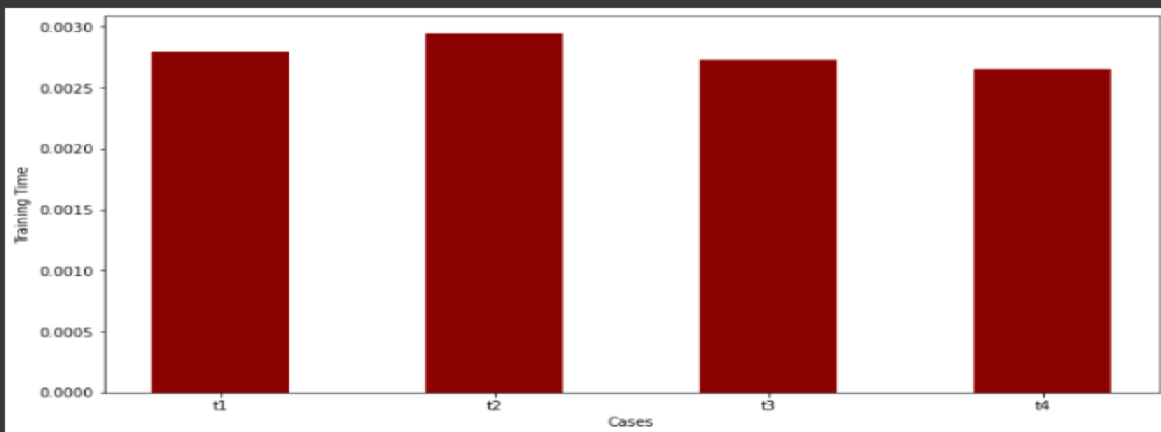
```
[26] import time
     # 10% training set , k=2
     start1 = time.time()
     knn = KNeighborsClassifier(n_neighbors=2)
     knn.fit(x_train.iloc[:100], y_train.iloc[:100])
     end1 = time.time()
     t1 = end1 - start1

     start2 =time.time()
     pred_t = knn.predict(X_test)
     end2 = time.time()
     p1 = end2 - start2

     # 100% training set , k=2
     start11 = time.time()
     knn = KNeighborsClassifier(n_neighbors=2)
     knn.fit(x_train, y_train)
     end11 = time.time()
     t2 = end11 - start11

     start22 =time.time()
     pred_t = knn.predict(X_test)
     end22 = time.time()
     p2 = end22 - start22
```

```
# 10% training set , k=10
start3 = time.time()
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(x_train.iloc[:100], y_train.iloc[:100])
end3 = time.time()
t3 = end3 - start3

start4 =time.time()
pred_t = knn.predict(X_test)
end4 = time.time()
p3 = end4 - start4

# 100% training set , k=10
start33 = time.time()
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(x_train, y_train)
end33 = time.time()
t4 = end33 - start33

start44 =time.time()
pred_t = knn.predict(X_test)
end44 = time.time()
p4 = end44 - start44


print("Training time: ",t1)
print("Training time: ",t2)
print("Training time: ",t3)
print("Training time: ",t4)
print("Perdicitoin time: ",p1)
print("Perdicitoin time: ",p2)
print("Perdicitoin time: ",p3)
print("Perdicitoin time: ",p4)
```

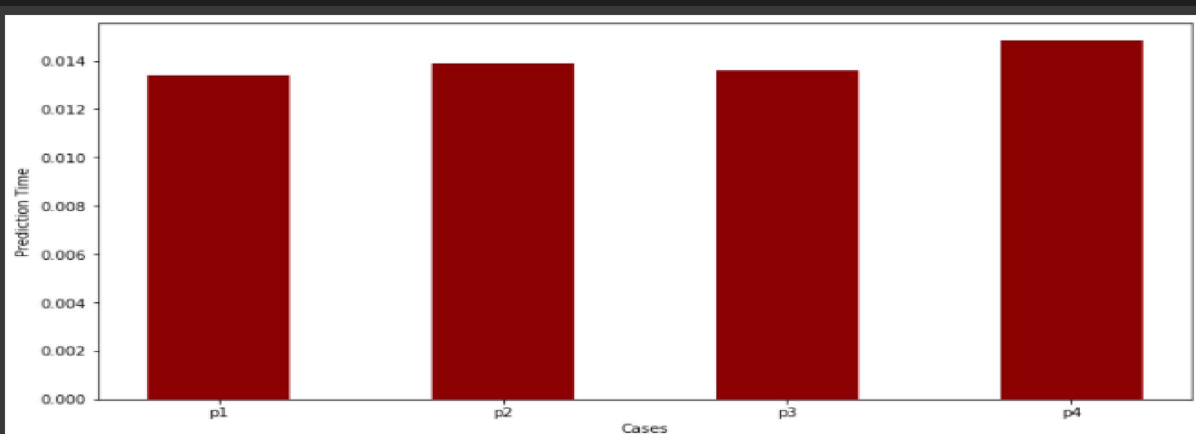
```
Training time:  0.011065006256103516
Training time:  0.016573190689086914
Training time:  0.0035552978515625
Training time:  0.004053831100463867
Perdicitoin time:  0.0411374568939209
Perdicitoin time:  0.051938533782958984
Perdicitoin time:  0.04671621322631836
Perdicitoin time:  0.05570387840270996
```

2.12.1. Plotting the Training and Prediction Time to 4 cases

```
plt.figure(figsize=(12, 6))
cases = ['t1','t2','t3','t4']
times = [t1,t2,t3,t4]
plt.bar(cases, times ,color='darkred', width = 0.5)
plt.xlabel('Cases')
plt.ylabel('Training Time')
plt.show()
```




```
plt.figure(figsize=(12, 6))
cases = ['p1','p2','p3','p4']
times = [p1,p2,p3,p4]
plt.bar(cases, times ,color='darkred', width = 0.5)
plt.xlabel('Cases')
plt.ylabel('Prediction Time')
plt.show()
```



Conclusion

From applying KNN model, we found that:

1. According to point(c), after we are modeling the car evaluation with KNN algorithm. when we use 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100% of the training set for 10 separate KNN classifiers we notice that the performance increase as training set increase.
    - ✓ **Note: The variance and testing sets become more closer to each other which avoid overfitting.**

2. According to point(d), when we use 100% training set and varying with k from 1 to 10, we notice that the best k in 5 and 8 but 5 is more efficient because the variance between testing and validation is very small than k=8. Another reason makes k=5 is best is avoiding our model to become confused in future unseen data if the two classes have the same amount of points (4 points in each).
    - ✓ **Note: It prefers selecting an odd value for K to get the best accuracy.**

3. According to point(e), when we analyze the training time by using different number of training samples, we notice the training time become little bit larger when the k =2 which is make sense because it makes more calculation.
    - ✓ **Note: The size of the samples doesn’t have affection on the time of training and prediction**