TEACHNOOK

MAJOR PROJECT

(FEB – BATCH)

MADE BY GROUP– DS-02-SPB3

EXTERNAL DATA ANALYSIS

Dataset: -

PROBLEM STATEMENT: -

Q - In this kernel, we try to make some predictions where we have to determine whether a person makes an income over 50K in a year. We implement Random Forest Classification with Python and Scikit-Learn. So, to answer this question, we had built some Random Forest classifier to predict whether a person can make their income over 50K in a Year or not?

```
[29] import pandas as pd
     import numpy as np
     import seaborn as sns
     import sklearn as sk
     import matplotlib.pyplot as plt
```

```
[2] path = "/content/drive/MyDrive/data_csv.csv"
```

```
df = pd.read_csv(path)
print(df)
```

```
   age         workclass  fnlwgt  education  education-num  \
0   39         state-gov   77526  bachelors             13
1   50  self-emp-not-inc   83311  bachelors             13
2   38           private  215646    hs-grad              9
3   53           private  234721       11th              7
4   28           private  338409  bachelors             13

       marital-status         occupation   relationship   race     sex
0       never-married       adm-cierical  not-in-family  white    male
1  married-civ-spouse    exec-managerial        husband  white    male
2            divorced  handlers-cleaners  not-in-family  white    male
3  married-civ-spouse  handlers-cleaners        husband  black    male
4  married-civ-spouse     prof-specially           wife  black  female

   capital-gain  capital-loss  hours-per-week native-country income
0          2174             0              40             US  <=50k
1             0             0              13             US  <=50k
2             0             0              40             US  <=50k
3             0             0              40             US  <=50k
4             0             0              40           Cuba  <=50k
```

GOING THROUGH BASIC INFORMATIONS: -

#1 Print the ‘Workclass” column from the dataset.

```
#print workclass column form the dataset

print(df['workclass'].unique())
```

```
['state-gov' 'self-emp-not-inc' 'private']
```

#2 Show the dimensions of the given data set.

```
#view the dimensions of the data
print('The shape of the dataset : ', df.shape)
```

```
The shape of the dataset :  (5, 15)
```

#3 show the preview the dataset.

```
#Priview the dataset
df.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	state-gov	77526	bachelors	13	never-married	adm-cierical	not-in-family	white	male	2174	0	40	US	<=50k
1	50	self-emp-not-inc	83311	bachelors	13	married-civ-spouse	exec-managerial	husband	white	male	0	0	13	US	<=50k
2	38	private	215646	hs-grad	9	divorced	handlers-cleaners	not-in-family	white	male	0	0	40	US	<=50k
3	53	private	234721	11th	7	married-civ-spouse	handlers-cleaners	husband	black	male	0	0	40	US	<=50k
4	28	private	338409	bachelors	13	married-civ-spouse	prof-specially	wife	black	female	0	0	40	Cuba	<=50k

#4 Renaming columns.

```
#Rename the column names
col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',
             'marital_status', 'occupation', 'relationship',
             'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week',
             'native_country', 'income']

df.columns = col_names

df.columns
```

```
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')
```

#5 Show the summery of the dataset.

```
#View summery
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             5 non-null      int64 
 1   workclass       5 non-null      object
 2   fnlwgt          5 non-null      int64 
 3   education       5 non-null      object
 4   education_num   5 non-null      int64 
 5   marital_status  5 non-null      object
 6   occupation      5 non-null      object
 7   relationship    5 non-null      object
 8   race            5 non-null      object
 9   sex             5 non-null      object
 10  capital_gain    5 non-null      int64 
 11  capital_loss    5 non-null      int64 
 12  hours_per_week  5 non-null      int64 
 13  native_country  5 non-null      object
 14  income          5 non-null      object
dtypes: int64(6), object(9)
memory usage: 728.0+ bytes
```

#6 Check the datatype of each column in the dataset.

```
#Check the data type of the column
df.dtypes
```

```
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
```

#7 Show the statistical properties of the dataset.

```
#view statistical properties of the data set
df.describe()
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
count	5.00000	5.000000	5.000000	5.000000	5.0	5.000000
mean	41.60000	189922.600000	11.000000	434.800000	0.0	34.600000
std	10.06479	110358.314006	2.828427	972.242357	0.0	12.074767
min	28.00000	77526.000000	7.000000	0.000000	0.0	13.000000
25%	38.00000	83311.000000	9.000000	0.000000	0.0	40.000000
50%	39.00000	215646.000000	13.000000	0.000000	0.0	40.000000
75%	50.00000	234721.000000	13.000000	0.000000	0.0	40.000000
max	53.00000	338409.000000	13.000000	2174.000000	0.0	40.000000

#8 Check if there is any missing value in the given dataset.

```
#Check the missing value
df.isnull().sum()
```

```
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
```

#9 Find the categorical variable.

```
#Find categorical variable
categorical = [var for var in df.columns if df[var].dtype=='O']

print('There are {} categorical variables\n'.format(len(categorical)))

print('The categorical variables are :\n\n', categorical)
```

```
There are 9 categorical variables

The categorical variables are :

 ['workclass', 'education', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'native_country', 'income']
```

#10 Review the categorical variable.

```
#Priview categorical variable
df[categorical].head()
```

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	income
0	state-gov	bachelors	never-married	adm-cierical	not-in-family	white	male	US	<=50k
1	self-emp-not-inc	bachelors	married-civ-spouse	exec-managerial	husband	white	male	US	<=50k
2	private	hs-grad	divorced	handlers-cleaners	not-in-family	white	male	US	<=50k
3	private	11th	married-civ-spouse	handlers-cleaners	husband	black	male	US	<=50k
4	private	bachelors	married-civ-spouse	prof-specially	wife	black	female	Cuba	<=50k

#11 Check the missing value from 'income' column.

```
#Exploring the variables
#Explore income target variable
# check for missing values

df['income'].isnull().sum()
```

```
0
```

#12 show any unique number from the column 'Income'.

```
#We can see that there are no missing values in the income target variable.

# view number of unique values

df['income'].nunique()
```

```
1
```

#13 Check the unique value from the column 'income'.

```
#There are 1 unique values in the income variable.

# view the unique values

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

```
array(['<=50k'], dtype=object)
```

#14 show the percentage of frequency distribution of values.

```
# view percentage of frequency distribution of values

df['income'].value_counts()/len(df)
```

```
<=50k    1.0
Name: income, dtype: float64
```

#15 Count the number of rows in the given dataset.

```
#show number of rows in a given dataset
len(df)
```

```
5
```

#16 Show the minimum 'age' from the given dataset

```
#Show the minimum 'age'
print(min(df['age']))
```

```
28
```

#17 Show the maximum 'age' from the given dataset.

```
#Show the maximun 'age'
print(max(df['age']))
```

```
53
```

#18 See if there is any 'capital_gain' in the given data set.

```
# See if there is any 'capital_gain'.

print(df['capital_gain'])
```

```
0    2174
1       0
2       0
3       0
4       0
Name: capital_gain, dtype: int64
```

#19 See if there is any ‘capital_loss’ in the given data set.

```
#See if there is any 'capital_loss'.
print(df['capital_loss'])
```

```
0    0
1    0
2    0
3    0
4    0
Name: capital_loss, dtype: int64
```

#20 See the Maximum ‘capital_gain’.

```
#See the Maximum 'capital_gain'.
print(df['capital_gain'].max())
```

```
2174
```

#21 See the Minimum ‘capital_loss’.

```
#See the minimum 'capital_loss'.
print(df['capital_gain'].min())
```

```
0
```

#22 Show which 'workclass' has maximum 'capital_gain'

```
#The is 'capital_gain' in the given data set with the 'workclass'.

print(df[['workclass','capital_gain']].max())
```

```
workclass       state-gov
capital_gain         2174
dtype: object
```

Predictions: -

1. The problem statement

In this kernel, We try to make predictions where the prediction task is to determine whether a person makes over 50K a year. We implement Random Forest Classification with Python and Scikit-Learn. So, to answer the question, We build a Random Forest classifier to predict whether a person makes over 50K a year or not.

2. Importing the libraries

In [124]:
```
import numpy as np
import pandas as pd
```

In [125]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

sns.set(style="whitegrid")
```

In [126]:
```
import warnings

warnings.filterwarnings('ignore')
```

4. Exploratory data analysis (EDA)

In [28]:
```
# print the shape
print('The shape of the dataset : ', df.shape)
```

```
The shape of the dataset :  (32561, 15)
```

We can see that there are 32561 instances and 15 attributes in the data set.

In [29]:
```
df.head()
```

Out[29]:

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

3. Importing the dataset

In [127]:
```
path = r'file:///C:\Users\hp\Downloads\income_evaluation.csv'

df = pd.read_csv(path)
```


4. Exploratory data analysis (EDA)

In [28]:
```
# print the shape
print('The shape of the dataset : ', df.shape)
```

```
The shape of the dataset :  (32561, 15)
```

We can see that there are 32561 instances and 15 attributes in the data set.

In [29]:
```
df.head()
```

Out[29]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
							Handlers-							United-	

```
In [129]: col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status', 'occupation', 'relationship',
                       'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income']

          df.columns = col_names

          df.columns

Out[129]: 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 [130]: df.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 32561 entries, 0 to 32560
          Data columns (total 15 columns):
           #   Column          Non-Null Count  Dtype
          ---  ------          --------------  -----
           0   age             32561 non-null  int64
           1   workclass       32561 non-null  object
           2   fnlwgt          32561 non-null  int64
           3   education       32561 non-null  object
           4   education_num   32561 non-null  int64
           5   marital_status  32561 non-null  object
           6   occupation      32561 non-null  object
           7   relationship    32561 non-null  object
           8   race            32561 non-null  object
           9   sex             32561 non-null  object
           10  capital_gain    32561 non-null  int64
```


```
In [129]: col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status', 'occupation', 'relationship',
                       'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income']

          df.columns = col_names

          df.columns

Out[129]: 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 [130]: df.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 32561 entries, 0 to 32560
          Data columns (total 15 columns):
           #   Column          Non-Null Count  Dtype
          ---  ------          --------------  -----
           0   age             32561 non-null  int64
           1   workclass       32561 non-null  object
           2   fnlwgt          32561 non-null  int64
           3   education       32561 non-null  object
           4   education_num   32561 non-null  int64
           5   marital_status  32561 non-null  object
           6   occupation      32561 non-null  object
           7   relationship    32561 non-null  object
           8   race            32561 non-null  object
           9   sex             32561 non-null  object
           10  capital_gain    32561 non-null  int64
```

FINDINGS: -

Findings

- We can see that the dataset contains 9 character variables and 6 numerical variables.
- `income` is the target variable.
- There are no missing values in the dataset. I will explore this later,

In [131]:
```
df.dtypes
```

Out[131]:
```
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
```

In [134]:
```
df.describe(include='all')
```

Out[134]:

In [130]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education_num   32561 non-null  int64 
 5   marital_status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital_gain    32561 non-null  int64 
 11  capital_loss    32561 non-null  int64 
 12  hours_per_week  32561 non-null  int64 
 13  native_country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [36]:

```
# check for missing values

df.isnull().sum()
```

Out[36]:

```
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 [134]:

```
df.describe(include='all')
```

Out[134]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss
count	32561.000000	32561	3.256100e+04	32561	32561.000000	32561	32561	32561	32561	32561	32561.000000	32561.000000
unique	NaN	9	NaN	16	NaN	7	15	6	5	2	NaN	NaN
top	NaN	Private	NaN	HS-grad	NaN	Married-civ-spouse	Prof-specialty	Husband	White	Male	NaN	NaN
freq	NaN	22696	NaN	10501	NaN	14976	4140	13193	27816	21790	NaN	NaN
mean	38.581647	NaN	1.897784e+05	NaN	10.080679	NaN	NaN	NaN	NaN	NaN	1077.648844	87.303830
std	13.640433	NaN	1.055500e+05	NaN	2.572720	NaN	NaN	NaN	NaN	NaN	7385.292085	402.960219
min	17.000000	NaN	1.228500e+04	NaN	1.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000
25%	28.000000	NaN	1.178270e+05	NaN	9.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000
50%	37.000000	NaN	1.783560e+05	NaN	10.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000
75%	48.000000	NaN	2.370510e+05	NaN	12.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000
max	90.000000	NaN	1.484705e+06	NaN	16.000000	NaN	NaN	NaN	NaN	NaN	99999.000000	4356.000000

In [38]:
```python
def initial_eda(df):
    if isinstance(df, pd.DataFrame):
        total_na = df.isna().sum().sum()
        print("Dimensions : %d rows, %d columns" % (df.shape[0], df.shape[1]))
        print("Total NA Values : %d " % (total_na))
        print("%38s %10s     %10s %10s" % ("Column Name", "Data Type", "#Distinct", "NA Values"))
        col_name = df.columns
        dtyp = df.dtypes
        uniq = df.nunique()
        na_val = df.isna().sum()
        for i in range(len(df.columns)):
            print("%38s %10s   %10s %10s" % (col_name[i], dtyp[i], uniq[i], na_val[i]))

    else:
        print("Expect a DataFrame but got a %15s" % (type(df)))
```

In [39]:
```python
initial_eda(df)
```

```
Dimensions : 32561 rows, 15 columns
Total NA Values : 0 
                           Column Name  Data Type      #Distinct  NA Values
                                   age      int64             73          0
                             workclass     object              9          0
                                fnlwgt      int64          21648          0
                             education     object             16          0
                         education_num      int64             16          0
                        marital_status     object              7          0
                            occupation     object             15          0
                          relationship     object              6          0
                                  race     object              5          0
```

INTERPRETATION: -

Interpretation

We can see that there are no missing values in the dataset.

In [37]:
```python
#assert that there are no missing values in the dataframe

assert pd.notnull(df).all().all()
```

Interpretation

- The above command does not throw any error. Hence, it is confirmed that there are no missing or negative values in the dataset.
- All the values are greater than or equal to zero excluding character values.

In [38]:
```python
def initial_eda(df):
    if isinstance(df, pd.DataFrame):
        total_na = df.isna().sum().sum()
        print("Dimensions : %d rows, %d columns" % (df.shape[0], df.shape[1]))
        print("Total NA Values : %d " % (total_na))
        print("%38s %10s     %10s %10s" % ("Column Name", "Data Type", "#Distinct", "NA Values"))
        col_name = df.columns
        dtyp = df.dtypes
        uniq = df.nunique()
        na_val = df.isna().sum()
        for i in range(len(df.columns)):
            print("%38s %10s   %10s %10s" % (col_name[i], dtyp[i], uniq[i], na_val[i]))

    else:
        print("Expect a DataFrame but got a %15s" % (type(df)))
```

```
In [39]: initial_eda(df)

Dimensions : 32561 rows, 15 columns
Total NA Values : 0
                     Column Name  Data Type      #Distinct  NA Values
                             age     int64          73          0
                       workclass    object           9          0
                          fnlwgt     int64       21648          0
                       education    object          16          0
                   education_num     int64          16          0
                  marital_status    object           7          0
                      occupation    object          15          0
                    relationship    object           6          0
                            race    object           5          0
                             sex    object           2          0
                    capital_gain     int64         119          0
                    capital_loss     int64          92          0
                  hours_per_week     int64          94          0
                  native_country    object          42          0
                          income    object           2          0
```

EXPLORE CATEGORICAL VARIABLES: -

4.1. Explore Categorical Variables

```
In [135]: categorical = [var for var in df.columns if df[var].dtype=='O']

print(f"There are {len(categorical)} categorical variables\n")

print('The categorical variables are :\n\n', categorical)

There are 9 categorical variables

The categorical variables are :

 ['workclass', 'education', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'native_country', 'income']
```

4.2 Preview categorical variables

```
In [41]: df[categorical].head()
```

Out[41]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	income
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States	<=50K
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	<=50K
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	<=50K
3	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States	<=50K
4	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba	<=50K

In [39]:
```
initial_eda(df)
```

```
Dimensions : 32561 rows, 15 columns
Total NA Values : 0
```

Column Name	Data Type	#Distinct	NA Values
age	int64	73	0
workclass	object	9	0
fnlwgt	int64	21648	0
education	object	16	0
education_num	int64	16	0
marital_status	object	7	0
occupation	object	15	0
relationship	object	6	0
race	object	5	0
sex	object	2	0
capital_gain	int64	119	0
capital_loss	int64	92	0
hours_per_week	int64	94	0
native_country	object	42	0
income	object	2	0

There are 2 unique values in the `income` variable.

In [46]:
```
# view the unique values

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

Out[46]:
```
array([' <=50K', ' >50K'], dtype=object)
```

The two unique values are `<=50K` and `>50K`.

In [47]:
```
# view the frequency distribution of values

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

Out[47]:
```
 <=50K    24720
 >50K      7841
Name: income, dtype: int64
```

In [136]:
```
# visualize frequency distribution of income variable

f,ax=plt.subplots(1,2,figsize=(18,8))

ax[0] = df['income'].value_counts().plot.pie(explode=[0,0],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Income Share')


#f, ax = plt.subplots(figsize=(6, 8))
ax[1] = sns.countplot(x="income", data=df, palette="ocean_r")
ax[1].set_title("Frequency distribution of income variable")
```

4.3 Summary of categorical variables

- There are 9 categorical variables in the dataset.
- The categorical variables are given by `workclass`, `education`, `marital_status`, `occupation`, `relationship`, `race`, `sex`, `native_country` and `income`.
- `income` is the target variable.

4.4 Explore the variables

Explore `income` target variable

```
In [44]: # check for missing values

df['income'].isnull().sum()
```

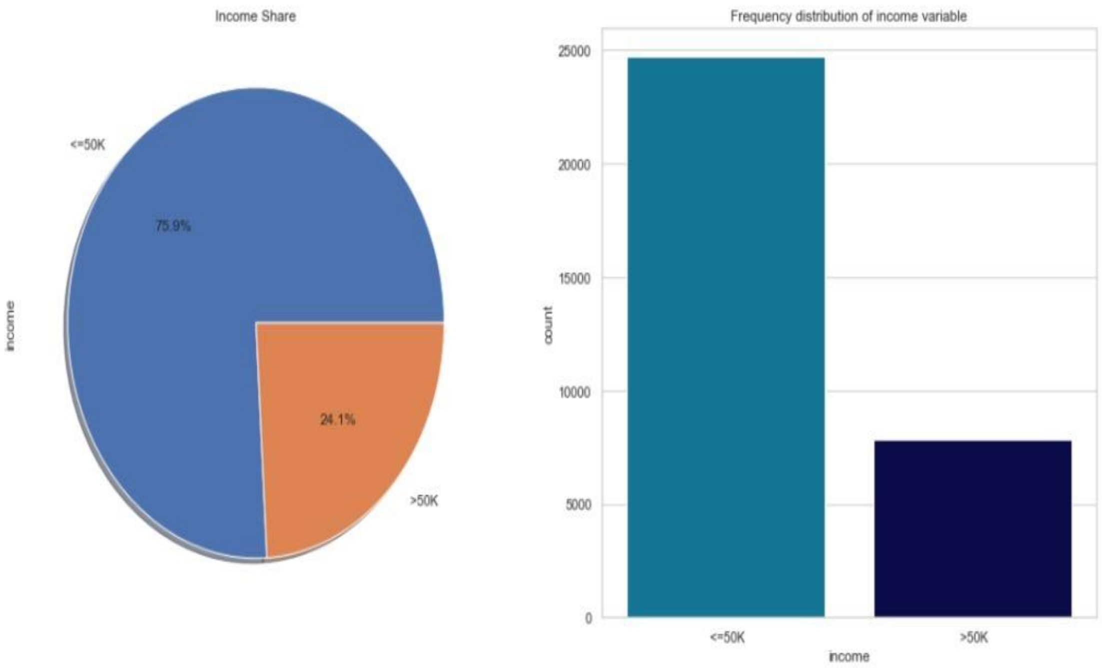
```
Out[44]: 0
```

We can see that there are no missing values in the `income` target variable.

```
In [45]: # view number of unique values

df['income'].nunique()
```

```
Out[45]: 2
```

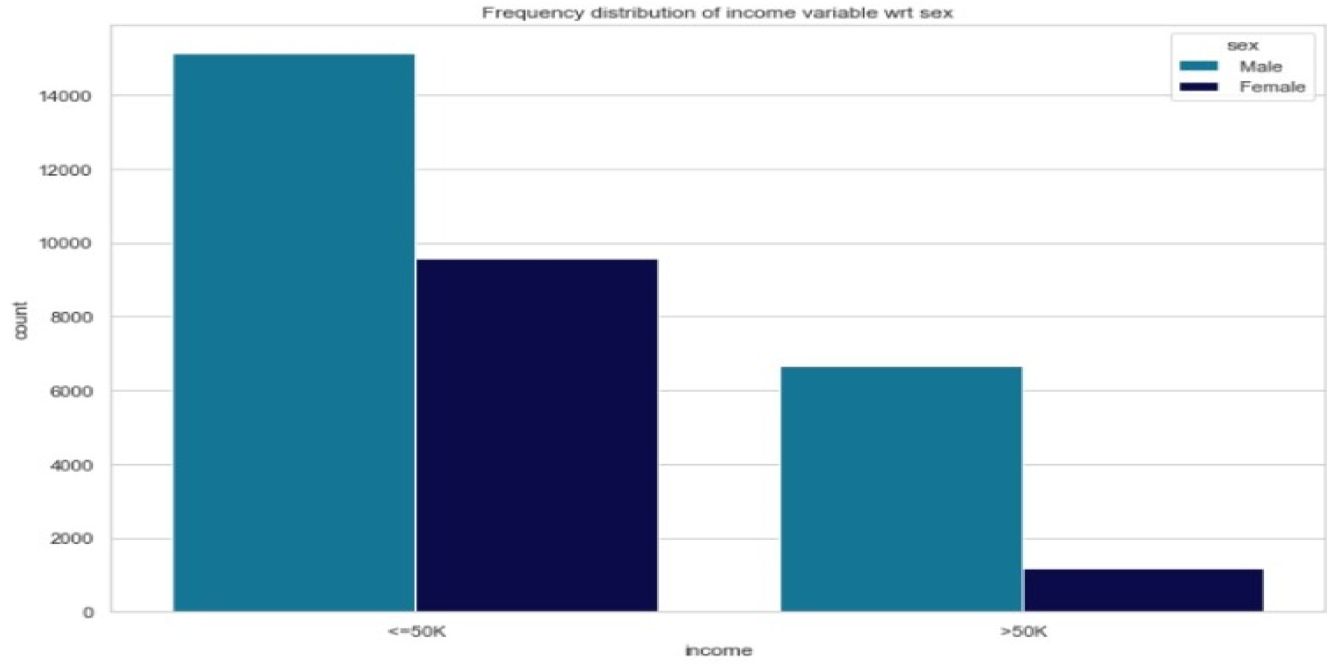
```
plt.show()
```



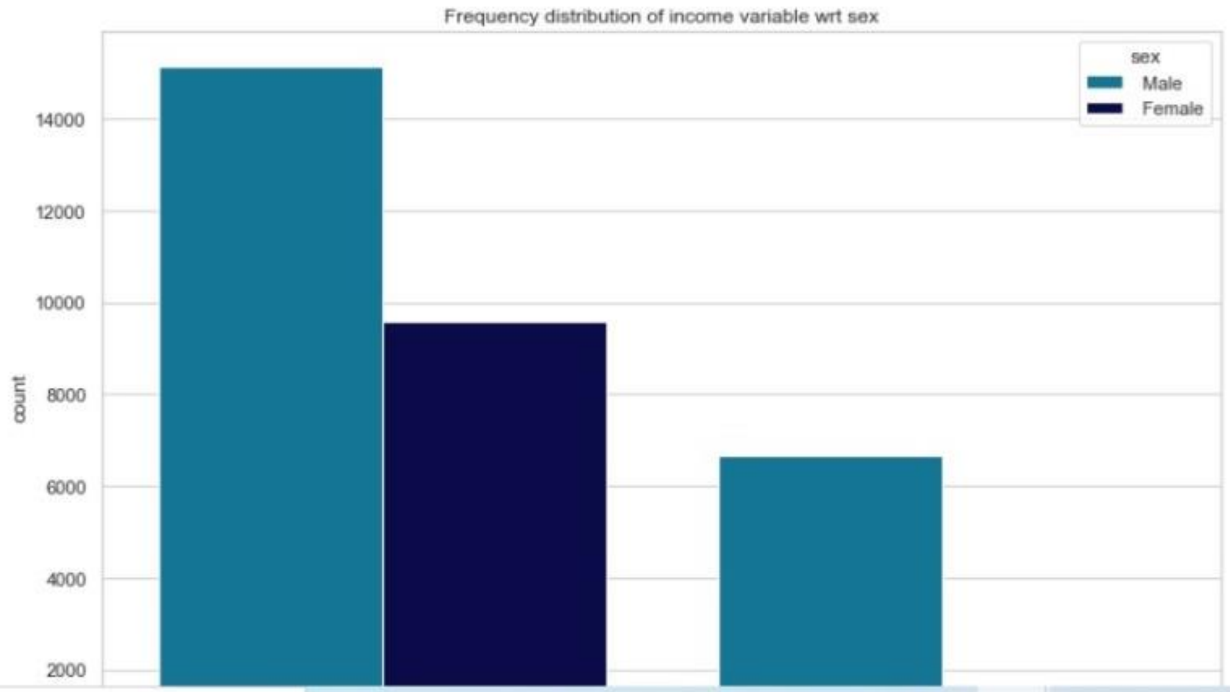

```
In [47]: # view the frequency distribution of values

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

```
Out[47]:  <=50K     24720
 >50K      7841
Name: income, dtype: int64
```

```
In [136]: # visualize frequency distribution of income variable

f,ax=plt.subplots(1,2,figsize=(18,8))

ax[0] = df['income'].value_counts().plot.pie(explode=[0,0],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Income Share')


#f, ax = plt.subplots(figsize=(6, 8))
ax[1] = sns.countplot(x="income", data=df, palette="ocean_r")
ax[1].set_title("Frequency distribution of income variable")

plt.show()
```



In [143]:
```
categorical_features=["sex","race"]
for feature in categorical_features:
    f,ax = plt.subplots(figsize=(12, 8))
    ax = sns.countplot(x="income", hue=feature, data=df, palette="ocean_r")
    ax.set_title(f"Frequency distribution of income variable wrt {feature}")
    plt.show()
```



Interpretation

- We can see that whites make more money than non-whites in both the income categories.

Explore `workclass` variable

In [53]:
```
# check number of unique labels

df.workclass.nunique()
```

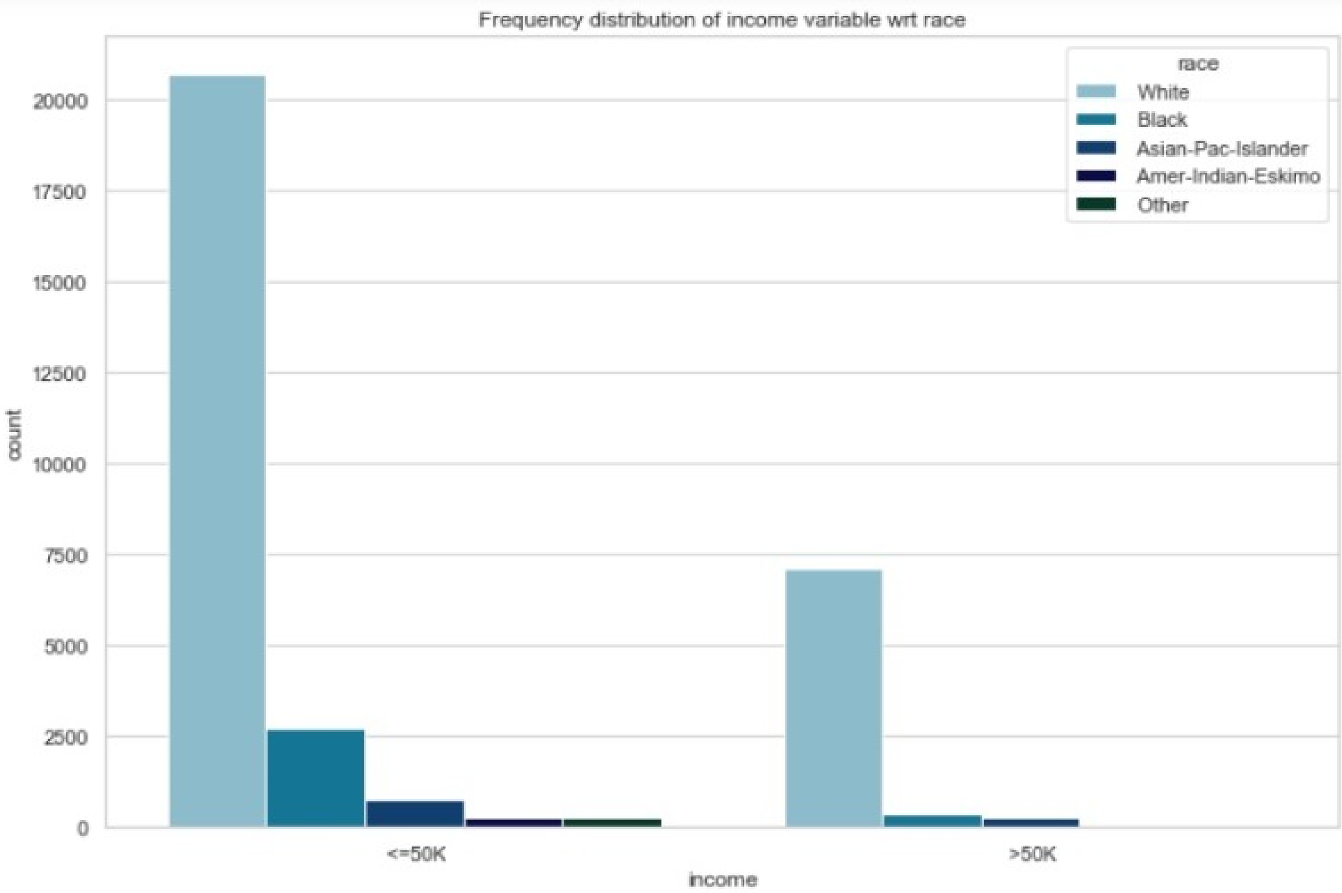
Out[53]:
```
9
```

In [54]:
```
# view the unique labels

df.workclass.unique()
```

Out[54]:
```
array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',
       ' Local-gov', ' ?', ' Self-emp-inc', ' Without-pay',
       ' Never-worked'], dtype=object)
```

In [55]:
```
# view frequency distribution of values

df.workclass.value_counts()
```

Out[55]:
```
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1298
```



In [56]:
```
# replace '?' values in workclass variable with `NaN`

df['workclass'].replace(' ?', np.NaN, inplace=True)
```

In [57]:
```
# again check the frequency distribution of values in workclass variable

df.workclass.value_counts()
```

Out[57]:
```
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: workclass, dtype: int64
```

- Now, we can see that there are no values encoded as `?` in the workclass variable.
- I will adopt similar approach with `occupation` and `native_country` column.

Visualize `workclass` variable

In [146]:
```
f, ax = plt.subplots(figsize=(10, 6))
ax = df.workclass.value_counts().plot(kind="bar", color="green")
ax.set_title("Frequency distribution of workclass variable")
ax.set_xticklabels(df.workclass.value_counts().index, rotation=30)
```

In [54]:
```
# view the unique labels

df.workclass.unique()
```

Out[54]:
```
array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',
       ' Local-gov', ' ?', ' Self-emp-inc', ' Without-pay',
       ' Never-worked'], dtype=object)
```

In [55]:
```
# view frequency distribution of values

df.workclass.value_counts()
```

Out[55]:
```
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: workclass, dtype: int64
```

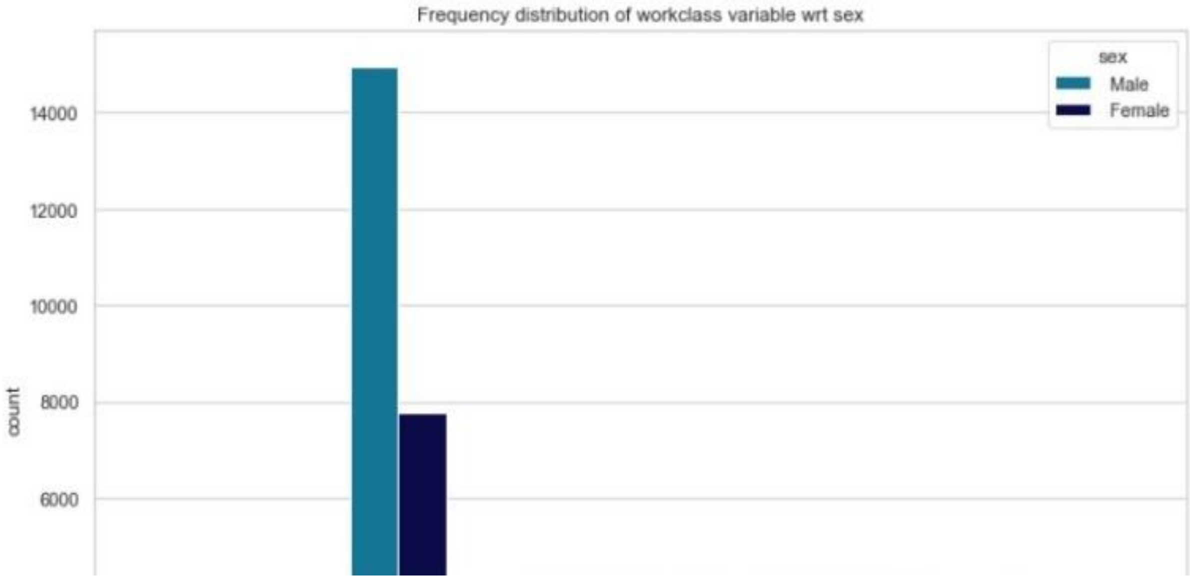
We can see that there are 1836 values encoded as `?` in workclass variable. I will replace these `?` with `NaN`.

In [56]:
```
# replace '?' values in workclass variable with `NaN`

df['workclass'].replace(' ?', np.NaN, inplace=True)
```

Interpretation

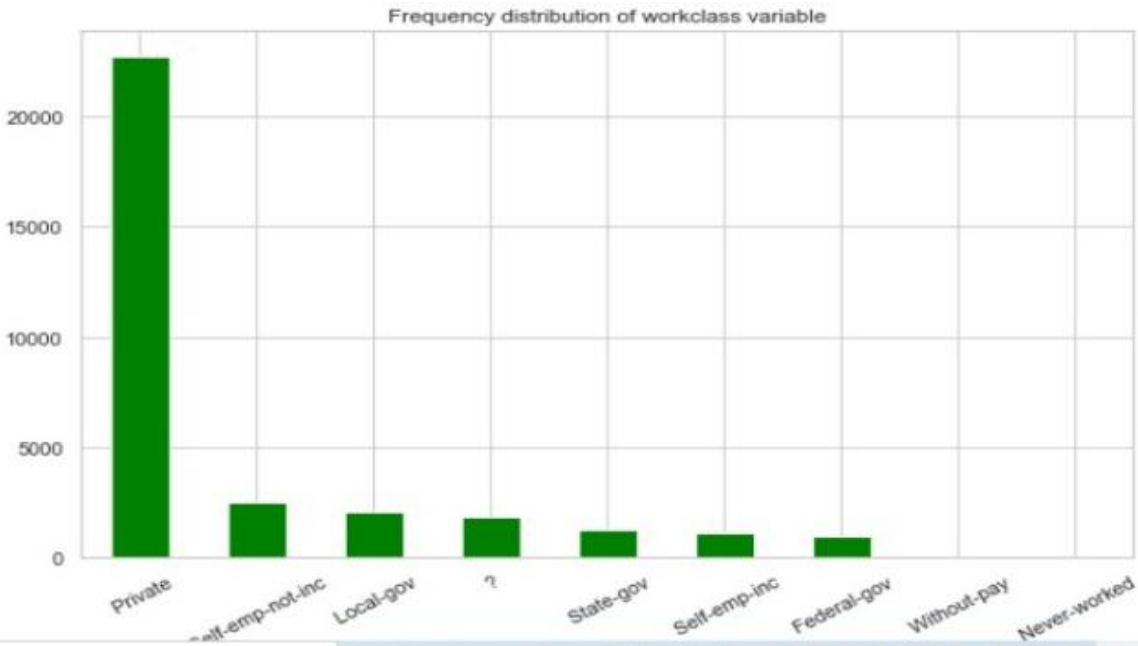
- We can see that there are lot more private workers than other category of workers.

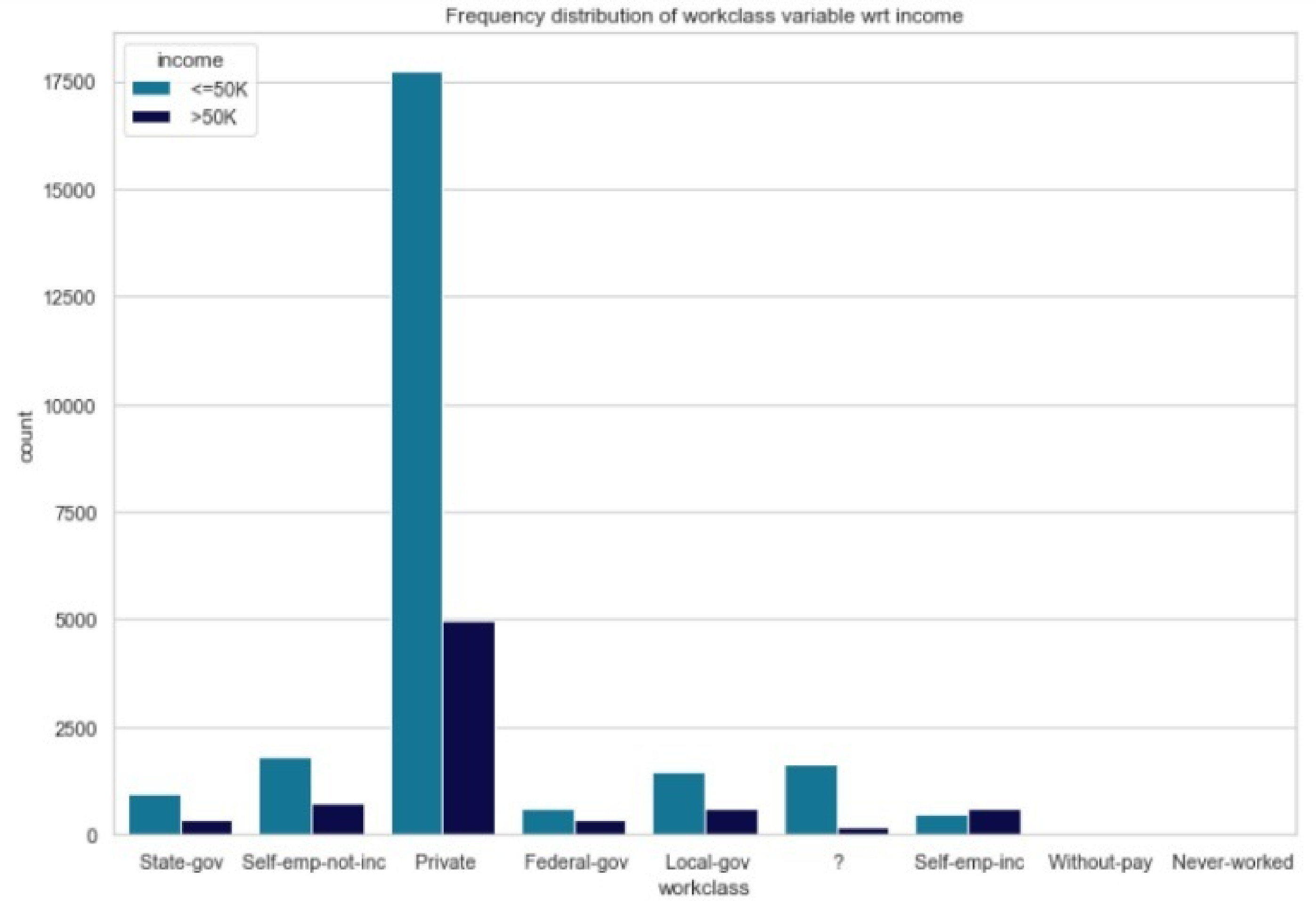
In [144]:
```
categorical_features=["sex","income"]
for feature in categorical_features:
    f,ax = plt.subplots(figsize=(12, 8))
    ax = sns.countplot(x="workclass", hue=feature, data=df, palette="ocean_r")
    ax.set_title(f"Frequency distribution of workclass variable wrt {feature}")
    plt.show()
```

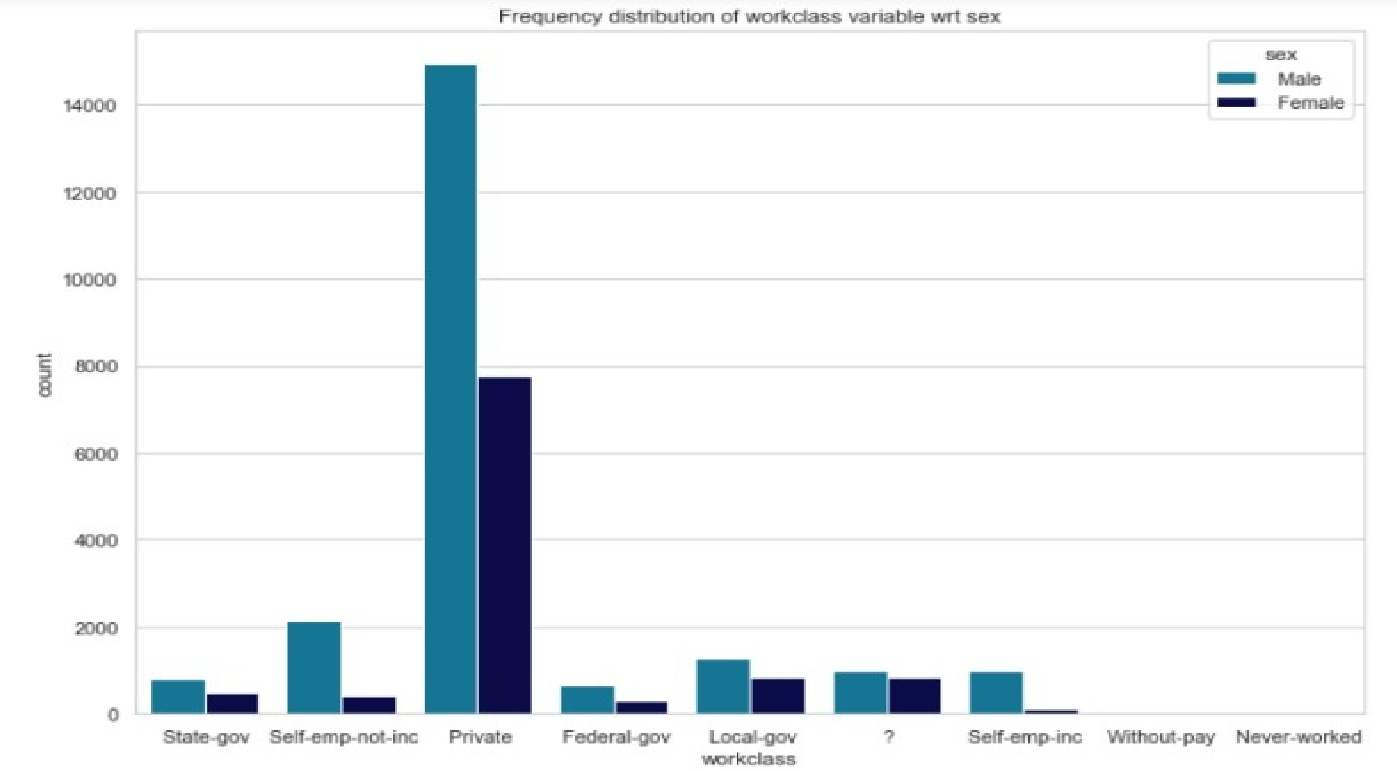


Visualize `workclass` variable

In [146]:
```
f, ax = plt.subplots(figsize=(10, 6))
ax = df.workclass.value_counts().plot(kind="bar", color="green")
ax.set_title("Frequency distribution of workclass variable")
ax.set_xticklabels(df.workclass.value_counts().index, rotation=30)
plt.show()
```







Frequency distribution of workclass variable wrt income

In [63]:
```
# view frequency distribution of values

df.occupation.value_counts()
```

Out[63]:
```
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3770
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 ?                    1843
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: occupation, dtype: int64
```

We can see that there are 1843 values encoded as `?` in occupation variable. I will replace these `?` with `NaN`.

In [64]:
```
# replace '?' values in occupation variable with `NaN`

df['occupation'].replace(' ?', np.NaN, inplace=True)
```

In [65]:
```
# again check the frequency distribution of values
```

Interpretation

- We can see that workers make less than equal to 50k in most of the working categories.
- But this trend is more appealing in Private `workclass` category.

- We can see that there are more male workers than female workers in all the working category.
- The trend is more appealing in Private sector.

Explore `occupation` variable

In [61]:
```
# check number of unique labels

df.occupation.nunique()
```

Out[61]: 15

In [62]:
```
# view unique labels

df.occupation.unique()
```

Out[62]:
```
array([' Adm-clerical', ' Exec-managerial', ' Handlers-cleaners',
       ' Prof-specialty', ' Other-service', ' Sales', ' Craft-repair',
       ' Transport-moving', ' Farming-fishing', ' Machine-op-inspct',
       ' Tech-support', ' ?', ' Protective-serv', ' Armed-Forces',
       ' Priv-house-serv'], dtype=object)
```

```
In [65]: # again check the frequency distribution of values

         df.occupation.value_counts()
```

```
Out[65]: Prof-specialty       4140
         Craft-repair         4099
         Exec-managerial      4066
         Adm-clerical         3770
         Sales                3650
         Other-service        3295
         Machine-op-inspct    2002
         Transport-moving     1597
         Handlers-cleaners    1370
         Farming-fishing       994
         Tech-support          928
         Protective-serv       649
         Priv-house-serv       149
         Armed-Forces            9
         Name: occupation, dtype: int64
```

```
In [147]: # visualize frequency distribution of `occupation` variable

          f, ax = plt.subplots(figsize=(12, 8))
          ax = sns.countplot(x="occupation", data=df, palette="ocean_r")
          ax.set_title("Frequency distribution of occupation variable")
          ax.set_xticklabels(df.occupation.value_counts().index, rotation=30)
          plt.show()
```

```
In [69]: # check frequency distribution of values

         df.native_country.value_counts()
```

```
Out[69]: United-States         29170
         Mexico                  643
         ?                       583
         Philippines             198
         Germany                 137
         Canada                  121
         Puerto-Rico             114
         El-Salvador             106
         India                   100
         Cuba                     95
         England                  90
         Jamaica                  81
         South                    80
         China                    75
         Italy                    73
         Dominican-Republic       70
         Vietnam                  67
         Guatemala                64
         Japan                    62
         Poland                   60
         Columbia                 59
         Taiwan                   51
         Haiti                    44
         Iran                     43
         Portugal                 37
         Nicaragua                34
```

In [70]:
```python
# replace '?' values in native_country variable with `NaN`

df['native_country'].replace(' ?', np.NaN, inplace=True)
```

In [71]:
```python
# again check the frequency distribution of values

df.native_country.value_counts()
```

Out[71]:
```
United-States         29170
Mexico                  643
Philippines             198
Germany                 137
Canada                  121
Puerto-Rico             114
El-Salvador             106
India                   100
Cuba                     95
England                  90
Jamaica                  81
South                    80
China                    75
Italy                    73
Dominican-Republic       70
Vietnam                  67
Guatemala                64
Japan                    62
Poland                   60
Columbia                 59
Taiwan                   51
Haiti                    44
Iran                     43
```


```
Columbia                         59
Taiwan                           51
Haiti                            44
Iran                             43
```


```
Portugal                         37
Nicaragua                        34
Peru                             31
France                           29
Greece                           29
Ecuador                          28
Ireland                          24
Hong                             20
Cambodia                         19
Trinadad&Tobago                  19
Laos                             18
Thailand                         18
Yugoslavia                       16
Outlying-US(Guam-USVI-etc)       14
Honduras                         13
Hungary                          13
Scotland                         12
Holand-Netherlands                1
Name: native_country, dtype: int64
```

We can see that there are 583 values encoded as `?` in native_country variable. I will replace these `?` with `NaN`.

In [70]:
```python
# replace '?' values in native_country variable with `NaN`

df['native_country'].replace(' ?', np.NaN, inplace=True)
```

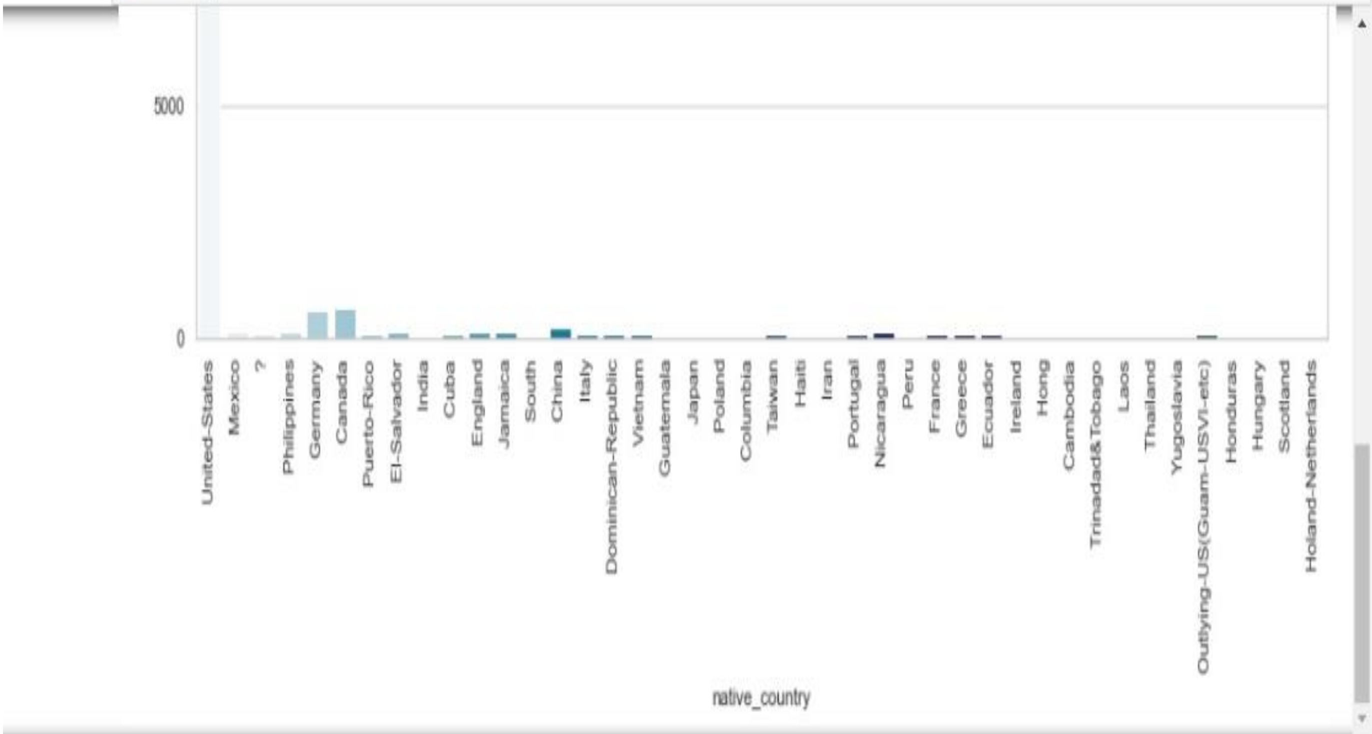
```
Peru                          31
France                        29
Greece                        29
Ecuador                       28
Ireland                       24
Hong                          20
Cambodia                      19
Trinadad&Tobago               19
Laos                          18
Thailand                      18
Yugoslavia                    16
Outlying-US(Guam-USVI-etc)    14
Honduras                      13
Hungary                       13
Scotland                      12
Holand-Netherlands             1
Name: native_country, dtype: int64
```

In [148]:

```python
# visualize frequency distribution of `native_country` variable

f, ax = plt.subplots(figsize=(16, 12))
ax = sns.countplot(x="native_country", data=df, palette="ocean_r")
ax.set_title("Frequency distribution of native_country variable")
ax.set_xticklabels(df.native_country.value_counts().index, rotation=90)
plt.show()
```

In [148]:

```python
# visualize frequency distribution of `native_country` variable

f, ax = plt.subplots(figsize=(12, 12))
ax = sns.countplot(x="native_country", data=df, palette="ocean_r")
ax.set_title("Frequency distribution of native_country variable")
ax.set_xticklabels(df.native_country.value_counts().index, rotation=90)
plt.show()
```



We can see that `United-States` dominate amongst the `native_country` variables.

In [73]:
```
#Checking missing categorical values
df[categorical].isnull().sum()
```

Out[73]:
```
workclass         1836
education            0
marital_status       0
occupation        1843
relationship         0
race                 0
sex                  0
native_country     583
income               0
dtype: int64
```

Now, we can see that `workclass`, `occupation` and `native_country` variable contains missing values.

6. Explore Numerical Variables

In [75]:
```
numerical = [var for var in df.columns if df[var].dtype!='O']

print('There are {} numerical variables\n'.format(len(numerical)))

print('The numerical variables are :\n\n', numerical)
```

```
There are 6 numerical variables
```


In [75]:
```
numerical = [var for var in df.columns if df[var].dtype!='O']

print('There are {} numerical variables\n'.format(len(numerical)))

print('The numerical variables are :\n\n', numerical)
```


```
There are 6 numerical variables

The numerical variables are :

 ['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss', 'hours_per_week']
```

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

Out[76]:

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
0	39	77516	13	2174	0	40
1	50	83311	13	0	0	13
2	38	215646	9	0	0	40
3	53	234721	7	0	0	40
4	28	338409	13	0	0	40

In [77]:
```
df[numerical].isnull().sum()
```

Out[77]:
```
age               0
fnlwgt            0
education_num     0
capital_gain      0
```

In [77]:
```
df[numerical].isnull().sum()
```

Out[77]:
```
age               0
fnlwgt            0
education_num     0
capital_gain      0
capital_loss      0
hours_per_week    0
dtype: int64
```

We can see that there are no missing values in the numerical variables.

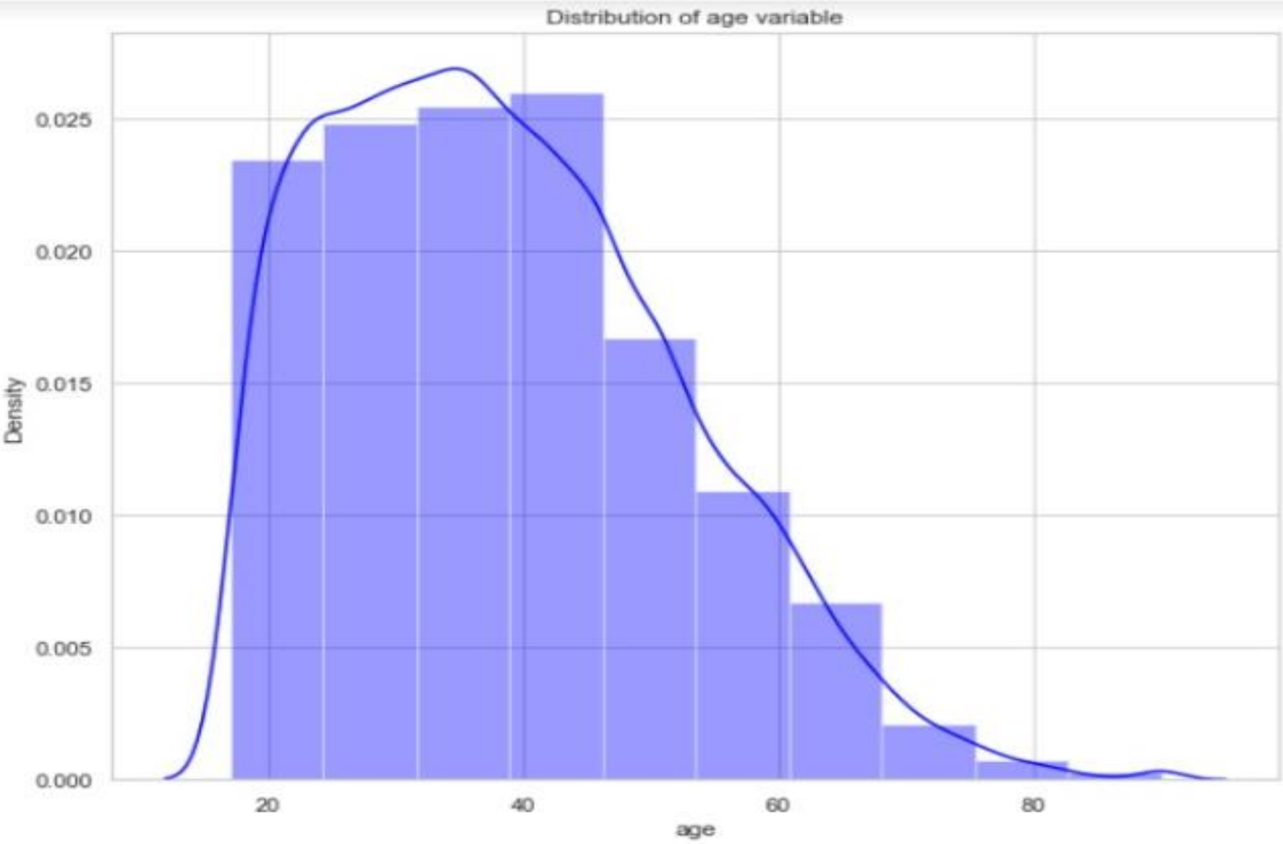
Explore `age` variable

In [78]:
```
df['age'].nunique()
```

Out[78]:
```
73
```

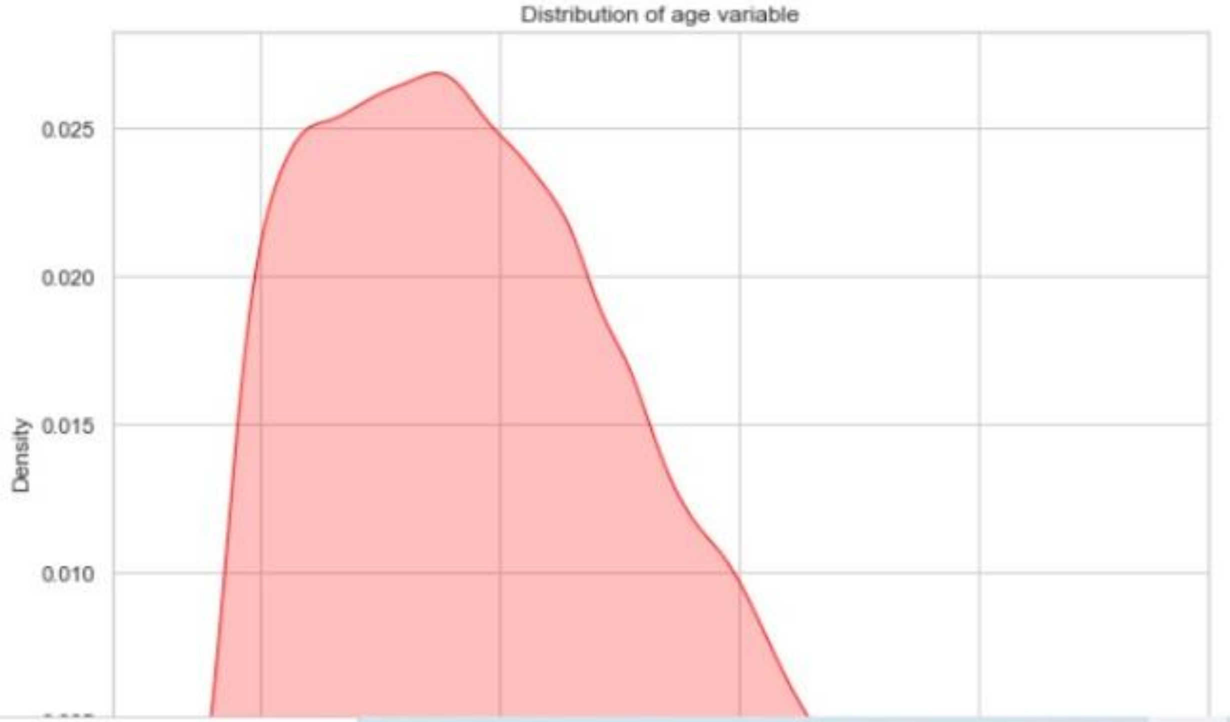
View the distribution of `age` variable

In [79]:
```
f, ax = plt.subplots(figsize=(10,8))
x = df['age']
ax = sns.distplot(x, bins=10, color='blue')
ax.set_title("Distribution of age variable")
plt.show()
```



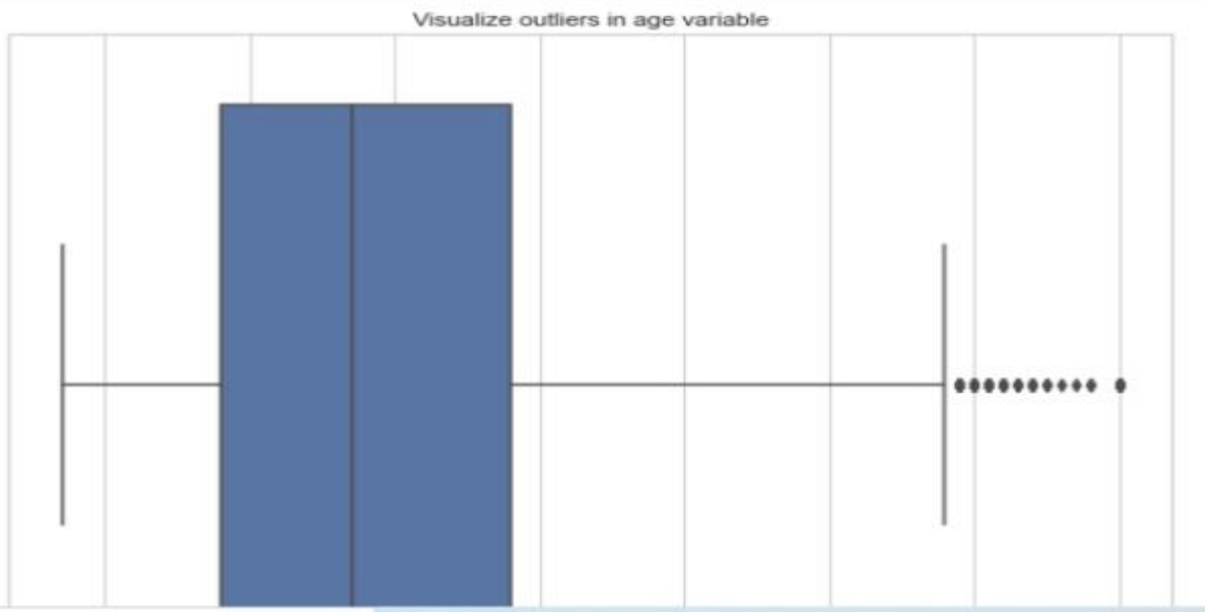
We can shade under the density curve and use a different color as follows:-

In [81]:
```
f, ax = plt.subplots(figsize=(10,8))
x = df['age']
x = pd.Series(x, name="Age variable")
ax = sns.kdeplot(x, shade=True, color='red')
ax.set_title("Distribution of age variable")
plt.show()
```



Detect outliers in `age` variable with boxplot

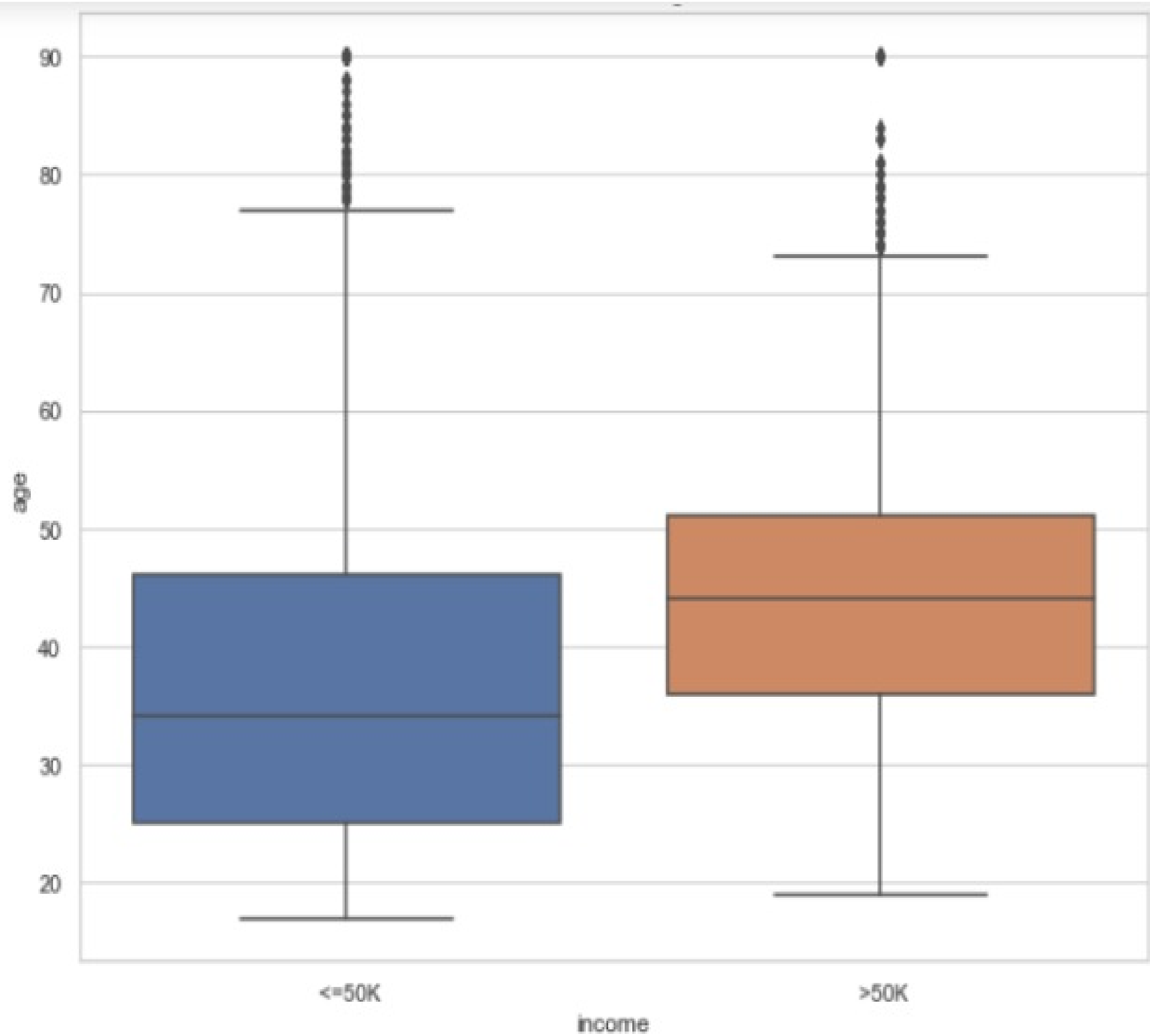
In [82]:
```
f, ax = plt.subplots(figsize=(10,8))
x = df['age']
ax = sns.boxplot(x)
ax.set_title("Visualize outliers in age variable")
plt.show()
```



We can see that there are lots of outliers in `age` variable.

Explore relationship between `age` and `income` variables

In [149]:
```
f, ax = plt.subplots(figsize=(10, 8))
ax = sns.boxplot(x="income", y="age", data=df)
ax.set_title("Visualize income wrt age variable")
plt.show()


f, ax = plt.subplots(figsize=(10, 8))
ax = sns.boxplot(x="income", y="age", data=df)
ax.set_title("Visualize income wrt age variable")
plt.show()
```



Visualize income wrt age variable

\

Interpretation

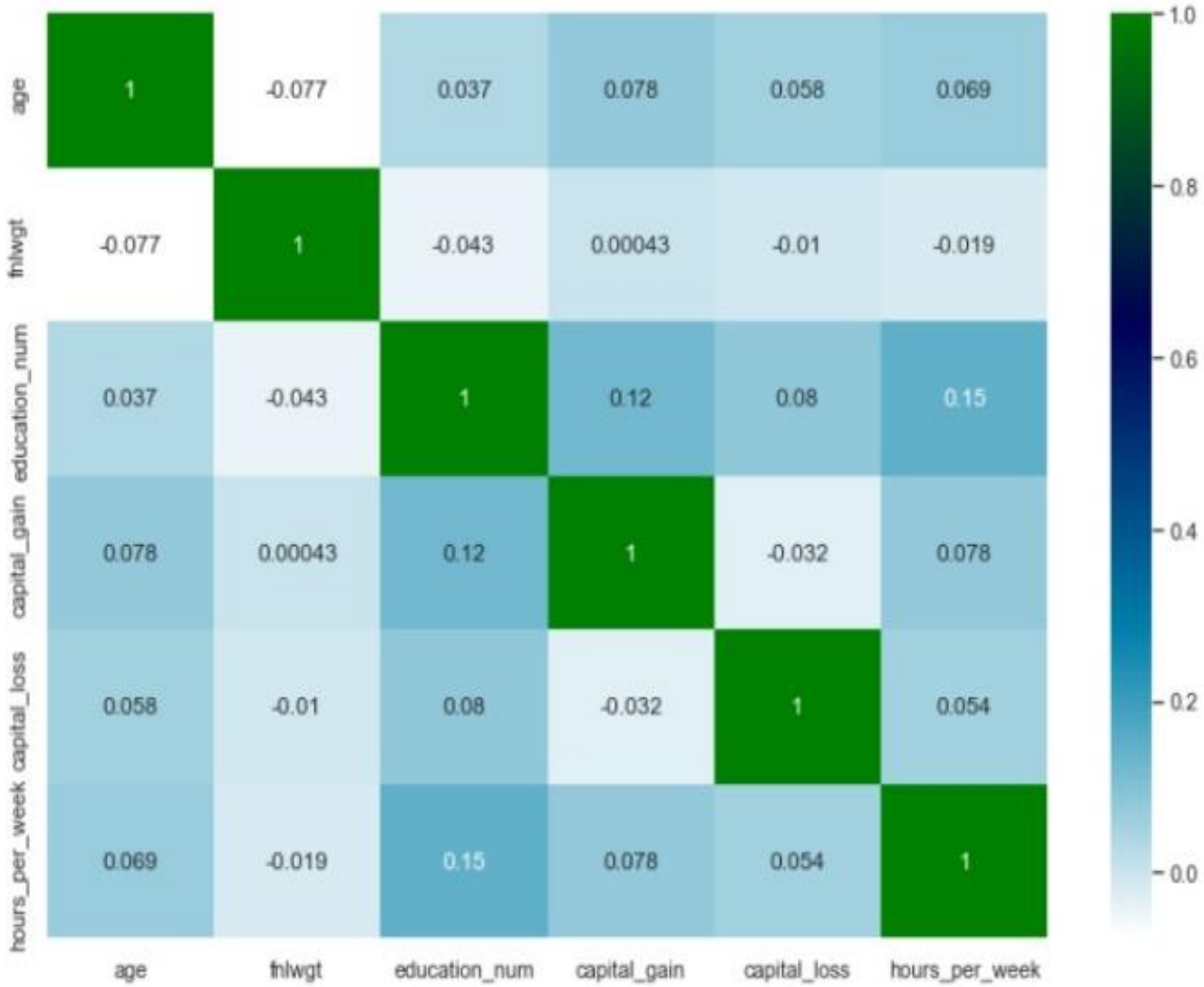
- As expected, younger people make less money as compared to senior people.

- Whites are more older than other groups of people.

Find out the correlations

In [137]:
```
# plot correlation heatmap to find out correlations
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(),annot=True,cmap="ocean_r")
```

Out[137]: <AxesSubplot:>

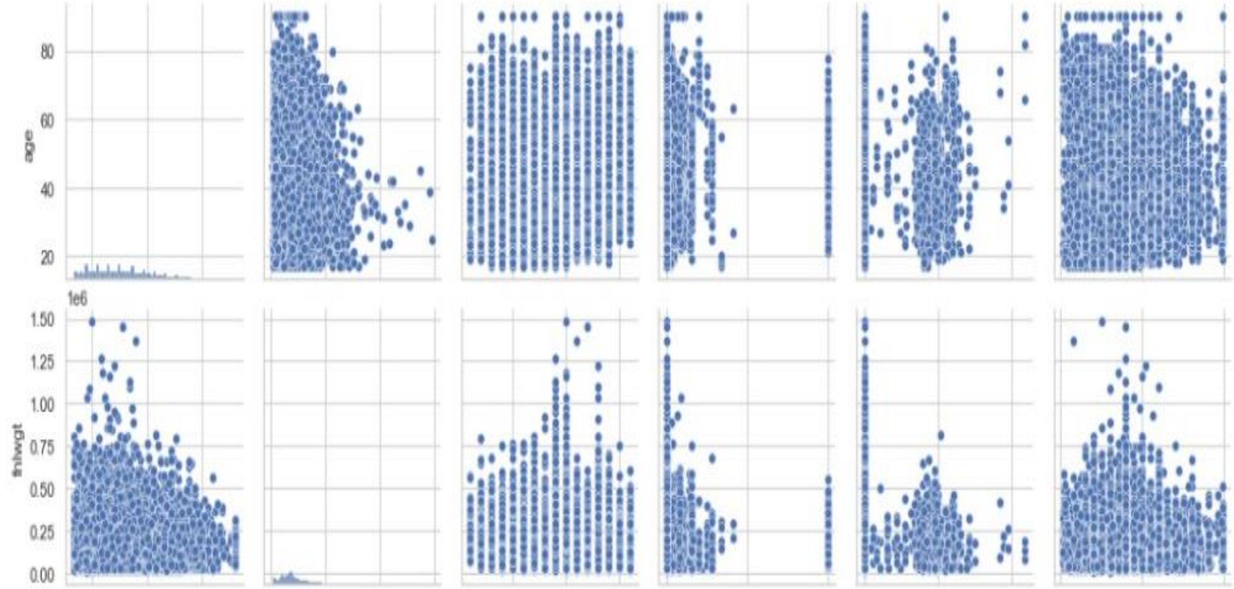


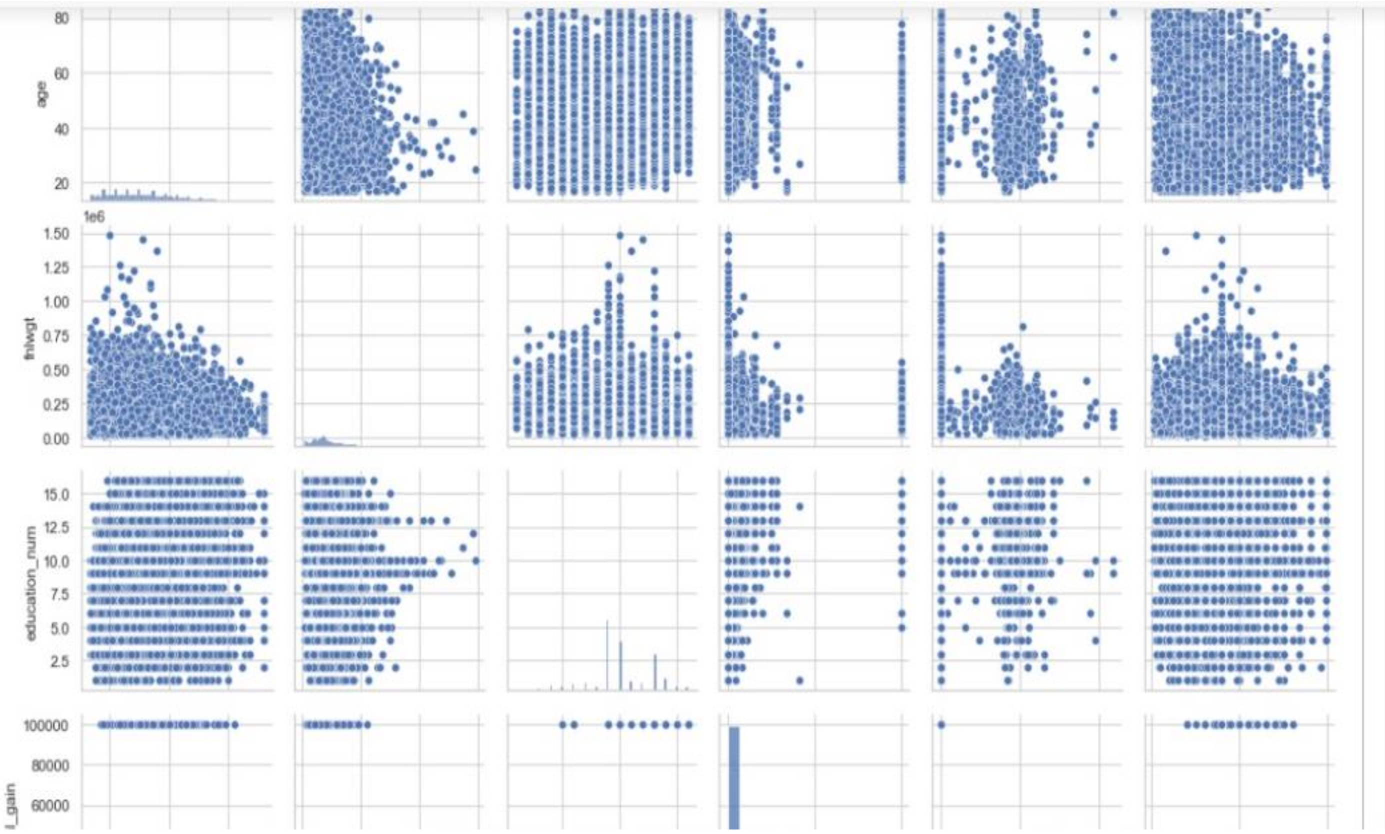
Interpretation

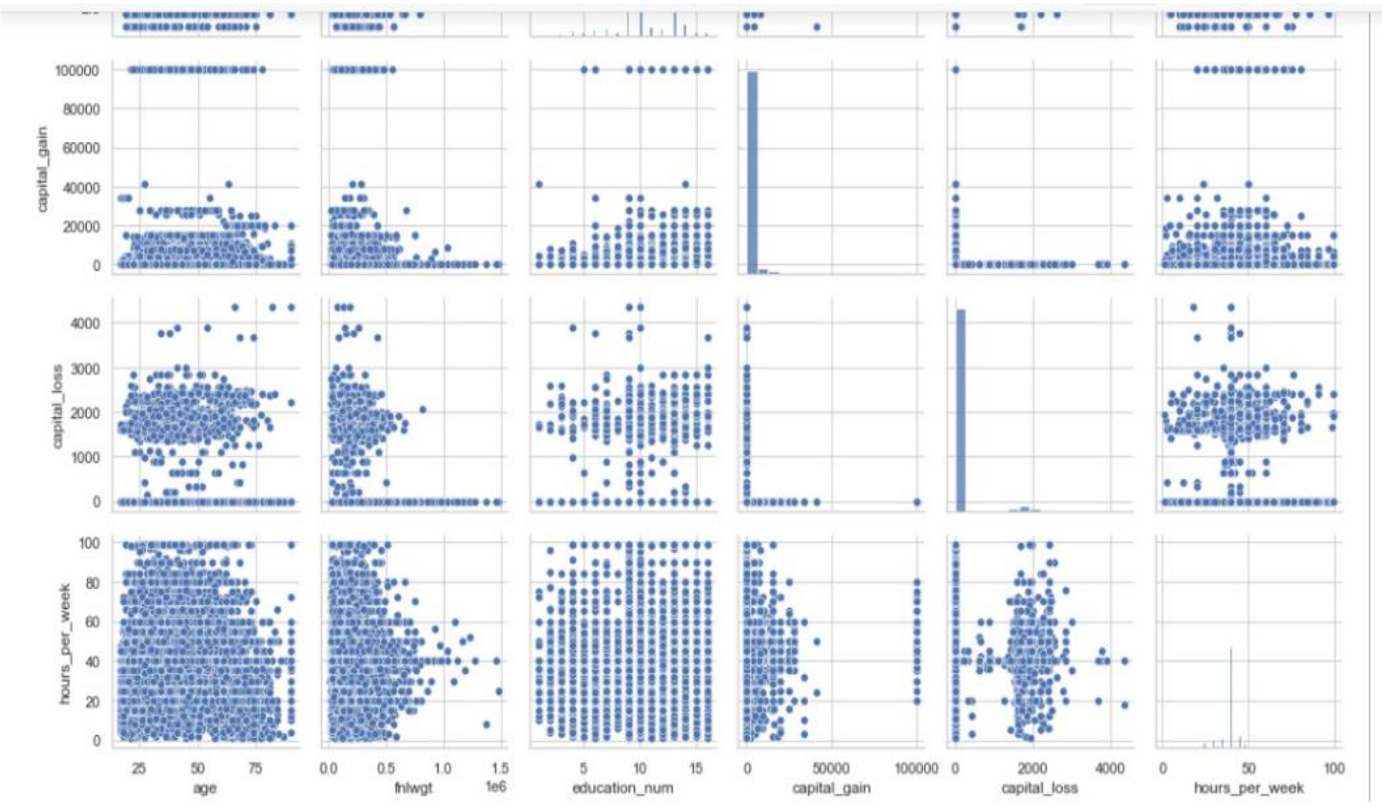
- We can see that there is no strong correlation between variables.

Plot pairwise relationships in dataset

In [88]:
```
sns.pairplot(df)
plt.show()
```



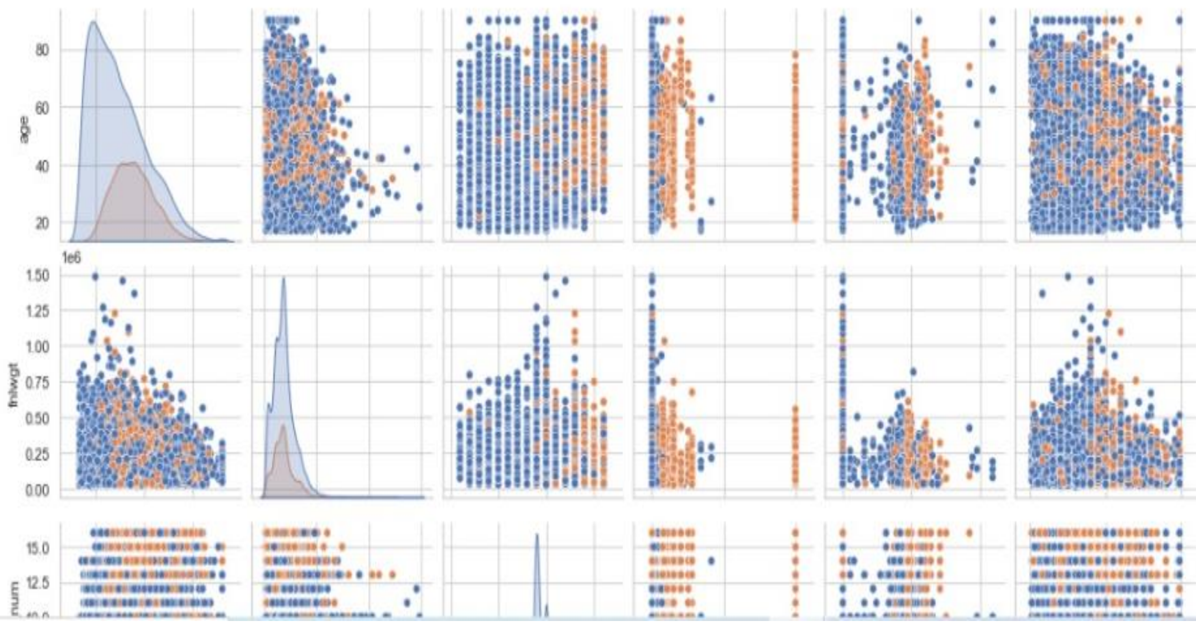


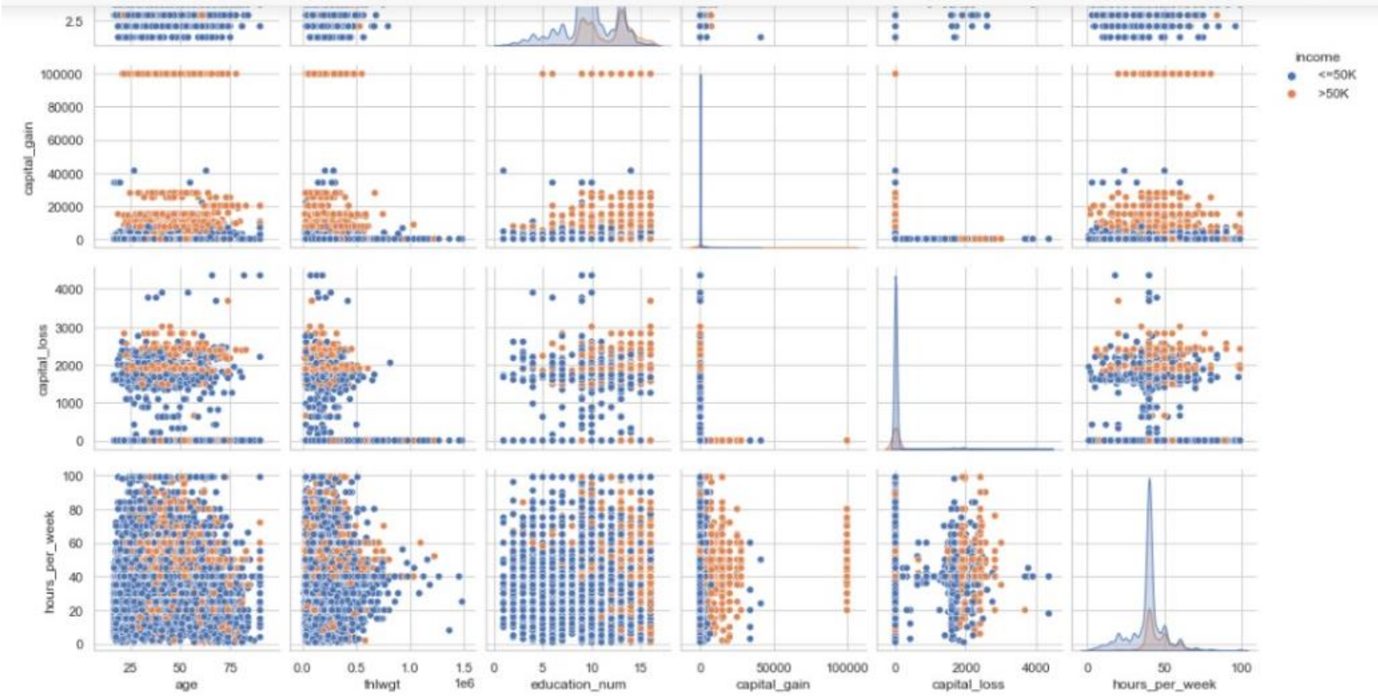


Interpretation

- We can see that `age` and `fnlwgt` are positively skewed.
- The variable `education_num` is negatively skewed while `hours_per_week` is normally distributed.
- There exists weak positive correlation between `capital_gain` and `education_num` (correlation coefficient=0.1226).

In [89]:
```
sns.pairplot(df, hue="income")
plt.show()
```




income
<=50K
>50K
capital_gain
capital_loss
hours_per_week
age
fnlwgt
1e6
education_num
capital_gain
capital_loss
hours_per_week


income
<=50K
>50K
education_num
capital_gain
capital_loss

7. Declare feature vector and target variable

In [91]:
```
X = df.drop(['income'], axis=1)

y = df['income']
```

8. Split data into separate training and test set

In [92]:
```
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 [93]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[93]:
```
((22792, 14), (9769, 14))
```

9. Feature Engineering .

9.1 Display categorical variables in training set

9. Feature Engineering .

9.1 Display categorical variables in training set

In [94]:
```
categorical = [col for col in X_train.columns if X_train[col].dtypes == 'O']

categorical
```

Out[94]:
```
['workclass',
 'education',
 'marital_status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native_country']
```

9.2 Display numerical variables in training set

In [95]:
```
numerical = [col for col in X_train.columns if X_train[col].dtypes != 'O']

numerical
```

Out[95]:
```
['age',
 'fnlwgt',
 'education_num',
 'capital_gain',
 'capital_loss',
```

9.3 Engineering missing values in categorical variables

In [96]:
```
# print percentage of missing values in the categorical variables in training set

X_train[categorical].isnull().mean()
```

Out[96]:
```
workclass         0.055985
education         0.000000
marital_status    0.000000
occupation        0.056072
relationship      0.000000
race              0.000000
sex               0.000000
native_country    0.018164
dtype: float64
```

In [97]:
```
# print categorical variables with missing data

for col in categorical:
    if X_train[col].isnull().mean()>0:
        print(col, (X_train[col].isnull().mean()))
```

```
workclass 0.055984555984555984
occupation 0.05607230607230607
native_country 0.018164268164268166
```

In [98]:
```
# impute missing categorical variables with most frequent value

for df2 in [X_train, X_test]:
    df2['workclass'].fillna(X_train['workclass'].mode()[0], inplace=True)
    df2['occupation'].fillna(X_train['occupation'].mode()[0], inplace=True)
    df2['native_country'].fillna(X_train['native_country'].mode()[0], inplace=True)
```

In [99]:
```
# check missing values in categorical variables in X_train

X_train[categorical].isnull().sum()
```

Out[99]:
```
workclass         0
education         0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
native_country    0
dtype: int64
```

In [100]:
```
# check missing values in categorical variables in X_test

X_test[categorical].isnull().sum()
```

Out[100]:
```
workclass         0
education         0
marital_status    0
occupation        0
```

In [101]:
```
# check missing values in X_train

X_train.isnull().sum()
```

Out[101]:
```
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
dtype: int64
```

In [102]:
```
# check missing values in X_test

X_test.isnull().sum()
```

Out[102]:
```
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
dtype: int64
```

We can see that there are no missing values in X_train and X_test.

9.4 Encode categorical variables

In [103]:
```python
# preview categorical variables in X_train

X_train[categorical].head()
```

Out[103]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_country
32098	Private	HS-grad	Married-civ-spouse	Craft-repair	Husband	White	Male	United-States
25206	State-gov	HS-grad	Divorced	Adm-clerical	Unmarried	White	Female	United-States
23491	Private	Some-college	Married-civ-spouse	Sales	Husband	White	Male	United-States
12367	Private	HS-grad	Never-married	Craft-repair	Not-in-family	White	Male	Guatemala
7054	Private	7th-8th	Never-married	Craft-repair	Not-in-family	White	Male	Germany

In [104]:
```python
# import category encoders

import category_encoders as ce
```

In [105]:
```python
# encode categorical variables with one-hot encoding

encoder = ce.OneHotEncoder(cols=['workclass', 'education', 'marital_status', 'occupation', 'relationship',
                                 'race', 'sex', 'native_country'])

X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
```

In [106]:
```python
X_train.head()
```

Out[106]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	workclass_8	fnlwgt	...	native_country_32	native_c
32098	45	1	0	0	0	0	0	0	0	170871	...	0	
25206	47	0	1	0	0	0	0	0	0	108890	...	0	
23491	48	1	0	0	0	0	0	0	0	187505	...	0	
12367	29	1	0	0	0	0	0	0	0	145592	...	0	
7054	23	1	0	0	0	0	0	0	0	203003	...	0	

5 rows × 105 columns

In [107]:
```python
X_train.shape
```

In [109]:
```
X_test.shape
```
Out[109]:
```
(9769, 105)
```

- We now have training and testing set ready for model building. Before that, we should map all the feature variables onto the same scale. It is called **feature scaling**. We will do it as follows.

10. Feature Scaling

In [110]:
```
cols = X_train.columns
```

In [111]:
```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [112]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [113]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

We now have X_train dataset ready to be fed into the Random Forest classifier. We will do it as follows.

In [107]:
```
X_train.shape
```
Out[107]:
```
(22792, 105)
```

We can see that from the initial 14 columns, we now have 105 columns in training set.

Similarly, I will take a look at the X_test set.

In [108]:
```
X_test.head()
```
Out[108]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	workclass_8	fnlwgt	...	native_country_32	native_c
22278	27	1	0	0	0	0	0	0	0	177119	...	0	
8950	27	1	0	0	0	0	0	0	0	216481	...	0	
7838	25	1	0	0	0	0	0	0	0	256263	...	0	
16505	46	1	0	0	0	0	0	0	0	147640	...	0	
19140	45	1	0	0	0	0	0	0	0	172822	...	0	

5 rows × 105 columns

In [109]:
```
X_test.shape
```
Out[109]:
```
(9769, 105)
```

We now have X_train dataset ready to be fed into the Random Forest classifier. We will do it as follows.

11. Random Forest Classifier model with default parameters

In [114]:
```
# import Random Forest classifier

from sklearn.ensemble import RandomForestClassifier


# instantiate the classifier

rfc = RandomForestClassifier(random_state=0)


# fit the model

rfc.fit(X_train, y_train)


# Predict the Test set results

y_pred = rfc.predict(X_test)
```

Here, I have build the Random Forest Classifier model with default parameter of `n_estimators = 10`. So, I have used 10 decision-trees to build the model. Now, I will increase the number of decision-trees and see its effect on accuracy.

12. Random Forest Classifier model with 100 Decision Trees)

In [115]:
```
# instantiate the classifier with n_estimators = 100

rfc_100 = RandomForestClassifier(n_estimators=100, random_state=0)


# fit the model to the training set

rfc_100.fit(X_train, y_train)


# Predict on the test set results

y_pred_100 = rfc_100.predict(X_test)


# Check accuracy score

print('Model accuracy score with 100 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred_100)))
```

The model accuracy score with 10 decision-trees is 0.8446 but the same with 100 decision-trees is 0.8521. So, as expected `accuracy increases` with number of decision-trees in the model.

13. Find important features with Random Forest model

Until now, We have used all the features given in the model. Now, I will select only the important features, build the model using these features and see its effect on accuracy.

First, We will create the Random Forest model as follows:-

In [116]:
```
# create the classifier with n_estimators = 100

clf = RandomForestClassifier(n_estimators=100, random_state=0)



# fit the model to the training set

clf.fit(X_train, y_train)
```

Out[116]:
```
RandomForestClassifier(random_state=0)
```

Now, We will use the feature importance variable to see feature importance scores.

In [117]:
```
# view the feature scores
```

Now, I will build the random forest model again and check accuracy.

In [120]:
```
# instantiate the classifier with n_estimators = 100

clf = RandomForestClassifier(n_estimators=100, random_state=0)



# fit the model to the training set

clf.fit(X_train, y_train)


# Predict on the test set results

y_pred = clf.predict(X_test)



# Check accuracy score

print('Model accuracy score with native_country_41 variable removed : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with native_country_41 variable removed : 0.8544
```

Interpretation

- I have removed the `native_country_41` variable from the model, rebuild it and checked its accuracy.

Out[117]:
```
fnlwgt               0.159772
age                  0.149074
capital_gain         0.091299
hours_per_week       0.086339
education_num        0.065130
                       ...
native_country_16    0.000028
occupation_14        0.000015
native_country_35    0.000009
workclass_8          0.000008
native_country_41    0.000000
Length: 105, dtype: float64
```

We can see that the most important feature is `fnlwgt` and least important feature is `native_country_41`.

14. Build the Random Forest model on selected features

Now, We will drop the least important feature `native_country_41` from the model, rebuild the model and check its effect on accuracy.

In [119]:
```python
# drop the least important feature from X_train and X_test

X_train = X_train.drop(['native_country_41'], axis=1)

X_test = X_test.drop(['native_country_41'], axis=1)
```

Now, I will build the random forest model again and check accuracy.

Interpretation

- I have removed the `native_country_41` variable from the model, rebuild it and checked its accuracy.
- The accuracy of the model now comes out to be 0.8544.
- The accuracy of the model with all the variables taken into account is 0.8521.
- So, we can see that the model accuracy has been improved with `native_country_41` variable removed from the model.

Now, based on the above analysis we can conclude that our classification model accuracy is very good. Our model is doing a very good job in terms of predicting the class labels.

But, it does not give the underlying distribution of values. Also, it does not tell anything about the type of errors our classifer is making.

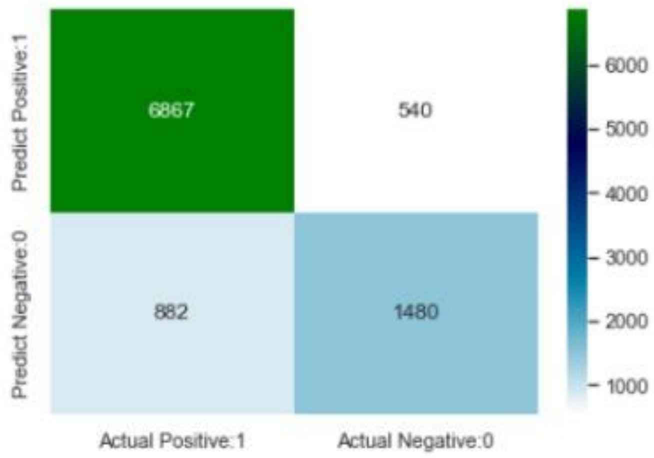
We have another tool called `Confusion matrix` that comes to our rescue.

In [121]:
```python
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)
```

```
Confusion matrix

 [[6867  540]
 [ 882 1480]]
```

```python
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='ocean_r')
```

Out[139]: <AxesSubplot:>



15. Classification Report

In [123]:
```python
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```


15. Classification Report

In [123]:
```python
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```


```
              precision    recall  f1-score   support

       <=50K       0.89      0.93      0.91      7407
        >50K       0.73      0.63      0.68      2362

    accuracy                           0.85      9769
   macro avg       0.81      0.78      0.79      9769
weighted avg       0.85      0.85      0.85      9769
```

17. Results and Conclusion

1. In this project, We build a Random Forest Classifier to predict the income of a person. We build two models, one with 10 decision-trees and another one with 100 decision-trees.
2. The model accuracy score with `10 decision-trees is 0.8446` but the same with `100 decision-trees is 0.8521`. So, as expected accuracy increases with number of decision-trees in the model.
3. We have used the Random Forest model to find only the important features, build the model using these features and see its effect on accuracy.
4. We have removed the `native_country_41` variable from the model, rebuild it and checked its accuracy. The `accuracy of the model with native_country_41 variable removed is 0.8544`. So, we can see that the model accuracy has been improved with `native_country_41` variable removed from the model.
5. Confusion matrix and classification report are another tool to visualize the model performance. They yield good performance.

That is the end of this kernel. We hope you find it useful and enjoyable.

Your feedback and comments are most welcome.

THANKS TO ALL THE GROUP MEMBERS: =

a. **Riddhey vyas**
b. **Deeptimayee Maharana**
c. **483 chaithanya**
d. **Akhil Kumar reddy**
e. **Sumanth polieni**
f. **Thota samanvitha**
g. **Neeha shaik**
h. **Usha Sri**
i. **Ranhith T**

Thank you