K Nearest Neighbors Classifier Model (KNN)

Objective

- The objective is to build model to explore the possibility in predicting income level based on the individual’s personal information.

K Nearest Neighbors (KNN)

- The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

Distance Metrics used in KNN algorithm

- Euclidean Distance - The Euclidean distance between two points is the length of the straight line segment connecting them. This metric helps us calculate the net displacement done between the two states of an object.
- Manhattan Distance - The Manhattan distance between two points is the sum of the absolute differences between the x and y coordinates of each point. Used to measure the minimum distance by summing the length of all the intervals needed to get from one location to another in a city, it’s also known as the taxicab distance.
- Minkowski Distance - Minkowski distance generalizes the Euclidean and Manhattan distances. It adds a parameter called “order” that allows different distance measures to be calculated. Minkowski distance indicates a distance between two points in a normed vector space
- Hamming distance - Hamming distance is used to compare two binary vectors (also called data strings or bitstrings). To calculate it, data first has to be translated into a binary system.
- Cosine Distance- This distance metric is used mainly to calculate similarity between two vectors. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in the same direction.

How does KNN work?

- Step-1: Select the number K of the neighbors
- Step-2: Calculate the Euclidean distance of K number of neighbors
- Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
- Step-4: Among these k neighbors, count the number of the data points in each category.
- Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Dataset source & brief

- Adult income dataset is sourced from Kaggle. The dataset contains 15 columns. The target variable is Income, it is divided into two classes: <=50K and >50K. Other 14 attribues are the demographics and other features to describe a person like Age, Workclass, Final Weight, Education, Education Number of Years, Marital-status, Occupation, Relationship, Race, Sex, Capital-gain, Capital-loss, Hours-per-week, Native-country. The dataset contains missing values that are marked with a question mark character (?). The target class is imbalanced, making '>50K': majority class at approximately 25% & '<=50K': minority class at approximately 75%.

Import required libraries

In [1]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Import dataset

In [2]:

```
df=pd.read_csv(r"C:\Users\manme\Documents\Priya\Stats and ML\Dataset\adult.csv")
df.head(2)
```

Out[2]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White

Check basic information

In [3]:

```
df.shape # check shape
```

Out[3]:

```
(48842, 15)
```

In [4]:

```
df.describe().T  #statistical summary of numerical columns
```

Out[4]:

	count	mean	std	min	25%	50%	75%
age	48842.0	38.643585	13.710510	17.0	28.0	37.0	48.0
fnlwgt	48842.0	189664.134597	105604.025423	12285.0	117550.5	178144.5	237642.0
educational-num	48842.0	10.078089	2.570973	1.0	9.0	10.0	12.0
capital-gain	48842.0	1079.067626	7452.019058	0.0	0.0	0.0	0.0
capital-loss	48842.0	87.502314	403.004552	0.0	0.0	0.0	0.0
hours-per-week	48842.0	40.422382	12.391444	1.0	40.0	40.0	45.0

In [5]:

```
df.select_dtypes(include=['object']).describe(include='all') #summary of categorical col
```

Out[5]:

	workclass	education	marital-status	occupation	relationship	race	gender	native-country	inc
count	48842	48842	48842	48842	48842	48842	48842	48842	4
unique	9	16	7	15	6	5	2	42	
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	United-States	<
freq	33906	15784	22379	6172	19716	41762	32650	43832	3

In [6]:

```
df.duplicated().sum()  #check duplicates
```

Out[6]:

52

In [7]:

```
df=df.drop_duplicates()  # remove duplicates
```

In [8]:

```
for i in df.columns:
    print("**********************************", i ,
          "*******************************************")
    print()
    print(set(df[i].tolist()))
    print()
```

```
********************************** age ********************************
***********

{17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 3
4, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90}

********************************** workclass **************************
*****************

{'Self-emp-not-inc', 'Local-gov', 'Never-worked', 'Federal-gov', 'Priva
te', '?', 'Without-pay', 'State-gov', 'Self-emp-inc'}

********************************** fnlwgt *****************************
**************

{262153, 262158, 131088, 131091, 393248, 131117, 393264, 262196, 26220
```

In [9]:

```
df.replace('?', np.nan, inplace=True)  # work,occupation & native-country '?' Replacing
```

In [10]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48790 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   age              48790 non-null  int64 
 1   workclass        45995 non-null  object
 2   fnlwgt           48790 non-null  int64 
 3   education        48790 non-null  object
 4   educational-num  48790 non-null  int64 
 5   marital-status   48790 non-null  object
 6   occupation       45985 non-null  object
 7   relationship     48790 non-null  object
 8   race             48790 non-null  object
 9   gender           48790 non-null  object
 10  capital-gain     48790 non-null  int64 
 11  capital-loss     48790 non-null  int64 
 12  hours-per-week   48790 non-null  int64 
 13  native-country   47934 non-null  object
 14  income           48790 non-null  object
dtypes: int64(6), object(9)
memory usage: 6.0+ MB
```

In [11]:

```
df.isnull().sum()/len(df)*100     # check null values percentage
```

Out[11]:

```
age                 0.000000
workclass           5.728633
fnlwgt              0.000000
education           0.000000
educational-num     0.000000
marital-status      0.000000
occupation          5.749129
relationship        0.000000
race                0.000000
gender              0.000000
capital-gain        0.000000
capital-loss        0.000000
hours-per-week      0.000000
native-country      1.754458
income              0.000000
dtype: float64
```

In [12]:

```
# dropping educational- num variable
df.drop(['educational-num'],axis=1, inplace=True)
```

In [13]:

```
# Handling missing value with mode
df['workclass']=df['workclass'].fillna('Private')
df['occupation']=df['occupation'].fillna('Prof-specialty')
df['native-country']=df['native-country'].fillna('United-States')
```

In [14]:

```
df['workclass'].value_counts()
```

Out[14]:

```
Private             36655
Self-emp-not-inc     3861
Local-gov            3136
State-gov            1981
Self-emp-inc         1694
Federal-gov          1432
Without-pay            21
Never-worked           10
Name: workclass, dtype: int64
```

In [15]:

```
df['education'].unique()
```

Out[15]:

```
array(['11th', 'HS-grad', 'Assoc-acdm', 'Some-college', '10th',
       'Prof-school', '7th-8th', 'Bachelors', 'Masters', 'Doctorate',
       '5th-6th', 'Assoc-voc', '9th', '12th', '1st-4th', 'Preschool'],
      dtype=object)
```

In [16]:

```
#replacing names in education column
df['education'].replace(['11th','10th','7th-8th','5th-6th','9th','12th','1st-4th','Presc
df['education'].replace('HS-grad', 'Highschool_Grad',inplace=True)
df['education'].replace(['Assoc-acdm','Assoc-voc'],'Associate-degree', inplace = True)
df['education'].replace('Some-college', 'College',inplace=True)
df['education'].replace('Bachelors', 'Bachelors',inplace=True)
df['education'].replace(['Masters','Prof-school'],'Masters', inplace = True)
df['education'].replace('Doctorate', 'Doctorate',inplace=True)
```

In [17]:

```
df['marital-status'].unique()
```

Out[17]:

```
array(['Never-married', 'Married-civ-spouse', 'Widowed', 'Divorced',
       'Separated', 'Married-spouse-absent', 'Married-AF-spouse'],
      dtype=object)
```

In [18]:

```
# replacing name in 'marital-status' column
df['marital-status'].replace(['Married-civ-spouse','Married-spouse-absent', 'Married-AF-
df['marital-status'].replace('Never-married', 'UnMarried',inplace=True)
df['marital-status'].replace(['Divorced','Separated'], 'Separated',inplace=True)
df['marital-status'].replace(['Widowed'], 'Widowed',inplace=True)
```

In [19]:

```
df['occupation'].unique()
```

Out[19]:

```
array(['Machine-op-inspct', 'Farming-fishing', 'Protective-serv',
       'Prof-specialty', 'Other-service', 'Craft-repair', 'Adm-clerical',
       'Exec-managerial', 'Tech-support', 'Sales', 'Priv-house-serv',
       'Transport-moving', 'Handlers-cleaners', 'Armed-Forces'],
      dtype=object)
```

In [20]:

```
df['relationship'].unique()
```

Out[20]:

```
array(['Own-child', 'Husband', 'Not-in-family', 'Unmarried', 'Wife',
       'Other-relative'], dtype=object)
```

In [21]:

```
df['race'].unique()
```

Out[21]:

```
array(['Black', 'White', 'Asian-Pac-Islander', 'Other',
       'Amer-Indian-Eskimo'], dtype=object)
```

In [22]:

```
df['native-country'].unique()
```

Out[22]:

```
array(['United-States', 'Peru', 'Guatemala', 'Mexico',
       'Dominican-Republic', 'Ireland', 'Germany', 'Philippines',
       'Thailand', 'Haiti', 'El-Salvador', 'Puerto-Rico', 'Vietnam',
       'South', 'Columbia', 'Japan', 'India', 'Cambodia', 'Poland',
       'Laos', 'England', 'Cuba', 'Taiwan', 'Italy', 'Canada', 'Portugal',
       'China', 'Nicaragua', 'Honduras', 'Iran', 'Scotland', 'Jamaica',
       'Ecuador', 'Yugoslavia', 'Hungary', 'Hong', 'Greece',
       'Trinadad&Tobago', 'Outlying-US(Guam-USVI-etc)', 'France',
       'Holand-Netherlands'], dtype=object)
```

In [23]:

```
df['native-country'].value_counts()
```

Out[23]:

```
United-States                 44648
Mexico                          943
Philippines                     294
Germany                         206
Puerto-Rico                     184
Canada                          182
El-Salvador                     155
India                           151
Cuba                            138
England                         127
China                           122
South                           115
Jamaica                         106
Italy                           105
Dominican-Republic              103
Japan                            92
Poland                           87
Vietnam                          86
Guatemala                        86
Columbia                         85
Haiti                            75
Portugal                         67
Taiwan                           65
Iran                             59
Greece                           49
Nicaragua                        49
Peru                             46
Ecuador                          45
France                           38
Ireland                          37
Hong                             30
Thailand                         30
Cambodia                         28
Trinadad&Tobago                  27
Yugoslavia                       23
Outlying-US(Guam-USVI-etc)       23
Laos                             23
Scotland                         21
Honduras                         20
Hungary                          19
Holand-Netherlands                1
Name: native-country, dtype: int64
```

In [24]:

```
# As United-States makes more than 90% data so replacing other countries name in 'native
df['native-country'].replace(['Peru', 'Guatemala', 'Mexico','Dominican-Republic', 'Irela
                              'Thailand', 'Haiti', 'El-Salvador', 'Puerto-Rico', 'Vietna
                              'Japan', 'India', 'Cambodia', 'Poland','Laos', 'England',
                              'Canada', 'Portugal', 'China', 'Nicaragua', 'Honduras', 'I
                              'Ecuador', 'Yugoslavia', 'Hungary', 'Hong', 'Greece', 'Tri
                              'Outlying-US(Guam-USVI-etc)', 'France','Holand-Netherlands
```

Encoding

In [25]:

```
#label encoder for target variable
df['income']=df['income'].astype('category')
df['income']=df['income'].cat.codes
```

In [26]:

```
df['gender'].value_counts()
```

Out[26]:

```
Male      32614
Female    16176
Name: gender, dtype: int64
```

In [27]:

```
#label encoder for gender variable
df['gender']=df['gender'].astype('category')
df['gender']=df['gender'].cat.codes
```

In [28]:

```
# encode categorical columns
from sklearn.preprocessing import LabelEncoder
df = df.apply(LabelEncoder().fit_transform)
df.head()
```

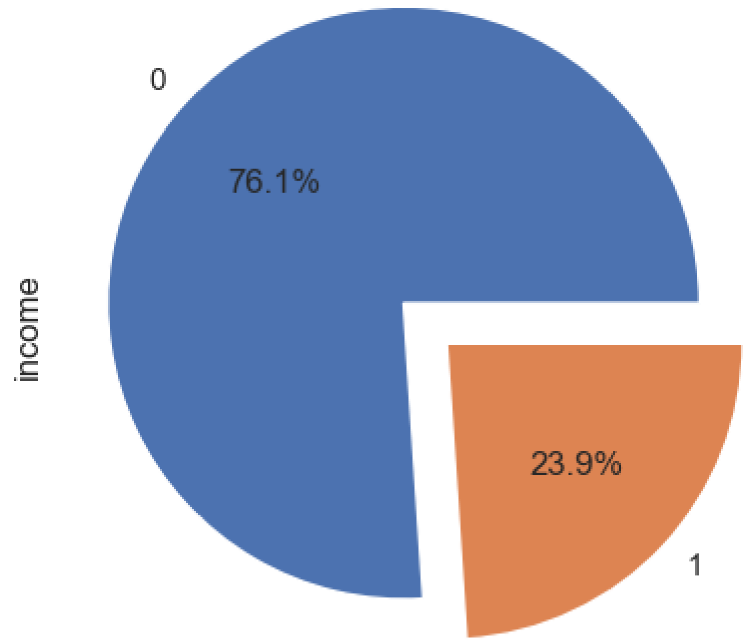
Out[28]:

	age	workclass	fnlwgt	education	marital-status	occupation	relationship	race	gender	capita gai
0	8	3	19329	0	2	6	3	2	1	
1	21	3	4212	5	0	4	0	4	1	
2	11	1	25340	1	0	10	0	4	1	
3	27	3	11201	3	0	6	0	2	1	9
4	1	3	5411	3	2	9	3	4	0	

Exploratory Data Analysis

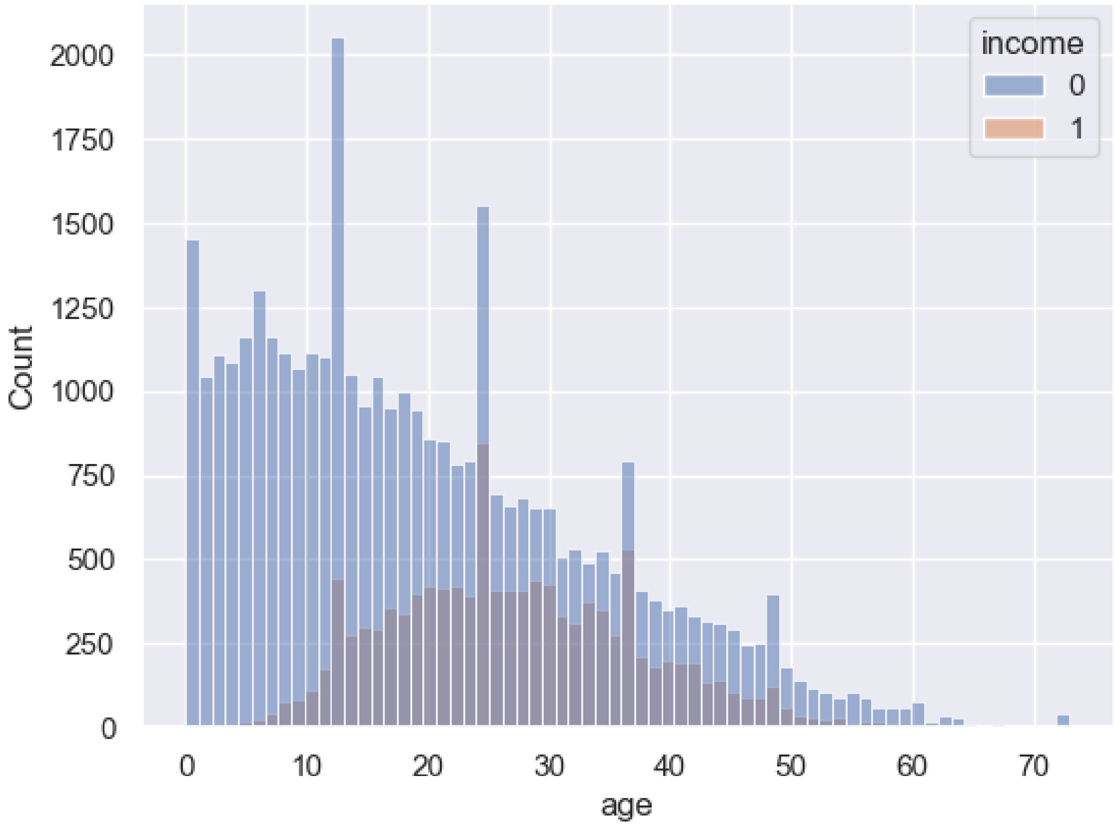
In [34]:

```
df['income'].value_counts().plot(kind='pie',explode=[0.1,0.1],autopct='%0.1f%%')
plt.show()
```



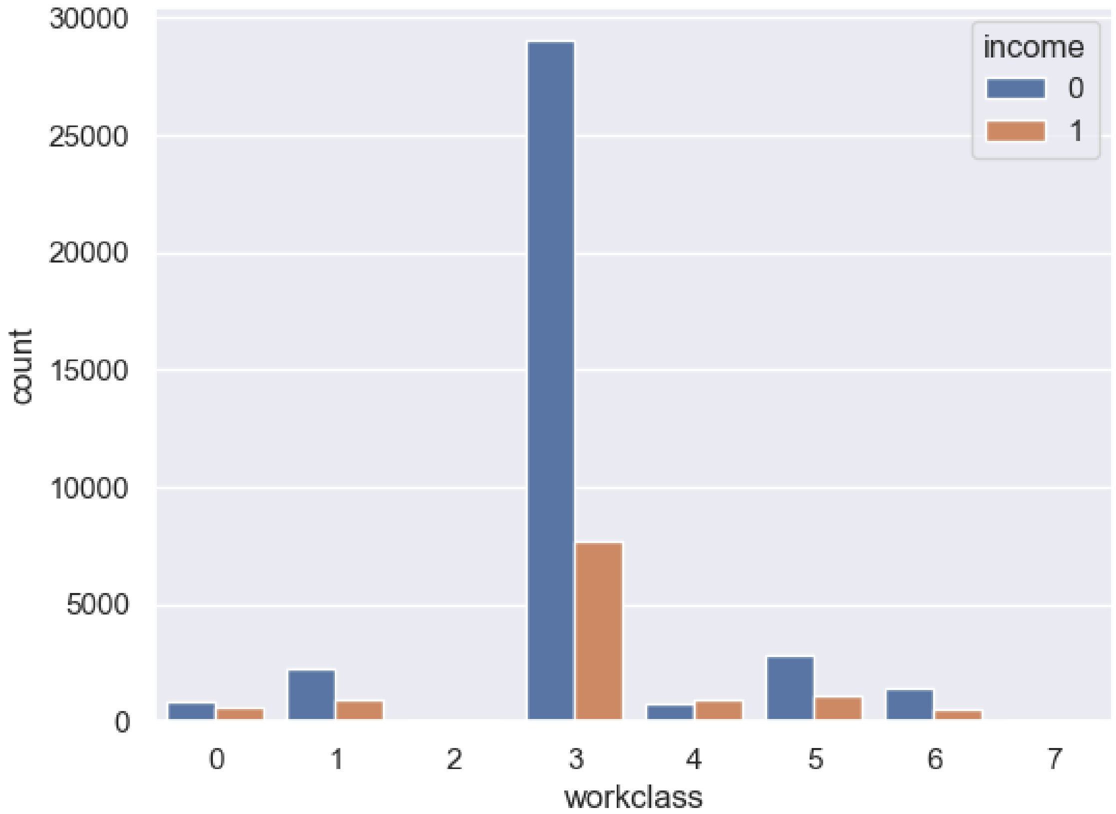
In [35]:

```
sns.histplot(x='age',hue='income', data=df)
plt.show()
```



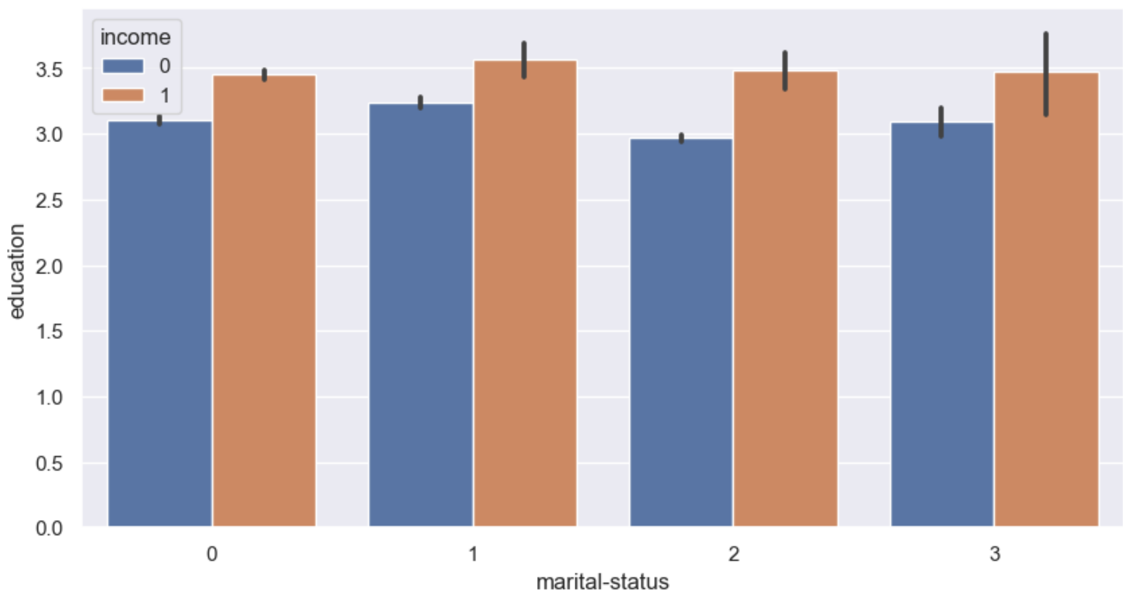
In [36]:

```
sns.countplot(x='workclass',hue='income', data=df)
plt.show()
```



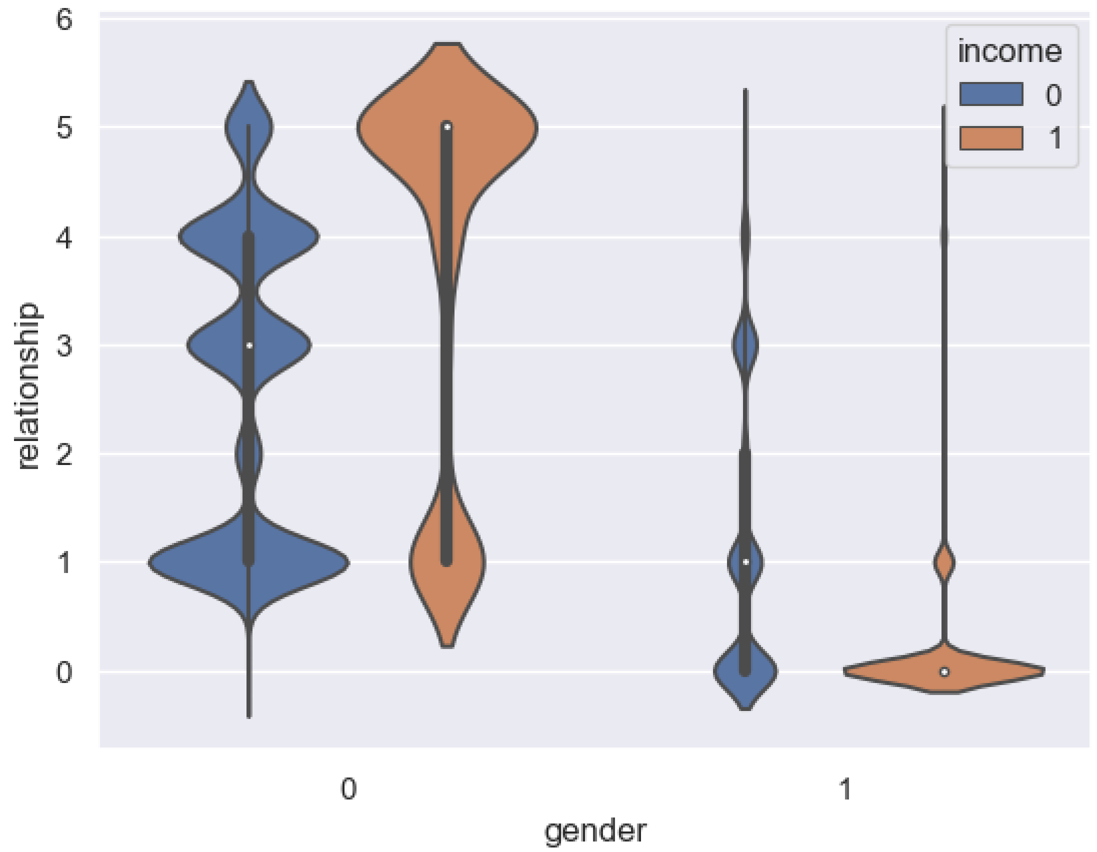
In [37]:

```
plt.figure(figsize=(10,5),dpi=100)
sns.barplot(y='education',x='marital-status',hue='income',data=df)
plt.show()
```



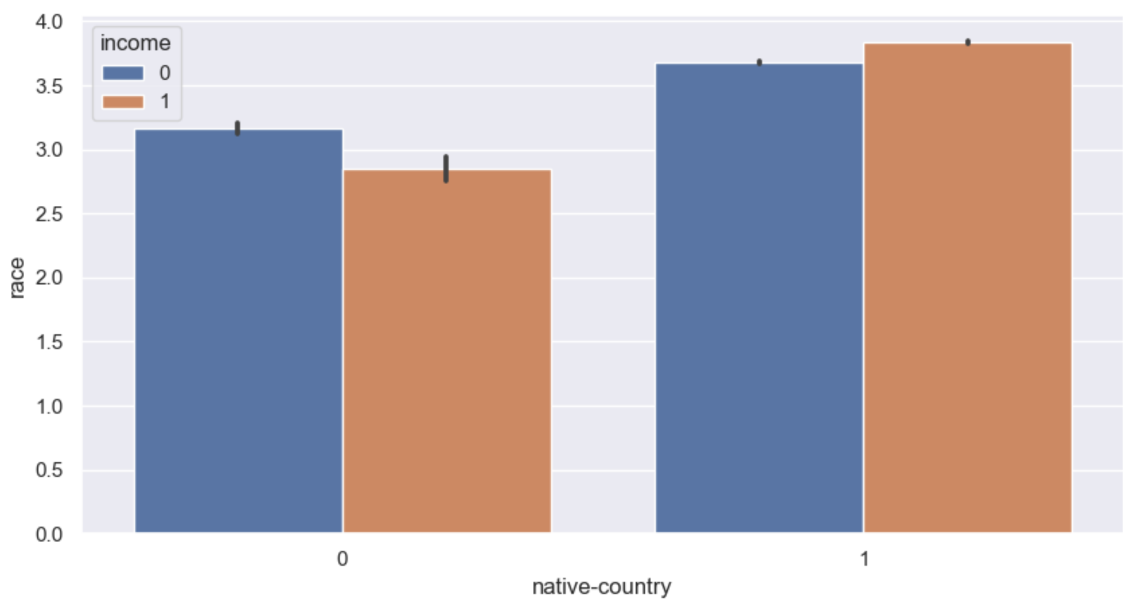
In [38]:

```
sns.violinplot(x='gender', y='relationship', hue='income', data=df)
plt.show()
```



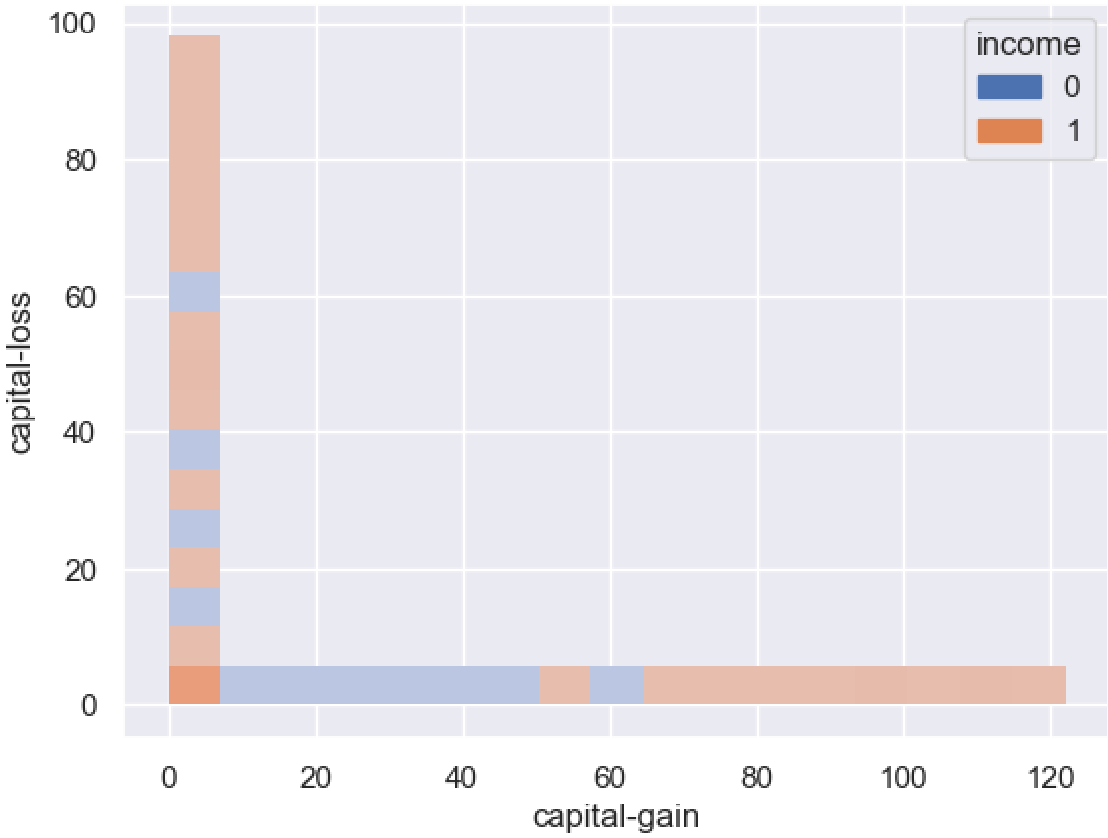
In [39]:

```
plt.figure(figsize=(10,5),dpi=100)
sns.barplot(y='race',x='native-country',hue='income',data=df)
plt.show()
```



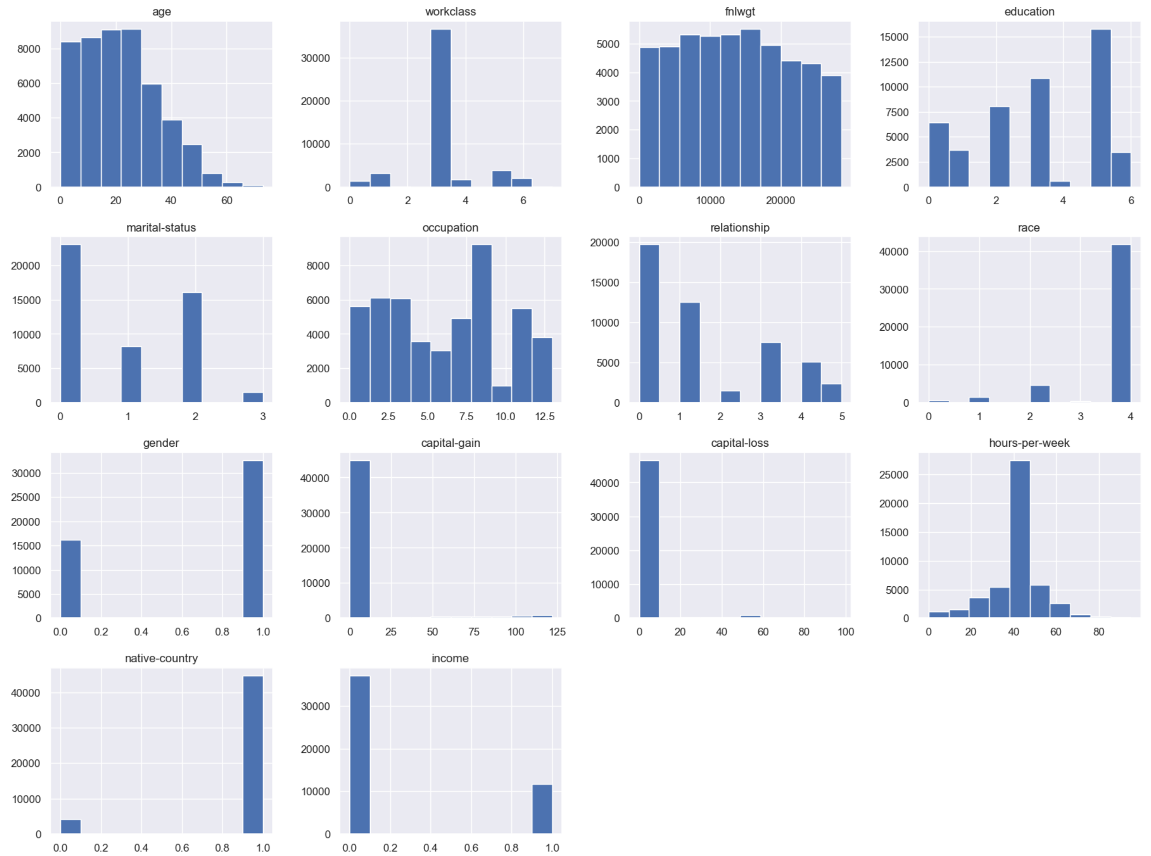
In [40]:

```
sns.histplot(x='capital-gain',y='capital-loss', hue='income', data=df)
plt.show()
```



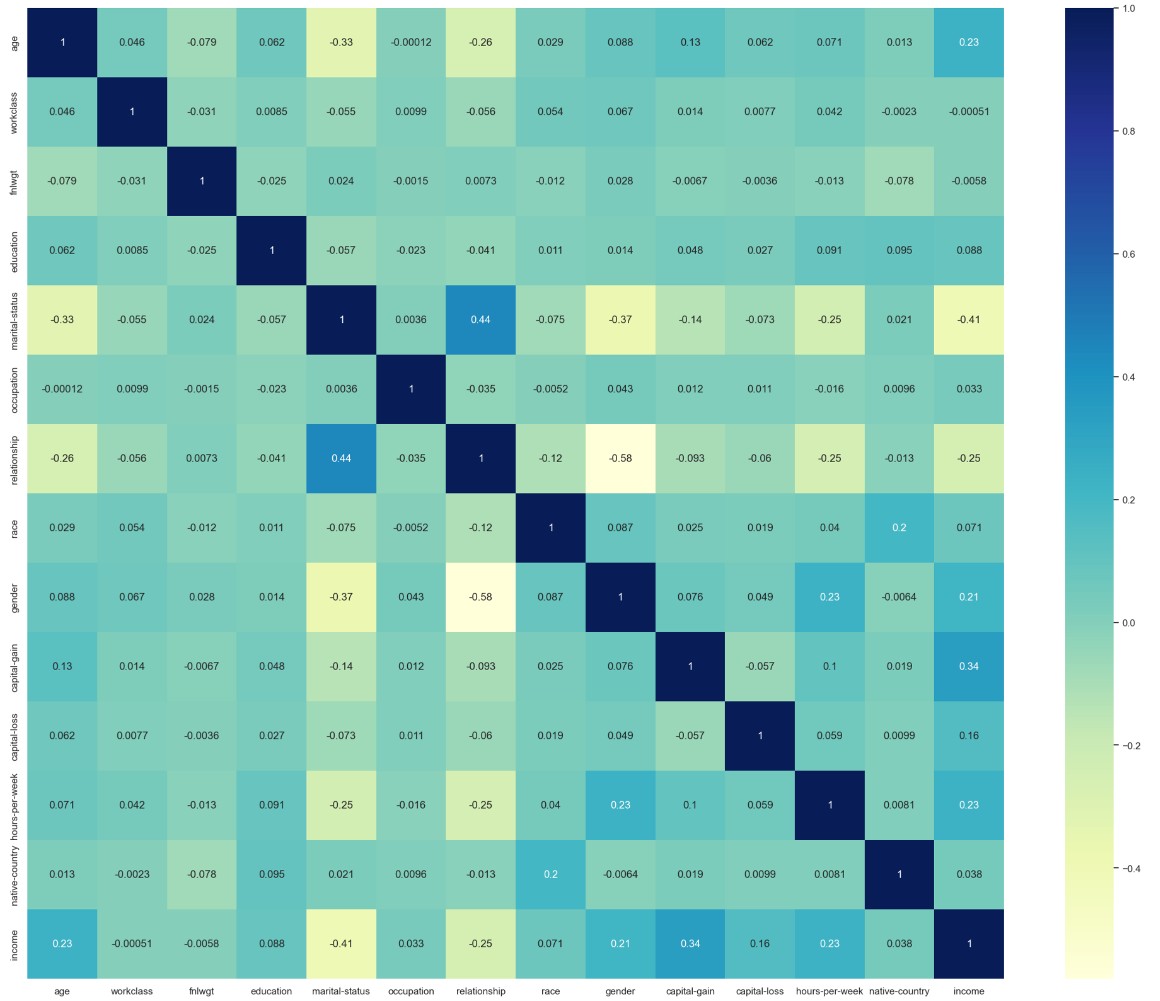
In [42]:

```
df.hist(figsize=(20,15))
plt.show()
```



In [44]:

```
plt.figure(figsize=(25,20))
sns.heatmap(df.corr(),annot=True,cmap='YlGnBu')  # Correlation by using Heatmap
plt.show()
```



Data Splitting

In [45]:

```
# split the data into independent and dependent variable
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [46]:

```
x.head(2)
```

Out[46]:

	age	workclass	fnlwgt	education	marital-status	occupation	relationship	race	gender	capita gai
0	8	3	19329	0	2	6	3	2	1	
1	21	3	4212	5	0	4	0	4	1	

In [47]:

```
y.head(2)
```

Out[47]:

```
0    0
1    0
Name: income, dtype: int64
```

Feature Scaling

- It helps in preventing features with larger magnitudes from dominating the distance calculations.

In [48]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x1=sc.fit_transform(x)
pd.DataFrame(x1).head()
```

Out[48]:

	0	1	2	3	4	5	6	7	
0	-0.995947	-0.089723	0.706349	-1.676215	1.126372	-0.038681	0.971279	-1.971227	0.7(
1	-0.047620	-0.089723	-1.196279	0.971463	-0.959154	-0.542555	-0.900732	0.392492	0.7(
2	-0.777103	-1.889345	1.462895	-1.146679	-0.959154	0.969066	-0.900732	0.392492	0.7(
3	0.390069	-0.089723	-0.316642	-0.087608	-0.959154	-0.038681	-0.900732	-1.971227	0.7(
4	-1.506585	-0.089723	-1.045373	-0.087608	1.126372	0.717129	0.971279	0.392492	-1.4

In [49]:

```
df['income'].value_counts() # data is imbalanced
```

Out[49]:

```
0    37109
1    11681
Name: income, dtype: int64
```

In [50]:

```
# Handle imbalance data
import imblearn
from imblearn.over_sampling import RandomOverSampler
ros=RandomOverSampler()
x_ovr,y_ovr=ros.fit_resample(x1,y)
print(x_ovr.shape,y_ovr.shape,y.shape)
```

```
(74218, 13) (74218,) (48790,)
```

In [51]:

```
y_ovr.value_counts() # imbalance data handled
```

Out[51]:

```
0    37109
1    37109
Name: income, dtype: int64
```

In [52]:

```
# split the data into training and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_ovr,y_ovr, test_size=0.20, random_
```

Building KNN Classifier Model

In [53]:

```
from sklearn.neighbors import KNeighborsClassifier
```

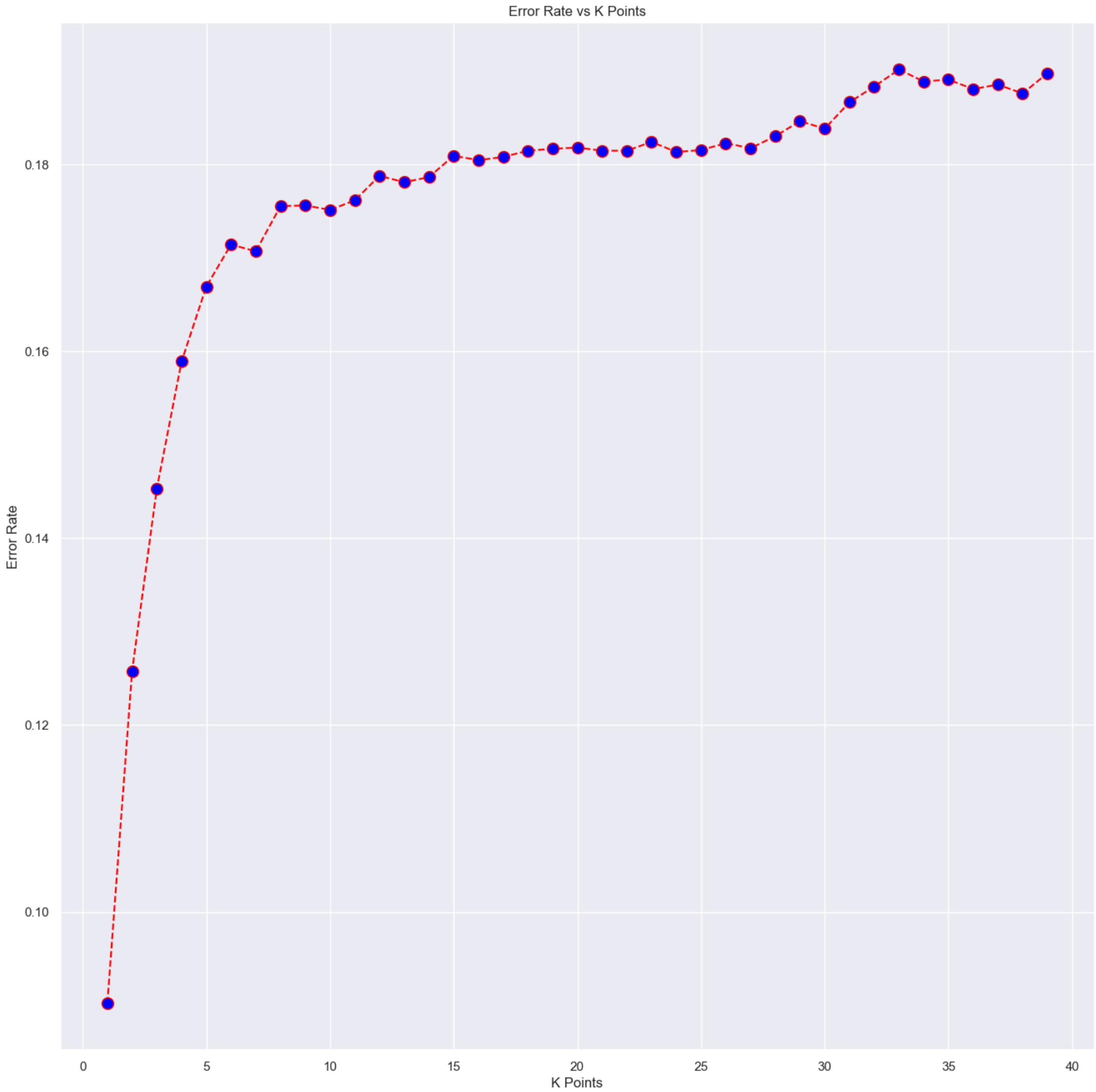
In [54]:

```
error_rate=[]

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

In [57]:

```
# Plotting the error graph
plt.figure(figsize=(16,16))
plt.plot(range(1,40), error_rate, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate vs K Points')
plt.xlabel('K Points')
plt.ylabel('Error Rate')
plt.show()
```



Building KNN Classification model using k=3

In [79]:

```
KNN = KNeighborsClassifier(n_neighbors=3)
KNN.fit(x_train, y_train)
```

Out[79]:

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [80]:

```
y_pred_train = KNN.predict(x_train)
y_pred_test = KNN.predict(x_test)
```

In [81]:

```
# Evaluate
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,prec
```

In [82]:

```
print(confusion_matrix(y_train,y_pred_train))
print(confusion_matrix(y_test,y_pred_test))
```

```
[[25721  3966]
 [  503 29184]]
[[5795 1627]
 [ 530 6892]]
```

In [83]:

```
# Evaluate train data
acc= accuracy_score(y_train,y_pred_train)
print('k=3 train data Accuracy score is',acc)
prec= precision_score(y_train,y_pred_train)
print('k=3 train data Precision score is',prec)
rec= recall_score(y_train,y_pred_train)
print('k=3 train data Recall score is',rec)
f1= f1_score(y_train,y_pred_train)
print('k=3 train data F1-Score is',f1)
```

```
k=3 train data Accuracy score is 0.9247313638966551
k=3 train data Precision score is 0.8803619909502263
k=3 train data Recall score is 0.9830565567420083
k=3 train data F1-Score is 0.9288794818339514
```

In [84]:

```
# Evaluate test data
acc= accuracy_score(y_test,y_pred_test)
print('k=3 test data Accuracy score is',acc)
prec= precision_score(y_test,y_pred_test)
print('k=3 test data Precision score is',prec)
rec= recall_score(y_test,y_pred_test)
print('k=3 test data Recall score is',rec)
f1= f1_score(y_test,y_pred_test)
print('k=3 test data F1-Score is',f1)
```

```
k=3 test data Accuracy score is 0.8546887631366209
k=3 test data Precision score is 0.8090151426223735
k=3 test data Recall score is 0.9285906763675559
k=3 test data F1-Score is 0.8646885389875164
```

In [86]:

```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(KNN, x_train, y_train, cv=10)
test_accuracy = cross_val_score(KNN, x_test, y_test, cv=10)
print("Training Accuracy after CV with k=3 :", training_accuracy.mean())
print('--------------------'*5)
print("Test Accuracy after CV with k=3:", test_accuracy.mean())
```

```
Training Accuracy after CV with k=3 : 0.8489407414883333
--------------------------------------------------------------------------
------------------------------
Test Accuracy after CV with k=3: 0.7902186737092397
```

Rebuild KNN Classification model using k=5

In [67]:

```
KNN = KNeighborsClassifier(n_neighbors=5)
KNN.fit(x_train, y_train)
```

Out[67]:

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [68]:

```
y_pred_train = KNN.predict(x_train)
y_pred_test = KNN.predict(x_test)
```

In [69]:

```
# Evaluate train data
acc= accuracy_score(y_train,y_pred_train)
print('k=5 train data Accuracy score is',acc)
prec= precision_score(y_train,y_pred_train)
print('k=5 train data Precision score is',prec)
rec= recall_score(y_train,y_pred_train)
print('k=5 train data Recall score is',rec)
f1= f1_score(y_train,y_pred_train)
print('k=5 train data F1-Score is',f1)
```

```
k=5 train data Accuracy score is 0.8868696735945026
k=5 train data Precision score is 0.8420190589636688
k=5 train data Recall score is 0.9524370936773672
k=5 train data F1-Score is 0.8938309071079709
```

In [70]:

```
# Evaluate test data
acc= accuracy_score(y_test,y_pred_test)
print('k=5 test data Accuracy score is',acc)
prec= precision_score(y_test,y_pred_test)
print('k=5 test data Precision score is',prec)
rec= recall_score(y_test,y_pred_test)
print('k=5 test data Recall score is',rec)
f1= f1_score(y_test,y_pred_test)
print('k=5 test data F1-Score is',f1)
```

```
k=5 test data Accuracy score is 0.8330638641875505
k=5 test data Precision score is 0.791991495393338
k=5 test data Recall score is 0.9033953112368633
k=5 test data F1-Score is 0.8440332326283988
```

Rebuild KNN Classification model using k=7

In [71]:

```
KNN = KNeighborsClassifier(n_neighbors=7)
KNN.fit(x_train, y_train)
```

Out[71]:

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

In [75]:

```
y_pred_train = KNN.predict(x_train)
y_pred_test = KNN.predict(x_test)
```

In [76]:

```
# Evaluate train data
acc= accuracy_score(y_train,y_pred_train)
print('k=7 train data Accuracy score is',acc)
prec= precision_score(y_train,y_pred_train)
print('k=7 train data Precision score is',prec)
rec= recall_score(y_train,y_pred_train)
print('k=7 train data Recall score is',rec)
f1= f1_score(y_train,y_pred_train)
print('k=7 train data F1-Score is',f1)
```

```
k=7 train data Accuracy score is 0.865766160272173
k=7 train data Precision score is 0.8238732961493721
k=7 train data Recall score is 0.930440933742042
k=7 train data F1-Score is 0.8739203341032049
```

In [77]:

```
# Evaluate test data
acc= accuracy_score(y_test,y_pred_test)
print('k=7 test data Accuracy score is',acc)
prec= precision_score(y_test,y_pred_test)
print('k=7 test data Precision score is',prec)
rec= recall_score(y_test,y_pred_test)
print('k=7 test data Recall score is',rec)
f1= f1_score(y_test,y_pred_test)
print('k=7 test data F1-Score is',f1)
```

```
k=7 test data Accuracy score is 0.8292912961465913
k=7 test data Precision score is 0.7912297426120114
k=7 test data Recall score is 0.8946375639989221
k=7 test data F1-Score is 0.8397622359934235
```

Conclusion

- In this project, I build a KNN classifier model to explore the possibility in predicting income level based on the individual’s personal information.
- The model yields good performance as indicated by the model accuracy which was found to be 92% for train data and 85% for test data with k=3.
- After cross validation of train data accuracy was at 84% and test data at 79% which is above the threshold value of 75%.
- After rebuilding the model with k=5 & k=7 I got less accuracy for both train and test data, making k=3 the most suitable one.

In [ ]:

```

```