IDS PROJECT

<u>CENSUS INCOME (ADULT) DATASET</u>

<u>GROUP MEMBERS</u>:-
Himanshi Sharma (21ucc052)
Vedant Gholap (21ucc112)
Arya Panchlinge (21ucc024)

PROJECT OBJECTIVES:

Applying ML Classification algorithms on the data set and getting inferences from the data.

SOURCE OF DATASET:

https://archive.ics.uci.edu/dataset/2/adult

ABOUT THE DATASET:

The dataset contains information from the 1994 census such as age, education, marital status, occupation, native country, income, etc. The task is to predict whether income exceeds $50K/year based on census data.

DATA DESCRIPTION:

- Dataset Characteristics: Multivariate
- No. of attributes:15
  - Age
  - Workclass
  - Fnlwgt
  - Education
  - Education-num

- ➢ Marital-status
- ➢ Occupation
- ➢ Relationship
- ➢ Race
- ➢ Sex
- ➢ Capital-gain
- ➢ Capital-loss
- ➢ Hours-per-week
- ➢ Native-country
- ➢ Income

- Attribute Characteristics: Integer, Categorical, Binary
- No. of instances: 48842 (rows)
- Presence Of Missing Values: True

IMPLEMENTATION:

In [ ]:
```
pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.10/dist-packages (0.0.3)
```

In [ ]:
```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

from ucimlrepo import fetch_ucirepo

# fetch dataset
adult = fetch_ucirepo(id=2)

# data (as pandas dataframes)
X = adult.data.features
y = adult.data.targets

# metadata
print(adult.metadata)

# variable information
print(adult.variables)
```


```
{'uci_id': 2, 'name': 'Adult', 'repository_url': 'https://archive.ics.uci.edu/dataset/2/adult', 'data_url': 'https://archive.i
cs.uci.edu/static/public/2/data.csv', 'abstract': 'Predict whether income exceeds $50K/yr based on census data. Also known as
"Census Income" dataset. ', 'area': 'Social Science', 'tasks': ['Classification'], 'characteristics': ['Multivariate'], 'num_i
nstances': 48842, 'num_features': 14, 'feature_types': ['Categorical', 'Integer'], 'demographics': ['Age', 'Income', 'Educatio
n Level', 'Other', 'Race', 'Sex'], 'target_col': ['income'], 'index_col': None, 'has_missing_values': 'yes', 'missing_values_s
ymbol': 'NaN', 'year_of_dataset_creation': 1996, 'last_updated': 'Mon Aug 07 2023', 'dataset_doi': '10.24432/C5XW20', 'creator
s': ['Barry Becker', 'Ronny Kohavi'], 'intro_paper': None, 'additional_info': {'summary': 'Extraction was done by Barry Becker
from the 1994 Census database.  A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) &&
(AGI>100) && (AFNLWGT>1)&& (HRSWK>0))\r\n\r\nPrediction task is to determine whether a person makes over 50K a year.\r\n', 'pu
rpose': None, 'funded_by': None, 'instances_represent': None, 'recommended_data_splits': None, 'sensitive_data': None, 'prepro
cessing_description': None, 'variable_info': 'Listing of attributes:\r\n\r\n>50K, <=50K.\r\n\r\nage: continuous.\r\nworkclass:
Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.\r\nfnlwgt: continuou
s.\r\neducation: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-
4th, 10th, Doctorate, 5th-6th, Preschool.\r\neducation-num: continuous.\r\nmarital-status: Married-civ-spouse, Divorced, Never
-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.\r\noccupation: Tech-support, Craft-repair, Other-servi
ce, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-mov
ing, Priv-house-serv, Protective-serv, Armed-Forces.\r\nrelationship: Wife, Own-child, Husband, Not-in-family, Other-relative,
Unmarried.\r\nrace: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.\r\nsex: Female, Male.\r\ncapital-gain: contin
uous.\r\ncapital-loss: continuous.\r\nhours-per-week: continuous.\r\nnative-country: United-States, Cambodia, England, Puerto-
```


First, we imported data from the UCI dataset website and saved it in a CSV file.

```
In [ ]:
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df=pd.read_csv('/content/adult_data.csv')  #Loading dataset
```

All important libraries such as Numpy, Pandas, etc are imported.

Numpy is used to perform a wide variety of mathematical operations on arrays.

Pandas is useful for data manipulation and analysis.

Seaborn is useful for making statistical plots/graphics in Python.

Matplotlib is useful for making static and interactive visualizations.

Through pd.read_csv the data is loaded into df dataframe.

```
df.head()  #1st 5 rows
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba

To see 1st 5 rows we used df.head() command.

```
In [ ]: df.isnull().sum()     #CHECKING FOR NULL VALUES
```

```
Out[ ]: age                 0
        workclass         963
        fnlwgt              0
        education           0
        education-num       0
        marital-status      0
        occupation        966
        relationship        0
        race                0
        sex                 0
        capital-gain        0
        capital-loss        0
        hours-per-week      0
        native-country    274
        income              0
        dtype: int64
```

We used df.isnull().sum() to get the count of null values in all columns.

```
In [ ]: df.tail()    #last 5 rows
```

Out[ ]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	c
48837	39	Private	215419	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	0	0	36	
48838	64	NaN	321403	HS-grad	9	Widowed	NaN	Other-relative	Black	Male	0	0	40	
48839	38	Private	374983	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	50	
48840	44	Private	83891	Bachelors	13	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	Male	5455	0	40	
48841	35	Self-emp-inc	182148	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	60	

To see the last 5 rows we used df.tail() command.

```
In [ ]: df.shape

Out[ ]: (48842, 15)

In [ ]: #dataset has 48842 rows and 15 columns.
```

df.shape() return no. of rows and columns in the dataset.

```
In [ ]: df.dtypes  #data is a mixture of numerical and categorical values.
```

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

df.dtypes is used to get the datatype of all columns.

```
In [ ]: df.dropna(subset=['workclass','occupation','native-country'],inplace=True)
```

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

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

```
In [ ]: #workclass,occupation and native-country are columns with categorical
        #values and so we remove null values from column dataset.
        #963,966,273 null rows are very small as compared to 48000 rows
        #so we remove them.
```

Data Preprocessing: Since the columns work-class, occupation and native-country had null values and all these columns contain categorical values, we drop those rows from the dataset.

In [ ]:
```
df.describe()
```

Out[ ]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	47621.000000	4.762100e+04	47621.000000	47621.000000	47621.000000	47621.000000
mean	38.640684	1.897271e+05	10.090821	1091.137649	87.853489	40.600050
std	13.558961	1.055695e+05	2.568320	7487.228336	404.010612	12.260345
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175840e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.782820e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.377200e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.000000

df.describe() gives count, mean, standard deviation, minimum, 1st quartile(25%), median, 3rd quartile(75%) and maximum values in the numerical columns of dataset.

Also, while going through the dataset we found '?' values in some columns of the dataset which need to be preprocessed.

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

Out[ ]:
```
Private             33717
Self-emp-not-inc     3838
Local-gov            3126
State-gov            1965
?                    1836
Self-emp-inc         1688
Federal-gov          1423
Without-pay            21
Never-worked            7
Name: workclass, dtype: int64
```

In [ ]:
```
#? is present in 1836 rows in workclass.
```

In [ ]:
```
df['occupation'].value_counts()
```

Out[ ]:
```
Prof-specialty       6110
Craft-repair         6089
Exec-managerial      6058
Adm-clerical         5589
Sales                5474
Other-service        4891
Machine-op-inspct    3006
Transport-moving     2341
Handlers-cleaners    2066
?                    1843
Farming-fishing      1485
Tech-support         1436
Protective-serv       981
Priv-house-serv       238
Armed-Forces           14
Name: occupation, dtype: int64
```

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

Out[ ]:
```
United-States                 42958
Mexico                          936
?                               583
Philippines                     293
Germany                         202
Puerto-Rico                     180
Canada                          177
El-Salvador                     153
India                           147
Cuba                            136
England                         123
```

In [ ]:
```
df.columns
```

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

In [ ]:
```
df['marital-status'].value_counts()
```
Out[ ]:
```
Married-civ-spouse       21966
Never-married            15555
Divorced                  6526
Separated                 1497
Widowed                   1443
Married-spouse-absent      600
Married-AF-spouse           34
Name: marital-status, dtype: int64
```
In [ ]:
```
df['sex'].value_counts()
```
Out[ ]:
```
Male      31937
Female    15684
Name: sex, dtype: int64
```
In [ ]:
```
df['race'].value_counts()
```
Out[ ]:
```
White                 40786
Black                  4535
Asian-Pac-Islander     1447
Amer-Indian-Eskimo      460
Other                   393
Name: race, dtype: int64
```
In [ ]:
```
df['income'].value_counts()
```
Out[ ]:
```
<=50K     24720
<=50K.    11360
>50K       7841
>50K.      3700
Name: income, dtype: int64
```

We can get all column header names by df. columns.

df[<column-name>].value_counts() is used to get different values in columns along with their counts.

We checked for '?' for this in all columns.

'?' is found in workclass, occupation, and native-country columns. Also, they are found in large numbers so their values need to be imputed by some suitable values. For categorical variables, mode is the best value for replacing/missing values.

In [ ]:
```
#'?' was found in workclass,occupation and native country columns.
df.mode()   #mode value for each column.
```
Out[ ]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	i
0	36	Private	203488	HS-grad	9	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	40	United-States	

We found the mode for all columns and replaced all '?' by the corresponding column's Mode values.

```
In [ ]: df['workclass']=df['workclass'].replace('?','Private')
        df['occupation']=df['occupation'].replace('?','Prof-specialty')
        df['native-country']=df['native-country'].replace('?','United-States')
```

```
In [ ]: df['workclass'].value_counts()  #checking
```

```
Out[ ]: Private             35553
        Self-emp-not-inc     3838
        Local-gov            3126
        State-gov            1965
        Self-emp-inc         1688
        Federal-gov          1423
        Without-pay            21
        Never-worked            7
        Name: workclass, dtype: int64
```

In the income column, we found 4 different options that needed to be corrected, So we used df['income'].replace() function to correct them.

```
In [ ]: df['income'].value_counts()
```

```
Out[ ]: <=50K     24720
        <=50K.    11360
        >50K       7841
        >50K.      3700
        Name: income, dtype: int64
```

```
In [ ]: #df['column name'] = df['column name'].replace(['old value'], 'new value')
        df['income']=df['income'].replace(['<=50K.'],'<=50K')
```

```
In [ ]: df['income']=df['income'].replace(['>50K.'],'>50K')
```

```
In [ ]: df['income'].value_counts()
```

```
Out[ ]: <=50K     36080
        >50K      11541
        Name: income, dtype: int64
```

```
In [ ]: #we preprocessed the income column values representing (<=50K. and <=50K)
        #(>50K and >50K.) as same values.
```

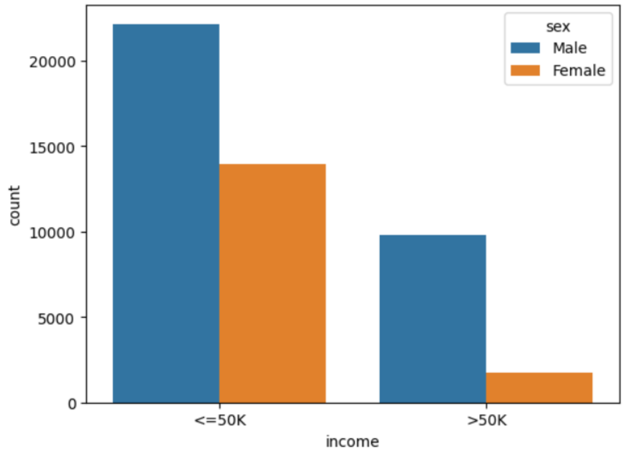
Above we replaced <=50k. by <=50k and >50k. with >50k

Our data preprocessing step is completed.

Now we perform Data Analysis And make many visualizations.

In [ ]:
```
sns.countplot(df,x='income',hue='sex')
```

Out[ ]: <Axes: xlabel='income', ylabel='count'>

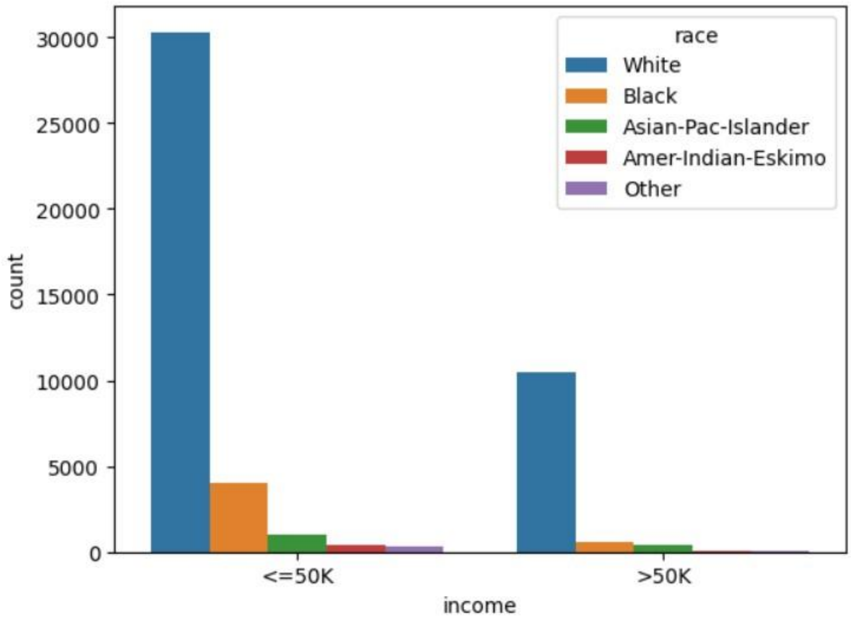


In [ ]:
```
#countplot of income w.r.to gender
#More no of males have income more than female in the 2 categories of income.
```

The countplot shows the count of males & females with respect to different income levels. Males have more income as compared to females.

In [ ]:
```
sns.countplot(df,x='income',hue='race')
```

Out[ ]: <Axes: xlabel='income', ylabel='count'>



In [ ]:
```
#Count of whites are more than any other race in the 2 categories of income.
```

The countplot shows the count of people of different race vs categories of income.

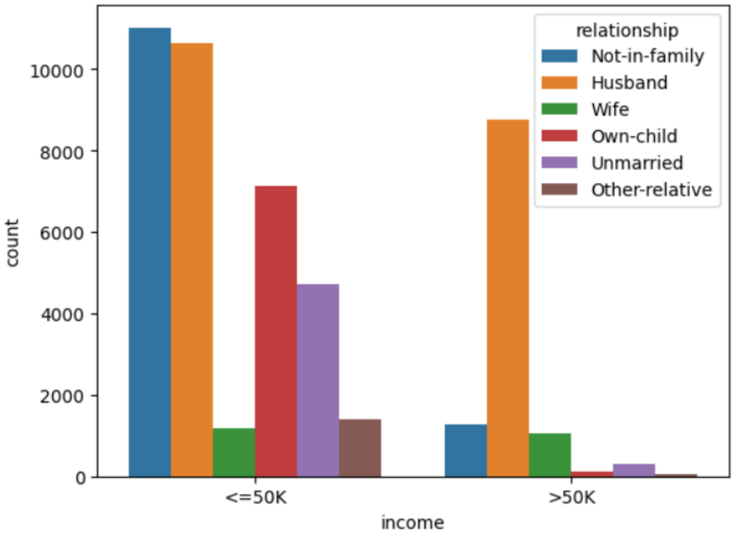
While raced people have the highest count in 2 categories of income.

In [ ]:

```
sns.countplot(df,x='income',hue='relationship')
```

Out[ ]:

```
<Axes: xlabel='income', ylabel='count'>
```



In [ ]:

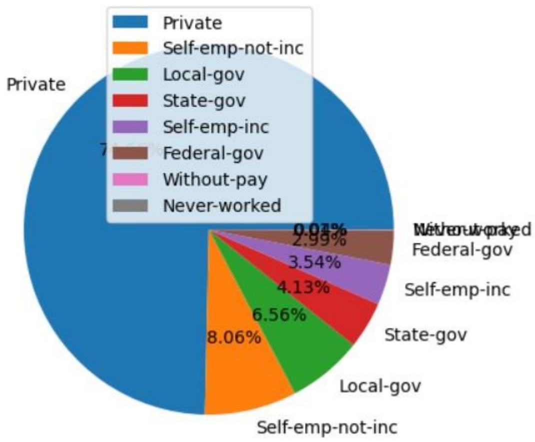
```
#In <=50K count of peole not-in-family is the maximum.
#In >50K count of relationship husband is more.
```

The count plot shows the count of people in different relationships vs income.

In <=50k level Not in family has the highest frequency and for >50k level of income Husband has the maximum frequency.

In [ ]:

```
#using pie chart
gender = df['workclass'].value_counts()
#  Plot the pie chart
labels=["Private","Self-emp-not-inc","Local-gov","State-gov","Self-emp-inc","Federal-gov","Without-pay","Never-worked"]
plt.pie(gender, labels=labels, autopct='%1.2f%%')
#plt.axis('equal')
plt.legend()
plt.show()
```



In [ ]:

```
#more than 70% of population works in private sector.
```

The pie chart shows the distribution of the population in the work class column. We find that more than 70% of the population works in the private sector. Nearly 13% of people work in local-gov, state-gov or federal-gov sectors.

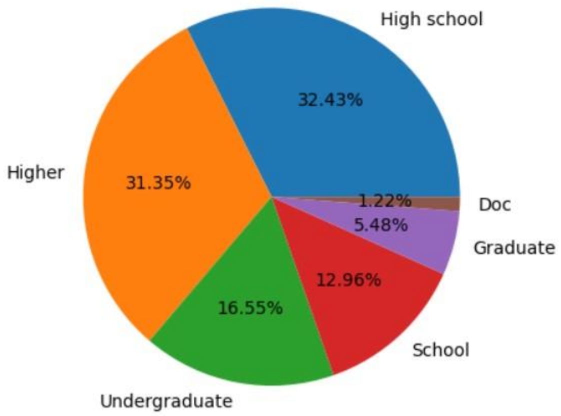
```
In [ ]: df['education'].value_counts()

Out[ ]: HS-grad         15444
        Some-college    10512
        Bachelors        7881
        Masters          2610
        Assoc-voc        2034
        11th             1746
        Assoc-acdm       1566
        10th             1336
        7th-8th           912
        Prof-school       819
        9th               735
        12th              633
        Doctorate         582
        5th-6th           494
        1st-4th           239
        Preschool          78
        Name: education, dtype: int64
```

Now we want to see distribution in the education column but it has many values so we group many values into one group only so that distribution and analysis are easily understandable.

```
In [ ]: #we have many columns which can be grouped into one column together(Feature Engineering).
        df.education= df.education.replace(['Preschool', '1st-4th', '5th-6th', '7th-8th', '9th','10th', '11th', '12th'], 'School')
        df.education= df.education.replace('HS-grad', 'High school')
        df.education= df.education.replace(['Assoc-voc', 'Assoc-acdm', 'Prof-school', 'Some-college'], 'Higher')
        df.education= df.education.replace('Bachelors', 'Undergraduate')
        df.education= df.education.replace('Masters', 'Graduate')
        df.education= df.education.replace('Doctorate', 'Doc')

In [ ]: df['education'].value_counts()

Out[ ]: High school      15444
        Higher           14931
        Undergraduate     7881
        School            6173
        Graduate          2610
        Doc                582
        Name: education, dtype: int64
```

We combined/replaced pre-school,1st-4th,5th-6th,7th-8th,9th-10th,11th,12th into one variable of ‘school’. HS-grad is replaced with ‘High school’. Other columns are also combined or their names are changed/replaced.

In [ ]:
```
#using pie chart
education = df['education'].value_counts()
#  Plot the pie chart
labels=["High school","Higher","Undergraduate","School","Graduate","Doc"]
plt.pie(education, labels=labels, autopct='%1.2f%%')
#plt.axis('equal')
plt.show()
```



In [ ]:
```
#pie-chart shows distribution of education status of population.
```

The pie chart shows the distribution of education levels in the population with 32% of the population in high school, 17% studying undergraduate, 13% of the population in school, and so on.

In [ ]:
```
#in marital-status also we can group many columns into one(Feature Engineering).
df['marital-status'].value_counts()
```

Out[ ]:
```
Married-civ-spouse       21966
Never-married            15555
Divorced                  6526
Separated                 1497
Widowed                   1443
Married-spouse-absent      600
Married-AF-spouse           34
Name: marital-status, dtype: int64
```

In [ ]:
```
df['marital-status']= df['marital-status'].replace(['Married-civ-spouse', 'Married-AF-spouse'], 'Married')
df['marital-status']= df['marital-status'].replace(['Never-married'], 'Unmarried')
df['marital-status']= df['marital-status'].replace(['Divorced', 'Separated','Widowed','Married-spouse-absent'], 'Other')
```

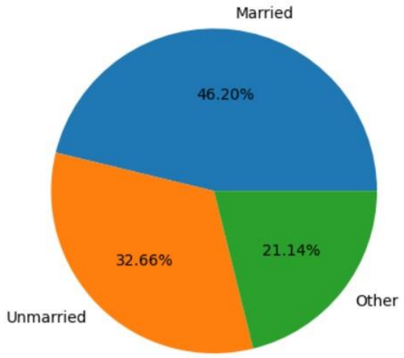
In [ ]:
```
df['marital-status'].value_counts()
```

Out[ ]:
```
Married      22000
Unmarried    15555
Other        10066
Name: marital-status, dtype: int64
```

We grouped various values of the marital-status column into 1 column as many values pointed to the same column values. We made divorced, separated, windowed, and married-spouse-absent into ‘Other’ for easy distribution and understanding of the data.

In [ ]:

```
#using pie chart
marital = df['marital-status'].value_counts()
#  Plot the pie chart
labels=["Married","Unmarried","Other"]
plt.pie(marital, labels=labels, autopct='%1.2f%%')
#plt.axis('equal')
plt.show()
```



In [ ]:

```
#pie-chart shows distribution of marital status of population.
```

The pie chart shows the distribution of marital status in the population with 46% of the population being married, 33% unmarried, and 21% in the ‘Other’ category.

In [ ]:

```
#Correlation
df.corr()
```

```
<ipython-input-201-7e6697e05da5>:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
```

Out[ ]:

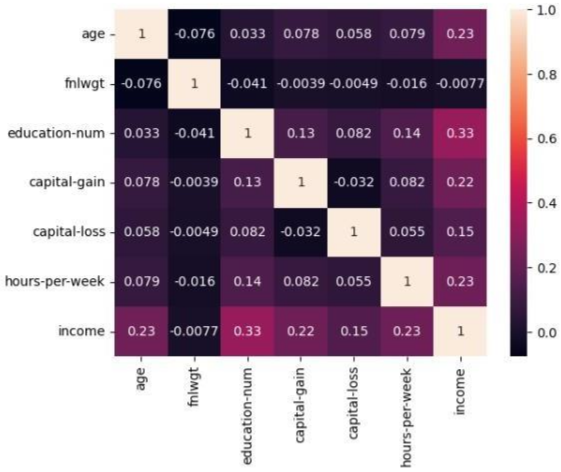
	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	income
age	1.000000	-0.075937	0.033327	0.078006	0.057772	0.079306	0.231986
fnlwgt	-0.075937	1.000000	-0.040933	-0.003890	-0.004949	-0.016144	-0.007687
education-num	0.033327	-0.040933	1.000000	0.125569	0.081799	0.143727	0.332845
capital-gain	0.078006	-0.003890	0.125569	1.000000	-0.031691	0.082279	0.222509
capital-loss	0.057772	-0.004949	0.081799	-0.031691	1.000000	0.055360	0.149078
hours-per-week	0.079306	-0.016144	0.143727	0.082279	0.055360	1.000000	0.227527
income	0.231986	-0.007687	0.332845	0.222509	0.149078	0.227527	1.000000

df. corr() is used to get correlation values among different columns or attributes of the dataset. This can be plotted into a heatmap by using the Seaborn library.

```
In [ ]: sns.heatmap(df.corr(),annot=True)
```

```
<ipython-input-202-8df7bcac526d>:1: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only
valid columns or specify the value of numeric_only to silence this warning.
```

```
Out[ ]: <Axes: >
```



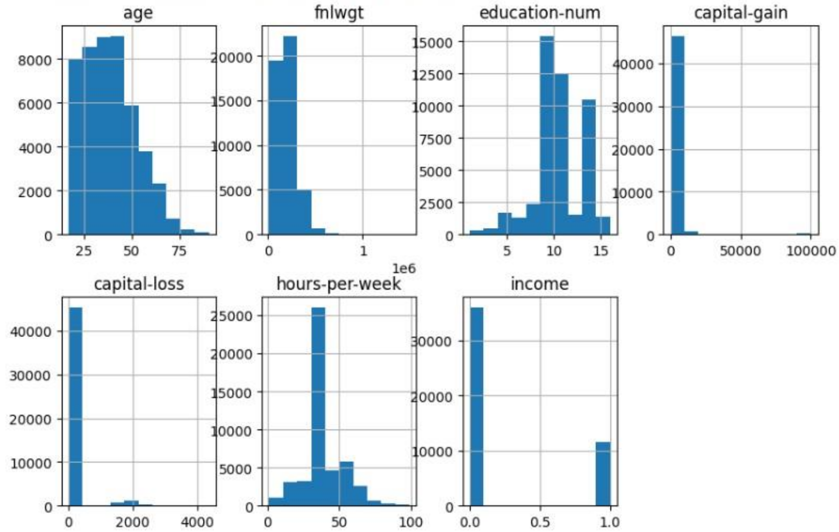

```
In [ ]: #The heatmap shows correlation among various columns of the dataset.
```

```
In [ ]: #From heatmap we find all the variables are weakly correlated
        #to each other as correlation coefficients or values are < 0.5 or negative
        #in most of the cases.
```

The heatmap shows the correlation among different columns and we find the correlation coefficients are < 0.5 or negative in some cases, so we conclude that the variables/columns are weakly correlated.

```
In [ ]: #histogram showing count in different columns.
        df.hist(figsize=(10,10),layout=(3,4),sharex=False)
        #all histograms to be arranged in 3 rows and 4 columns.
```

```
Out[ ]: array([[<Axes: title={'center': 'age'}>,
                <Axes: title={'center': 'fnlwgt'}>,
                <Axes: title={'center': 'education-num'}>,
                <Axes: title={'center': 'capital-gain'}>],
               [<Axes: title={'center': 'capital-loss'}>,
                <Axes: title={'center': 'hours-per-week'}>,
                <Axes: title={'center': 'income'}>, <Axes: >],
               [<Axes: >, <Axes: >, <Axes: >, <Axes: >]], dtype=object)
```




```
In [ ]: #most of the age-group lies in the age-group of 30 to 45.
        #maximum people work between 25 to 50 hours per week and few people work
        #for 75 hrs per week.
        #for income 0 represents <=50K and 1 represents >50K
        #so most people have income <=50K as is seen from histogram.
```

df.hist() plots histogram and here we plotted to count distribution for different columns. From the graphs, we can conclude that the majority population lies in the age group of 30 to 45. Maximum people work 25 to 50 hours per week but few people also work for 75 hours per week. For income 0 represents <=50k and 1 represents>50k income level.

In [ ]:
```
#The 2 categories of income
df['income'].value_counts()
```

Out[ ]:
```
<=50K    36080
>50K     11541
Name: income, dtype: int64
```

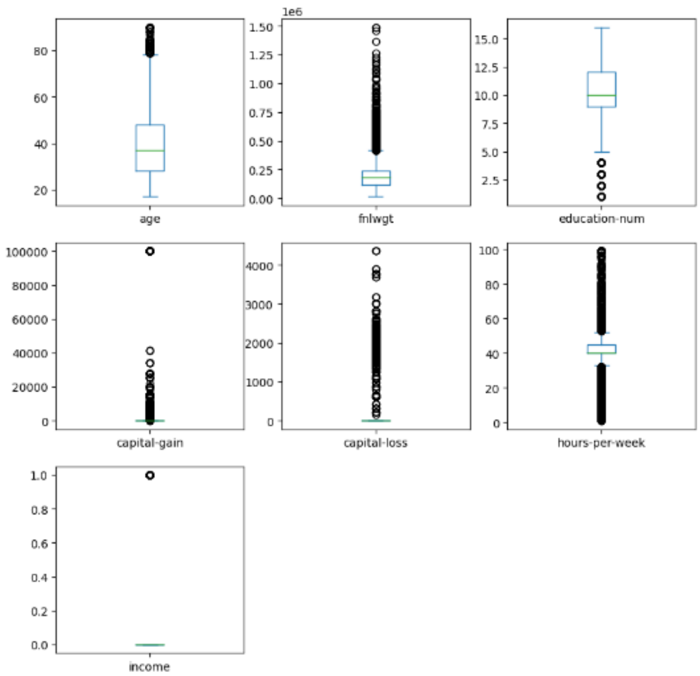
In [ ]:
```
#converting into 0 and 1 so that binary classification can be done.
df.income = df.income.replace('<=50K', 0)
df.income = df.income.replace('>50K', 1)
```

We changed it into 0s and 1s as it would be useful for binary classification of the 2 income levels(<=50k and >50k). So, by the graphs majority of people have income <= 50k.

In [ ]:
```
#most of the age-group lies in the age-group of 30 to 45.
#maximum people work between 25 to 50 hours per week and few people work
#for 75 hrs per week.
#for income 0 represents <=50K and 1 represents >50K
#so most people have income <=50K as is seen from histogram.
```

In [ ]:
```
df.plot(kind='box',figsize=(10,10),layout=(3,3),sharex=False,subplots=True)
```

Out[ ]:
```
age                    Axes(0.125,0.653529;0.227941x0.226471)
fnlwgt              Axes(0.398529,0.653529;0.227941x0.226471)
education-num       Axes(0.672059,0.653529;0.227941x0.226471)
capital-gain           Axes(0.125,0.381765;0.227941x0.226471)
capital-loss        Axes(0.398529,0.381765;0.227941x0.226471)
hours-per-week      Axes(0.672059,0.381765;0.227941x0.226471)
income                    Axes(0.125,0.11;0.227941x0.226471)
dtype: object
```

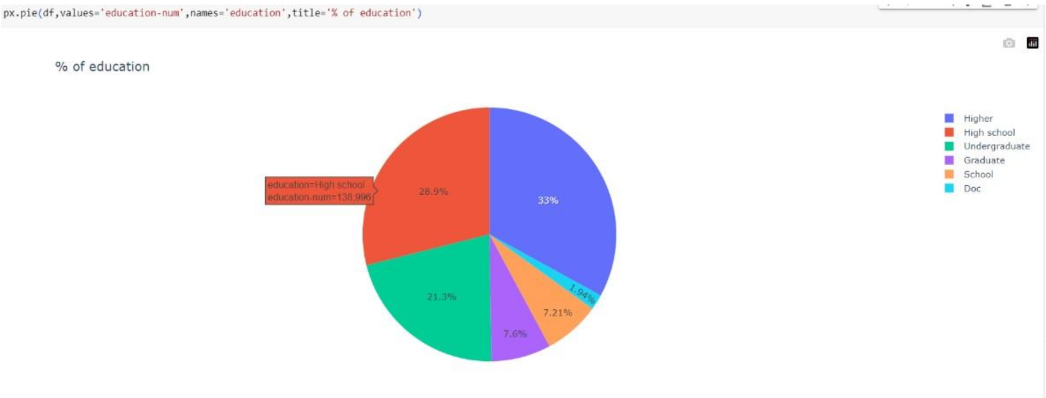


In [ ]:
```
#The boxplots are the best tool to represent outliers.
#The dots outside the boxplots represent the outliers.
#age has outliers as some people are aged more than 80 years.
#income has outliers as some people have income >50K.
#For hours-per-week most people work 40 hours per week
#some work more and some less as is clear from outliers above and below.
```

df.plot(kind='box',..) plots boxplots showing the distribution of different columns of the dataset. Based on the boxplots above we can conclude that the Age column has outliers as some people are aged > 80 years. The income column also has outliers as some people have income >50k while the majority population has income <50k. Also in hours-per-week, we find most people work 40 hours per week but some people work much more and some very much less as is seen from outliers.

```
In [ ]: import plotly.express as px
```

```
In [ ]: px.pie(df,values='education-num',names='education',title='% of education')
```



Plotly is used to create beautiful interactive web-based visualizations.

Based on the education-num column the 'Higher',' High school' and 'Undergraduate' have higher shares.

Now we move to **Machine Learning Classification**.

```
In [ ]: #separating target column i.e. income from other columns
x=df.drop(['income'],axis=1)
y=df['income']
```

```
In [ ]: from sklearn.preprocessing import StandardScaler,LabelEncoder
df1=df.copy()
df1=df1.apply(LabelEncoder().fit_transform)
#Label Encoder is used to convert categorical variables into numerical form.
df1.head()
```

Out[ ]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	incc
0	22	6	3377	5	12	2	0	1	4	1	27	0	39	38	
1	33	5	3698	5	12	0	3	0	4	1	0	0	12	38	
2	21	3	17968	2	8	1	5	1	4	1	0	0	39	38	
3	36	3	19585	4	6	0	5	0	2	1	0	0	39	38	
4	11	3	24894	5	12	0	9	5	2	0	0	0	39	4	

```
In [ ]: #we see all categorical variables converted into numerical forms.
```

First, we separated the target column i.e. income from the dataset and stored it in 'y'. We use a Label Encoder as the data has categorical values that cannot be understood by the ML algorithm so to convert the categorical values to numerical variables we use the label encoder.

```
In [ ]: ss= StandardScaler().fit(df1.drop('income', axis=1))
        X= ss.transform(df1.drop('income', axis=1))
        y= df['income']
        #StandardScaler removes the mean and scales each feature/variable to unit variance.

In [ ]: #In StandardScaler 1st we fit then we transform the data.
```

Standard Scaler removes the mean and fits each variable to unit variance.

For a standard scaler we first fit and then transform the data.

```
In [ ]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

In [ ]: #70% of the dataset given for training and 30% will be used for testing.
```

Using sci-kit-learn's train_test_split we split 70% of data into the training set and 30% into the test set. The y contains the target column and x has all other columns and random_state = 0 fixes the distribution of split and we can give any no. to the random state.

The target 'income' column has 2 values 0 and 1 denoting income <= 50k and income>=50k which is a case of binary classification. For this case, we can use various classification models such as Logistic Regression and Ensemble methods such as Random Forests.

<u>First, we proceed by Logistic Regression</u>.

```
In [ ]: from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import accuracy_score

In [ ]: lr=LogisticRegression()

In [ ]: model=lr.fit(X_train,y_train) #training

In [ ]: prediction=model.predict(X_test)

In [ ]: print("Accuracy of training data: {:,.5f}".format(lr.score(X_train,y_train)))
        print("Accuracy of test data: {:,.5f}".format(lr.score(X_test,y_test)))

        Accuracy of training data: 0.83845
        Accuracy of test data: 0.84104

In [ ]: #Both have nearly the same accuracy score for logistic regression.
```

From sklearn.linear_model we imported the LogisticRegression and from sklearn.metrics we imported accuracy_score.

The evaluation metric for classification is the ***Accuracy score*** which is (no. of correct predictions/total no. of input data points)*100%.

We called the lr=LogisticRegression() function. Then we fit the training dataset into that model and later predicted the results from the test dataset.

Using lr.score() we got the accuracy score for the training and test dataset.

The Accuracy was found to be nearly the same i.e. 84% .

```
In [ ]: #df.to_csv(r'Path where you want to store the exported CSV file\File Name.csv', index=False)
```

```
In [ ]: df1.to_csv(r'/content/df1data.csv',index=False)
```

```
In [ ]: input=(35,5,17376,2,8,0,3,0,4,1,0,0,44,38)#ONLY FEATURES OR INPUT VARIABLES HERE
        #NO TARGET VARIABLE PRESENT ABOVE.
        input_as_numpy_array=np.asarray(input)
        #reshaping the numpy array as we are predicting for only one instance.
        input_reshaped=input_as_numpy_array.reshape(1,-1)
        prediction=model.predict(input_reshaped)
        print(prediction)
        if(prediction[0]==0):
          print('<=50K')
        else:
          print('>50K')

[1]
>50K
```

```
In [ ]: #Above we made a predictive system using logistic regression
        #taking input as all columns other than income and output as
        #0 or 1 for income.
        #It has 84% accuracy.
        #Above it predicts correct output of 1.
        #35,5,17376,2,8,0,3,0,4,1,0,0,44,38,1
```

Above we stored the label-encoded and standard-scaled dataset in a CSV file and took one row from it then we predicted its income based on the model we built and we found the correct prediction that matches the dataset values.

Second, we use Random Forest Classifier.

Ensemble learning is a supervised learning technique used in ML to improve overall performance by combining the predictions from multiple models.

Random Forest is an ensemble learning method where multiple decision trees are constructed and then they are merged to get more accurate predictions. merging for regression is the average of all predictions of different decision trees in the random forest while merging for classification is the majority of the prediction made by different decision trees in the random forest.

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
```

```
In [ ]: rfc=RandomForestClassifier()
        model=rfc.fit(X_train,y_train)
        prediction1=model.predict(X_test)
```

```
In [ ]: print("Accuracy of training data: {:,.5f}".format(rfc.score(X_train,y_train)))
        print("Accuracy of test data: {:,.5f}".format(rfc.score(X_test,y_test)))

        Accuracy of training data: 0.99991
        Accuracy of test data: 0.85623
```

```
In [ ]: #By random forest classifier we get 85.9% accuracy which is better than
        #logistic regression.
```

From sklearn.ensemble we imported RandomForestClassifier.

We called RandomForestClassifier() in rfc.

Then we train rfc on training data and then prediction is made on the test data.

The accuracy score is found by rfc.score() for training and test sets and we find 86% accuracy for test set.

The Random Forest Classifier Model performs slightly better than the Logistic Regression Model.

The confusion matrix is a table with 2 rows and 2 columns that reports the no. of true positives, true negatives, false positives, and false negatives.

The classification report displays precision, recall, F1-score, and support scores.

precision=accuracy of positive predictions=(true positives)/(true positives + false positives)

recall=fraction of positives that were correctly identified=(true positives)/(true positives + false negatives)

F1 score=weighted harmonic mean of precision and recall=(2*precision*recall)/(precision+recall)

support=no. of actual occurrences of the class in the specified dataset.

```
from sklearn.metrics  import  confusion_matrix
from sklearn.metrics import classification_report
```

```
In [ ]: print(confusion_matrix(y_test, prediction1))
```

```
[[10133   753]
 [ 1301  2100]]
```

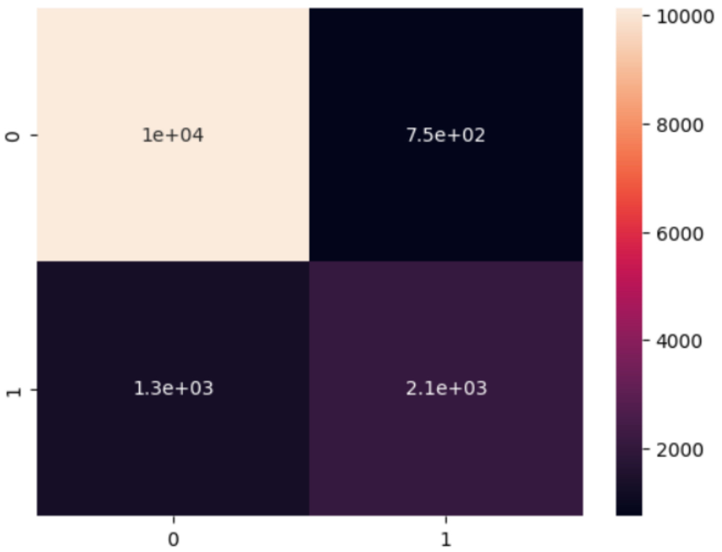
```
In [ ]: print(classification_report(y_test, prediction1))
```

```
              precision    recall  f1-score   support

           0       0.89      0.93      0.91     10886
           1       0.74      0.62      0.67      3401

    accuracy                           0.86     14287
   macro avg       0.81      0.77      0.79     14287
weighted avg       0.85      0.86      0.85     14287
```

We printed the confusion matrix and classification report on the test dataset.

```
In [ ]: cf_matrix=confusion_matrix(y_test, prediction1)
        sns.heatmap(cf_matrix,annot=True)
```

```
Out[ ]: <Axes: >
```




```
In [ ]: # confusion matrix is of the form
        #     TP | FN
        #     FP | TN
```

Above we printed a heatmap for the confusion matrix by the Seaborn Library.

```
In [ ]: # confusion matrix is of the form
        #     TP | FN
        #     FP | TN
```

```
In [ ]: print(confusion_matrix(y_test, prediction1))
```

```
[[10133   753]
 [ 1301  2100]]
```

```
In [250…: print("Precision= ",(10133)/(10133+1301))
```

```
Precision=  0.8862165471401084
```

```
In [251…: print("Recall= ",(10133)/(10133+753))
```

```
Recall=  0.9308285871761895
```

```
In [252…: #above is for class 0
          #now we do for class 1.
```

```
In [253…: print("Precision= ",(2100)/(2100+753))
```

```
Precision=  0.7360672975814931
```

```
In [254…: print("Recall= ",(2100)/(2100+1301))
```

```
Recall=  0.6174654513378418
```

```
In [255…: print("f1 score for class 0= ",2*0.8862165471401084*0.9308285871761895/(0.9308285871761895+0.8862165471401084))
```

```
f1 score for class 0=  0.9079749103942653
```

```
In [257…: print("f1 score for class 1= ",2*0.7360672975814931*0.6174654513378418/(0.6174654513378418+0.7360672975814931))
```

```
f1 score for class 1=  0.6715701950751519
```

We calculated the precision, recall, and f1 score for the 2 classes of income and found them to be exactly the same as those in the classification report.

CONCLUSIONS:

The ‘Census Income’ or ‘Adult’ dataset is a classification-related dataset.

We performed and made many visualizations on it and analyzed and got valuable insights from the dataset.

We used ML classification algorithms such as Logistic Regression (as the target column i.e. income column consists of binary values) and Random Forest Classifier for better accuracy.

We got the test data accuracy score of **84% by Logistic Regression and 86% by Random Forest Classifier**.

LINK TO THE PROJECT REPOSITORY :

https://github.com/himanshi-154/IDSproject