KNN- it is one of the simplest & widely used algorithm in which a new data point is classified based on similarity in ¶

the specific group of neighbouring data points.

what is k in KNN ?

it denotes the number of nearest neighbor which are voting class of the new data or the testing datset

In [2]:

```
# about the dataset
# This dataset contains the details of the users in a social networking site to find wheather a user buys a
# clicking the ad on the site based on thei gender, salary, age .
```

In [3]:

```
# importing necessary libraries
```

In [4]:

```
import pandas as pd           # to read files
import numpy as np     # for calculations
import matplotlib.pyplot as plt    # for visualization
import sklearn                     # for KNN
```

In [5]:

```
# reading the dataset
mydata=pd.read_csv('Social_Network_Ads.csv')
```

In [6]:

```
mydata.head(3)
```

Out[6]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0

checking basic info about dataset

In [7]:

```
mydata.describe()
```

Out[7]:

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

In [8]:

```
mydata.shape
```

Out[8]:

```
(400, 5)
```

In [9]:

```
np.sqrt(400)
```

Out[9]:

```
20.0
```

In [10]:
```
mydata.isnull().sum()
```

Out[10]:
```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

In [11]:
```
mydata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   User ID          400 non-null    int64 
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64 
 3   EstimatedSalary  400 non-null    int64 
 4   Purchased        400 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

In [12]:
```
# since the dataset containing charactr we need to convert it in numeric by using LabelEncoder
# also sepearting the dependent and indepedent variable
```

In [13]:
```
x_ind=mydata.iloc[:, [1,2,3]].values
y_dep=mydata.iloc[:,-1].values
```

In [14]:
```
x_ind
```

Out[14]:
```
array([['Male', 19, 19000],
       ['Male', 35, 20000],
       ['Female', 26, 43000],
       ...,
       ['Female', 50, 20000],
       ['Male', 36, 33000],
       ['Female', 49, 36000]], dtype=object)
```

In [15]:
```
y_dep
```

Out[15]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1], dtype=int64)
```

In [16]:
```
from sklearn.preprocessing import LabelEncoder
```

In [17]:
```
le= LabelEncoder()
```

In [18]:
```
x_ind[:,0]=le.fit_transform(x_ind[:,0])
```

In [19]:
```
x_ind
```

Out[19]:
```
array([[1, 19, 19000],
       [1, 35, 20000],
       [0, 26, 43000],
       ...,
       [0, 50, 20000],
       [1, 36, 33000],
       [0, 49, 36000]], dtype=object)
```

Now we are performing train_test_split basically dividing the data into 80% for training ans building the model

20% for evaluating and predicting the model

In [20]:

```
from sklearn.model_selection import train_test_split
```

In [21]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.8,random_state=0)
```

Next, we are doing feature scaling to the training and test set of independent variables for reducing the size to

smaller values

In [22]:

```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
```

Now we have to create and train the K Nearest Neighbor model with the training set

In [23]:

```
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='euclidean', p=2)
classifier.fit(x_train,y_train)
```

Out[23]:

```
KNeighborsClassifier(metric='euclidean')
```

In [24]:
```
# we are using 3 parameters here n_neighbors is setting as 5, which means 5 neighborhood points are requird
# a given point.
# the distance metric we are using euclidean
# we have to select p value also p=1 (Manhattan), p=2 (euclidean)
```

our model is created, now we have to predict the output for the test dataset

In [25]:
```
y_pred=classifier.predict(x_test)
```

In [26]:
```
# Comparing true and predicted values
```

In [27]:
```
y_test
```

Out[27]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0], dtype=int64)
```

In [28]:

```
y_pred
```

Out[28]:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0], dtype=int64)
```

Evaluating the model

In [29]:

```
 from sklearn.metrics import confusion_matrix, accuracy_score
```

In [30]:

```
cm= confusion_matrix(y_test,y_pred)
```

In [31]:

```
cm
```

Out[31]:

```
array([[190,  11],
       [ 43,  76]], dtype=int64)
```

In [32]:

```
ac=accuracy_score(y_test,y_pred)
```

In [33]:

```
ac
```

Out[33]:

```
0.83125
```

In [34]:

```
# we got the accuracy of 83% which means our model is performing well
```

In [35]:

```
# we have to find error rate also
# we use erorr rate to choose the optimal k value as there is no pre defined statistical method to find the
# value of K
```

In [36]:

```
error_rate=[]
for i in range (1,40):
    knn_new=KNeighborsClassifier(n_neighbors=i)
    knn_new.fit(x_train,y_train)
    y_pred_er=knn_new.predict(x_test)
    error_rate.append(np.mean(y_pred_er !=y_test))
```

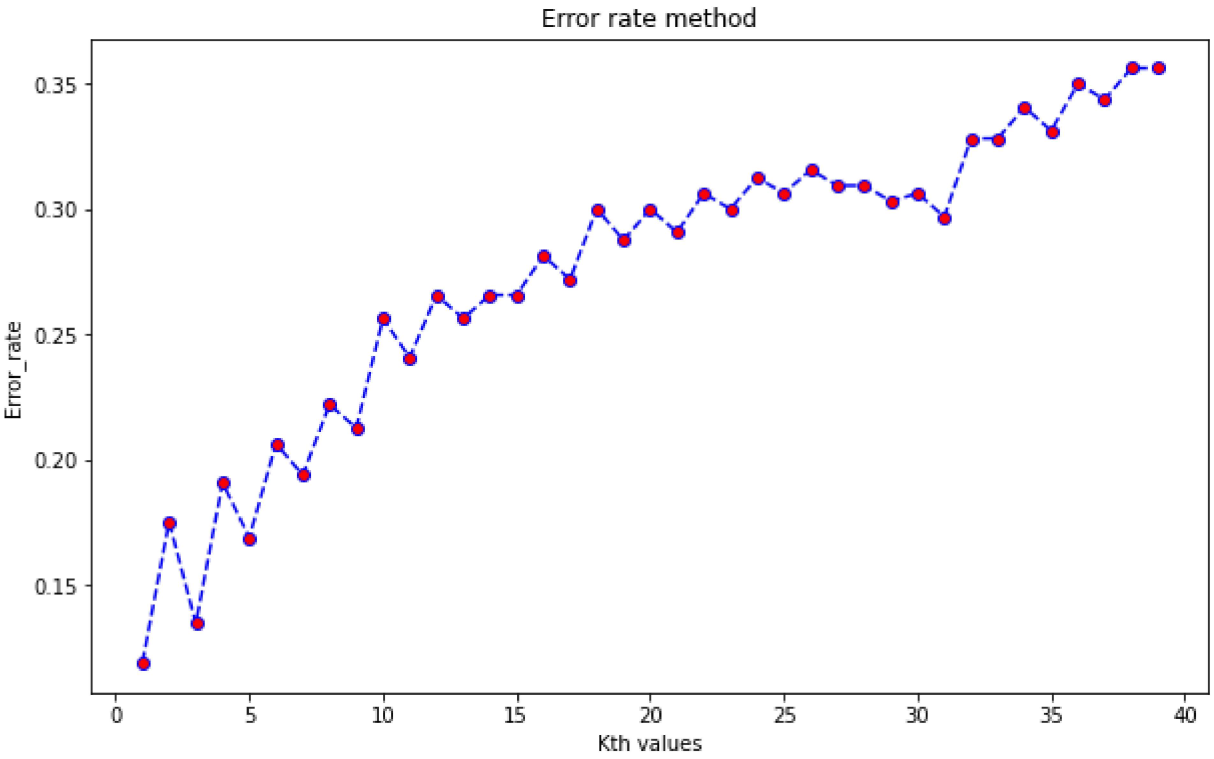
plt.figure(figsize=(10,6)) plt.plot(range(1,40),error_rate,color='blue',linestyle='dashed',marker='o', markerfacecolor='red') plt.title("Error rate method") plt.xlabel("Kth values") plt.ylabel("Error_rate")

In [38]:

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue',linestyle='dashed',marker='o', markerfacecolor='red')
plt.title("Error rate method")
plt.xlabel("Kth values")
plt.ylabel("Error_rate")
```

Out[38]:

```
Text(0, 0.5, 'Error_rate')
```



In [56]:

```
KNN1=KNeighborsClassifier(n_neighbors=5,p=2,metric='euclidean')
```

In [57]:

```
KNN1=KNN1.fit(x_train,y_train)
```

In [58]:

```
y_pred = KNN1.predict(x_test)
```

In [59]:

```
accuracy_score(y_test,y_pred)
```

Out[59]:

```
0.83125
```

In [47]:

```
# after error rate i have taken n_neighbors as 5 still the accuracy is 83% which means model is pretty good
```

Conclusion

In [ ]:

```
# kNN has advantages as well as disadvantages like
# advantages = 1) no traning period- as it stores/ memorise the training dataset and learns from it only at
# makinng real time prediction
# 2) new data point can be added seamlessly
# Disadvantages - 1) does not work with large datasets
# 2) need feature scaling- standardization or normalization before applying KNN algorithm.
```