DATA MINING PROJECT

OVERVIEW

Data mining is very important !

We have enormous amount of data and they are collected in enormous speed

Data mining helps us to make automated analysis of massive data and also make hyposthises formation.

OVERVIEW

In this project we are going to do several things with our data in two parts:

<u>Part1:</u>

- missing handling
- Remove noise
- Remove duplicate records
- Remove irrelevant attributes
- Remove correlated attributes
  - . Correlation rate greater than or equal 0.8 for positive correlation
  - . Correlation rate less than or equal -0.8 for negative correlation

-Apply discretization on numeric attributes as possible

- First, we are going to import important packages to make handling easy

```
[2]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
```

- Importing the dataset in python

```
[3]: #importing the dataset into kaggle
     df = pd.read_csv(r"C:\Users\Dell\Desktop\adult.csv")
```

[4]:
```
df.head()
```

[4]:
```
   age  workclass  fnlwgt     education  educational-num      marital-status  \
0   25    Private  226802          11th                7       Never-married
1   38    Private   89814       HS-grad                9  Married-civ-spouse
2   28  Local-gov  336951    Assoc-acdm               12  Married-civ-spouse
3   44    Private  160323  Some-college               10  Married-civ-spouse
4   18          ?  103497  Some-college               10       Never-married

          occupation relationship   race  gender  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male             0             0
1    Farming-fishing      Husband  White    Male             0             0
2    Protective-serv      Husband  White    Male             0             0
3  Machine-op-inspct      Husband  Black    Male          7688             0
4                  ?    Own-child  White  Female             0             0

   hours-per-week native-country income
0              40  United-States  <=50K
1              50  United-States  <=50K
2              40  United-States   >50K
3              40  United-States   >50K
4              30  United-States  <=50K
```

- Now, we want to see how the data like

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
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   educational-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   gender           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: 5.6+ MB
```

- The info( ) method prints information about the DataFrame.The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).

[6]:
```python
df.dtypes
```

[6]:
```
age                 int64
workclass          object
fnlwgt              int64
education          object
educational-num     int64
marital-status     object
occupation         object
relationship       object
race               object
gender             object
capital-gain        int64
capital-loss        int64
hours-per-week      int64
native-country     object
income             object
dtype: object
```

- To choose best suitable ways to handle data, we want to know all kinds of types these data contains

STEP 1: HANDLING MISSING VALUES

```
In [7]: df=df.replace('?',np.NaN)
        df.isnull().sum()

Out[7]: age                   0
        workclass          2799
        fnlwgt                0
        education             0
        educational-num       0
        marital-status        0
        occupation         2809
        relationship          0
        race                  0
        gender                0
        capital-gain          0
        capital-loss          0
        hours-per-week        0
        native-country      857
        income                0
        dtype: int64
```

- To handle missing values in the dataset, we want to know the total number of missing values for each column.

In [9]:
```python
percent_missing = df.isnull().sum() * 100 / len(df)
percent_missing
```

Out[9]:
```
age                0.000000
workclass          5.730724
fnlwgt             0.000000
education          0.000000
educational-num    0.000000
marital-status     0.000000
occupation         5.751198
relationship       0.000000
race               0.000000
gender             0.000000
capital-gain       0.000000
capital-loss       0.000000
hours-per-week     0.000000
native-country     1.754637
income             0.000000
dtype: float64
```

*** 2799 - 5.73 % null values in 'workclass' column**

*** 2809 - 5.75 %null values in 'occupation' column**

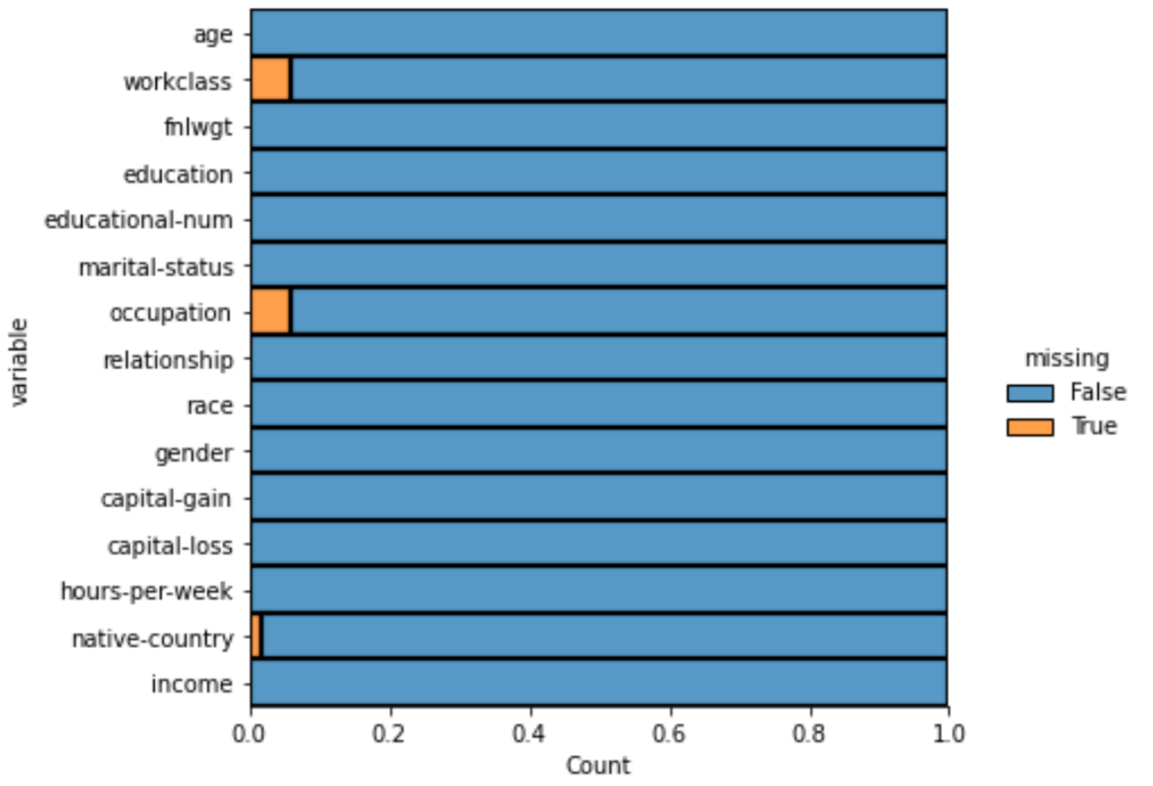
*** 457 - 1.75 % null values in 'native-country' column**

- In this code, we want to know the percentage of null values in every column. This step is important because we will know in what percentage null values affects our data

Visualization of null values

In [10]:
```
plt.figure(figsize=(10,6))
sns.displot(
    data=df.isna().melt(value_name="missing"),
    y="variable",
    hue="missing",
    multiple="fill",
    aspect=1.25
)
plt.savefig("distplot", dpi=100)
```

```
<Figure size 720x432 with 0 Axes>
```



- This visualization shows us how missing values affect our data

In [11]:
```
df=df.fillna(df.mode().iloc[0])
df.head(10)
```

Out[11]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	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
5	34	Private	198693	10th	6	Never-married	Other-service	Not-in-family	White	Male	0	0	30	United-States	<=50K
6	29	Private	227026	HS-grad	9	Never-married	Prof-specialty	Unmarried	Black	Male	0	0	40	United-States	<=50K
7	63	Self-emp-not-inc	104626	Prof-school	15	Married-civ-spouse	Prof-specialty	Husband	White	Male	3103	0	32	United-States	>50K
8	24	Private	369667	Some-college	10	Never-married	Other-service	Unmarried	White	Female	0	0	40	United-States	<=50K
9	55	Private	104996	7th-8th	4	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	10	United-States	<=50K

- After we prepare data, we will start handling missing phase
- First, fill missing value with the most frequent value of that column

```
In [12]: df.isnull().sum()

Out[12]: age                0
         workclass          0
         fnlwgt             0
         education          0
         educational-num    0
         marital-status     0
         occupation         0
         relationship       0
         race               0
         gender             0
         capital-gain       0
         capital-loss       0
         hours-per-week     0
         native-country     0
         income             0
         dtype: int64
```

- To make sure there is no null value in data

STEP 2: REMOVE NOISE:

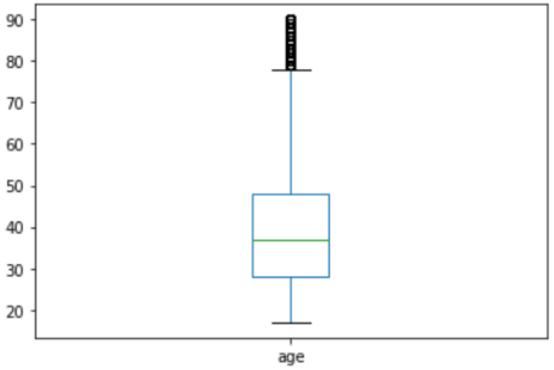
- We want to detect all the outliers and take action with it through two options :
- Remove outliers
- Handling outliers

- There are several ways to handle outliers but we will just replace the values with the median

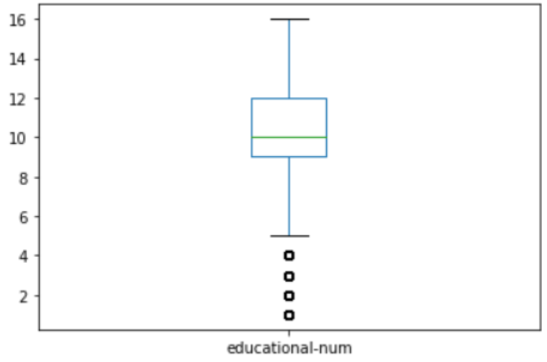
- Firstly, we are going to use boxplot to detect the outliers in each column. We choose to make the boxplot function which take the dataset and the feature's name ,then display the boxplot

In [13]:
```
def boxplot(df,fn):
    df.boxplot(column=[fn])
    plt.grid(False)
    plt.show()
```

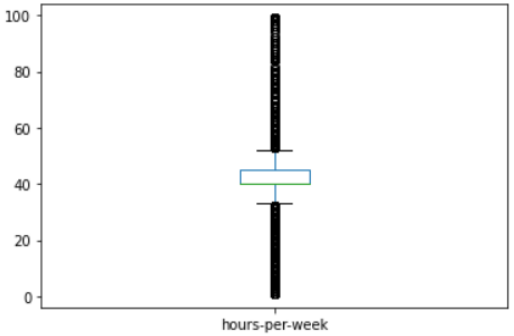
In [14]:
```
boxplot(df,"age")
```



In [16]:
```
boxplot(df,"educational-num")
```



In [19]:
```
boxplot(df,"hours-per-week")
```



- Secondly, we define a function called outliers which returns a list of indices of outliers in our dataset. This function will take the dataframe and function

(The function refers to the column we want to know how many outliers in it)

- In this step, we detect the outliers using IQR method and detect the outliers related to the upper and lower bound

```
In [20]: def outliers(df,fn):
             Q1=df[fn].quantile(0.25)
             Q3=df[fn].quantile(0.75)
             IQR=Q3-Q1
             lower_bound=Q1-1.5*IQR
             upper_bound=Q3+1.5*IQR
             list= df.index[ (df[fn]<lower_bound ) | (df[fn]>upper_bound) ]
             return list
```

- Thirdly, we create an *empty list* to store the indices of outliers in it and *for loop* to store all the indices of all columns

```
In [21]: indices=[]
         for col in['educational-num','age']:
             indices.extend(outliers(df,col))
```

- This is a small hint about how many the outliers are

In [22]:
```python
print(indices)
```

```
3908, 3934, 3956, 3972, 4005, 4032, 4045, 4053, 4062, 4093, 4102, 4170, 4191, 4228, 4237, 4264, 4283, 4308, 4409, 4420, 4426,
4445, 4453, 4467, 4599, 4601, 4606, 4629, 4643, 4646, 4650, 4661, 4711, 4712, 4727, 4729, 4743, 4774, 4795, 4862, 4931, 4944,
4949, 5034, 5080, 5087, 5088, 5113, 5177, 5184, 5238, 5348, 5353, 5372, 5442, 5445, 5482, 5484, 5486, 5504, 5536, 5563, 5631,
5668, 5690, 5714, 5736, 5760, 5778, 5786, 5794, 5795, 5829, 5840, 5911, 5922, 5959, 6013, 6016, 6021, 6034, 6055, 6112, 6130,
6188, 6239, 6264, 6288, 6373, 6403, 6427, 6491, 6529, 6534, 6560, 6561, 6575, 6590, 6605, 6636, 6657, 6689, 6762, 6763, 6826,
6843, 6849, 6875, 6886, 6979, 7049, 7054, 7086, 7170, 7216, 7252, 7297, 7307, 7328, 7333, 7340, 7362, 7377, 7438, 7485, 7511,
7539, 7559, 7589, 7600, 7628, 7663, 7677, 7683, 7712, 7736, 7744, 7769, 7773, 7800, 7811, 7830, 7834, 7892, 7948, 7965, 7966,
7983, 8038, 8055, 8068, 8099, 8136, 8206, 8261, 8281, 8343, 8346, 8349, 8371, 8374, 8375, 8469, 8470, 8537, 8559, 8607, 8629,
8640, 8677, 8690, 8721, 8751, 8917, 8930, 8956, 8973, 8988, 8990, 9124, 9162, 9163, 9270, 9277, 9281, 9289, 9356, 9366, 9393,
9394, 9440, 9472, 9508, 9512, 9517, 9541, 9543, 9563, 9621, 9639, 9647, 9658, 9689, 9707, 9729, 9732, 9735, 9744, 9745, 9767,
9769, 9808, 9827, 9853, 9857, 9888, 9901, 9923, 9993, 10010, 10029, 10039, 10102, 10160, 10162, 10166, 10195, 10199, 10226, 1
0262, 10283, 10289, 10304, 10330, 10359, 10421, 10433, 10457, 10546, 10603, 10612, 10646, 10666, 10674, 10676, 10694, 10710,
10721, 10777, 10864, 10874, 10895, 10905, 10915, 10954, 10990, 11012, 11082, 11090, 11099, 11101, 11130, 11145, 11148, 11162,
11169, 11176, 11201, 11244, 11282, 11343, 11434, 11452, 11456, 11457, 11481, 11494, 11537, 11603, 11632, 11677, 11694, 11706,
11713, 11754, 11873, 11895, 11896, 11940, 11972, 12024, 12038, 12051, 12060, 12063, 12123, 12130, 12179, 12234, 12244, 12335,
12360, 12367, 12387, 12398, 12467, 12471, 12476, 12500, 12507, 12512, 12541, 12581, 12682, 12691, 12726, 12736, 12802, 12815,
12989, 13006, 13009, 13022, 13025, 13031, 13036, 13065, 13132, 13188, 13200, 13209, 13248, 13270, 13314, 13331, 13334, 13340,
13341, 13363, 13395, 13438, 13473, 13476, 13484, 13511, 13518, 13525, 13546, 13558, 13568, 13582, 13632, 13660, 13677, 13737,
13774, 13802, 13833, 13873, 13875, 13951, 13953, 14023, 14064, 14069, 14095, 14114, 14127, 14147, 14153, 14245, 14263, 14266,
```

- Finally, we drop some of the indices
- Here, we choose to remove the outliers from a certain features called *educational-num* and *age*
- We make a function called *remove* which takes the dataframe and a list of the indices, then returns a dataframe without its outliers.
- Inside the function, we transfer the *list* into a *set* which is sorted and unique.

```
In [23]: def remove(df,list):
             list=sorted(set(list))
             df=df.drop(list)
             return df
```

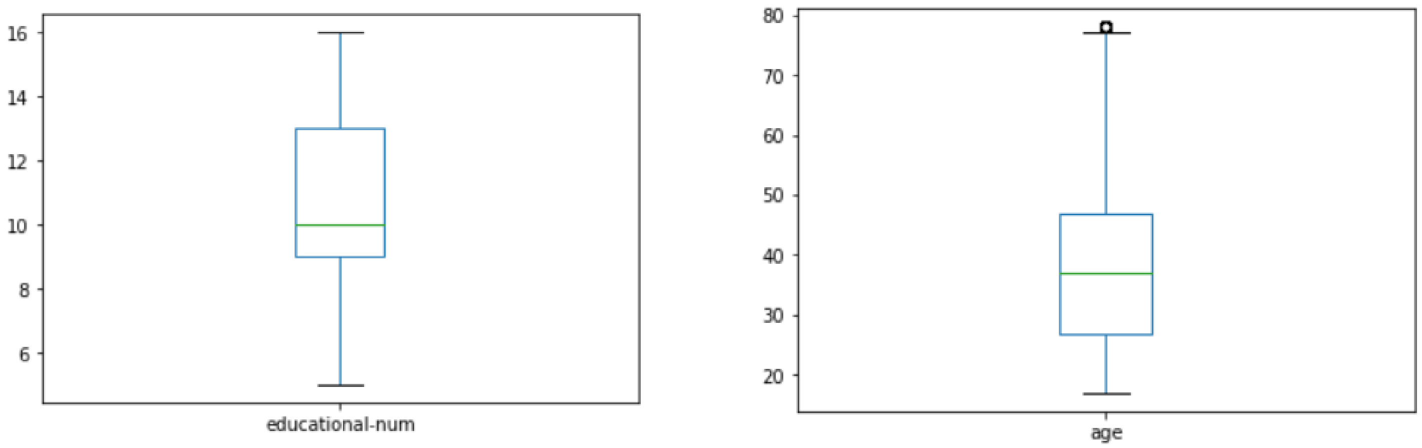
```
In [24]: df=remove(df,indices)
```

```
In [25]: df.shape
Out[25]: (46872, 15)
```

- Here, we notice that the number of rows is reduced to 46872 after removing rows contains outliers

- The boxplot of *educational-num* and *age* after removing outliers from it

```
In [30]: boxplot(df,"educational-num")
         boxplot(df,"age")
```



- Next step of handling, we replace the rest of outliers with median values using quantiles for the rest of features

```
In [27]: m1 = df['hours-per-week'].quantile(0.50)
         q1 = df['hours-per-week'].quantile(0.95)
         df['hours-per-week'] = np.where(df['hours-per-week'] >q1, m1, df['hours-per-week'])

         m2 = df['fnlwgt'].quantile(0.50)
         q2 = df['fnlwgt'].quantile(0.95)
         df['fnlwgt'] = np.where(df['fnlwgt'] >q2, m2, df['fnlwgt'])

         m3 = df['capital-gain'].quantile(0.50)
         q3 = df['capital-gain'].quantile(0.95)
         df['capital-gain'] = np.where(df['capital-gain'] >q3, m3, df['capital-gain'])

         m4 = df['capital-loss'].quantile(0.50)
         q4 = df['capital-loss'].quantile(0.95)
         df['capital-loss'] = np.where(df['capital-loss'] >q4, m4, df['capital-loss'])
```

- In this step, we are going to check if there is any unsuitable negative values in numerical features or not to handle them

In [29]:
```
print ((df['age'] < 0).any())
print ((df['fnlwgt'] < 0).any())
print ((df['educational-num'] < 0).any())
print ((df['capital-gain'] < 0).any())
print ((df['capital-gain'] < 0).any())
print ((df['hours-per-week'] < 0).any())
```

```
False
False
False
False
False
False
```

- As we see, there are no unsuitable negative values in the data set which may cause noise in data

STEP 3: REMOVE DUPLICATES

In [30]:
```
df.drop_duplicates()
```

Out[30]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802.0	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States	<=50K
1	38	Private	89814.0	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States	<=50K
2	28	Local-gov	336951.0	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States	>50K
3	44	Private	160323.0	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	0.0	0.0	40.0	United-States	>50K
4	18	Private	103497.0	Some-college	10	Never-married	Prof-specialty	Own-child	White	Female	0.0	0.0	30.0	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	Private	257302.0	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0.0	0.0	38.0	United-States	<=50K
48838	40	Private	154374.0	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0.0	0.0	40.0	United-States	>50K
48839	58	Private	151910.0	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0.0	0.0	40.0	United-States	<=50K
48840	22	Private	201490.0	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0.0	0.0	20.0	United-States	<=50K
48841	52	Self-emp-inc	287927.0	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	0.0	0.0	40.0	United-States	>50K

46767 rows × 15 columns

- Here, we notice that the number of rows is reduced to 46767 after removing duplications

STEP 4: NORMALIZATION:

- Normalization refers to rescaling numeric attributes into the range 0 and 1.
- In the following steps, we are going to use *MinMax* normalization.
- *MinMaxScaler( )* function takes range by default (0,1)
- *fit_transform( )* : Fit to data, then transform it.

In [31]:
```
from sklearn.preprocessing import MinMaxScaler
scaler =MinMaxScaler()
numerical=["age","educational-num","capital-gain",'capital-loss','hours-per-week']
df[numerical]=scaler.fit_transform(df[numerical])
df.head(5)
```

Out[31]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	0.131148	Private	226802.0	11th	0.181818	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	0.661017	United-States	<=50K
1	0.344262	Private	89814.0	HS-grad	0.363636	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	0.830508	United-States	<=50K
2	0.180328	Local-gov	336951.0	Assoc-acdm	0.636364	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	0.661017	United-States	>50K
3	0.442623	Private	160323.0	Some-college	0.454545	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	0.0	0.0	0.661017	United-States	>50K
4	0.016393	Private	103497.0	Some-college	0.454545	Never-married	Prof-specialty	Own-child	White	Female	0.0	0.0	0.491525	United-States	<=50K

STEP 5: REMOVE IRRELEVANT ATTRIBUTES

- We choose to make this step using feature selection
- There are two steps to make feature selection:
  - if we get correlation between features and target attributes that the attribute which is highly correlated with target is more important (remove irrelevant attribute)
  - if we get correlation between features and target attributes that the attribute which is highly correlated with target is more important (remove irrelevant attribute)
- Here, we are going to use pearson correlation coefficient to convert categorical attribute to numerical

In [32]:
```
import sklearn
from sklearn.preprocessing import OrdinalEncoder
enc=OrdinalEncoder()
df.columns
```

Out[32]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'educational-num',
       'marital-status', 'occupation', 'relationship', 'race', 'gender',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'income'],
      dtype='object')
```

- Importing *OrdinalEncoder* from *sklearn* used to transform categorical values

- *fit( )* : the training data will be used to estimate the minimum and maximum observable values

```
In [33]: enc.fit(df[['age', 'workclass', 'fnlwgt', 'education', 'educational-num',
          'marital-status', 'occupation', 'relationship', 'race', 'gender',
          'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
          'income']])

Out[33]: OrdinalEncoder()
```

- Transforming the categorical values to numbers so we could apply correlation
- *transform( )* : can use the normalized data to train your model

```
In [34]: #Transforming the categorical values to numbers so we could apply correlation
         df[['age', 'workclass', 'fnlwgt', 'education', 'educational-num',
          'marital-status', 'occupation', 'relationship', 'race', 'gender',
          'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
          'income']]=enc.transform(df[['age', 'workclass', 'fnlwgt', 'education', 'educational-num',
          'marital-status', 'occupation', 'relationship', 'race', 'gender',
          'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
          'income']])
```

- Transforming the categorical values to numbers so we could apply correlation

```
In [34]: df[['age', 'workclass', 'fnlwgt', 'education', 'educational-num',
          'marital-status', 'occupation', 'relationship', 'race', 'gender',
          'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
          'income']]=enc.transform(df[['age', 'workclass', 'fnlwgt', 'education', 'educational-num',
          'marital-status', 'occupation', 'relationship', 'race', 'gender',
          'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
          'income']])
```

- Getting the correlation between our target attribute "income" with other features and sorting them in descending order.

```
In [35]: corr_matrix=abs(df.corr())
         corr_matrix["income"].sort_values(ascending=False)
```

```
Out[35]: income             1.000000
         educational-num    0.340747
         relationship       0.260775
         age                0.256212
         hours-per-week     0.242660
         gender             0.220917
         marital-status     0.201358
         race               0.071919
         education          0.045429
         occupation         0.033907
         capital-gain       0.012811
         native-country     0.004281
         fnlwgt             0.002043
         workclass          0.000676
         capital-loss            NaN
         Name: income, dtype: float64
```

- Removing attribute that had the smallest correlation " relationship " , so that we dropped “ capital-loss ” as its correlation equals NaN

```
In [36]: df=df.drop(["capital-loss"],axis=1)
         df
```

Out[36]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	hours-per-week	native-country	income
0	8.0	3.0	18695.0	1.0	2.0	4.0	6.0	3.0	2.0	1.0	0.0	39.0	38.0	0.0
1	21.0	3.0	4067.0	8.0	4.0	2.0	4.0	0.0	4.0	1.0	0.0	49.0	38.0	0.0
2	11.0	1.0	24454.0	4.0	7.0	2.0	10.0	0.0	4.0	1.0	0.0	39.0	38.0	1.0
3	27.0	3.0	10790.0	11.0	5.0	2.0	6.0	0.0	2.0	1.0	0.0	39.0	38.0	1.0
4	1.0	3.0	5207.0	11.0	5.0	4.0	9.0	3.0	4.0	0.0	0.0	29.0	38.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	10.0	3.0	20844.0	4.0	7.0	2.0	12.0	5.0	4.0	0.0	0.0	37.0	38.0	0.0
48838	23.0	3.0	10194.0	8.0	4.0	2.0	6.0	0.0	4.0	1.0	0.0	39.0	38.0	1.0
48839	41.0	3.0	9931.0	8.0	4.0	6.0	0.0	4.0	4.0	0.0	0.0	39.0	38.0	0.0
48840	5.0	3.0	16254.0	8.0	4.0	4.0	0.0	3.0	4.0	1.0	0.0	19.0	38.0	0.0
48841	35.0	4.0	22516.0	8.0	4.0	2.0	3.0	5.0	4.0	0.0	0.0	39.0	38.0	1.0

46872 rows × 14 columns

- Now, we are going to remove correlated attribute

- what is correlation?

  ✓ Correlation is a statistical term which in common usage refers to how close two variables are to having a linear relationship with each other

- How does correlation help in feature selection?

  ✓ Features with high correlation are more linearly dependent and have almost the same effect on the dependent variable. So, when two features have high correlation, we can drop one of the two features.

- Now, we will use Pearson Correlation between numerical attribute

```
In [37]: df.corr()
```

Here is the output

Out[37]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	hours-per-week	native-country	income
age	1.000000	0.041173	-0.058294	0.014109	0.111925	-0.292310	-0.007615	-0.266949	0.034706	0.091815	0.044507	0.099712	-0.000773	0.256212
workclass	0.041173	1.000000	-0.028876	0.009983	0.018229	-0.020787	0.011313	-0.054238	0.052872	0.065741	-0.000068	0.005429	-0.005660	-0.000676
fnlwgt	-0.058294	-0.028876	1.000000	-0.007397	-0.012775	0.028631	0.004653	0.005638	-0.005311	0.019910	-0.005582	-0.000365	-0.046768	0.002043
education	0.014109	0.009983	-0.007397	1.000000	0.215842	-0.029101	-0.037319	-0.013364	0.011547	-0.019507	0.005054	0.044956	0.030652	0.045429
educational-num	0.111925	0.018229	-0.012775	0.215842	1.000000	-0.071505	0.087857	-0.127650	0.029865	0.031065	-0.001445	0.168176	-0.012105	0.340747
marital-status	-0.292310	-0.020787	0.028631	-0.029101	-0.071505	1.000000	0.030504	0.185489	-0.069501	-0.121219	-0.027016	-0.194592	-0.011612	-0.201358
occupation	-0.007615	0.011313	0.004653	-0.037319	0.087857	0.030504	1.000000	-0.037163	-0.003615	0.047003	-0.001624	-0.037743	-0.004175	0.033907
relationship	-0.266949	-0.054238	0.005638	-0.013364	-0.127650	0.185489	-0.037163	1.000000	-0.117589	-0.579294	-0.040345	-0.262709	-0.007196	-0.260775
race	0.034706	0.052872	-0.005311	0.011547	0.029865	-0.069501	-0.003615	-0.117589	1.000000	0.085716	0.011201	0.035971	0.128571	0.071919
gender	0.091815	0.065741	0.019910	-0.019507	0.031065	-0.121219	0.047003	-0.579294	0.085716	1.000000	0.026208	0.235534	-0.000432	0.220917
capital-gain	0.044507	-0.000068	-0.005582	0.005054	-0.001445	-0.027016	-0.001624	-0.040345	0.011201	0.026208	1.000000	0.015713	0.004214	-0.012811
hours-per-week	0.099712	0.005429	-0.000365	0.044956	0.168176	-0.194592	-0.037743	-0.262709	0.035971	0.235534	0.015713	1.000000	0.003130	0.242660
native-country	-0.000773	-0.005660	-0.046768	0.030652	-0.012105	-0.011612	-0.004175	-0.007196	0.128571	-0.000432	0.004214	0.003130	1.000000	0.004281
income	0.256212	-0.000676	0.002043	0.045429	0.340747	-0.201358	0.033907	-0.260775	0.071919	0.220917	-0.012811	0.242660	0.004281	1.000000

- Here, we compare the correlation between features and remove one of two features that have a correlation higher than 0.8

```
In [38]: plt.figure(figsize=(15,8))
         sns.heatmap(df.corr(), annot=True)
```

```
Out[38]: <AxesSubplot:>
```

- With the following function we can select highly correlated features
    - Set of all the names of correlated columns.
    - we are interested in absolute coefficient value.
    - Then, we get the name of column at *colname* variable

```
In [39]: def correlation(dataset, corr_rate):
             col_corr = set()
             corr_matrix = df.corr()
             for i in range(len(corr_matrix.columns)):
                 for j in range(i):
                     if abs(corr_matrix.iloc[i, j]) > corr_rate:
                         colname = corr_matrix.columns[i]
                         col_corr.add(colname)
             return col_corr
```

- Note that there is no correlation attribute greater than .8

```
In [40]: corr_features = correlation(df, 0.8)
         len(set(corr_features))

Out[40]: 0
```

STEP 5:DISCRETIZATION:

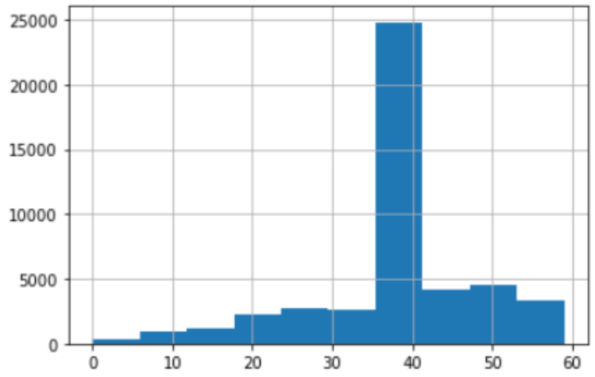
- Data discretization : is the process of converting continuous data into discrete buckets by grouping it.
- Discretization is also known for easy maintainability of the data. Training a model with discrete data becomes faster and more effective than when attempting the same with continuous data.

In [41]:
```
df = pd.DataFrame(df)
df['hours-per-week'].describe()
```

Out[41]:
```
count    46872.000000
mean        38.346604
std         10.346904
min          0.000000
25%         39.000000
50%         39.000000
75%         43.000000
max         59.000000
Name: hours-per-week, dtype: float64
```

In [42]:
```
df['hours-per-week'].hist()
```

Out[42]: <AxesSubplot:>



- *Pandas cut( )* : function is used to separate the array elements into different bins.
- parameters used in this code (x) : The input array that is to be binned.
- bins: defines the number of bin or the edges for the segmentation.
- labels : (optional) specifies the labels for the returned bins.

- Now, we will split the column into three bins (poorly effective, effective, highly effective) using *pandas.cut( )*

```
In [43]: df['hours-per-week_des'] = pd.cut(x = df['hours-per-week'], bins = 3, labels = ['poorly effective',
                                                                                'effective', 'heighly effective'])
```

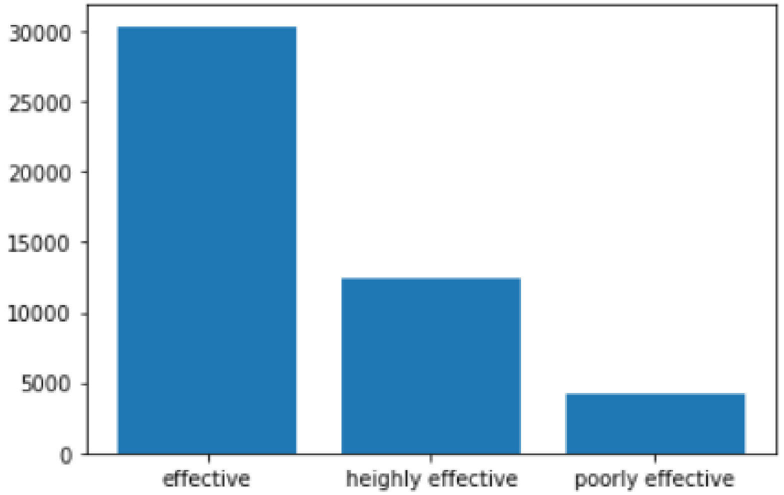
- Then, count the objects in every bin.

```
In [40]: df['hours-per-week_des'].value_counts()

Out[40]: effective            30327
         heighly effective    12365
         poorly effective      4180
         Name: hours-per-week_des, dtype: int64
```

- Then, create a function *(draw_barplot)* to draw the barplot of each bin.

```
In [45]: import matplotlib.pyplot as plt
         def draw_barplot(x):
             s = x.value_counts()
             plt.bar(s.index, s.values)
```

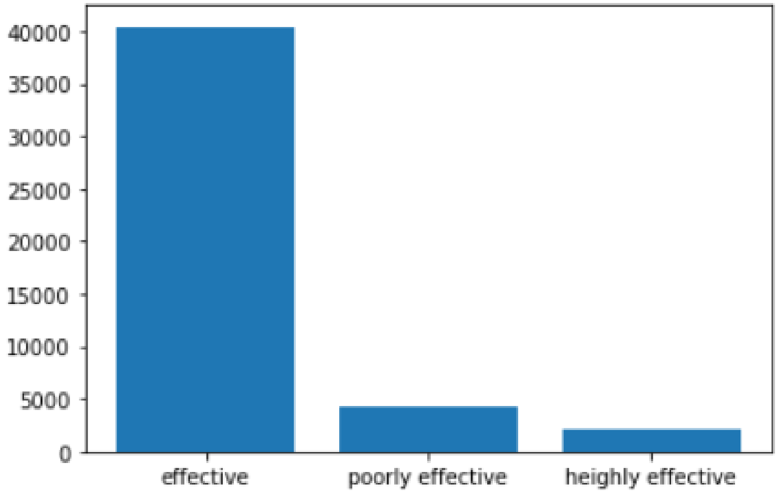
```
In [46]: draw_barplot(df['hours-per-week_des'])
```



- Now, define the edges for the segmentation.

```
In [47]: df['hours-per-week_des2'] = pd.cut(x = df['hours-per-week'], bins = [0, 20, 55, 80], labels = [
             'poorly effective', 'effective', 'heighly effective'])
```

```
In [48]: draw_barplot(df['hours-per-week_des2'])
```



- KBinsDiscretizer:
    - The discretization transform is available in the scikit-learn Python machine learning library via the KBinsDiscretizer class.
    - Parameters: n_binsint or n_features: The number of bins to produce.
    - encode{'onehot', 'onehot-dense', 'ordinal'}: Method used to encode the transformed result.
    - 'onehot': Encode the transformed result with one-hot encoding and return a sparse matrix.
    - 'onehot-dense': Encode the transformed result with one-hot encoding and return a dense array.
    - 'ordinal': Return the bin identifier encoded as an integer value.
    - strategy{'uniform', 'quantile', 'kmeans'}: Strategy used to define the widths of the bins.
    - 'uniform': All bins in each feature have identical widths.
    - 'quantile': All bins in each feature have the same number of points.
    - 'kmeans': Values in each bin have the same nearest center of a 1D k-means cluster.

- Equal-Width Discretization

  - Separating all possible values into ‘N’ number of bins, each having the same width. Formula for interval width:

    - Width = (maximum value - minimum value) / N
    - where N is the number of bins or intervals.

In [49]:
```
from sklearn.preprocessing import KBinsDiscretizer
```

In [50]:
```
discretizer  = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
```

- fit_transform() method:

  - Fit to data, then transform it.
  - Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.
  - parameters: x: array-like of shape (n_samples, n_features) Input samples.

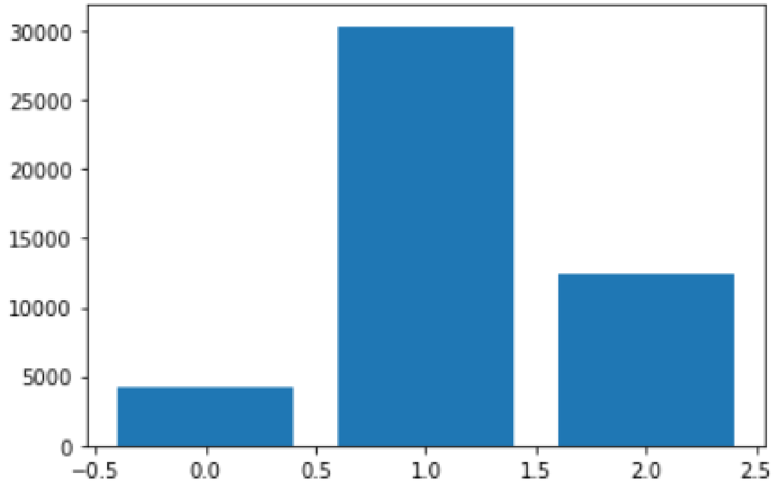
In [51]:
```
df['hours-per-week_discrete'] = discretizer.fit_transform(df['hours-per-week'].values.reshape(-1,1)).astype(int)
```

In [52]:
```
df.head()
```

Out[52]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	hours-per-week	native-country	income	hours-per-week_des	hours-per-week_des2
0	8.0	3.0	18695.0	1.0	2.0	4.0	6.0	3.0	2.0	1.0	0.0	39.0	38.0	0.0	effective	effective
1	21.0	3.0	4067.0	8.0	4.0	2.0	4.0	0.0	4.0	1.0	0.0	49.0	38.0	0.0	heighly effective	effective
2	11.0	1.0	24454.0	4.0	7.0	2.0	10.0	0.0	4.0	1.0	0.0	39.0	38.0	1.0	effective	effective
3	27.0	3.0	10790.0	11.0	5.0	2.0	6.0	0.0	2.0	1.0	0.0	39.0	38.0	1.0	effective	effective
4	1.0	3.0	5207.0	11.0	5.0	4.0	9.0	3.0	4.0	0.0	0.0	29.0	38.0	0.0	effective	effective

In [53]:
```
draw_barplot(df['hours-per-week_discrete'])
```



In [54]:
```
discretizer.bin_edges_
```

Out[54]:
```
array([array([ 0.        , 19.66666667, 39.33333333, 59.        ])],
      dtype=object)
```

- Equal frequency:

  - Separating all possible values into 'N' number of bins, each having the same amount of observations. Intervals may correspond to quantile values.

```
In [69]: discretizer  = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='quantile')
```

```
In [70]: df['hours-per-week_eq_freq'] = discretizer.fit_transform(df['hours-per-week'].values.reshape(-1,1)).astype(int)
```

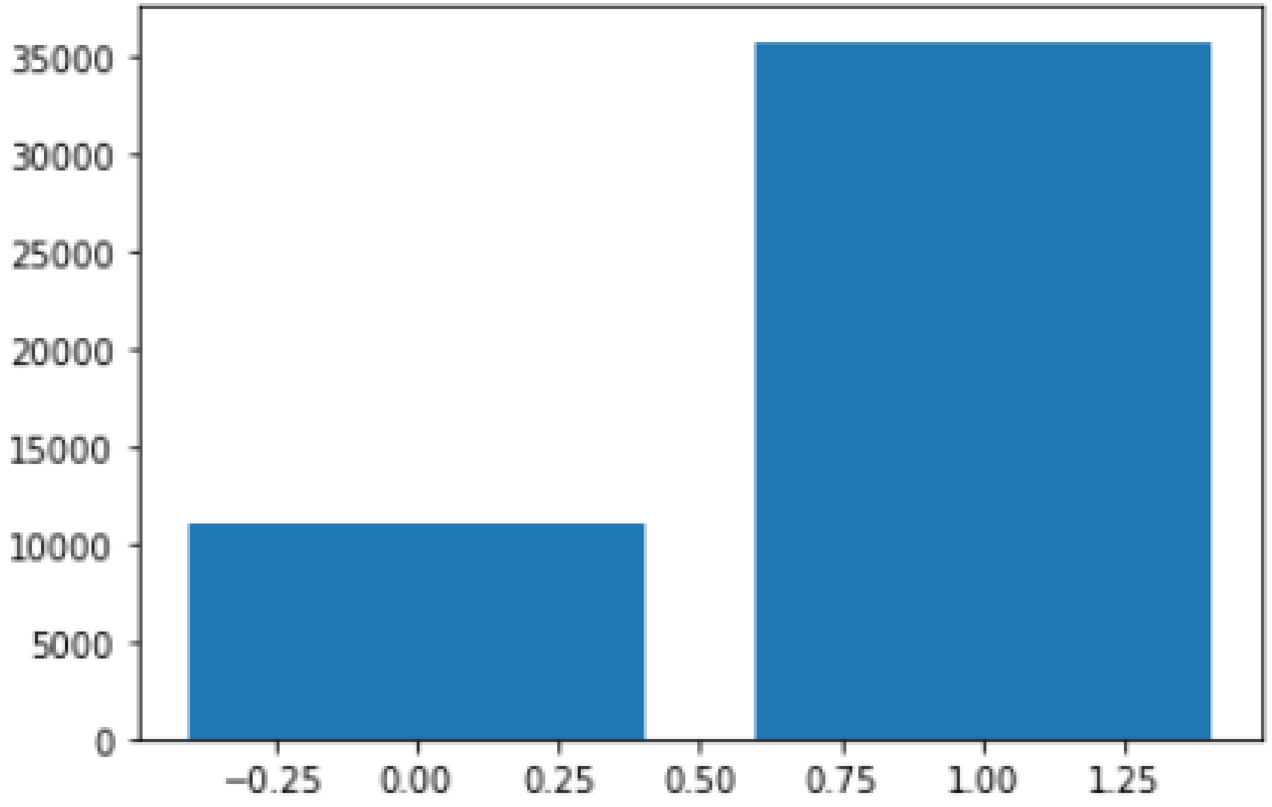
```
C:\Users\DELL\anaconda3\lib\site-packages\sklearn\preprocessing\_discretization.py:220: UserWarning: Bins whose width are too s
mall (i.e., <= 1e-8) in feature 0 are removed. Consider decreasing the number of bins.
  warnings.warn('Bins whose width are too small (i.e., <= '
```

```
In [71]: df.head()
```

Out[71]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	hours-per-week	native-country	income	hours-per-week_des	hours-per-week_des2	hours-per-week_discrete	hours-per-week_eq_freq
0	8.0	3.0	18695.0	1.0	2.0	4.0	6.0	3.0	2.0	1.0	0.0	39.0	38.0	0.0	effective	effective	1	1
1	21.0	3.0	4067.0	8.0	4.0	2.0	4.0	0.0	4.0	1.0	0.0	49.0	38.0	0.0	heighly effective	effective	2	1
2	11.0	1.0	24454.0	4.0	7.0	2.0	10.0	0.0	4.0	1.0	0.0	39.0	38.0	1.0	effective	effective	1	1
3	27.0	3.0	10790.0	11.0	5.0	2.0	6.0	0.0	2.0	1.0	0.0	39.0	38.0	1.0	effective	effective	1	1
4	1.0	3.0	5207.0	11.0	5.0	4.0	9.0	3.0	4.0	0.0	0.0	29.0	38.0	0.0	effective	effective	1	0

```
In [72]: draw_barplot(df['hours-per-week_eq_freq'])
```




```
In [73]: discretizer.bin_edges_

Out[73]: array([array([ 0., 39., 59.])], dtype=object)
```

- K-Means Discretization:

We apply K-Means clustering to the continuous variable, thus dividing it into discrete groups or clusters.

In [74]:
```
discretizer  = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='kmeans')
```

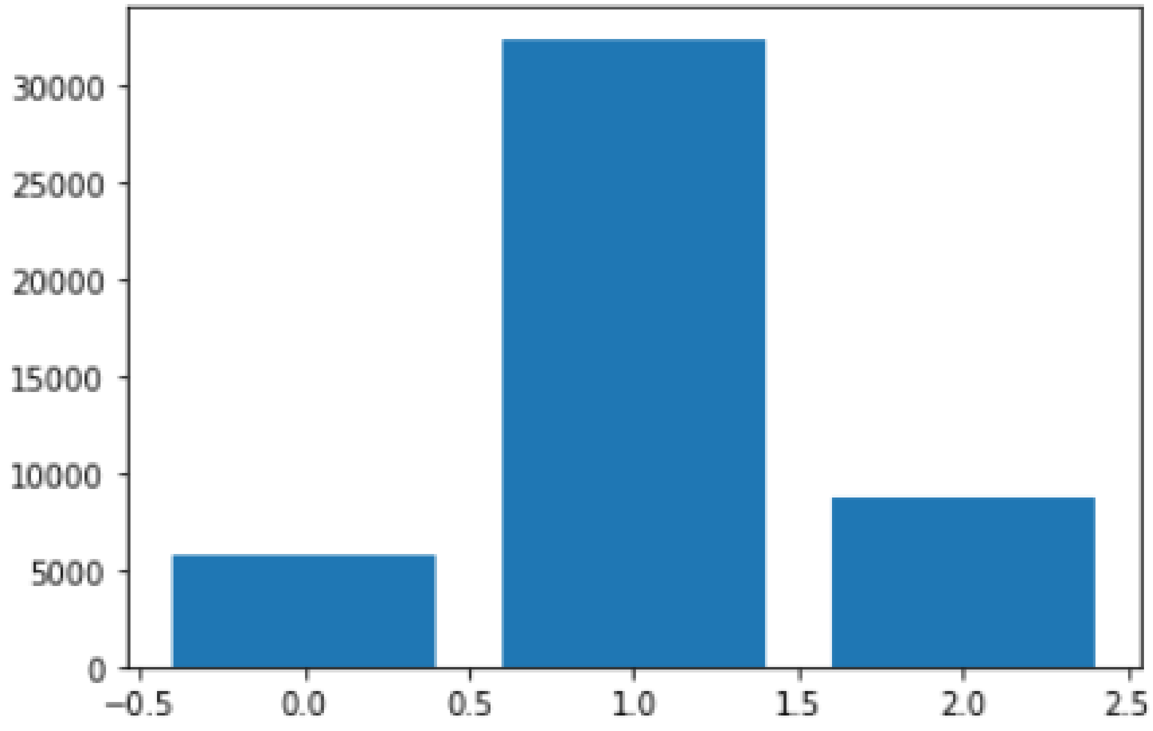
In [75]:
```
df['hours-per-week_k'] = discretizer.fit_transform(df['hours-per-week'].values.reshape(-1,1)).astype(int)
```

In [76]:
```
df.head()
```

Out[76]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	hours-per-week	native-country	income	hours-per-week_des	hours-per-week_des2	hours-per-week_discrete	hours-per-week_eq_freq	hours-per-week_k
0	8.0	3.0	18695.0	1.0	2.0	4.0	6.0	3.0	2.0	1.0	0.0	39.0	38.0	0.0	effective	effective	1	1	1
1	21.0	3.0	4067.0	8.0	4.0	2.0	4.0	0.0	4.0	1.0	0.0	49.0	38.0	0.0	heighly effective	effective	2	1	2
2	11.0	1.0	24454.0	4.0	7.0	2.0	10.0	0.0	4.0	1.0	0.0	39.0	38.0	1.0	effective	effective	1	1	1
3	27.0	3.0	10790.0	11.0	5.0	2.0	6.0	0.0	2.0	1.0	0.0	39.0	38.0	1.0	effective	effective	1	1	1
4	1.0	3.0	5207.0	11.0	5.0	4.0	9.0	3.0	4.0	0.0	0.0	29.0	38.0	0.0	effective	effective	1	0	1

In [77]:
```
draw_barplot(df['hours-per-week_k'])
```



In [78]:
```
discretizer.bin_edges_
```

Out[78]:
```
array([array([ 0.        , 27.69253179, 45.29151268, 59.        ])],
      dtype=object)
```

OVERVIEW

In this project we are going to do several things with our data in two parts:

Part 2:

1. Split your dataset into training and testing sets 80% and 20% for training and testing sets respectively and save each of these sets into separated files.

2. Muse the following classifiers :
   - KNN
   - Decision Tree
   - Naïve Bayes

3. Depend on your previous study on clustering technique (K means) use this technique on a suitable dataset.

APPLYING DECISION TREE:

- Decision tree is a supervised learning method used for classification.
- Our goal is to create model that predicts the value of the target variable by learning simple decision rules from data features .
- Here, we import the libraries we will use.

```
In [66]: from sklearn.model_selection import train_test_split
         x = df.drop('income',axis=1)
         y = df['income']
         x.head(5)
```

- From *sklearn* model selection, we import the train test split which helps us to divide our data into training set and test set.
- We assign x to all the data except for the target column which is income.
- we assign y to income.

- Here is the output…

Out[85]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	hours-per-week	native-country
0	8.0	3.0	18695.0	1.0	2.0	4.0	6.0	3.0	2.0	1.0	0.0	39.0	38.0
1	21.0	3.0	4067.0	8.0	4.0	2.0	4.0	0.0	4.0	1.0	0.0	49.0	38.0
2	11.0	1.0	24454.0	4.0	7.0	2.0	10.0	0.0	4.0	1.0	0.0	39.0	38.0
3	27.0	3.0	10790.0	11.0	5.0	2.0	6.0	0.0	2.0	1.0	0.0	39.0	38.0
4	1.0	3.0	5207.0	11.0	5.0	4.0	9.0	3.0	4.0	0.0	0.0	29.0	38.0

- And this is y which we assgine our target to it:

In [67]:
```
y.head(5)
```

Out[67]:
```
0    0.0
1    0.0
2    1.0
3    1.0
4    0.0
Name: income, dtype: float64
```

- Then, we need to Split our data set into training set and testing set.
- Here, we split our data set into 80% training and 20% testing.

In [87]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.20)
x_train.head()
```

Out[87]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	hours-per-week	native-country
9531	36.0	3.0	17666.0	5.0	6.0	4.0	0.0	2.0	2.0	0.0	0.0	39.0	38.0
19650	44.0	3.0	9731.0	8.0	4.0	2.0	3.0	0.0	4.0	1.0	0.0	49.0	38.0
20823	10.0	3.0	25435.0	5.0	6.0	2.0	4.0	0.0	4.0	1.0	0.0	39.0	38.0
21224	24.0	3.0	10836.0	5.0	6.0	2.0	3.0	0.0	4.0	1.0	0.0	39.0	38.0
27320	38.0	3.0	12992.0	9.0	9.0	2.0	9.0	0.0	4.0	1.0	0.0	39.0	38.0

- Here, we import decision tree classifier from the tree of the sklearn which will create our tree
- Then, we made an object of the decision tree classifier with max depth 5 which means that the tree will have 6 Levels at most.
- After that, we Performed training on the data using the *fit( )* function.

In [88]:
```
from sklearn.tree import DecisionTreeClassifier
tree1 = DecisionTreeClassifier(max_depth = 5)
tree1.fit(x_train,y_train)
```

Out[88]:
```
DecisionTreeClassifier(max_depth=5)
```

- Here, we used the Function *Predict( )* to make predictions , this is the test step .

- Then, we imported *accuracy_*score from *sklearn* metrices to measure the accuracy of the model, we find our accuracy to be 82% and it's a good percentage.

```
In [89]: y_predict = tree1.predict(x_test)

         from sklearn.metrics import accuracy_score
         print(accuracy_score(y_test,y_predict))

         0.8218666666666666
```

- Then, we will put our features in a list to use the in the feature_names parameter

```
In [90]: features = list(df.columns[1:])
         features

Out[90]: ['workclass',
          'fnlwgt',
          'education',
          'educational-num',
          'marital-status',
          'occupation',
          'relationship',
          'race',
          'gender',
          'capital-gain',
          'hours-per-week',
          'native-country',
          'income']
```

In [91]:
```python
from sklearn.tree import plot_tree
fig = plt.figure(figsize=(25,20))
plot = plot_tree(tree1,feature_names = features,
                filled = True,
                rounded = True,
                fontsize = 16)
```

- Here, we import *plot_tree* which will visualize our tree.

- *plot_tree()* function takes the tree, feature names, filled parameter, rounded parameter and font size.

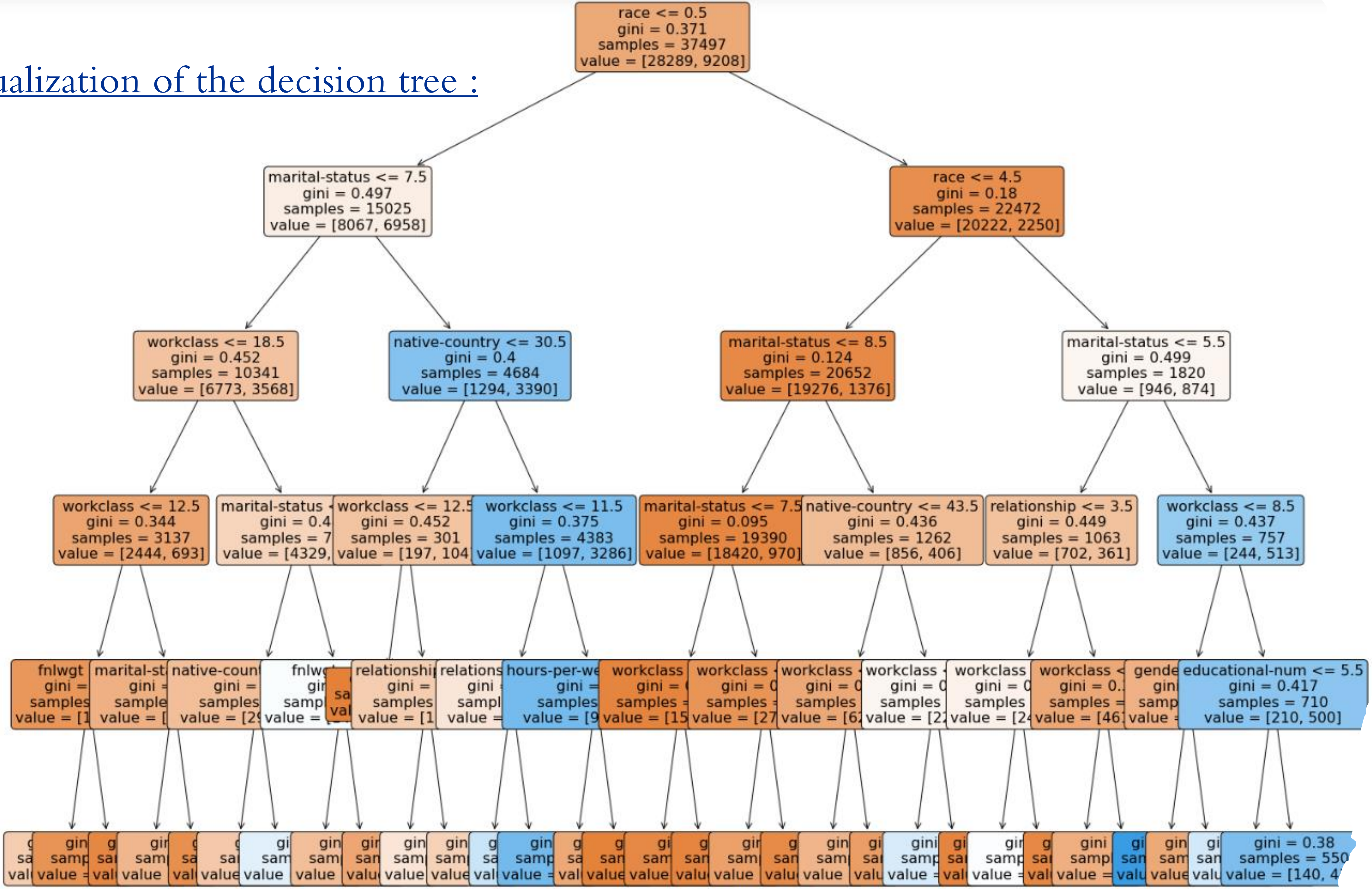
- *filed* parameter set for TRUE for painting nodes to indicate majority class for classification.

- *rounded* set for TRUE for drawing node boxes with rounded corners.

In [80]:
```python
fig.savefig("decistion_tree.png")
```

- At this Step we've saved the figure as png to our computer

<u>The visualization of the decision tree :</u>



APPLYING KNN:

- K nearest neighbor is another supervised learning method used in making predictions.

```
In [93]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors = 5)
         knn.fit(x_train,y_train)

Out[93]: KNeighborsClassifier()
```

- Here, we import the *k nearest neighbours* classifier from *sklearn* Library.
- We create instance from the *knn* classifier with 5 neighbors.
- *knn.fit( )* function to apply training on the training dataset we made in the last technique.

```
In [94]: y_predict2 = knn.predict(x_test)
         y_predict2

Out[94]: array([1., 1., 0., ..., 0., 0., 0.])

In [95]: print(accuracy_score(y_test,y_predict2))

         0.7345066666666666
```

- These two Steps is for predicting the values of the test set and measure the accuracy score
- The accuracy is equal to 73.4%

APPLYING NAIVE BAYES:

- It is a classification technique based on Bayes' theorem with an assumption of independence among predictors.

- Gaussian Naïve Bayes: It works with continuous attributes, it assumes the data normally distributed (Gaussian Distribution)

- Using Naïve Bayes, we can explore the possibility in predicting income level based on the individual's personal information

```
In [78]: from sklearn.naive_bayes import GaussianNB

         model = GaussianNB()

         model.fit(x_train,y_train)

Out[78]: GaussianNB()
```

- Here, we import *gaussian naive bayes model* from *sklearn Library* create a gaussian classifier and then use the *fit( )* function to train the model using training dataset.

- We predict the response value using test dataset.

```
In [82]: y_pred = model.predict(x_test)
         y_pred

Out[82]: array([0., 0., 1., ..., 0., 0., 0.])
```

- In this step, we calculate the accuracy.

```
In [80]: from sklearn import metrics
         from sklearn.metrics import accuracy_score

         print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))

         Model accuracy score: 0.7835
```

we find that our accuracy is 78.27%

- We import *scikit-learn metrics* module for *accuracy* calculation, then, checking accuracy using actual and predicted values.

- We find that our accuracy is 78.34% and it's not bad percentage

APPLYING K-MEANS:

- K-means Clustering : is one of the simplest and popularest unsupervised learning algorithms. It's about grouping the unlabeled dataset into different clusters. The letter K refers to the number of clusters.

```
In [7]: from sklearn.cluster import KMeans
        import pandas as pd
        from sklearn.preprocessing import MinMaxScaler
        from matplotlib import pyplot as plt
        %matplotlib inline
```

- As usual, we import the libraries we are going to use in our algorithm.
- We changed the dataset to iris because it is suitable for k-means algorithm.

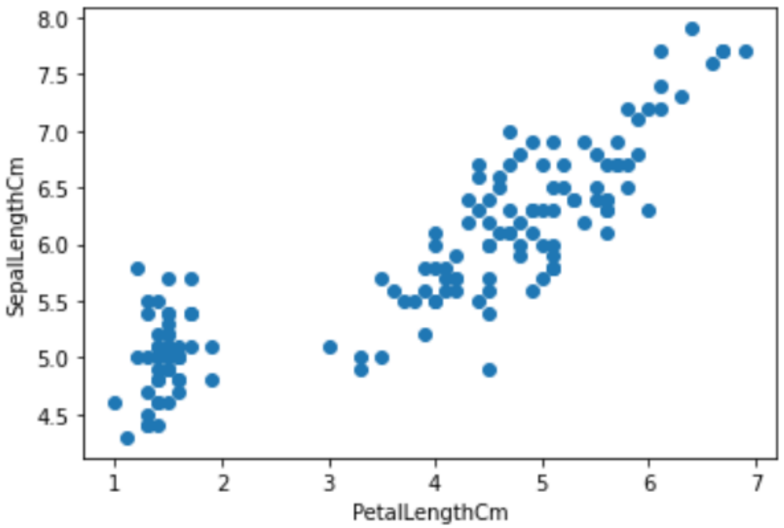
```
In [9]: df = pd.read_csv("C:\\Users\\DELL\\Desktop\\Iris.csv")
        df.head()
```

Out[9]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

- Here, We apply clustering according to *PetalLengthCm* and *SepalLengthCm* columns.
- This is a scatter plot. It shows the relationship between the two features.

```
In [11]: plt.scatter(df.PetalLengthCm,df['SepalLengthCm'])
         plt.xlabel('PetalLengthCm')
         plt.ylabel('SepalLengthCm')

Out[11]: Text(0, 0.5, 'SepalLengthCm')
```



- In this step, we used *kmeans()* function and chose the number of clusters to be 4, then we used *fit_predict()* function to set a cluster number to a certain value

```
In [12]: km = KMeans(n_clusters=4)
         y_predicted = km.fit_predict(df[['PetalLengthCm','SepalLengthCm']])
         y_predicted

Out[12]: array([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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 1, 3, 1, 3, 1, 3, 1, 3, 3, 3, 3, 1, 3, 1,
                3, 3, 1, 3, 1, 3, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 1, 3, 1, 1, 1,
                3, 3, 3, 1, 3, 3, 3, 3, 3, 1, 3, 3, 2, 1, 2, 1, 2, 2, 3, 2, 2, 2,
                1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 1, 2, 2, 2,
                1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1])
```

- Here, we added new column called cluster to the data frame. It assigns each row to a certain cluster.

```
In [13]: df['cluster']=y_predicted
         df.head()

Out[13]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	cluster
0	1	5.1	3.5	1.4	0.2	Iris-setosa	0
1	2	4.9	3.0	1.4	0.2	Iris-setosa	0
2	3	4.7	3.2	1.3	0.2	Iris-setosa	0
3	4	4.6	3.1	1.5	0.2	Iris-setosa	0
4	5	5.0	3.6	1.4	0.2	Iris-setosa	0

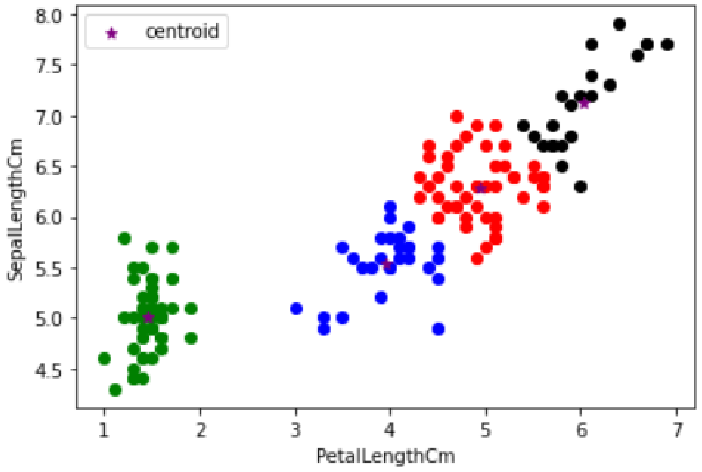
- This function is to calculate the center for each cluster.

```
In [14]: km.cluster_centers_

Out[14]: array([[1.464     , 5.006     ],
                [4.94      , 6.292     ],
                [6.03181818, 7.12272727],
                [3.96071429, 5.53214286]])
```

- In this step, we added each cluster values at a separate data frame, for each cluster we assigned different color and for each center in the cluster a purple color, then we plotted these cluster as a scatter plot

```
In [15]: df1 = df[df.cluster==0]
         df2 = df[df.cluster==1]
         df3 = df[df.cluster==2]
         df4 = df[df.cluster==3]
         plt.scatter(df1.PetalLengthCm,df1['SepalLengthCm'],color='green')
         plt.scatter(df2.PetalLengthCm,df2['SepalLengthCm'],color='red')
         plt.scatter(df3.PetalLengthCm,df3['SepalLengthCm'],color='black')
         plt.scatter(df4.PetalLengthCm,df4['SepalLengthCm'],color='blue')
         plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
         plt.xlabel('PetalLengthCm')
         plt.ylabel('SepalLengthCm')
         plt.legend()
```

```
Out[15]: <matplotlib.legend.Legend at 0x24e187faf40>
```



- Here is the output…

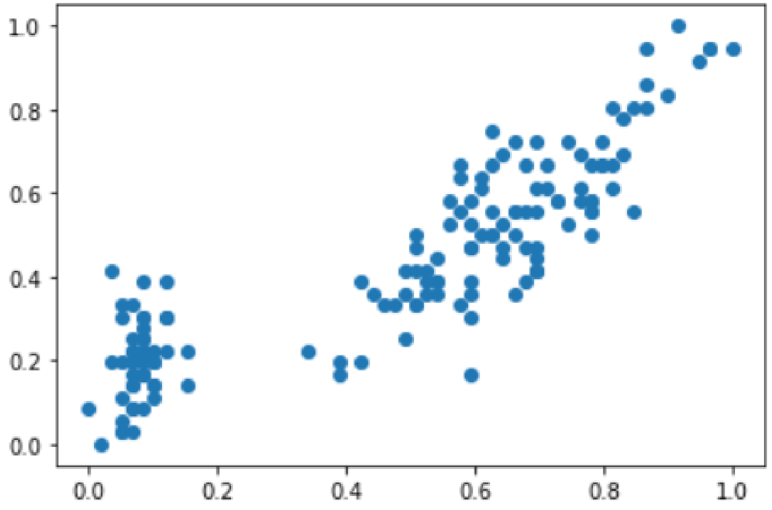
- Then, We use *MinMaxScaler()* function to Transform features by scaling each feature to a given range. the default values of range is [0,1]

```
In [16]: scaler = MinMaxScaler()

         scaler.fit(df[['SepalLengthCm']])
         df['SepalLengthCm'] = scaler.transform(df[['SepalLengthCm']])

         scaler.fit(df[['PetalLengthCm']])
         df['PetalLengthCm'] = scaler.transform(df[['PetalLengthCm']])

         df.head()
```

Out[16]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	cluster
0	1	0.222222	3.5	0.067797	0.2	Iris-setosa	0
1	2	0.166667	3.0	0.067797	0.2	Iris-setosa	0
2	3	0.111111	3.2	0.050847	0.2	Iris-setosa	0
3	4	0.083333	3.1	0.084746	0.2	Iris-setosa	0
4	5	0.194444	3.6	0.067797	0.2	Iris-setosa	0

```
In [17]: plt.scatter(df.PetalLengthCm,df['SepalLengthCm'])
Out[17]: <matplotlib.collections.PathCollection at 0x24e18874d90>
```



- After transforming columns, we repeat all the previous Steps...

In [18]:
```
km = KMeans(n_clusters=4)
y_predicted = km.fit_predict(df[['PetalLengthCm','SepalLengthCm']])
y_predicted
```

Out[18]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 3, 0, 3, 0, 3, 0, 3, 3, 3, 3, 0, 3, 0,
       3, 3, 0, 3, 0, 3, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 0, 3, 0, 0, 0,
       3, 3, 3, 0, 3, 3, 3, 3, 3, 0, 3, 3, 0, 0, 2, 0, 0, 2, 3, 2, 0, 2,
       0, 0, 0, 3, 0, 0, 0, 2, 2, 0, 2, 3, 2, 0, 0, 2, 0, 0, 0, 2, 2, 2,
       0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0])
```

In [19]:
```
df['cluster']=y_predicted
df.head()
```

Out[19]:

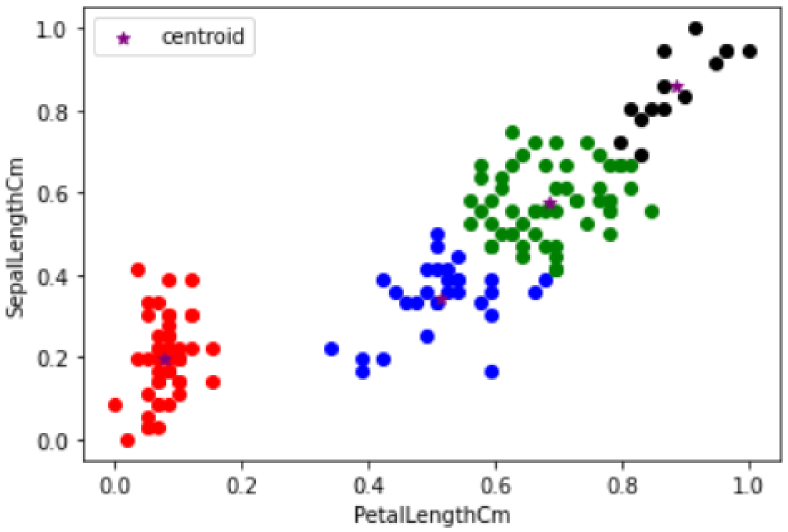
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	cluster
0	1	0.222222	3.5	0.067797	0.2	Iris-setosa	1
1	2	0.166667	3.0	0.067797	0.2	Iris-setosa	1
2	3	0.111111	3.2	0.050847	0.2	Iris-setosa	1
3	4	0.083333	3.1	0.084746	0.2	Iris-setosa	1
4	5	0.194444	3.6	0.067797	0.2	Iris-setosa	1

In [20]:
```
km.cluster_centers_
```

Out[20]:
```
array([[0.68583535, 0.57440476],
       [0.07864407, 0.19611111],
       [0.88619855, 0.85714286],
       [0.51299435, 0.34444444]])
```

In [21]:
```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
df4 = df[df.cluster==3]
plt.scatter(df1.PetalLengthCm,df1['SepalLengthCm'],color='green')
plt.scatter(df2.PetalLengthCm,df2['SepalLengthCm'],color='red')
plt.scatter(df3.PetalLengthCm,df3['SepalLengthCm'],color='black')
plt.scatter(df4.PetalLengthCm,df4['SepalLengthCm'],color='blue')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.xlabel('PetalLengthCm')
plt.ylabel('SepalLengthCm')
plt.legend()
```

Out[21]: <matplotlib.legend.Legend at 0x24e188f52e0>

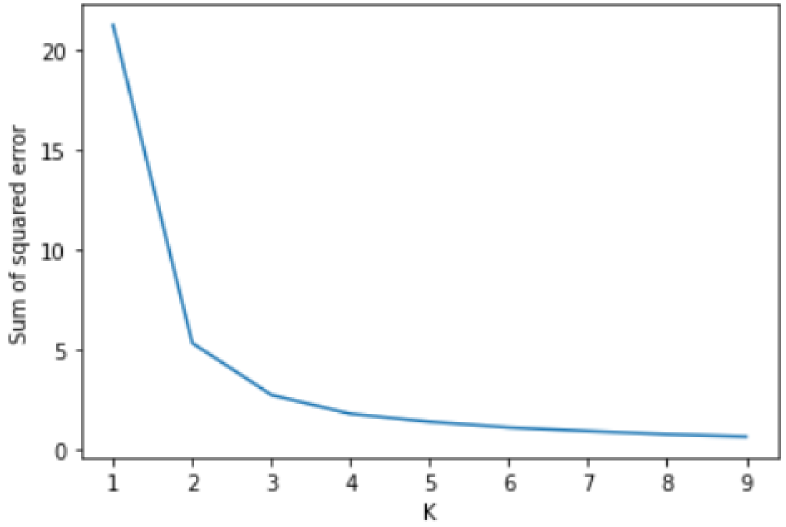


- Then, we use Elbow Plot to know the best value of number of clusters to apply kmeans algorithm

In [22]:
```
sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['PetalLengthCm','SepalLengthCm']])
    sse.append(km.inertia_)
    plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x24e189a5f40>]
```



- We use *sse.append(km.inertia_)* to show mean square error


THANK YOU
☺