In [1]:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
data = pd.read_csv("Churn.csv",na_values='Nan')
data
```

Out[2]:

	Unnamed: 0	state	area.code	account.length	voice.plan	voice.messages	intl.plan	intl.mins
0	1	KS	area_code_415	128	yes	25	no	10.0
1	2	OH	area_code_415	107	yes	26	no	13.7
2	3	NJ	area_code_415	137	no	0	no	12.2
3	4	OH	area_code_408	84	no	0	yes	6.6
4	5	OK	area_code_415	75	no	0	yes	10.1
...	...	...	...	...	...	...	...	...
4995	4996	HI	area_code_408	50	yes	40	no	9.9
4996	4997	WV	area_code_415	152	no	0	no	14.7
4997	4998	DC	area_code_415	61	no	0	no	13.6
4998	4999	DC	area_code_510	109	no	0	no	8.5
4999	5000	VT	area_code_415	86	yes	34	no	9.3

5000 rows × 21 columns

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Unnamed: 0      5000 non-null   int64  
 1   state           5000 non-null   object 
 2   area.code       5000 non-null   object 
 3   account.length  5000 non-null   int64  
 4   voice.plan      5000 non-null   object 
 5   voice.messages  5000 non-null   int64  
 6   intl.plan       5000 non-null   object 
 7   intl.mins       5000 non-null   float64
 8   intl.calls      5000 non-null   int64  
 9   intl.charge     5000 non-null   float64
 10  day.mins        5000 non-null   float64
 11  day.calls       5000 non-null   int64  
 12  day.charge      4993 non-null   float64
 13  eve.mins        4976 non-null   float64
```

```
 14  eve.calls       5000 non-null   int64
 15  eve.charge      5000 non-null   float64
 16  night.mins      5000 non-null   float64
 17  night.calls     5000 non-null   int64
 18  night.charge    5000 non-null   float64
 19  customer.calls  5000 non-null   int64
 20  churn           5000 non-null   object
dtypes: float64(8), int64(8), object(5)
memory usage: 820.4+ KB
```

In [4]:

```
data.isna().sum()
```

Out[4]:

```
Unnamed: 0         0
state              0
area.code          0
account.length     0
voice.plan         0
voice.messages     0
intl.plan          0
intl.mins          0
intl.calls         0
intl.charge        0
day.mins           0
day.calls          0
day.charge         7
eve.mins          24
eve.calls          0
eve.charge         0
night.mins         0
night.calls        0
night.charge       0
customer.calls     0
churn              0
dtype: int64
```

In [5]:

```
data.dropna(inplace=True)
```

In [6]:

```
data.columns
```

Out[6]:

```
Index(['Unnamed: 0', 'state', 'area.code', 'account.length', 'voice.plan',
       'voice.messages', 'intl.plan', 'intl.mins', 'intl.calls', 'intl.charge',
       'day.mins', 'day.calls', 'day.charge', 'eve.mins', 'eve.calls',
       'eve.charge', 'night.mins', 'night.calls', 'night.charge',
       'customer.calls', 'churn'],
      dtype='object')
```

In [7]:

```
data.rename(columns={'area.code':'area_code','account.length':'account_length','voice.pl
                     'voice.messages':'voice_messages','intl.plan':'intl_plan','intl.min
                    'intl.calls':'intl_calls','intl.charge':'intl_charge','day.mins':'da
                    'day.charge':'day_charge','eve.mins':'eve_mins','eve.calls':'eve_cal
                    'night.mins':'night_mins','night.calls':'night_calls','night.charge'
                    },inplace=True)
data
```

Out[7]:

	Unnamed: 0	state	area_code	account_length	voice_plan	voice_messages	intl_plan	intl_min
0	1	KS	area_code_415	128	yes	25	no	10.
1	2	OH	area_code_415	107	yes	26	no	13.
2	3	NJ	area_code_415	137	no	0	no	12.
3	4	OH	area_code_408	84	no	0	yes	6.
4	5	OK	area_code_415	75	no	0	yes	10.
...	...	...	...	...	...	...	...	.
4995	4996	HI	area_code_408	50	yes	40	no	9.
4996	4997	WV	area_code_415	152	no	0	no	14.
4997	4998	DC	area_code_415	61	no	0	no	13.
4998	4999	DC	area_code_510	109	no	0	no	8.
4999	5000	VT	area_code_415	86	yes	34	no	9.

4969 rows × 21 columns

In [8]:

```
data.drop('Unnamed: 0',axis=1,inplace=True)
data
```

Out[8]:

	state	area_code	account_length	voice_plan	voice_messages	intl_plan	intl_mins	intl_calls
0	KS	area_code_415	128	yes	25	no	10.0	3
1	OH	area_code_415	107	yes	26	no	13.7	3
2	NJ	area_code_415	137	no	0	no	12.2	5
3	OH	area_code_408	84	no	0	yes	6.6	7
4	OK	area_code_415	75	no	0	yes	10.1	3
...	...	...	...	...	...	...	...	...
4995	HI	area_code_408	50	yes	40	no	9.9	5
4996	WV	area_code_415	152	no	0	no	14.7	2
4997	DC	area_code_415	61	no	0	no	13.6	4
4998	DC	area_code_510	109	no	0	no	8.5	6
4999	VT	area_code_415	86	yes	34	no	9.3	16

4969 rows × 20 columns

In [9]:

```
le = preprocessing.LabelEncoder()
objlist = ['area_code','intl_plan','voice_plan','churn','state']
data[objlist] = data[objlist].apply(le.fit_transform)
```

In [10]:

```
data
```

Out[10]:

	state	area_code	account_length	voice_plan	voice_messages	intl_plan	intl_mins	intl_calls	in
0	16	1	128	1	25	0	10.0	3	
1	35	1	107	1	26	0	13.7	3	
2	31	1	137	0	0	0	12.2	5	
3	35	0	84	0	0	1	6.6	7	
4	36	1	75	0	0	1	10.1	3	
...	...	...	...	...	...	...	...	...	
4995	11	0	50	1	40	0	9.9	5	
4996	49	1	152	0	0	0	14.7	2	
4997	7	1	61	0	0	0	13.6	4	
4998	7	2	109	0	0	0	8.5	6	
4999	46	1	86	1	34	0	9.3	16	

4969 rows × 20 columns

In [11]:

```
x = data.iloc[:,0:-1]
x
```

Out[11]:

	state	area_code	account_length	voice_plan	voice_messages	intl_plan	intl_mins	intl_calls	in
0	16	1	128	1	25	0	10.0	3	
1	35	1	107	1	26	0	13.7	3	
2	31	1	137	0	0	0	12.2	5	
3	35	0	84	0	0	1	6.6	7	
4	36	1	75	0	0	1	10.1	3	
...	...	...	...	...	...	...	...	...	
4995	11	0	50	1	40	0	9.9	5	
4996	49	1	152	0	0	0	14.7	2	
4997	7	1	61	0	0	0	13.6	4	
4998	7	2	109	0	0	0	8.5	6	
4999	46	1	86	1	34	0	9.3	16	

4969 rows × 19 columns

In [12]:

```
y = data.iloc[:,-1]
y
```

Out[12]:
```
0       0
1       0
2       0
3       0
4       0
       ..
4995    0
4996    1
4997    0
4998    0
4999    0
Name: churn, Length: 4969, dtype: int32
```

In [13]:
```
from sklearn.model_selection import train_test_split, cross_val_score
```

In [14]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3, random_state=15)
```

In [15]:
```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn import preprocessing
from sklearn import metrics
```

In [49]:
```
model=LogisticRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(classification_report(y_test,y_pred))
from sklearn.metrics import precision_recall_fscore_support as score
precision,recall,fscore,support = score(y_test,y_pred,average='macro')
LR=fscore
LR
```

```
              precision    recall  f1-score   support

           0       0.87      0.99      0.93      1290
           1       0.44      0.07      0.12       201

    accuracy                           0.86      1491
   macro avg       0.65      0.53      0.52      1491
weighted avg       0.81      0.86      0.82      1491
```

Out[49]:
```
0.5227995509877178
```

In [17]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score as ac
confusion_matrix=confusion_matrix(y_test,y_pred)
print(confusion_matrix)
```

```
[[1272   21]
 [ 173   25]]
```

In [19]:

```
#KNN
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
%matplotlib inline
k_range = [2*i+1 for i in range(0,20)]
k_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn,x_train , y_train, cv = 10)
    k_scores.append(scores.mean())
    print("K value=",k)
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(x_train,y_train)
    pred = model.predict(x_test)
    print(classification_report(y_test,pred))
```

```
K value= 1
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      1293
           1       0.34      0.38      0.36       198

    accuracy                           0.82      1491
   macro avg       0.62      0.63      0.63      1491
weighted avg       0.83      0.82      0.82      1491

K value= 3
              precision    recall  f1-score   support

           0       0.90      0.96      0.93      1293
           1       0.55      0.31      0.40       198

    accuracy                           0.88      1491
   macro avg       0.73      0.64      0.66      1491
weighted avg       0.85      0.88      0.86      1491

K value= 5
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1293
           1       0.69      0.31      0.43       198

    accuracy                           0.89      1491
   macro avg       0.80      0.65      0.68      1491
weighted avg       0.87      0.89      0.87      1491

K value= 7
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1293
           1       0.72      0.30      0.42       198

    accuracy                           0.89      1491
   macro avg       0.81      0.64      0.68      1491
weighted avg       0.88      0.89      0.87      1491
```

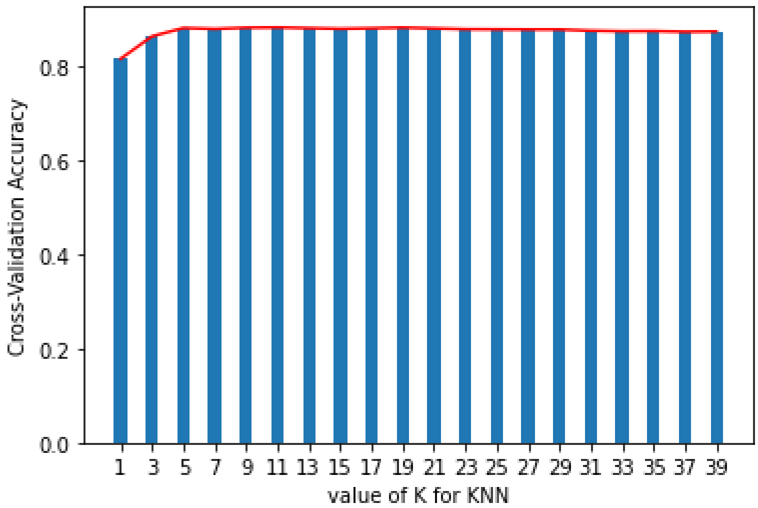
```
K value= 9
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      1293
           1       0.76      0.28      0.41       198

    accuracy                           0.89      1491
   macro avg       0.83      0.63      0.68      1491
weighted avg       0.88      0.89      0.87      1491

K value= 11
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      1293
           1       0.77      0.28      0.41       198

    accuracy                           0.89      1491
   macro avg       0.83      0.63      0.68      1491
weighted avg       0.88      0.89      0.87      1491

K value= 13
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      1293
           1       0.77      0.28      0.41       198

    accuracy                           0.89      1491
   macro avg       0.84      0.63      0.68      1491
weighted avg       0.88      0.89      0.87      1491

K value= 15
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      1293
           1       0.79      0.26      0.39       198

    accuracy                           0.89      1491
   macro avg       0.84      0.63      0.67      1491
weighted avg       0.88      0.89      0.87      1491

K value= 17
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      1293
           1       0.80      0.26      0.39       198

    accuracy                           0.89      1491
   macro avg       0.85      0.62      0.67      1491
weighted avg       0.88      0.89      0.87      1491

K value= 19
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      1293
           1       0.83      0.25      0.39       198

    accuracy                           0.89      1491
   macro avg       0.86      0.62      0.66      1491
weighted avg       0.89      0.89      0.87      1491
```

```
K value= 21
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      1293
           1       0.83      0.25      0.39       198

    accuracy                           0.89      1491
   macro avg       0.86      0.62      0.66      1491
weighted avg       0.89      0.89      0.87      1491

K value= 23
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1293
           1       0.82      0.24      0.37       198

    accuracy                           0.89      1491
   macro avg       0.86      0.61      0.65      1491
weighted avg       0.89      0.89      0.86      1491

K value= 25
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1293
           1       0.80      0.23      0.35       198

    accuracy                           0.89      1491
   macro avg       0.85      0.61      0.65      1491
weighted avg       0.88      0.89      0.86      1491

K value= 27
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1293
           1       0.84      0.23      0.36       198

    accuracy                           0.89      1491
   macro avg       0.87      0.61      0.65      1491
weighted avg       0.89      0.89      0.86      1491

K value= 29
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1293
           1       0.83      0.22      0.35       198

    accuracy                           0.89      1491
   macro avg       0.86      0.61      0.65      1491
weighted avg       0.88      0.89      0.86      1491

K value= 31
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1293
           1       0.83      0.22      0.34       198

    accuracy                           0.89      1491
   macro avg       0.86      0.61      0.64      1491
```

```
weighted avg       0.88      0.89      0.86      1491

K value= 33
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1293
           1       0.84      0.21      0.34       198

    accuracy                           0.89      1491
   macro avg       0.87      0.60      0.64      1491
weighted avg       0.88      0.89      0.86      1491

K value= 35
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1293
           1       0.84      0.21      0.34       198

    accuracy                           0.89      1491
   macro avg       0.87      0.60      0.64      1491
weighted avg       0.88      0.89      0.86      1491

K value= 37
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1293
           1       0.84      0.21      0.34       198

    accuracy                           0.89      1491
   macro avg       0.87      0.60      0.64      1491
weighted avg       0.88      0.89      0.86      1491

K value= 39
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1293
           1       0.83      0.22      0.34       198

    accuracy                           0.89      1491
   macro avg       0.86      0.61      0.64      1491
weighted avg       0.88      0.89      0.86      1491
```

In [20]:

```
plt.bar(k_range, k_scores)
plt.plot(k_range, k_scores, color = "red")

plt.xlabel('value of K for KNN')
plt.ylabel('Cross-Validation Accuracy')
plt.xticks(k_range)
plt.show()
```



In [21]:

```
model = KNeighborsClassifier(n_neighbors=5)
model.fit(x_train,y_train)
pred = model.predict(x_test)
```

In [51]:

```
print(classification_report(y_test,pred))
precision,recall,fscore,support = score(y_test,pred,average='macro')
KNN=fscore
KNN
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1290
           1       0.93      0.78      0.85       201

    accuracy                           0.96      1491
   macro avg       0.95      0.89      0.91      1491
weighted avg       0.96      0.96      0.96      1491
```

Out[51]:

```
0.9149499531734697
```

In [52]:

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
model = DecisionTreeClassifier(criterion='entropy',min_samples_split=5)
model.fit(x_train,y_train)
pred = model.predict(x_test)
metrics.accuracy_score(pred,y_test)
print(classification_report(y_test,pred))
precision,recall,fscore,support = score(y_test,pred,average='macro')
DT=fscore
DT
```

```
              precision    recall  f1-score   support

           0       0.96      0.95      0.95      1290
           1       0.69      0.73      0.71       201

    accuracy                           0.92      1491
   macro avg       0.83      0.84      0.83      1491
```

```
weighted avg       0.92      0.92      0.92      1491
```

Out[52]:
0.8327714404470512

In [53]:
```
# Random Forest Classification

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=7)

num_trees = 140
max_features = 6

model = RandomForestClassifier(n_estimators=num_trees,max_samples=0.8, max_features=max_

model.fit(x_train,y_train)

pred = model.predict(x_test)

print(classification_report(y_test,pred))
precision,recall,fscore,support = score(y_test,pred,average='macro')
RD=fscore
RD
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1074
           1       0.94      0.80      0.86       169

    accuracy                           0.97      1243
   macro avg       0.95      0.90      0.92      1243
weighted avg       0.96      0.97      0.96      1243
```

Out[53]:
0.9214157486080257

In [25]:
```
#SVM
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# clf = SVC()
# param_grid = [{'kernel':["rbf"],'gamma':[50,5,10,0.5,1,0.001,0.0001,0.00001],'C':[1,15
# gsv = GridSearchCV(clf,param_grid,cv=10)
# gsv.fit(x_train,y_train)
```

In [55]:
```
clf = SVC(C= 15, gamma = 1,kernel="rbf")
clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
print(classification_report(y_test, y_pred))
precision,recall,fscore,support = score(y_test,y_pred,average='macro')
SVM=fscore
SVM
```

```
              precision    recall  f1-score   support

           0       0.86      1.00      0.93      1074
           1       0.00      0.00      0.00       169

    accuracy                           0.86      1243
   macro avg       0.43      0.50      0.46      1243
weighted avg       0.75      0.86      0.80      1243
```

Out[55]:

```
0.46353042727665084
```

In [58]:

```python
#ANN
import tensorflow as tf
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(15,input_dim=19,activation = 'sigmoid'))
model.add(tf.keras.layers.Dense(10,activation='sigmoid'))
model.add(tf.keras.layers.Dense(1,activation='sigmoid'))
```

In [59]:

```python
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_3 (Dense)             (None, 15)                300       
                                                                 
 dense_4 (Dense)             (None, 10)                160       
                                                                 
 dense_5 (Dense)             (None, 1)                 11        
                                                                 
=================================================================
Total params: 471
Trainable params: 471
Non-trainable params: 0
_________________________________________________________________
```

In [60]:

```python
# Compile model
model.compile(loss ='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [61]:

```python
history = model.fit(x,y,validation_split=0.20, epochs=50, batch_size=100)
```

```
Epoch 1/50
40/40 [==============================] - 1s 7ms/step - loss: 1.0661 - accuracy: 0.1419 -
val_loss: 0.8822 - val_accuracy: 0.1419
Epoch 2/50
40/40 [==============================] - 0s 5ms/step - loss: 0.7453 - accuracy: 0.3603 -
val_loss: 0.6273 - val_accuracy: 0.8561
Epoch 3/50
40/40 [==============================] - 0s 4ms/step - loss: 0.5636 - accuracy: 0.8571 -
val_loss: 0.5116 - val_accuracy: 0.8581
Epoch 4/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4833 - accuracy: 0.8581 -
val_loss: 0.4599 - val_accuracy: 0.8581
Epoch 5/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4459 - accuracy: 0.8581 -
```

```
val_loss: 0.4344 - val_accuracy: 0.8581
Epoch 6/50
40/40 [==============================] - 0s 6ms/step - loss: 0.4276 - accuracy: 0.8581 -
val_loss: 0.4217 - val_accuracy: 0.8581
Epoch 7/50
40/40 [==============================] - 0s 6ms/step - loss: 0.4182 - accuracy: 0.8581 -
val_loss: 0.4151 - val_accuracy: 0.8581
Epoch 8/50
40/40 [==============================] - 0s 5ms/step - loss: 0.4132 - accuracy: 0.8581 -
val_loss: 0.4115 - val_accuracy: 0.8581
Epoch 9/50
40/40 [==============================] - 0s 6ms/step - loss: 0.4106 - accuracy: 0.8581 -
val_loss: 0.4095 - val_accuracy: 0.8581
Epoch 10/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4091 - accuracy: 0.8581 -
val_loss: 0.4086 - val_accuracy: 0.8581
Epoch 11/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4081 - accuracy: 0.8581 -
val_loss: 0.4079 - val_accuracy: 0.8581
Epoch 12/50
40/40 [==============================] - 0s 6ms/step - loss: 0.4074 - accuracy: 0.8581 -
val_loss: 0.4075 - val_accuracy: 0.8581
Epoch 13/50
40/40 [==============================] - 0s 6ms/step - loss: 0.4069 - accuracy: 0.8581 -
val_loss: 0.4071 - val_accuracy: 0.8581
Epoch 14/50
40/40 [==============================] - 0s 5ms/step - loss: 0.4067 - accuracy: 0.8581 -
val_loss: 0.4069 - val_accuracy: 0.8581
Epoch 15/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4066 - accuracy: 0.8581 -
val_loss: 0.4063 - val_accuracy: 0.8581
Epoch 16/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4063 - accuracy: 0.8581 -
val_loss: 0.4064 - val_accuracy: 0.8581
Epoch 17/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4062 - accuracy: 0.8581 -
val_loss: 0.4063 - val_accuracy: 0.8581
Epoch 18/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4060 - accuracy: 0.8581 -
val_loss: 0.4056 - val_accuracy: 0.8581
Epoch 19/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4057 - accuracy: 0.8581 -
val_loss: 0.4060 - val_accuracy: 0.8581
Epoch 20/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4055 - accuracy: 0.8581 -
val_loss: 0.4059 - val_accuracy: 0.8581
Epoch 21/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4052 - accuracy: 0.8581 -
val_loss: 0.4045 - val_accuracy: 0.8581
Epoch 22/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4049 - accuracy: 0.8581 -
val_loss: 0.4041 - val_accuracy: 0.8581
Epoch 23/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4046 - accuracy: 0.8581 -
val_loss: 0.4040 - val_accuracy: 0.8581
Epoch 24/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4044 - accuracy: 0.8581 -
val_loss: 0.4038 - val_accuracy: 0.8581
Epoch 25/50
```

```
40/40 [==============================] - 0s 5ms/step - loss: 0.4039 - accuracy: 0.8581 -
val_loss: 0.4031 - val_accuracy: 0.8581
Epoch 26/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4035 - accuracy: 0.8581 -
val_loss: 0.4032 - val_accuracy: 0.8581
Epoch 27/50
40/40 [==============================] - 0s 3ms/step - loss: 0.4033 - accuracy: 0.8581 -
val_loss: 0.4023 - val_accuracy: 0.8581
Epoch 28/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4026 - accuracy: 0.8581 -
val_loss: 0.4017 - val_accuracy: 0.8581
Epoch 29/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4022 - accuracy: 0.8581 -
val_loss: 0.4012 - val_accuracy: 0.8581
Epoch 30/50
40/40 [==============================] - 0s 3ms/step - loss: 0.4018 - accuracy: 0.8581 -
val_loss: 0.4012 - val_accuracy: 0.8581
Epoch 31/50
40/40 [==============================] - 0s 3ms/step - loss: 0.4012 - accuracy: 0.8581 -
val_loss: 0.4010 - val_accuracy: 0.8581
Epoch 32/50
40/40 [==============================] - 0s 4ms/step - loss: 0.4005 - accuracy: 0.8581 -
val_loss: 0.3983 - val_accuracy: 0.8581
Epoch 33/50
40/40 [==============================] - 0s 3ms/step - loss: 0.3987 - accuracy: 0.8581 -
val_loss: 0.3979 - val_accuracy: 0.8581
Epoch 34/50
40/40 [==============================] - 0s 3ms/step - loss: 0.3976 - accuracy: 0.8581 -
val_loss: 0.3955 - val_accuracy: 0.8581
Epoch 35/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3968 - accuracy: 0.8581 -
val_loss: 0.3983 - val_accuracy: 0.8581
Epoch 36/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3961 - accuracy: 0.8581 -
val_loss: 0.3939 - val_accuracy: 0.8581
Epoch 37/50
40/40 [==============================] - 0s 3ms/step - loss: 0.3952 - accuracy: 0.8581 -
val_loss: 0.3926 - val_accuracy: 0.8581
Epoch 38/50
40/40 [==============================] - 0s 3ms/step - loss: 0.3944 - accuracy: 0.8581 -
val_loss: 0.3918 - val_accuracy: 0.8581
Epoch 39/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3926 - accuracy: 0.8581 -
val_loss: 0.3938 - val_accuracy: 0.8581
Epoch 40/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3920 - accuracy: 0.8581 -
val_loss: 0.3883 - val_accuracy: 0.8581
Epoch 41/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3920 - accuracy: 0.8581 -
val_loss: 0.3892 - val_accuracy: 0.8581
Epoch 42/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3911 - accuracy: 0.8581 -
val_loss: 0.3878 - val_accuracy: 0.8581
Epoch 43/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3906 - accuracy: 0.8581 -
val_loss: 0.3858 - val_accuracy: 0.8581
Epoch 44/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3893 - accuracy: 0.8581 -
val_loss: 0.3885 - val_accuracy: 0.8581
```

```
Epoch 45/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3879 - accuracy: 0.8581 -
val_loss: 0.3838 - val_accuracy: 0.8581
Epoch 46/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3866 - accuracy: 0.8581 -
val_loss: 0.3835 - val_accuracy: 0.8581
Epoch 47/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3857 - accuracy: 0.8581 -
val_loss: 0.3850 - val_accuracy: 0.8581
Epoch 48/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3844 - accuracy: 0.8581 -
val_loss: 0.3855 - val_accuracy: 0.8581
Epoch 49/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3848 - accuracy: 0.8581 -
val_loss: 0.3816 - val_accuracy: 0.8581
Epoch 50/50
40/40 [==============================] - 0s 4ms/step - loss: 0.3823 - accuracy: 0.8581 -
val_loss: 0.3810 - val_accuracy: 0.8581
```

In [62]:

```
model.save_weights("mywt.kmw")
```

In [71]:

```
#evaluate the model
scores = model.evaluate(x,y)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
ANN=model.metrics_names[1], scores[1]*100
ANN=ANN[1]
ANN
```

```
156/156 [==============================] - 0s 1ms/step - loss: 0.3811 - accuracy: 0.8581
accuracy: 85.81%
```

Out[71]:

```
85.8120322227478
```

In [33]:

```
#Naive_Bayes
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train, y_train)
```

Out[33]:

```
▾ GaussianNB

GaussianNB()
```

In [34]:

```
y_pred =  nb.predict(x_test)
```

In [35]:

```
from sklearn.metrics import confusion_matrix
conf_matrix1=confusion_matrix(y_test, y_pred)
conf_matrix1
```

Out[35]:

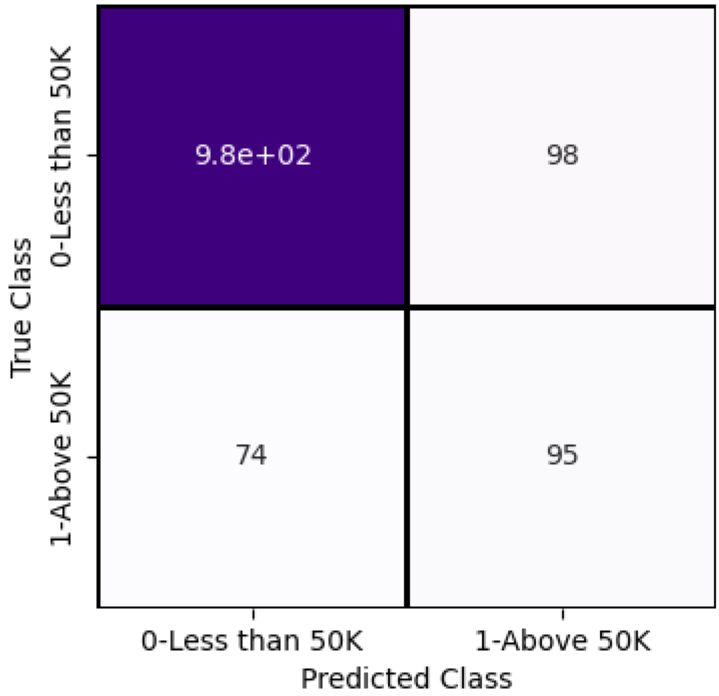
```
array([[976,  98],
       [ 74,  95]], dtype=int64)
```

In [36]:

```
import seaborn as sns
%matplotlib inline
fig, ax= plt.subplots(1,1, figsize=(4,4),dpi=100)

xlabels=['0-Less than 50K','1-Above 50K']
ylabels=['0-Less than 50K','1-Above 50K']

sns.heatmap(conf_matrix1,annot=True, cbar=None, cmap="Purples",xticklabels=xlabels,ytick
            linewidths=1,linecolor='black')
ax.set_xlabel('Predicted Class')
ax.set_ylabel('True Class')
```

Out[36]:

```
Text(20.722222222222214, 0.5, 'True Class')
```



In [72]:

```
from sklearn.metrics import classification_report
names=['0-Less than 50K','1-Above 50K']
print(classification_report(y_test, y_pred,target_names=names))
precision,recall,fscore,support = score(y_test,y_pred,average='macro')
NB=fscore
NB
```

```
                 precision    recall  f1-score   support

0-Less than 50K       0.86      1.00      0.93      1074
    1-Above 50K       0.00      0.00      0.00       169

       accuracy                           0.86      1243
      macro avg       0.43      0.50      0.46      1243
   weighted avg       0.75      0.86      0.80      1243
```

Out[72]:

```
0.46353042727665084
```

In [73]:

```
#Bagging

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=2)

num_trees = 100
model = BaggingClassifier(max_samples=0.8, n_estimators=num_trees,random_state=8)

model.fit(x_train,y_train)
pred = model.predict(x_test)
print(classification_report(y_test,pred))
precision,recall,fscore,support = score(y_test,pred,average='macro')
BG=fscore
BG
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      1290
           1       0.91      0.76      0.83       201

    accuracy                           0.96      1491
   macro avg       0.93      0.87      0.90      1491
weighted avg       0.96      0.96      0.96      1491
```

Out[73]:

```
0.9012623649683373
```

In [74]:

```
#AdaBoost Classifier

from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(random_state=96,base_estimator=RandomForestClassifier(random_st
                        n_estimators=100,learning_rate=0.01)
clf.fit(x_train,y_train)
pred = clf.predict(x_test)
print(classification_report(y_test,pred))
report = classification_report(y_test,pred)
precision,recall,fscore,support = score(y_test,pred,average='macro')
ADB=fscore
ADB
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1290
           1       0.93      0.78      0.85       201

    accuracy                           0.96      1491
   macro avg       0.95      0.89      0.91      1491
weighted avg       0.96      0.96      0.96      1491
```

Out[74]:

```
0.9149499531734697
```

In [45]:

```
# from sklearn.metrics import precision_recall_fscore_support as score
# precision,recall,fscore,support = score(y_test,pred,average='macro')
# ada=fscore
```

In [46]:

```
# ada
```

Out[46]:

0.9149499531734697

Accuracy Scores All Models

In [79]:

```
# initialize list elements
data = [['Logistic Regression',LR],['KNearest Nighbour',KNN],
        ['Decision Tree',DT],['Random Forest',RD],['Support Vector Machine',SVM],['Artif
        ['Navie Bais',NB],['Bagging',BG],['AdaBoosting',ADB]]
# Create the pandas DataFrame with column name is provided explicitly
df = pd.DataFrame(data, columns=['Algorithm Names','Accuracy'])
df
```

Out[79]:

	Algorithm Names	Accuracy
0	Logistic Regression	0.522800
1	KNearest Nighbour	0.914950
2	Decision Tree	0.832771
3	Random Forest	0.921416
4	Support Vector Machine	0.463530
5	Artificial Neural Network	85.812032
6	Navie Bais	0.463530
7	Bagging	0.901262
8	AdaBoosting	0.914950

In [ ]: