Introduction to Data Science

Problem Statement:-

Applying ML Classification algorithms on the data set and getting inferences from the data. You may use the appropriate ML algorithm packages available in R or Python. But know which algorithm you use and know the concept behind it.

Group Members:-

1). Akshat Gupta(20ucc013)
2). Aryan Shrivastava(20ucc023)
3). Saurabh Goyal(20ucs175)
4). Sunny Saxena(20ucs205)

Dataset:

This data was extracted from the census bureau database found at
http://www.census.gov/ftp/pub/DES/www/welcome.html.
48842 instances, a mix of continuous and discrete (train=32561, test=16281),
45222 if instances with unknown values are removed (train=30162, test=15060
Prediction task is to determine whether a person makes over 50K a year.

Dataset Source: https://archive.ics.uci.edu

Data Preprocessing:- It is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, lacking in certain behaviours or trends, and likely to contain many errors. Data preprocessing is a proven method of resolving such issues.

It mainly involves six steps:-

1. **Getting Dataset**
2. **Importing Libraries**
3. **Importing Dataset**
4. **Finding Missing Value**
5. **Encoding Categorical Data**
6. **Splitting Dataset into Training and Test Set**

1. **Getting Dataset:-** As stated above dataset for adult incomes is taken from archive.ics.uci.edu.
2. **Importing Necessary Libraries:-**

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pycountry
import plotly.express as px
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn import decomposition
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
import missingno as mn
```

3. Importing Dataset:-

In [2]:
```
Names_of_columns = ['Age', 'Workclass', 'fnlwgt', 'Education', 'Education-num', 'Marital-status', 'occupation',
                    'relationship', 'race', 'sex', 'capital-gain','capital-loss', 'hours-per-week', 'native-country', 'income']
training_data = pd.read_csv('adult.csv', names = Names_of_columns)
test_data = pd.read_csv('test.csv',names=Names_of_columns,  skiprows=1)
dataset = pd.concat([test_data,training_data])
dataset
```

Out[2]:

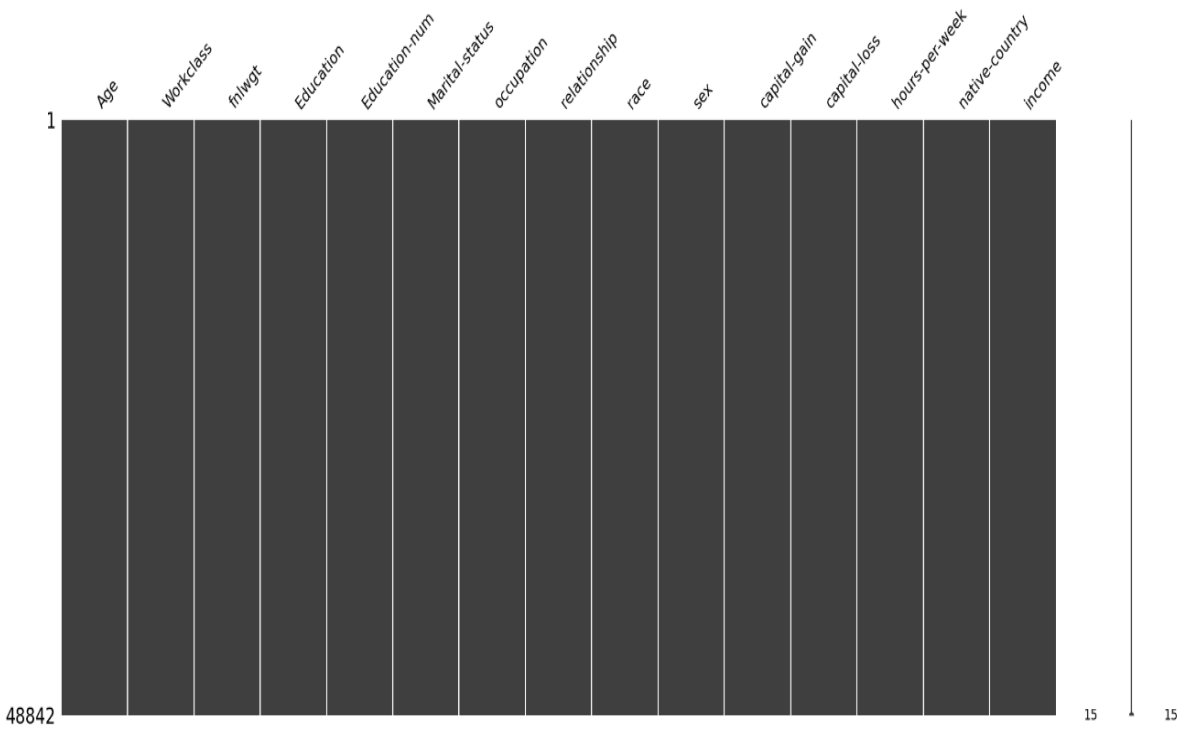
	Age	Workclass	fnlwgt	Education	Education-num	Marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K.
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K.
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K.
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K.
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

48842 rows × 15 columns

4. Finding Missing Values:-

Checking for Missing Values

```
In [9]: mn.matrix(dataset)
Out[9]: <AxesSubplot:>
```




```
In [10]: dataset.isnull().values.any()
Out[10]: False
```

It can be seen there are no missing values in the dataset.

Detailed Analysis of Dataset:-

- **Information about the dataset, including the index dtype and columns, non-values, and memory usage.**

```
In [3]: dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48842 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             48842 non-null  int64
 1   Workclass       48842 non-null  object
 2   fnlwgt          48842 non-null  int64
 3   Education       48842 non-null  object
 4   Education-num   48842 non-null  int64
 5   Marital-status  48842 non-null  object
 6   occupation      48842 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             48842 non-null  object
 10  capital-gain    48842 non-null  int64
 11  capital-loss    48842 non-null  int64
 12  hours-per-week  48842 non-null  int64
 13  native-country  48842 non-null  object
 14  income          48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 6.0+ MB
```

- **The descriptive statistics summarize the central tendency, dispersion, and shape of the dataset's distribution.**

```
In [7]: dataset.describe()
```

Out[7]:

	Age	fnlwgt	Education-num	capital-gain	capital-loss	hours-per-week
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.422382
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.391444
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.000000

- **Correlation of Attributes**

```
In [8]: dataset.corr()
```

Out[8]:

	Age	fnlwgt	Education-num	capital-gain	capital-loss	hours-per-week
Age	1.000000	-0.076628	0.030940	0.077229	0.056944	0.071558
fnlwgt	-0.076628	1.000000	-0.038761	-0.003706	-0.004366	-0.013519
Education-num	0.030940	-0.038761	1.000000	0.125146	0.080972	0.143689
capital-gain	0.077229	-0.003706	0.125146	1.000000	-0.031441	0.082157
capital-loss	0.056944	-0.004366	0.080972	-0.031441	1.000000	0.054467
hours-per-week	0.071558	-0.013519	0.143689	0.082157	0.054467	1.000000

- **Removing Duplicates in the data**

Dropping Duplicates

In [12]:
```
dataset.drop_duplicates(keep = 'last',inplace = True)
dataset
```

Out[12]:

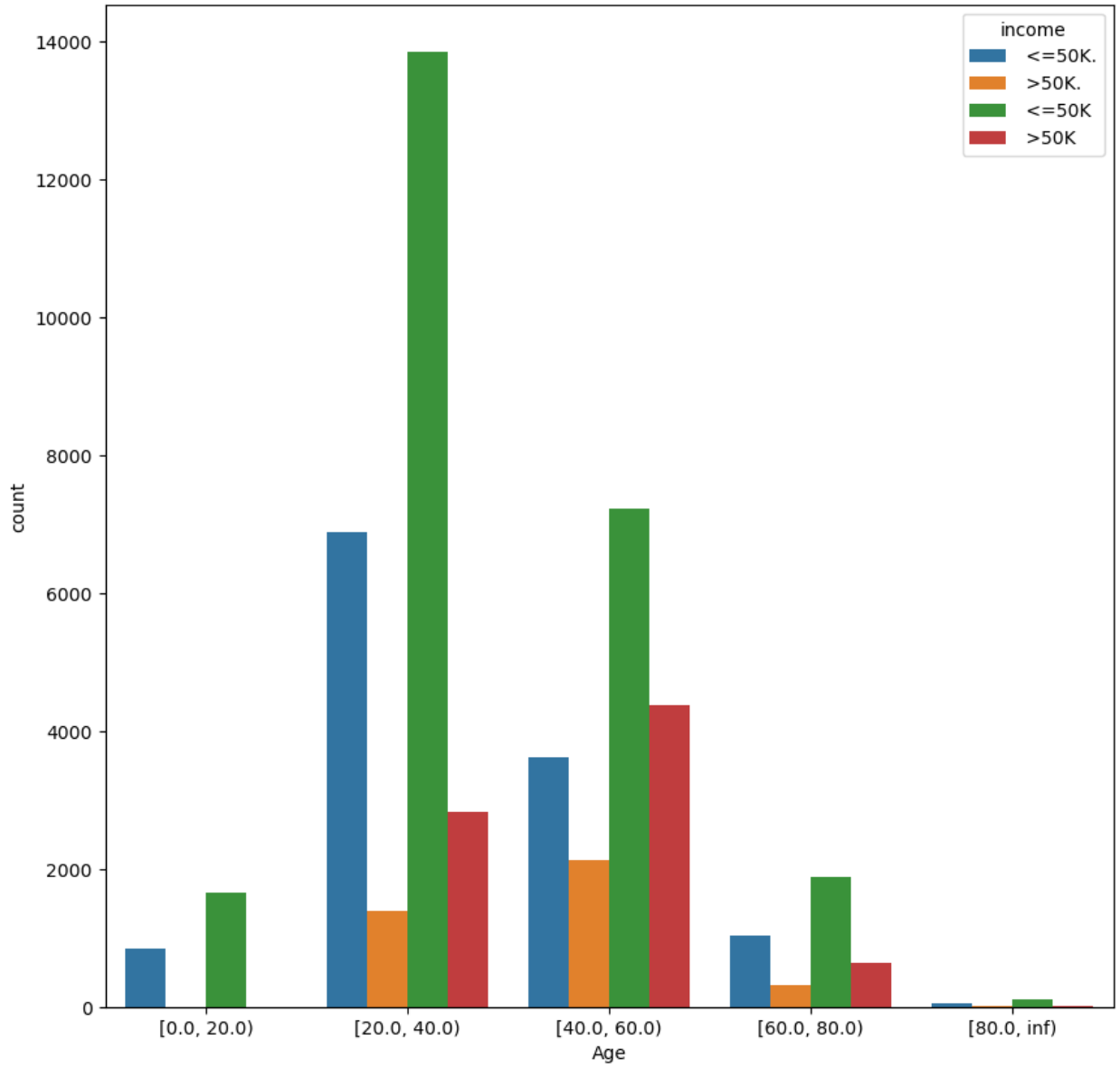
	Age	Workclass	fnlwgt	Education	Education-num	Marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K.
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K.
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K.
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K.
4	18	Private	103497	Some-college	10	Never-married	Prof-specialty	Own-child	White	Female	0	0	30	United-States	<=50K.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

48813 rows × 15 columns

Visual Analysis of the dataset:

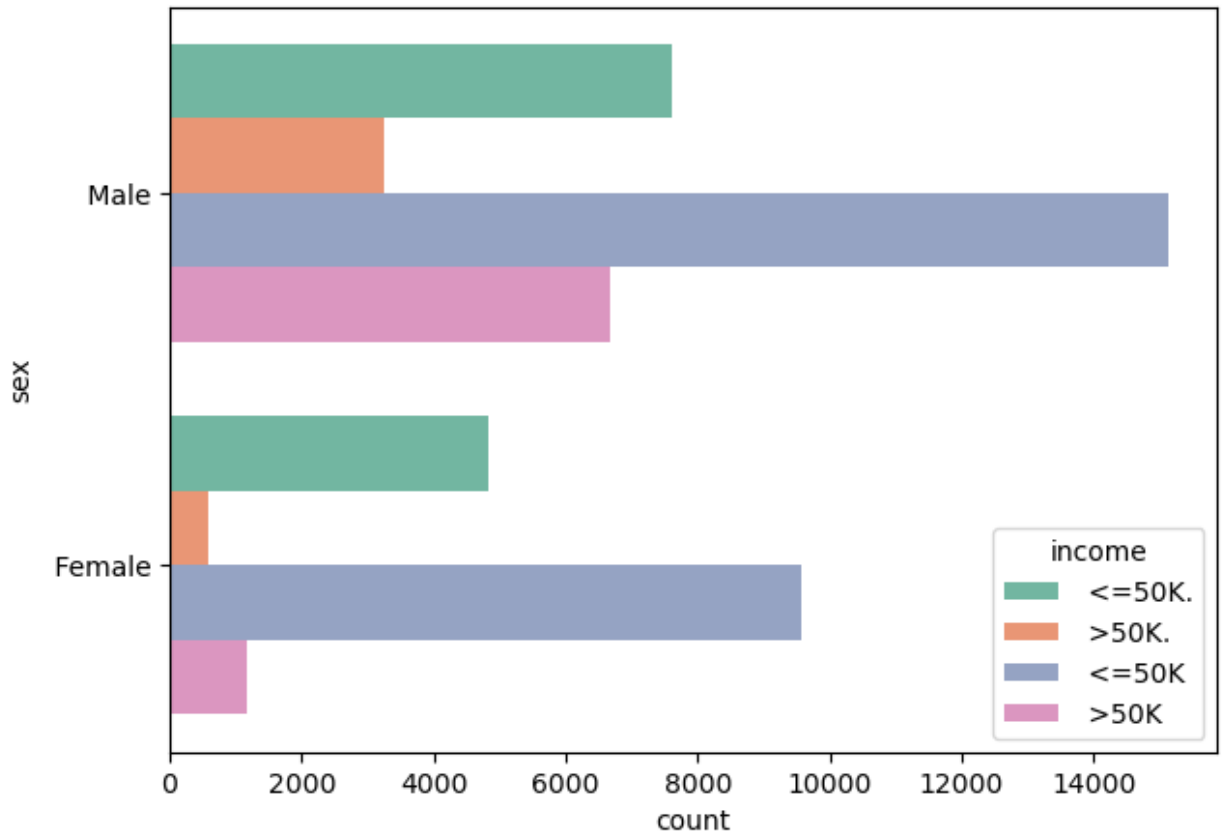
Here we are visually analyzing, with the help of certain graphs, the distribution of income based on various attributes.

- **Distribution of income based on Age** -

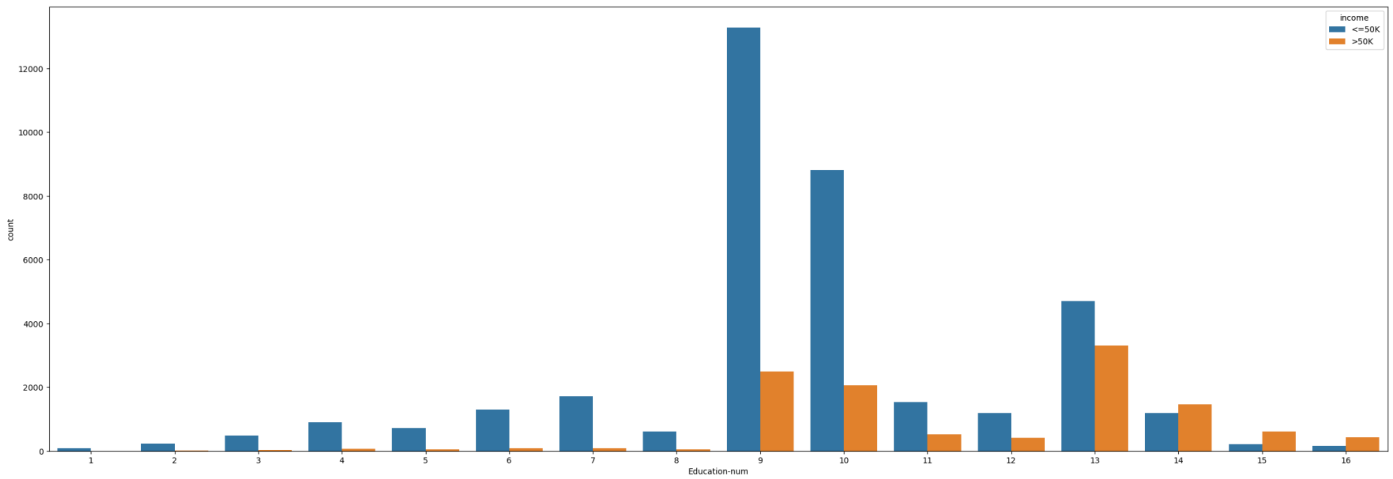


-

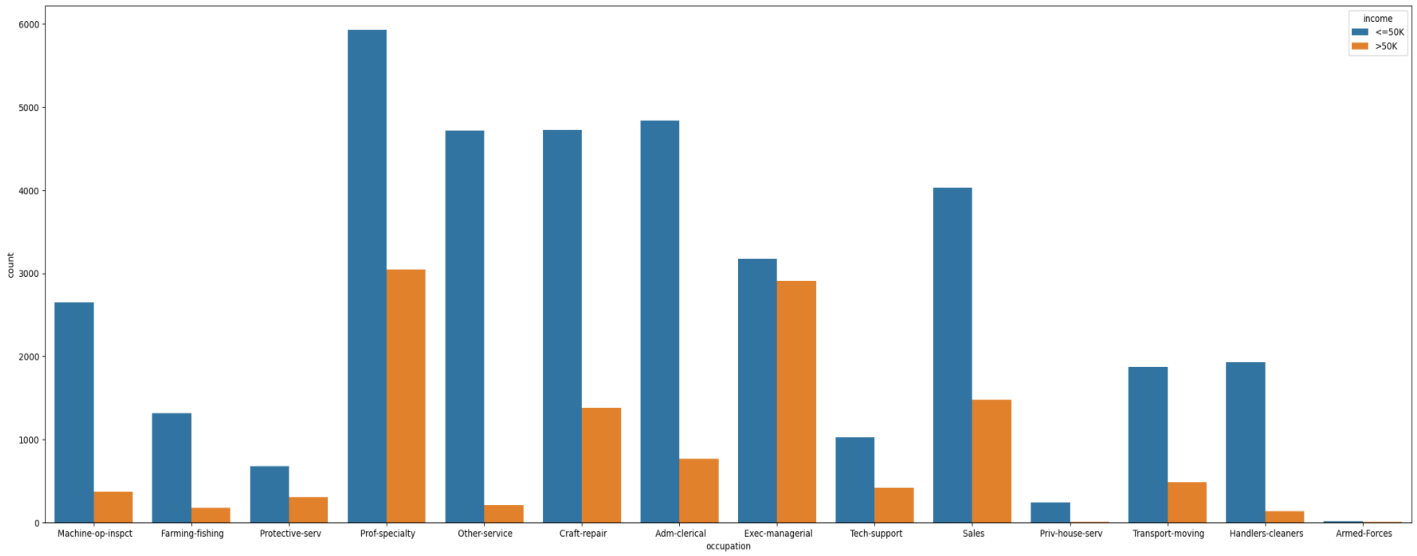
- **Distribution of Income based on Gender**:



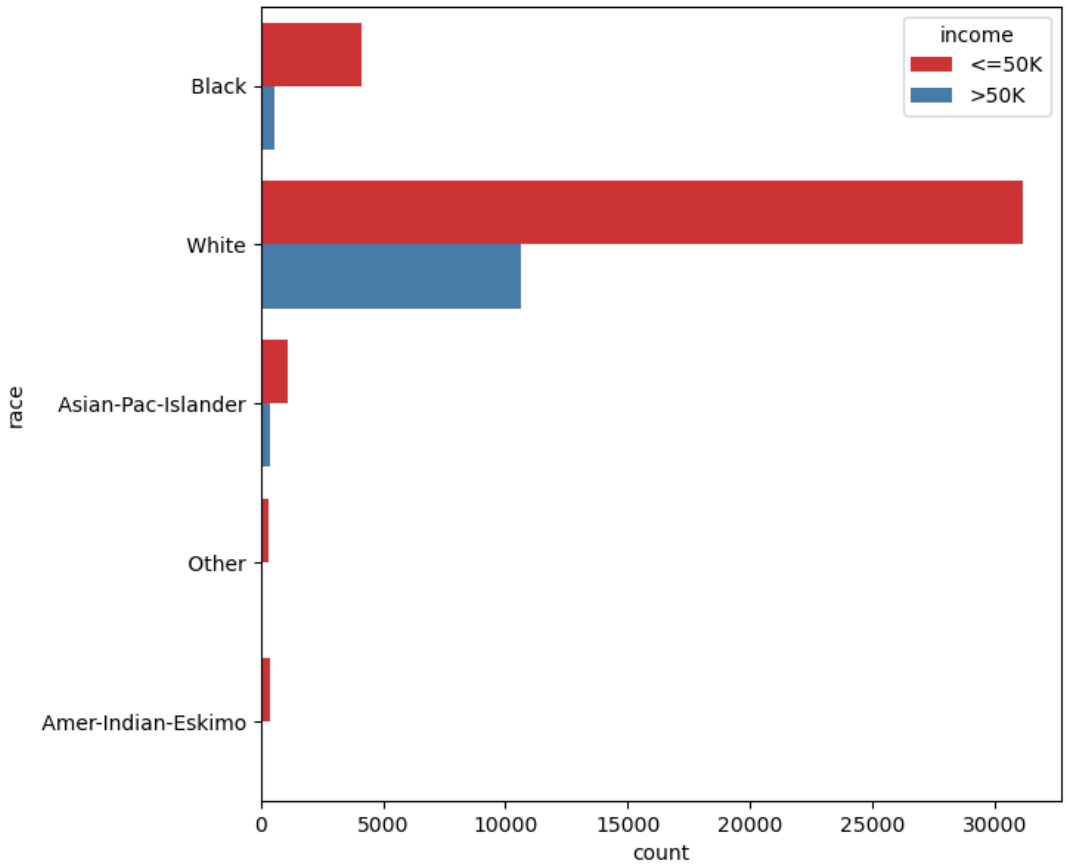
- **Income based on Education-num**:



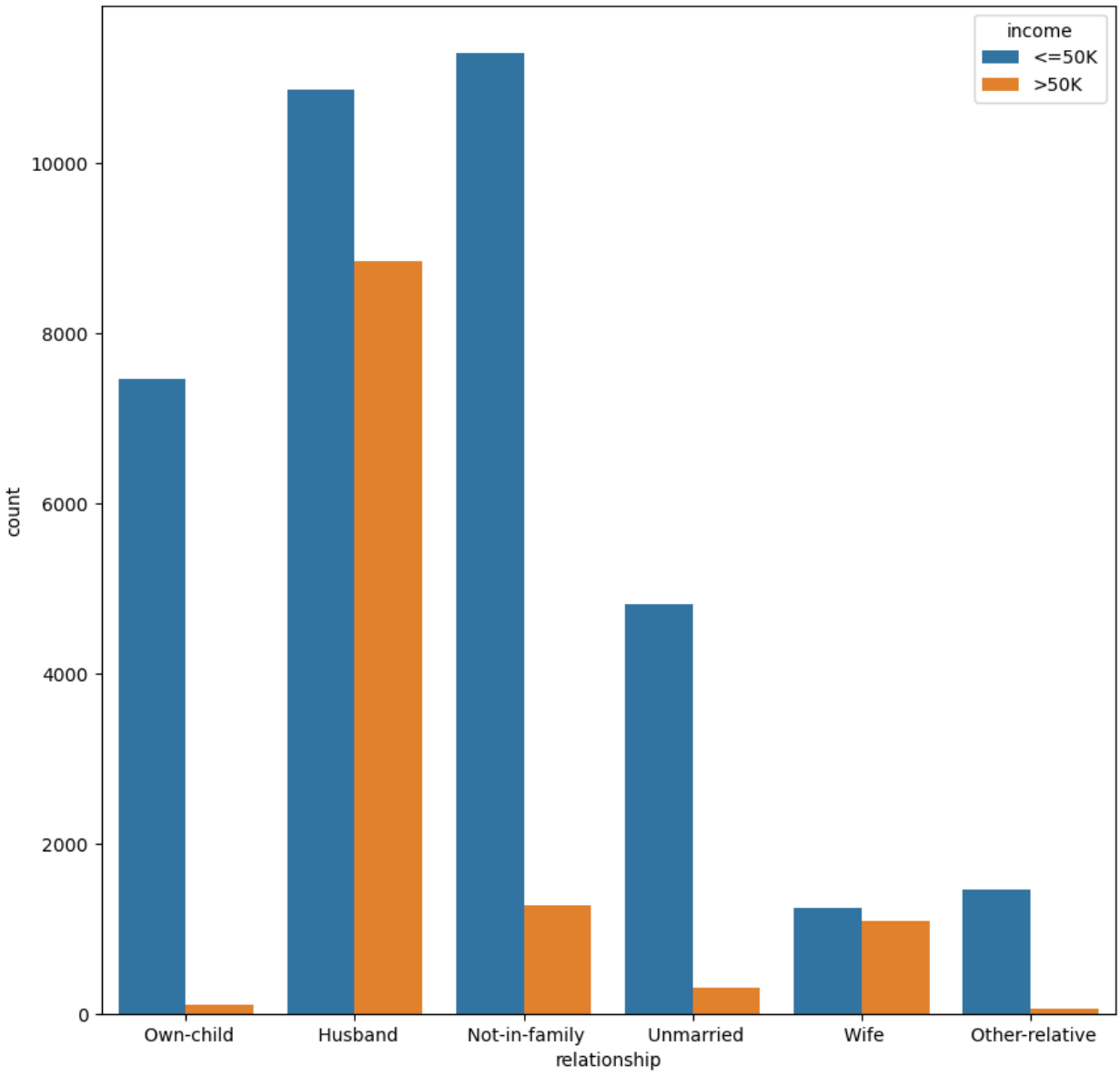
- **Income Based on Occupation** :



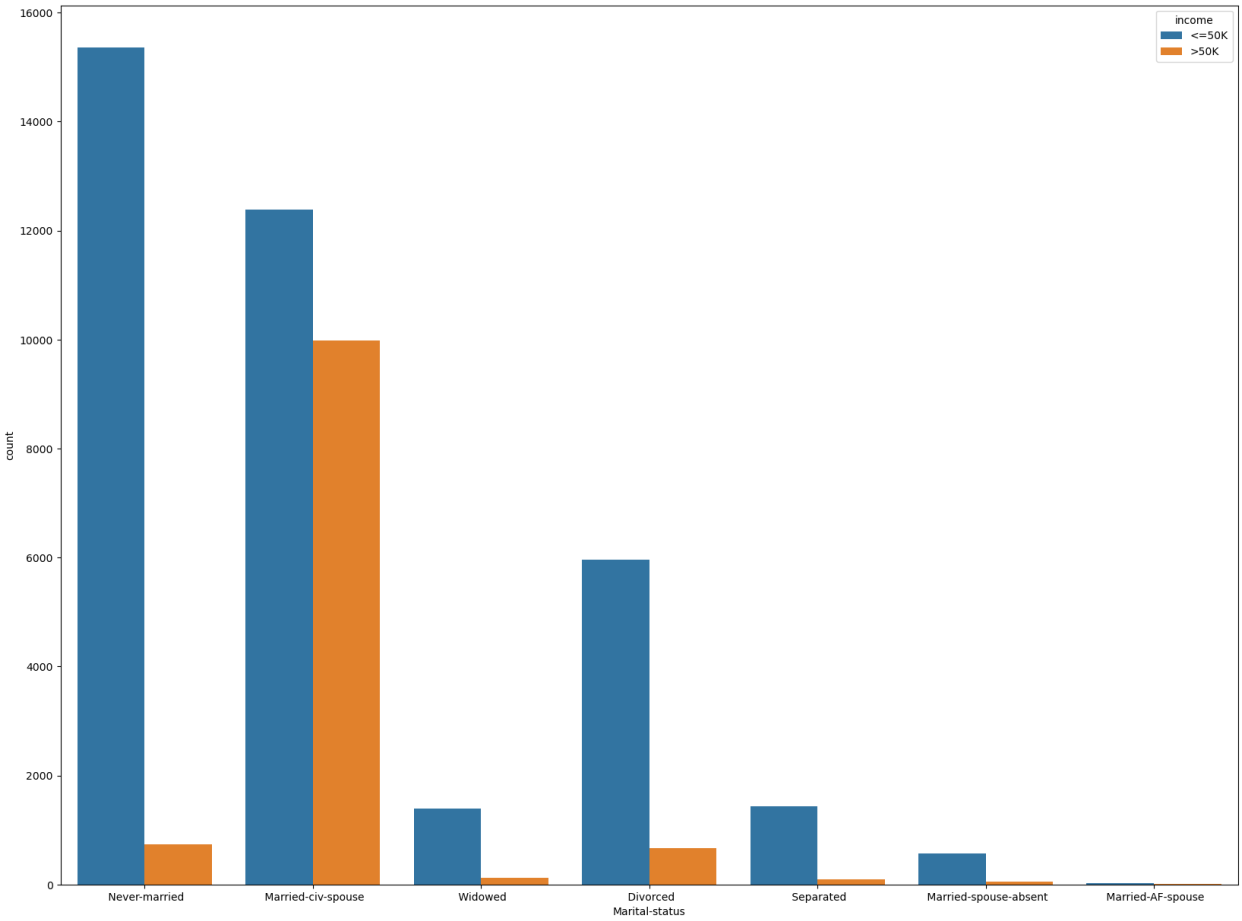
- **Income based on Race** :



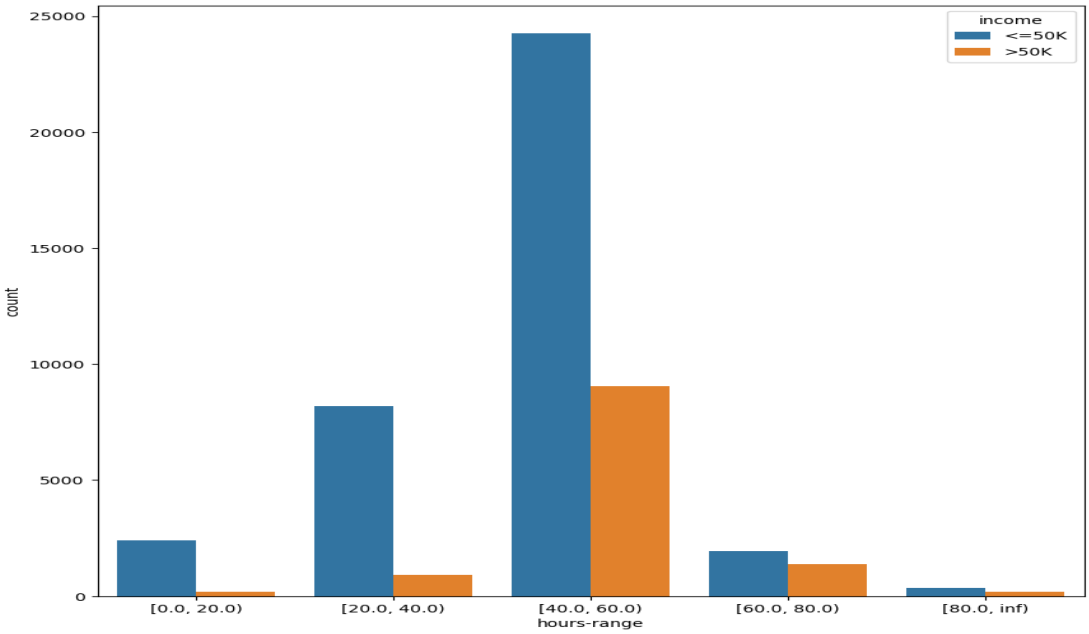
- **Income based on Relationship**:



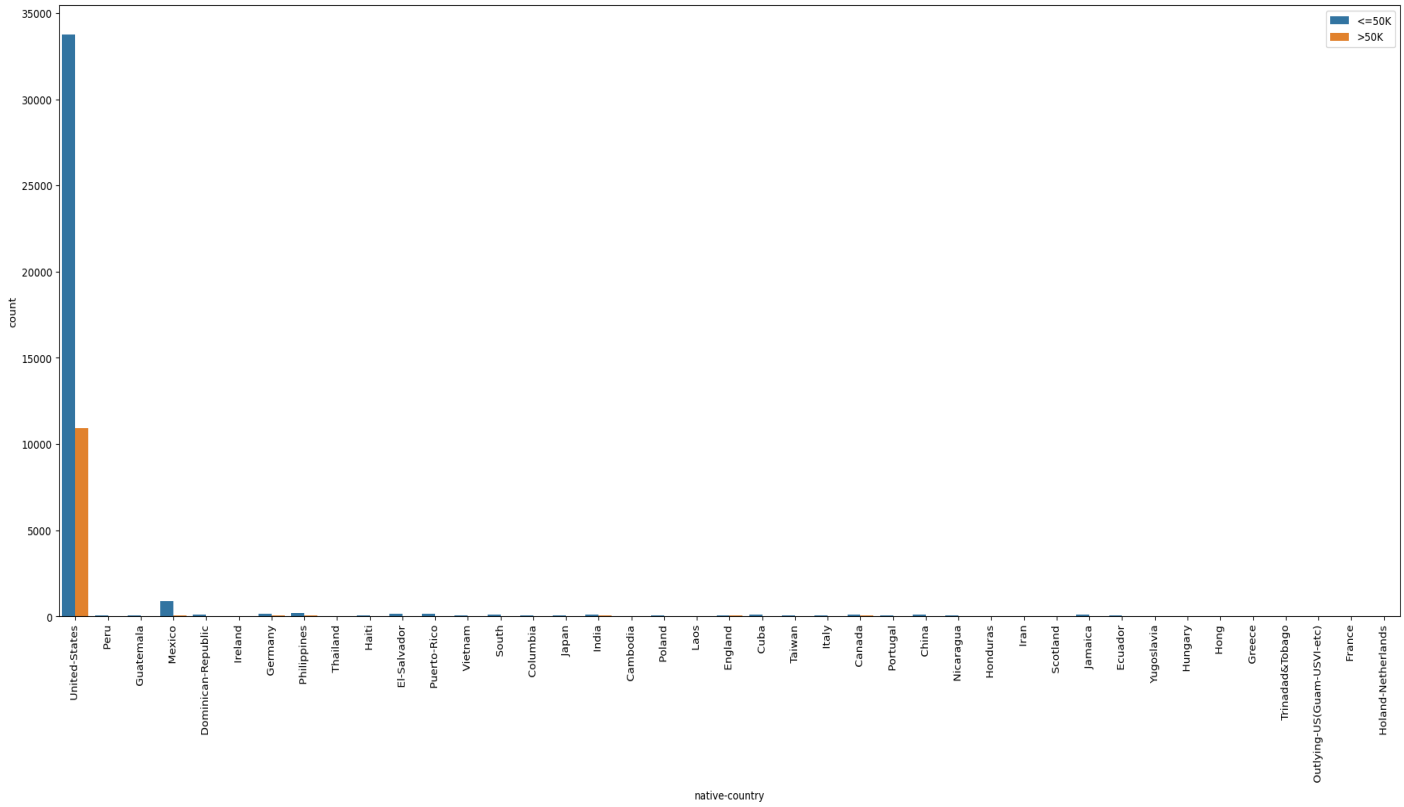
- **Income Based on Marital Status**:



- **Income based on Hours Per-Week**:



- **Income Based on Native Country**:



HEATMAP: (for analysing the correlation between Attributes)

- There is not a significant correlation between numerical features.
- Also, the correlation of fnlwgt gives a negative value for each column. Therefore it can be dropped.
- There is some correlation between education-num and hours-per-week, education-num and capital gain.

5. Encoding Categorical Data:

In [71]:
```
label_encoder = preprocessing.LabelEncoder()

dataset.drop(['hours-range', 'age'], axis=1, inplace = True)

columns = ['workclass', 'fnlwgt', 'education', 'marital-status', 'occupation',
           'relationship', 'race', 'sex', 'native-country']

for column in columns:
    dataset[column]= label_encoder.fit_transform(dataset[column])

dataset
```

Feature Selection

Feature Selection

In [72]:
```
scaler = StandardScaler()
features = dataset.drop(['income', 'fnlwgt', 'education'],axis = 1)
target = dataset['income']
scaled_data = scaler.fit_transform(features)
stand_data = scaled_data
```

Oversampling Data

In [73]:
```
smote = SMOTE(sampling_strategy= 0.7,random_state = 22)

x_smote, y_smote = smote.fit_resample(stand_data,target)

print('Original dataset shape', Counter(target))
print('Resample dataset shape', Counter(y_smote))
```

```
Original dataset shape Counter({'<=50K': 37108, '>50K': 11681})
Resample dataset shape Counter({'<=50K': 37108, '>50K': 25975})
```

- It can be seen in the output that oversampling the data counter for values above 50k has increased significantly.
- A popular technique for oversampling is being used here, i.e. SMOTE(Synthetic Minority Oversampling Technique). This creates synthetic samples by randomly sampling the characteristics from occurrences in the minority class.

6. Splitting Dataset into Training and Test Set:

```
In [74]: x_train, x_test, y_train, y_test = train_test_split(x_smote, y_smote, test_size = 0.3, random_state = 22)
```

ML Classification Algorithms:-

- **Decision Tree**
- **Random Forest**
- **Support Vector Machine**
- **Logistic Regression**
- **K-Nearest Neighbour**

1. Decision Tree:-

Variable selection criterion here can be done via two approaches:

1. **Entropy**:- In the context of Decision Trees, entropy is a measure of disorder or impurity in a node. Maximum entropy is given by one and minimum entropy by value 0.
2. **Gini Index**:- The Gini Index or impurity measures the probability of a random instance being misclassified when chosen randomly.The lower the Gini Index, the better, and the lower the likelihood of misclassification.

Gini:

In [76]:
```
dt_gini_model = DecisionTreeClassifier(criterion='gini',max_depth=5, random_state=22)
dt_gini_model.fit(x_train, y_train)

dt_gini_train_pred = dt_gini_model.predict(x_train)
dt_gini_test_pred = dt_gini_model.predict(x_test)

print(f'Train score: {accuracy_score(y_train, dt_gini_train_pred) *100:.2f}%')
print(f'Test score: {accuracy_score(y_test, dt_gini_test_pred) *100:.2f}%')

dt_gini_eval = evaluate(y_test, dt_gini_test_pred)
dt_gini_df = round(pd.DataFrame([dt_gini_eval], index = ['Decision Tree(Gini)']),4)
display(dt_gini_df)
```

```
Train score: 80.57%
Test score: 80.35%
```

	accuracy	precision	recall	f_measure	error_rate
Decision Tree(Gini)	0.8035	0.784	0.7207	0.751	0.1965

In [77]:
```
print(classification_report(y_test, dt_gini_test_pred))
```

```
              precision    recall  f1-score   support

       <=50K       0.82      0.86      0.84     11144
        >50K       0.78      0.72      0.75      7781

    accuracy                           0.80     18925
   macro avg       0.80      0.79      0.79     18925
weighted avg       0.80      0.80      0.80     18925
```

Entropy:

In [78]:
```
dt_entropy_model = DecisionTreeClassifier(criterion='entropy',max_depth=5, random_state=22)
dt_entropy_model.fit(x_train, y_train)

dt_entropy_train_pred = dt_entropy_model.predict(x_train)
dt_entropy_test_pred = dt_entropy_model.predict(x_test)

print(f'Train score: {accuracy_score(y_train, dt_entropy_train_pred) *100:.2f}%')
print(f'Test score: {accuracy_score(y_test, dt_entropy_test_pred) *100:.2f}%')

dt_entropy_eval = evaluate(y_test, dt_entropy_test_pred)
dt_entropy_df = round(pd.DataFrame([dt_entropy_eval], index = ['Decision Tree(Entropy)']),4)
display(dt_entropy_df)
```

```
Train score: 79.32%
Test score: 78.99%
```

	accuracy	precision	recall	f_measure	error_rate
Decision Tree(Entropy)	0.7899	0.7002	0.8549	0.7699	0.2101

```
In [95]: print(classification_report(y_test, dt_entropy_test_pred))
```

```
                precision    recall  f1-score   support

         <=50K       0.88      0.74      0.81     11144
          >50K       0.70      0.85      0.77      7781

      accuracy                           0.79     18925
     macro avg       0.79      0.80      0.79     18925
  weighted avg       0.81      0.79      0.79     18925
```

2. Random Forest:-

Gini :

```
In [79]: clf = RandomForestClassifier(criterion='gini',n_estimators = 200, max_depth=6, random_state = 22)
clf.fit(x_train, y_train)

rf_gini_train_pred = clf.predict(x_train)
rf_gini_test_pred = clf.predict(x_test)

print(f'Train score: {accuracy_score(y_train, rf_gini_train_pred)*100:.2f}%')
print(f'Test score: {accuracy_score(y_test, rf_gini_test_pred)*100:.2f}%')

rf_gini_eval = evaluate(y_test, rf_gini_test_pred)
rf_gini_df = round(pd.DataFrame([rf_gini_eval], index = ['Random Forest(Gini)']),4)
display(rf_gini_df)
```

```
Train score: 82.06%
Test score: 81.89%
```

	accuracy	precision	recall	f_measure	error_rate
Random Forest(Gini)	0.8189	0.7669	0.8039	0.785	0.1811

```
In [80]: print(classification_report(y_test, rf_gini_test_pred))
```

```
                precision    recall  f1-score   support

         <=50K       0.86      0.83      0.84     11144
          >50K       0.77      0.80      0.78      7781

      accuracy                           0.82     18925
     macro avg       0.81      0.82      0.81     18925
  weighted avg       0.82      0.82      0.82     18925
```

Entropy:

In [81]:
```
clf = RandomForestClassifier(criterion='entropy',n_estimators = 200, max_depth=6, random_state = 22)
clf.fit(x_train, y_train)

rf_entropy_train_pred = clf.predict(x_train)
rf_entropy_test_pred = clf.predict(x_test)

print(f'Train score: {accuracy_score(y_train, rf_entropy_train_pred)*100:.2f}%')
print(f'Test score: {accuracy_score(y_test, rf_entropy_test_pred)*100:.2f}%')

rf_entropy_eval = evaluate(y_test, rf_entropy_test_pred)
rf_entropy_df = round(pd.DataFrame([rf_entropy_eval], index = ['Random Forest(Entropy)']),4)
display(rf_entropy_df)
```

```
Train score: 81.89%
Test score: 81.73%
```

	accuracy	precision	recall	f_measure	error_rate
Random Forest(Entropy)	0.8173	0.761	0.8101	0.7847	0.1827

In [82]:
```
print(classification_report(y_test, rf_entropy_test_pred))
```

```
              precision    recall  f1-score   support

       <=50K       0.86      0.82      0.84     11144
        >50K       0.76      0.81      0.78      7781

    accuracy                           0.82     18925
   macro avg       0.81      0.82      0.81     18925
weighted avg       0.82      0.82      0.82     18925
```

3. SVM:-

In [83]:
```
svm_model = SVC()
svm_model.fit(x_train, y_train)

svm_train_pred = svm_model.predict(x_train)
svm_test_pred = svm_model.predict(x_test)

print(f'Train score: {accuracy_score(y_train, svm_train_pred)*100:.2f}%')
print(f'Test score: {accuracy_score(y_test, svm_test_pred)*100:.2f}%')

svm_eval = evaluate(y_test, svm_test_pred)
svm_df = round(pd.DataFrame([svm_eval], index = ['SVM']),4)
display(svm_df)
```

```
Train score: 81.38%
Test score: 80.76%
```

	accuracy	precision	recall	f_measure	error_rate
SVM	0.8076	0.7379	0.825	0.779	0.1924

```
In [84]: print(classification_report(y_test,svm_test_pred))
```

	precision	recall	f1-score	support
<=50K	0.87	0.80	0.83	11144
>50K	0.74	0.82	0.78	7781
accuracy			0.81	18925
macro avg	0.80	0.81	0.80	18925
weighted avg	0.81	0.81	0.81	18925

4. Logistic Regression:-

- Multinomial Regression:- It is used to predict a nominal dependent variable given one or more independent variables. It has certain assumptions.
  A. The dependent variable should be measured at the nominal level.
  B. Independent variables should be continuous, ordinal or nominal.
  C. Dependent variables should have mutually exclusive and exhaustive categories.
  D. No multicollinearity
  E. No outliers, high leverage values or highly influential values.

In [85]:
```
lr = LogisticRegression(multi_class = 'multinomial', max_iter = 200)
lr.fit(x_train, y_train)

lr_multin_train_pred = lr.predict(x_train)
lr_multin_test_pred = lr.predict(x_test)

print(f'Train score: {accuracy_score(y_train, lr_multin_train_pred)*100:.2f}%')
print(f'Test score: {accuracy_score(y_test, lr_multin_test_pred)*100:.2f}%')

lr_multin_eval = evaluate(y_test, lr_multin_test_pred)
lr_multin_df = round(pd.DataFrame([lr_multin_eval], index = ['Logistic Regression(multinomial)']),4)
display(lr_multin_df)
```

```
Train score: 76.12%
Test score: 76.17%
```

	accuracy	precision	recall	f_measure	error_rate
Logistic Regression(multinomial)	0.7617	0.7483	0.6336	0.6862	0.2383

In [86]:
```
print(classification_report(y_test,lr_multin_test_pred))
```

```
              precision    recall  f1-score   support

       <=50K       0.77      0.85      0.81     11144
        >50K       0.75      0.63      0.69      7781

    accuracy                           0.76     18925
   macro avg       0.76      0.74      0.75     18925
weighted avg       0.76      0.76      0.76     18925
```

- OVR(One v/s Rest):-
  In this logistic regression, a separate model is trained for each class predicted whether an observation is that class or not (making it binary classification). It assumes that each classification problem is independent.

In [87]:
```
lr = LogisticRegression(multi_class = 'ovr', max_iter = 200)
lr.fit(x_train, y_train)

lr_ovr_train_pred = lr.predict(x_train)
lr_ovr_test_pred = lr.predict(x_test)

print(f'Train score: {accuracy_score(y_train, lr_ovr_train_pred)*100:.2f}%')
print(f'Test score: {accuracy_score(y_test, lr_ovr_test_pred)*100:.2f}%')

lr_ovr_eval = evaluate(y_test, lr_ovr_test_pred)
lr_ovr_df = round(pd.DataFrame([lr_ovr_eval], index = ['Logistic Regression(ovr)']),4)
display(lr_ovr_df)
```

```
Train score: 76.12%
Test score: 76.18%
```

	accuracy	precision	recall	f_measure	error_rate
Logistic Regression(ovr)	0.7618	0.7484	0.6336	0.6862	0.2382

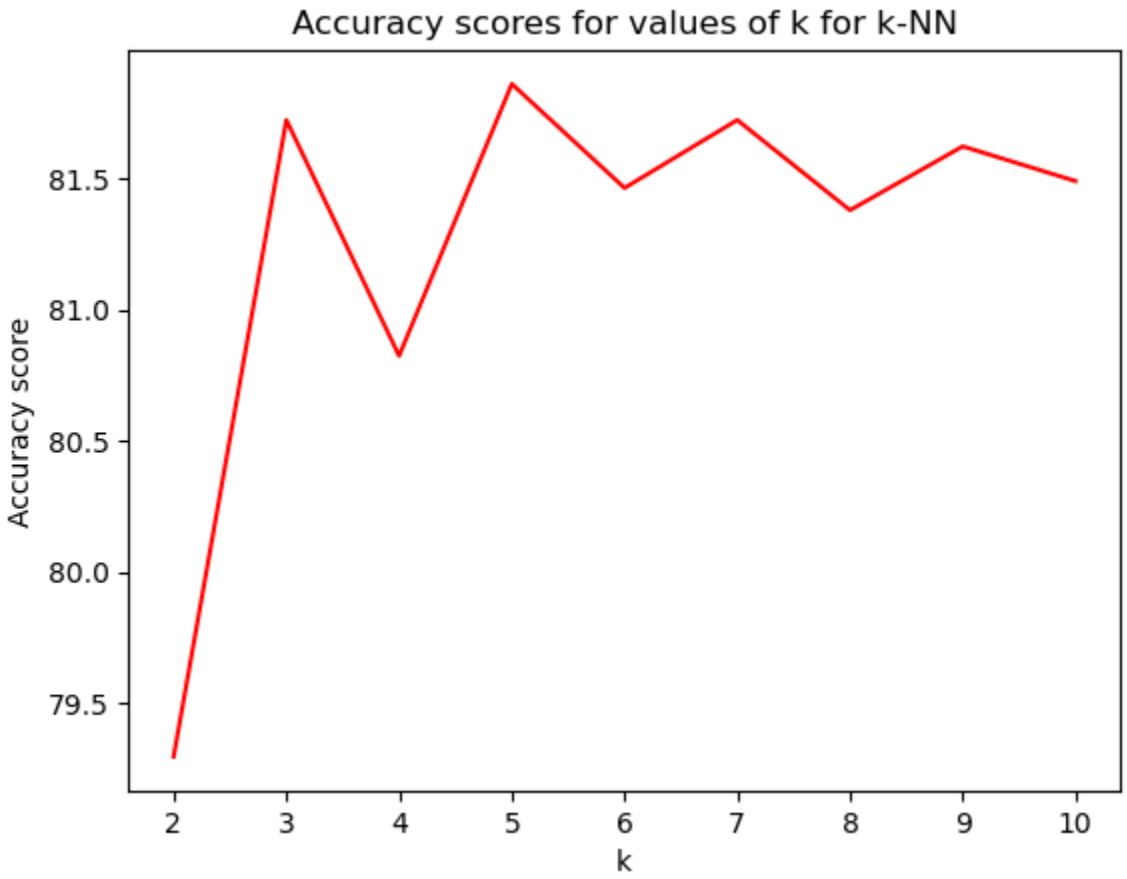
In [88]:
```
print(classification_report(y_test,lr_ovr_test_pred))
```

```
              precision    recall  f1-score   support

       <=50K       0.77      0.85      0.81     11144
        >50K       0.75      0.63      0.69      7781

    accuracy                           0.76     18925
   macro avg       0.76      0.74      0.75     18925
weighted avg       0.76      0.76      0.76     18925
```

5. k-Nearest Neighbour:-




```
Train score: 86.73%
Test score: 81.72%
```

	accuracy	precision	recall	f_measure	error_rate
k Nearest Neighbour	0.8172	0.7776	0.7779	0.7778	0.1828

- When K = 5, it gives the best accuracy.

```
In [92]: print(classification_report(y_test, knn_test_pred))
```

```
                precision    recall  f1-score   support

         <=50K       0.84      0.84      0.84     11144
          >50K       0.78      0.78      0.78      7781

      accuracy                           0.82     18925
     macro avg       0.81      0.81      0.81     18925
  weighted avg       0.82      0.82      0.82     18925
```

Overall Analysis of all Classification Algos:-

```
In [93]: analysis_df = pd.concat([dt_gini_df, dt_entropy_df, rf_gini_df, rf_entropy_df, svm_df, lr_multin_df, lr_ovr_df, knn_df ])
         analysis_df
```

Out[93]:

	accuracy	precision	recall	f_measure	error_rate
Decision Tree(Gini)	0.8035	0.7840	0.7207	0.7510	0.1965
Decision Tree(Entropy)	0.7899	0.7002	0.8549	0.7699	0.2101
Random Forest(Gini)	0.8189	0.7669	0.8039	0.7850	0.1811
Random Forest(Entropy)	0.8173	0.7610	0.8101	0.7847	0.1827
SVM	0.8076	0.7379	0.8250	0.7790	0.1924
Logistic Regression(multinomial)	0.7617	0.7483	0.6336	0.6862	0.2383
Logistic Regression(ovr)	0.7618	0.7484	0.6336	0.6862	0.2382
k Nearest Neighbour	0.8172	0.7776	0.7779	0.7778	0.1828

Sorted list according to f_measure and accuracy:

In [94]:
```
analysis_df.sort_values(by = ['f_measure', 'accuracy'], ascending = False, inplace = True)
analysis_df
```

Out[94]:

	accuracy	precision	recall	f_measure	error_rate
Random Forest(Gini)	0.8189	0.7669	0.8039	0.7850	0.1811
Random Forest(Entropy)	0.8173	0.7610	0.8101	0.7847	0.1827
SVM	0.8076	0.7379	0.8250	0.7790	0.1924
k Nearest Neighbour	0.8172	0.7776	0.7779	0.7778	0.1828
Decision Tree(Entropy)	0.7899	0.7002	0.8549	0.7699	0.2101
Decision Tree(Gini)	0.8035	0.7840	0.7207	0.7510	0.1965
Logistic Regression(ovr)	0.7618	0.7484	0.6336	0.6862	0.2382
Logistic Regression(multinomial)	0.7617	0.7483	0.6336	0.6862	0.2383

- Here, Random Forest has the best accuracy and f_measure
- Logistic regression underperforms because of the less linear relations with features.
- SVM performs well because there is a margin of separation between classes.