IDS Project Report :-

ML Classification on Census Income Dataset

Team Members:

1. Shail Kardani – 19ucs217
2. Kush Gandhi – 19ucs131
3. Manan Gandhi – 19ucs130
4. Gunjan Pagdhar – 19ucs101

INDEX:-

1) Introduction:

This data was extracted from the 1994 Census Bureau database by Ronny Kohavi and Barry Becker.
A set of reasonable clean records was extracted using the following conditions:
((AAGE>16) && (AGI>100) && (AFNLWGT>1) && (HRS WK>0)).

Task:- The prediction task is to determine whether a person makes over $50k a year.

Dataset needs to be explored to find insights and significant relationships between different attributes and target variables (which in our case is to predict whether a person earns more than $50k a year). Then apply ML classification algorithms on the data to train proper models and get accurate inferences. The dataset is publicly available for research purposes on various platforms.

This dataset was taken from Kaggle:- Data Set Link.

Github link for code(coded in Jupyter Notebook) :- Github Link.

Description:

The dataset contains 15 rows (features) of which the last features is our y-label (whether a person makes more than $50k) and a total of 32561 rows (patterns).

1 - **age:** (numeric)

2 - **workclass :** employment type (categorical: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.)

3 - **fnlwgt :** continuous.

4 - **education** : type of education (categorical : Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.)

5 - **education-num** : continuous.

6 - **marital-status** : (categorical: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.)

7 - **occupation** : (categorical : Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspect, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.)

8 - **relationship** : (categorical : Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.)

9 - **race** : (categorical : White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.)

10 - **Sex** : (categorical : Male, Female)

11 - **capital-gain** : continuous.

12 - **capital-loss** : continuous.

13 - **hours-per-week** : continuous.

14 - **native-country** : (categorical : United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad Tobago, Peru, Hong, Holland-Netherlands.

2) Importing The Data:

The dataset is taken from Kaggle (https://www.kaggle.com/uciml/adult-census-income).

Before importing the dataset, we need to import proper libraries. For data importing and data visualization, we have used the **pandas**, **numpy**, **matplotlib** and **seaborn** library.

Importing Necessary Libraries

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
```

```
In [2]: data = pd.read_csv('adult.csv')
```

3) Exploring the Data

Let's have a look at the values of few of the features of our datasets.

```
In [3]: data.head(10)
```

Out[3]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356	40	Unit
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18	Unit
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356	40	Unit
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	40	Unit
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	40	Unit
5	34	Private	216864	HS-grad	9	Divorced	Other-service	Unmarried	White	Female	0	3770	45	Unit
6	38	Private	150601	10th	6	Separated	Adm-clerical	Unmarried	White	Male	0	3770	40	Unit
7	74	State-gov	88638	Doctorate	16	Never-married	Prof-	Other-	White	Female	0	3683	20	Unit

Now Let's look at the shape and description of our data.

```
In [4]: data.shape
Out[4]: (32561, 15)
```

Thus the data has total of 32561 rows/instances and 15 columns including the y-label

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

```
 #   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
```

It is important to get the information of the dataset in order to figure out the total number of null values that we need to take care of before training the model. In our case there are null values that need to be accounted for.

Now, we will look at the description of the data (**count**, **mean**, **standard deviation** for the continuous values)

```
In [6]: data.describe()
```

Out[6]:

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

Next up, we will take a look at the total number of columns(features) that we have,

```
In [8]: data.columns
```

```
Out[8]: 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')
```

Observations:- As we have already seen there are no null values in our dataset. But there are some unwanted values that need to be accounted for before training our model or visualization. For example, some features contain values as '?', which is in unwanted format. To remove this, we have written a function to first list all the features containing these values and then handle them with appropriate methods.

```
In [10]: for column in data.columns:
             print(f"{column} = {data[data[column] == '?'].shape[0]}")
```

```
age = 0
workclass = 1836
fnlwgt = 0
education = 0
education.num = 0
marital.status = 0
occupation = 1843
relationship = 0
race = 0
sex = 0
capital.gain = 0
capital.loss = 0
hours.per.week = 0
native.country = 583
income = 0
```

There are various methods to handle these values. The easiest one is to simply remove the pattern. We have chosen to replace them with the mode of the features in order to preserve the number of patterns.

```
In [11]: data["workclass"][data["workclass"] == "?"] = data["workclass"].mode()[0] #Replace the '?' values with the mode
         data["occupation"][data["occupation"] == "?"] = data["occupation"].mode()[0]
         data["native.country"][data["native.country"] == "?"] = data["native.country"].mode()[0]
```

In [12]:
```
for column in data.columns:
    print(f"{column} = {data[data[column] == '?'].shape[0]}")
```

```
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
```

At this point no column in the data has any null values. This concludes our initial data preprocessing and now we will visualize some of the features to get further understanding of the data.

4) Data Visualization

```
In [13]: data.dtypes
```

```
Out[13]: 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
```

As we can see that there are two kinds of features, continuous and categorical.
We will first start with categorical data.

Categorical data

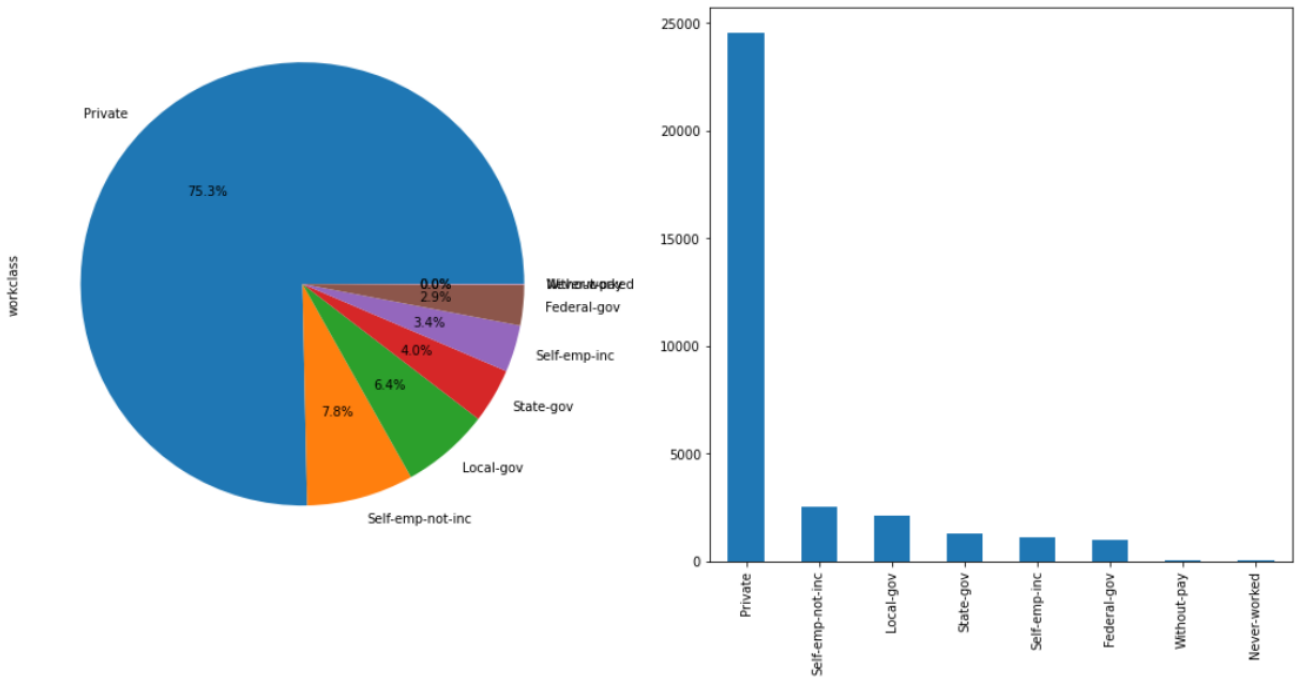
Analyzing few of the important categorical features

```
In [14]: categorical_features = ['workclass','education','marital.status','occupation','relationship','race','sex','native.country','incom
```

1) Workclass:

```
In [15]: f,ax = plt.subplots(1,2,figsize=(18,8))
         data['workclass'].value_counts().plot.pie(autopct='%1.1f%%', ax=ax[0])
         data['workclass'].value_counts().plot.bar(ax=ax[1])
```

Out[15]: <AxesSubplot:>

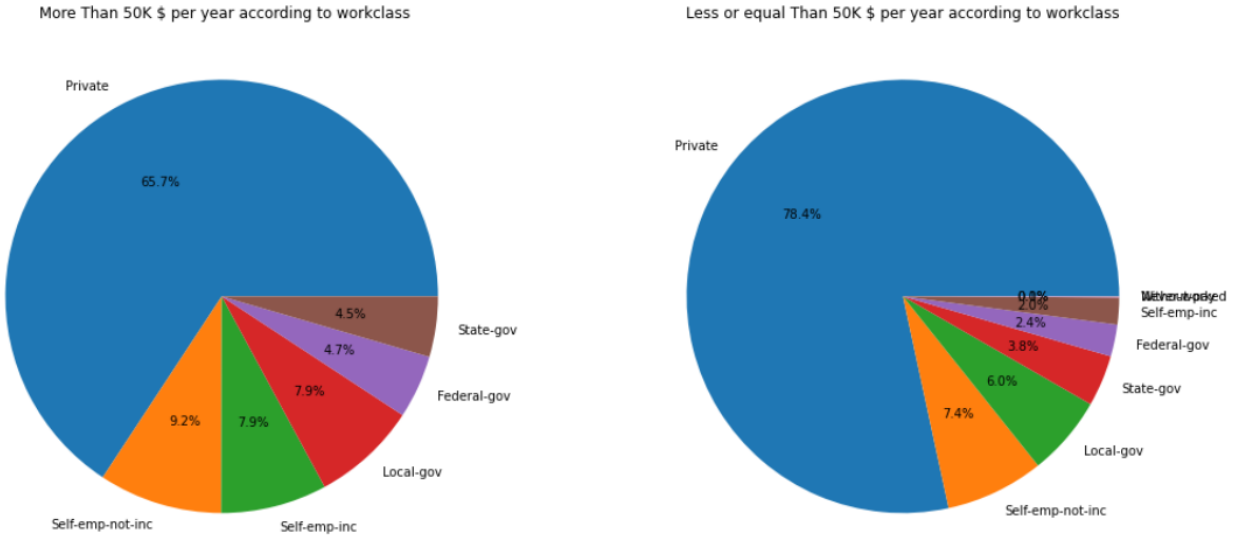


Observation : This pie chart and bar graph shows us the initial distribution of people in workclass. More than 70% of the population works in the private sector whereas less than 1% belongs to ‘never-worked’.

This graph doesn't give us more information about how the population is divided in workclass in each part(>$50k and $50k).

In [16]:
```
f,ax=plt.subplots(1,2,figsize=(18,8))
data[data['income'] == '>50K']['workclass'].value_counts().plot.pie(autopct='%1.1f%%', ax=ax[0])
ax[0].set_title(' More Than 50K $ per year according to workclass')
ax[0].set_ylabel('')

data[data['income'] == '<=50K']['workclass'].value_counts().plot.pie(autopct='%1.1f%%', ax=ax[1])
ax[1].set_title('Less or equal Than 50K $ per year according to workclass')
ax[1].set_ylabel('')
```

Out[16]:
```
Text(0, 0.5, '')
```



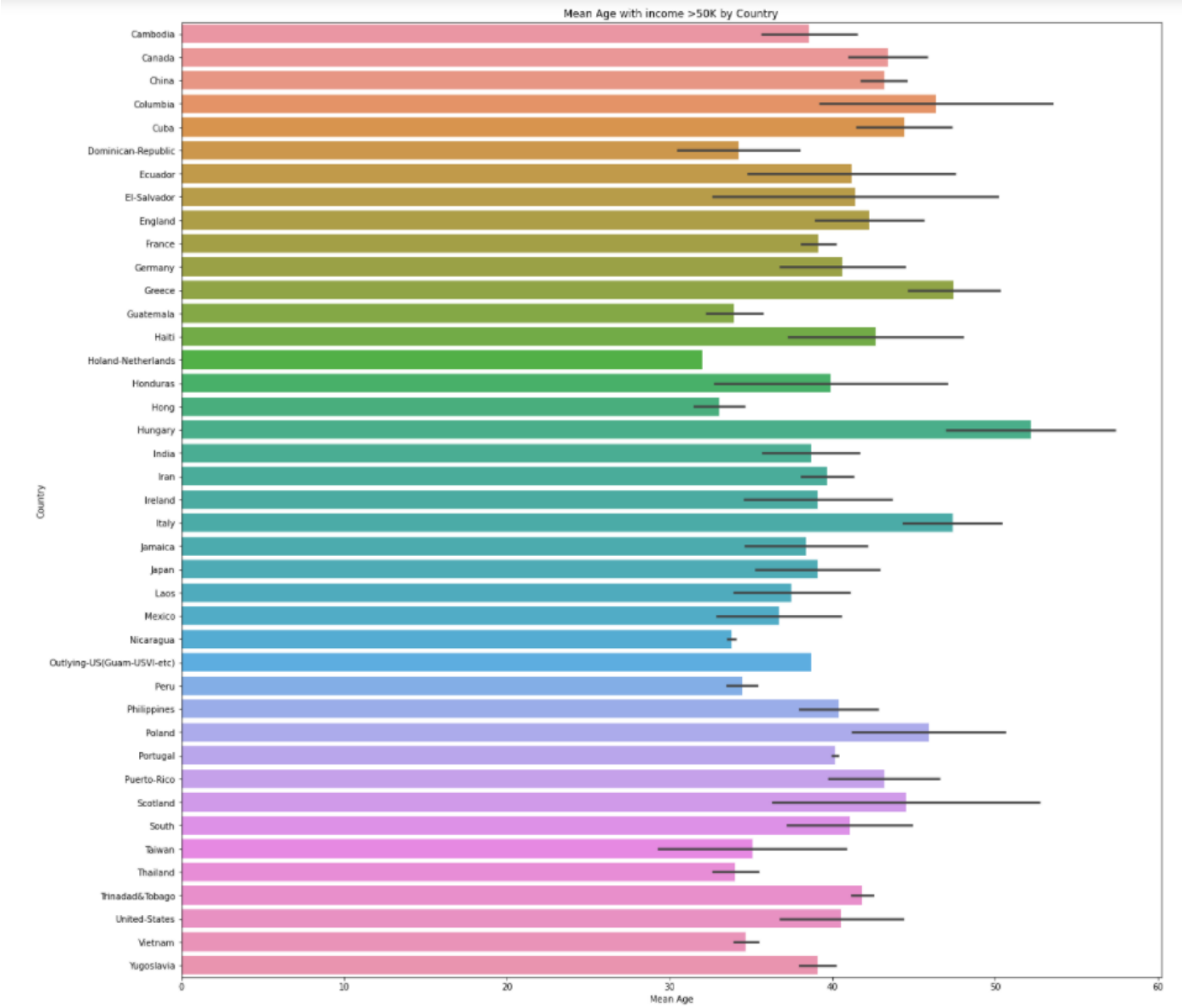
Observation :
From the above pie chart we can observe many assumption including the fact that:-

Most of the high paying job(>$ 50k) can be found at Private Sector and at the same time most of the low paying job are also found at public sector.

2) Age:

Another important feature that can give us insight on how population is distributed is the ‘Age’.

```
In [17]: temp_data = data[["native.country","income","age"]].groupby(["native.country","income"]).mean()

         temp_data = temp_data.reset_index()

         plt.figure(figsize = (20,20))
         sns.barplot(x = "age", y = "native.country", data = temp_data)
         plt.xlabel("Mean Age")
         plt.ylabel("Country")
         plt.title("Mean Age with income >50K by Country")
         plt.show()
```

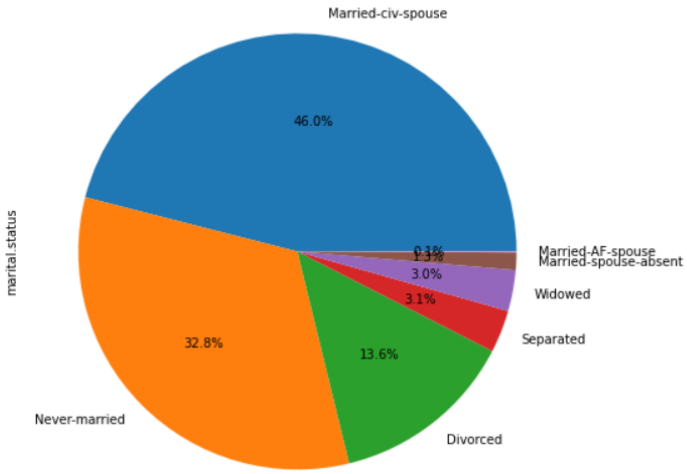


Observation : This plot is used to visualize Mean Age with income >$50k by country.
Hungary has the highest mean age with the values more than 55 and Dominican-Republic and Nicaragua are

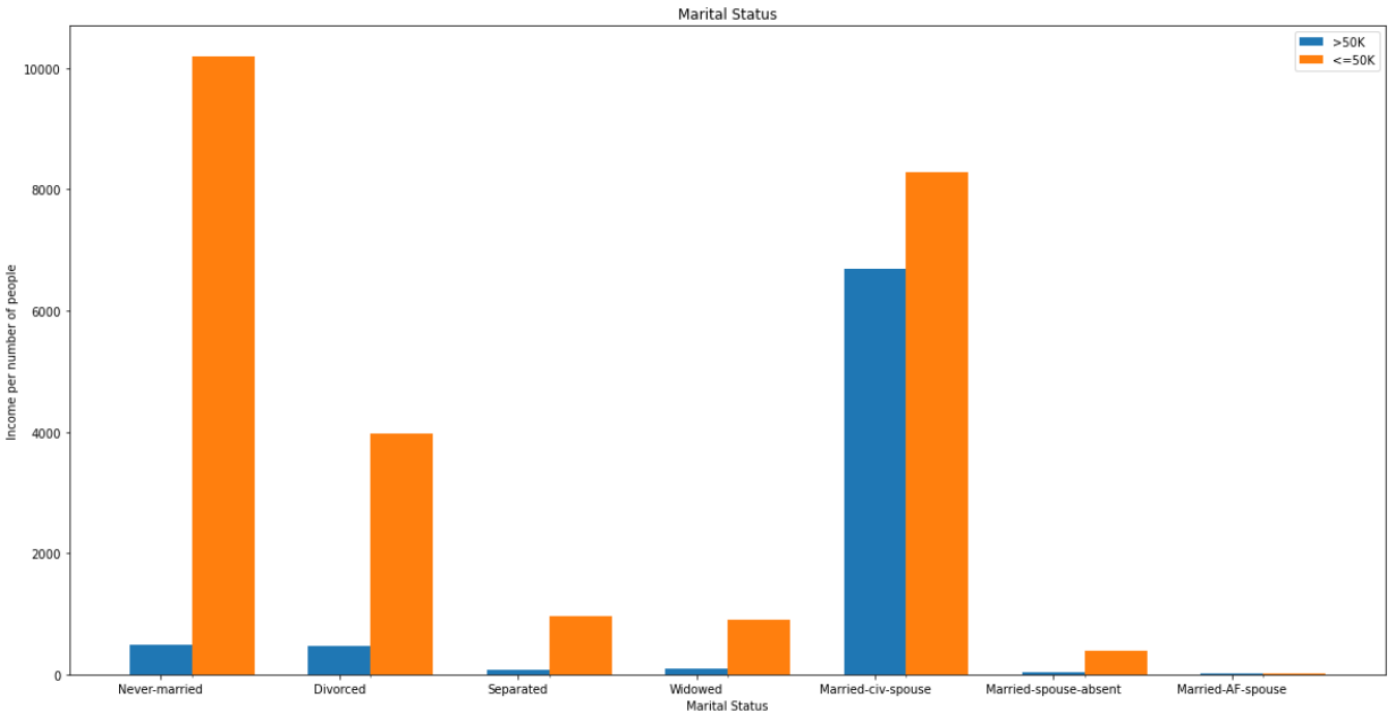
among the lowest with the value close to 30.

3) Marital Status:

```
In [19]: plt.figure(figsize=(8,8))
         data['marital.status'].value_counts().plot.pie(autopct = '%1.1f%%')
Out[19]: <AxesSubplot:ylabel='marital.status'>
```



Observations:- We begin by analyzing a simple pie chart that shows the distribution of population in terms of marital status. More than 45% of the population are Married.



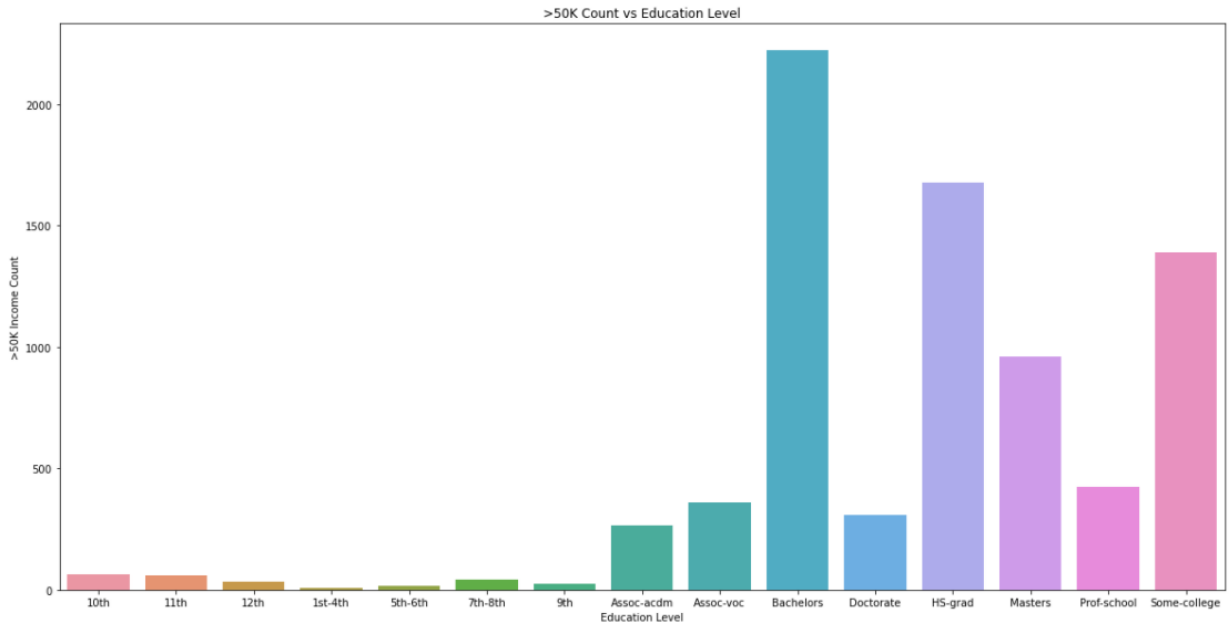
Observations:- We compared the two bar charts(>$50k and <$50k) for better understanding.This is a very telling chart. As we can see, almost half of people who are married earn more than $ 50K, most people who are separated, divorced or single earn less than $50K.
The code to produce the above bar graph with matplotlib is:

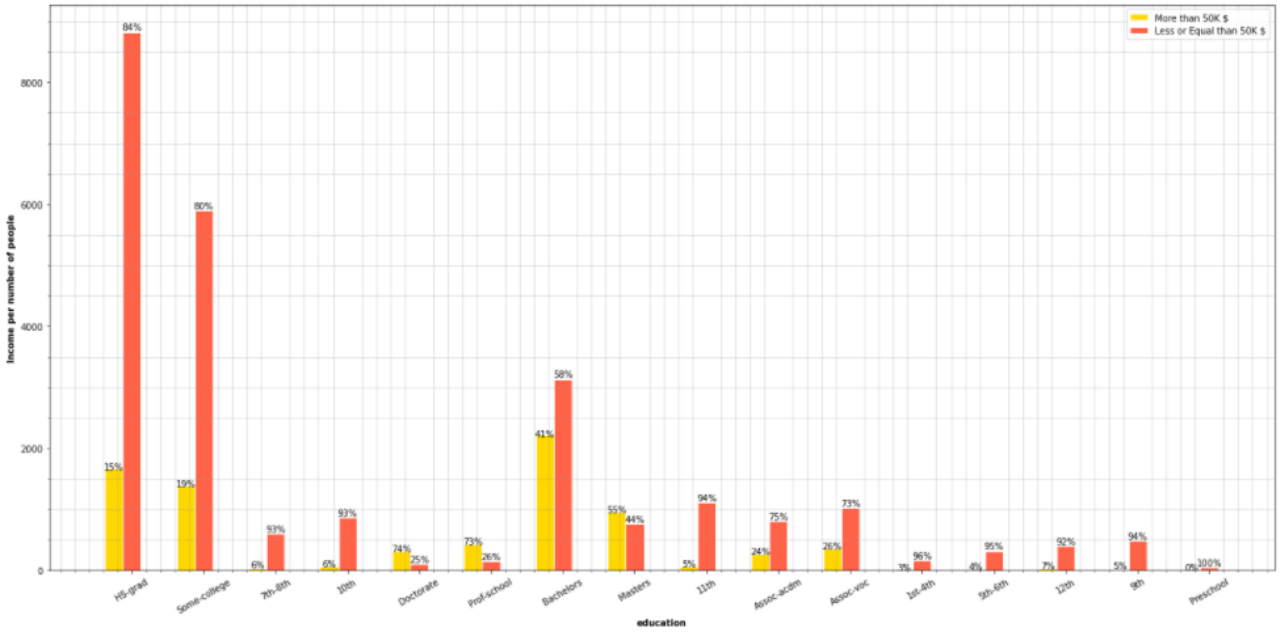
In [21]:
```
mt = ['Never-married','Divorced','Separated','Widowed','Married-civ-spouse','Married-spouse-absent','Married-AF-spouse']
temp1 = data[data['income']=='>50K']
ls1 = []
ls2 = []
for marital_type in mt:
    d = temp1[temp1['marital.status'] == marital_type]
    ls1.append(marital_type)
    count = d.shape[0]
    ls2.append(count)


temp2 = data[data['income']=='<=50K']
ls3 = []
ls4 = []
for marital_type in mt:
    d = temp2[temp2['marital.status'] == marital_type]
    ls3.append(marital_type)
    count = d.shape[0]
    ls4.append(count)
```

In [22]:
```
labels = ['Never-married','Divorced','Separated','Widowed','Married-civ-spouse','Married-spouse-absent','Married-AF-spouse']
men_means = [20, 34, 30, 35, 27]
women_means = [25, 32, 34, 20, 25]

x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

plt.figure(figsize=(20,5))
fig, ax = plt.subplots()
fig.set_figwidth(20)
fig.set_figheight(10)
rects1 = ax.bar(x - width/2, ls2, width, label='>50K',tick_label=labels)
rects2 = ax.bar(x + width/2, ls4, width, label='<=50K')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Income per number of people')
ax.set_xlabel('Marital Status')
ax.set_title('Marital Status')
ax.set_xticks(x, labels)
ax.legend()

plt.show()
```

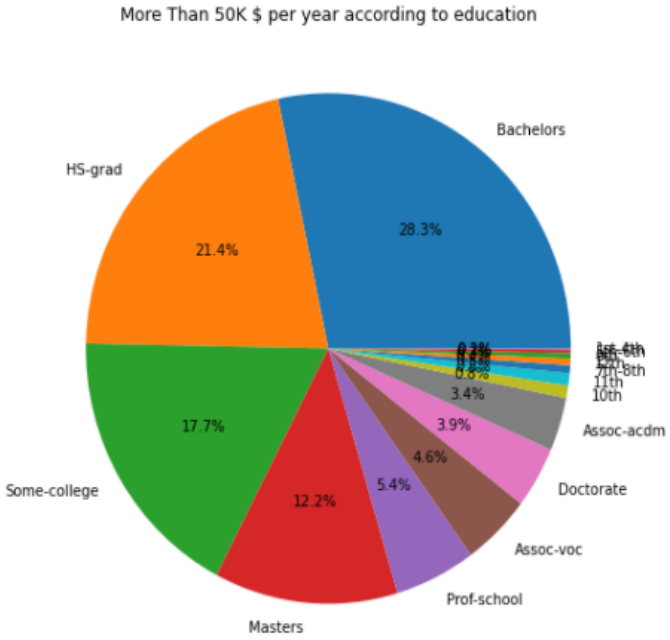
4) Education:

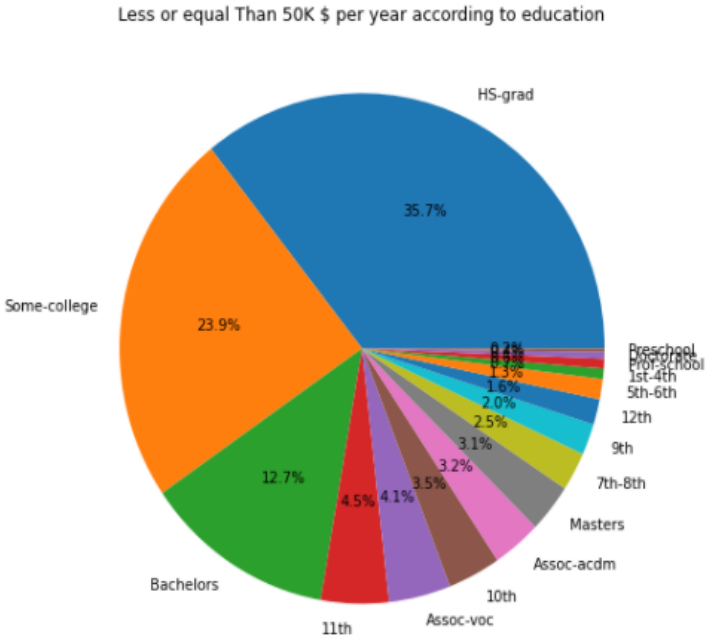
In [95]:
```
plt.figure(figsize = (20,10))
temp = data[data['income']=='>50K']
ed_data = temp.groupby("education")["income"].count()
ed_data = ed_data.reset_index()


sns.barplot(x = "education", y ="income", data = ed_data)
plt.xlabel("Education Level")
plt.ylabel(">50K Income Count")
plt.title(">50K Count vs Education Level")
plt.show()
```





Observations:- The first image shows the general distribution of population. But it doesn't give us much insight about the data. The second graph however shows us some interesting information. We can see that most people who are school professors and most people holding a Master degree or a PhD earn more than $50K per year. It's interesting that 41% of people owning a bachelor's degree tend to earn more than 50,000\$ a year. The observations we can draw here is that people who went to college and have professional degrees tend to earn more than $50K per year.





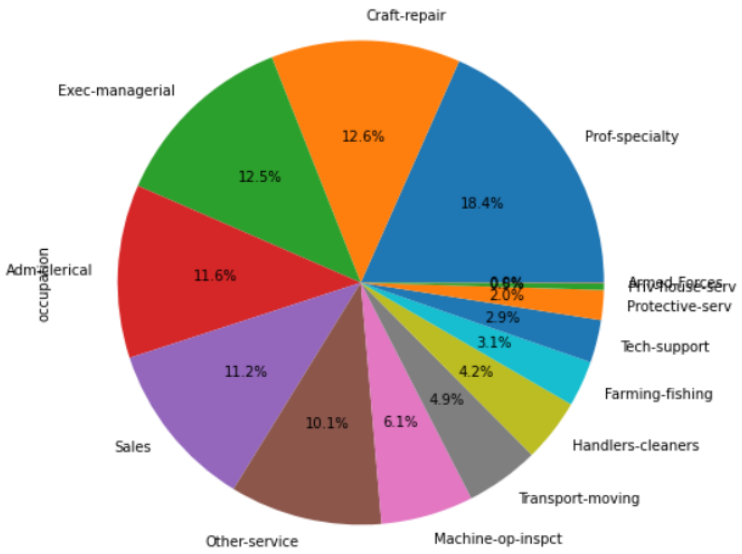
Observations:- Now, if we look at the charts above, among people earning more than $50K grouped by education we can see that half of the people have, at least, a college degree or are high school graduates (HS-grad). On the other hand, the other pie chart presents a similar distribution but, as we saw in the previous charts, we can see that people earning a Master's degree or a PhD tend to earn more than $50K.

5) Occupation:

Next up, we will see what kind of impact does occupation have on the salary.

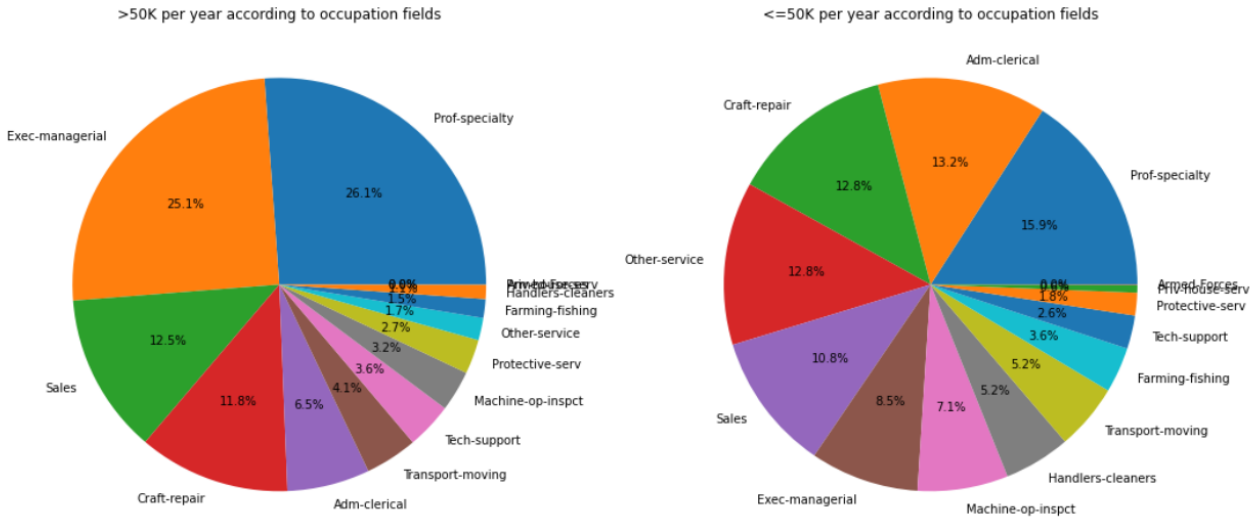
```
In [25]: plt.figure(figsize=(8, 8))
         data['occupation'].value_counts().plot.pie(autopct='%1.1f%%')
```

```
Out[25]: <AxesSubplot:ylabel='occupation'>
```



Observations:- As we can clearly see from the pie chart, that most of the occupation types are evenly distributed.

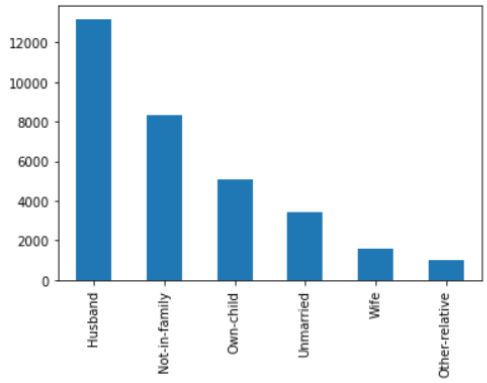
```
In [26]: f,ax=plt.subplots(1,2,figsize=(18,8))
         data[data['income'] == '>50K']['occupation'].value_counts().plot.pie(autopct='%1.1f%%', ax=ax[0])
         ax[0].set_title('>50K per year according to occupation fields')
         ax[0].set_ylabel('')
         data[data['income'] == '<=50K']['occupation'].value_counts().plot.pie(autopct='%1.1f%%', ax=ax[1])
         ax[1].set_title('<=50K per year according to occupation fields')
         ax[1].set_ylabel('')
Out[26]: Text(0, 0.5, '')
```



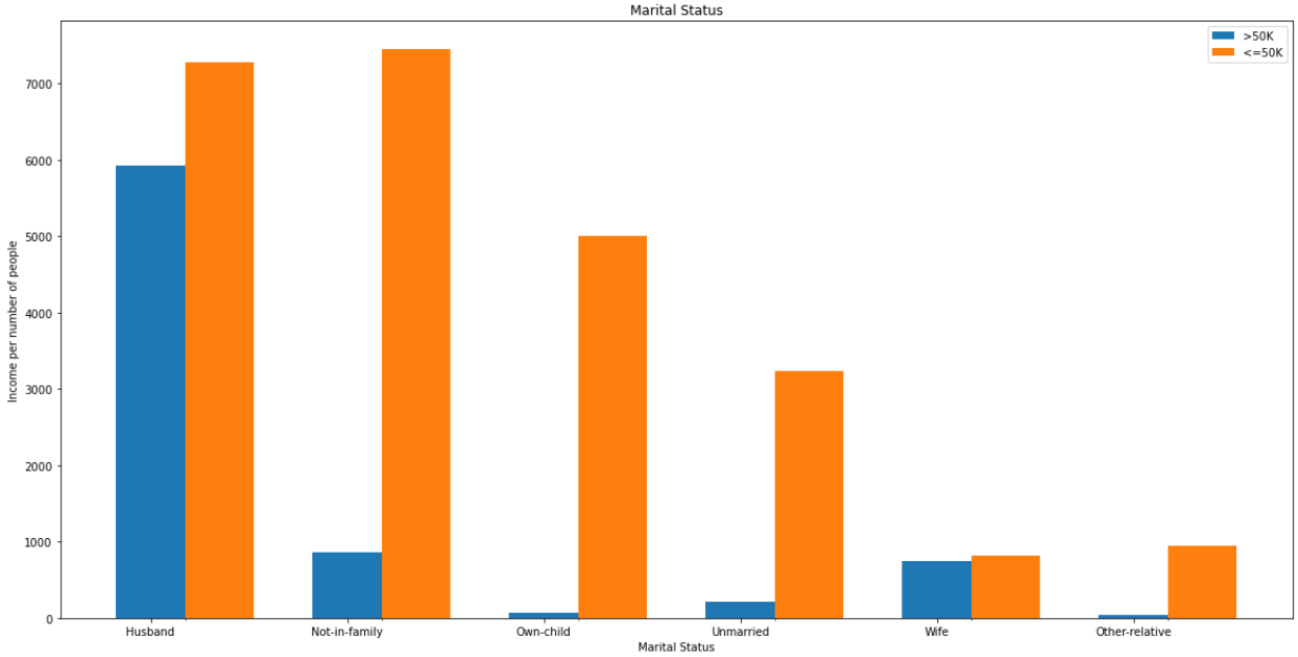
Observations:- In the above pie chart, we separated the population based on salary(label) and then made pie chart for each class.
We can see that most well paid jobs are related to Executive Managers, specialized professors, technology engineers and protection services.

6) Relation

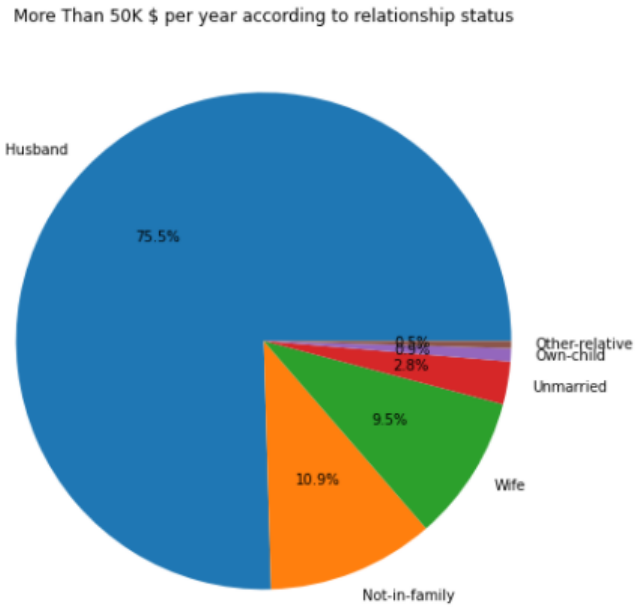
```
In [27]: data['relationship'].value_counts().plot.bar()
         plt.ylabel('')
Out[27]: Text(0, 0.5, '')
```

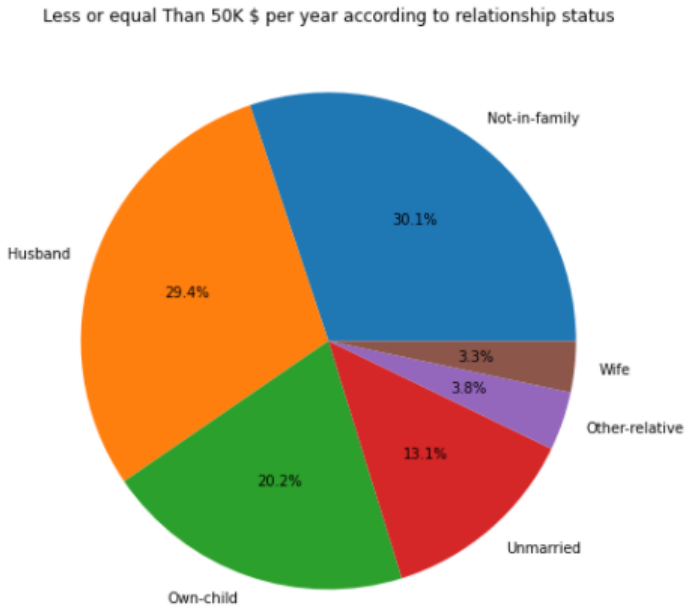


Observations:- We begin the analysis with a simple bar graph that shows that most of the family members that earn are husbands.



Observations:- An interesting fact is that 44% of people earning more than $50K are married men, but it's even more interesting that the percentage of married women earning $50K is slightly higher. Let's divide the information by groups of people who earn more and less than 50,000$.





Observations:- The pie charts show that, in general, most people earning more than $50K are married men. On the other pie charts the information is much more distributed.

This concludes an analysis of some of the important features. Now we will move on the continuous values.

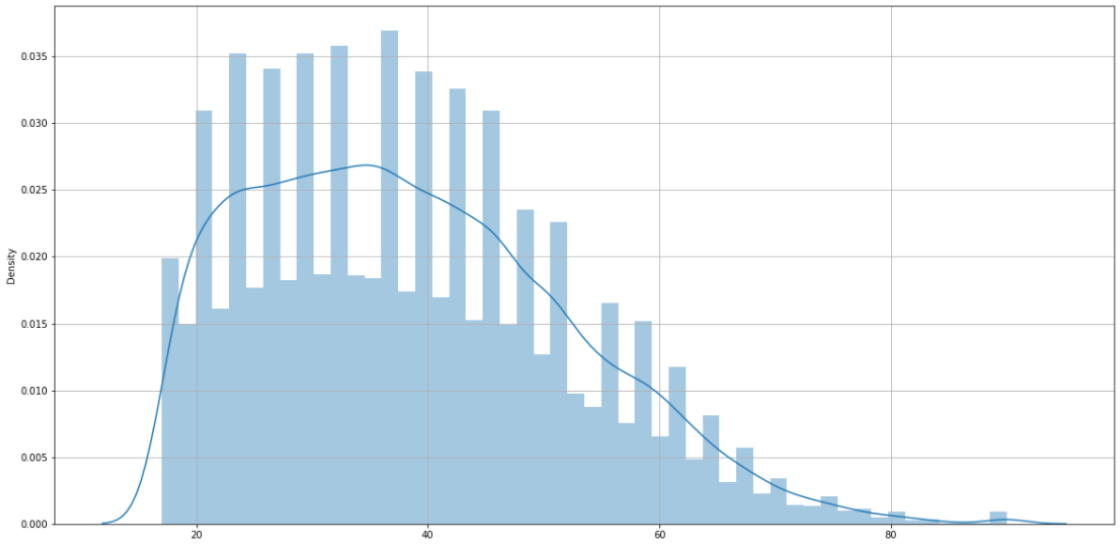
Numerical Analysis:

1) Age:

In [30]:
```
plt.figure(figsize=(20,10))
plt.grid()
sns.distplot(data['age'])
```

```
C:\Users\shail\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

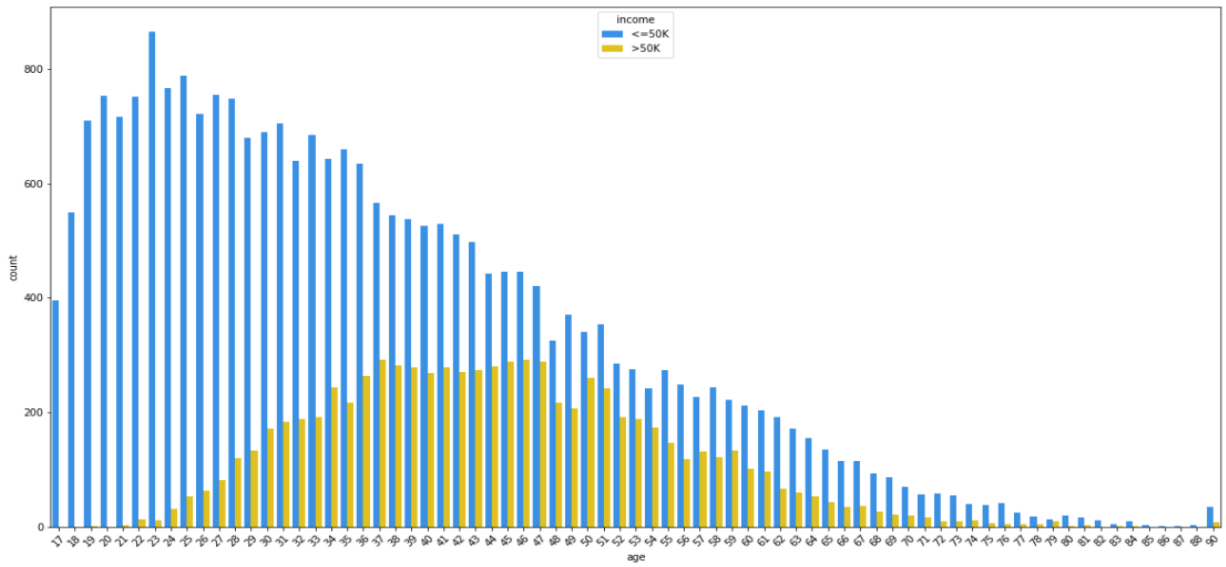
Out[30]: <AxesSubplot:xlabel='age', ylabel='Density'>



In [31]:
```
plt.figure(figsize=(20, 10))
plt.xticks(rotation=45)
sns.countplot(data['age'], hue=data['income'], palette=['dodgerblue', 'gold'])
```

```
C:\Users\shail\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[31]: <AxesSubplot:xlabel='age', ylabel='count'>



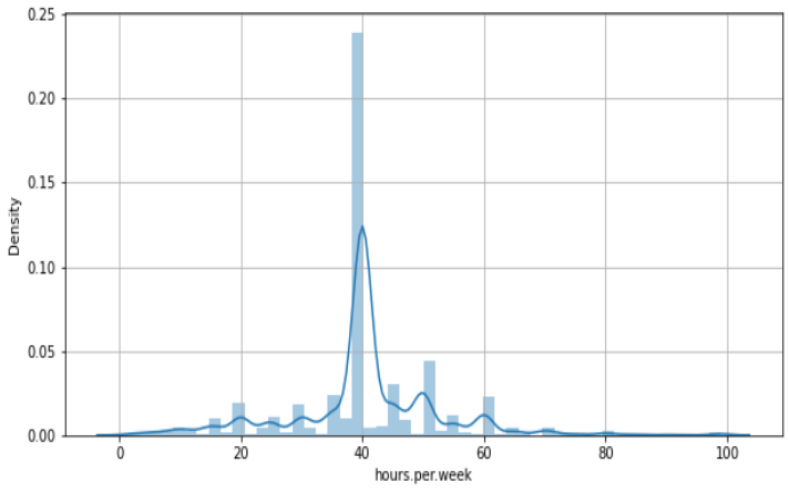
Observations:- In the first graph we can see, the age distribution collected in the census is concentrated from 20 y/o to the 50 y/o interval.
The second graph gives us very useful information. As age grows, there are more people earning more than $50K, so we can say that, generally, income is correlated with age.

2) Hours Per Week:

In [32]:
```
plt.figure(figsize=(10,5))
plt.grid()
sns.distplot(data['hours.per.week'])
```

```
C:\Users\shail\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[32]: <AxesSubplot:xlabel='hours.per.week', ylabel='Density'>



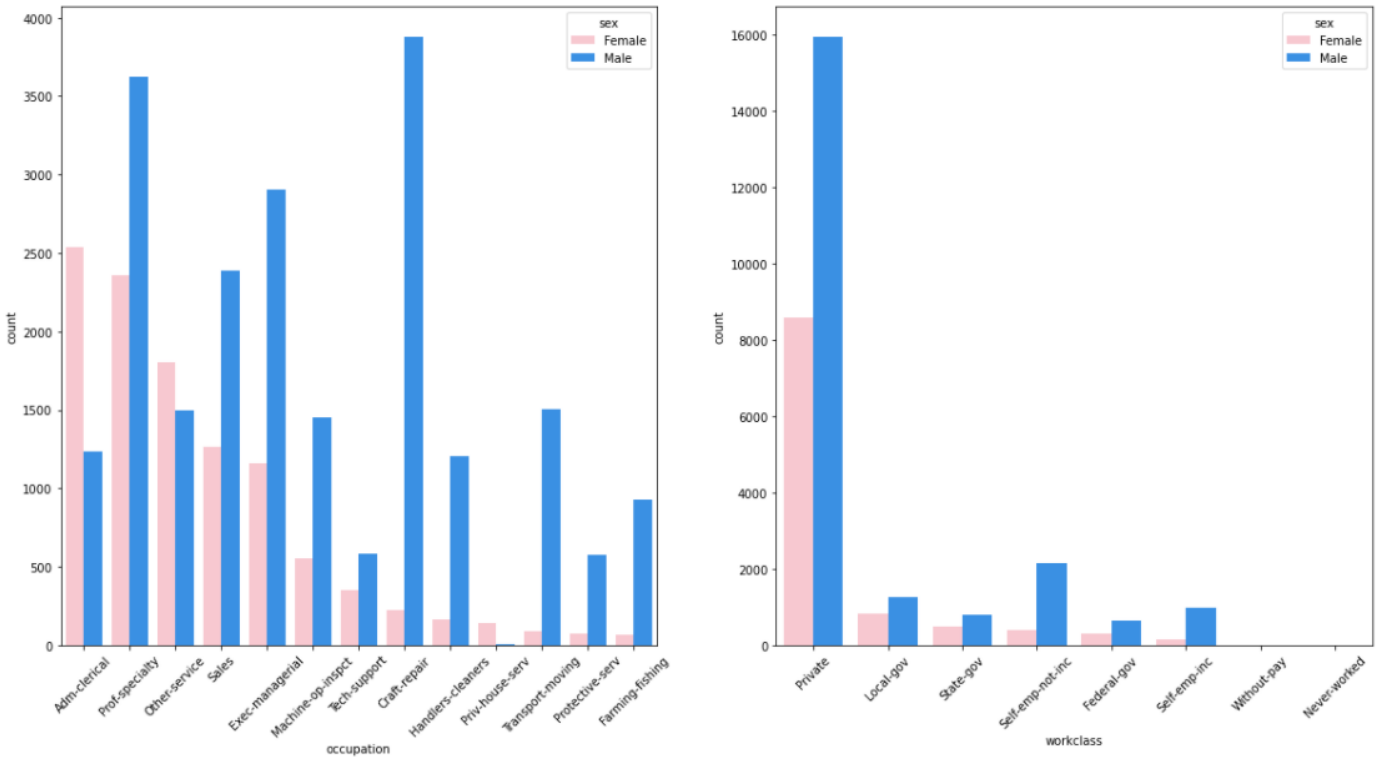
Observations:- The plot shows that most people in the census work 40 hours per week. Now, we'd like to know the hours per week distribution of the people earning more than $50K. Normally, people who earn more than $50K per year have a 40 hours/week routine. There are also a lot working for 45, 50 and 60 hours/week.

Analysis Based on Gender and Age:

After analyzing some of the important features, we realized that men tend to earn more than women according to the given dataset. Hence we decided to execute a better analysis considering these two features and draw some useful information.

3) Gender and Workclass:

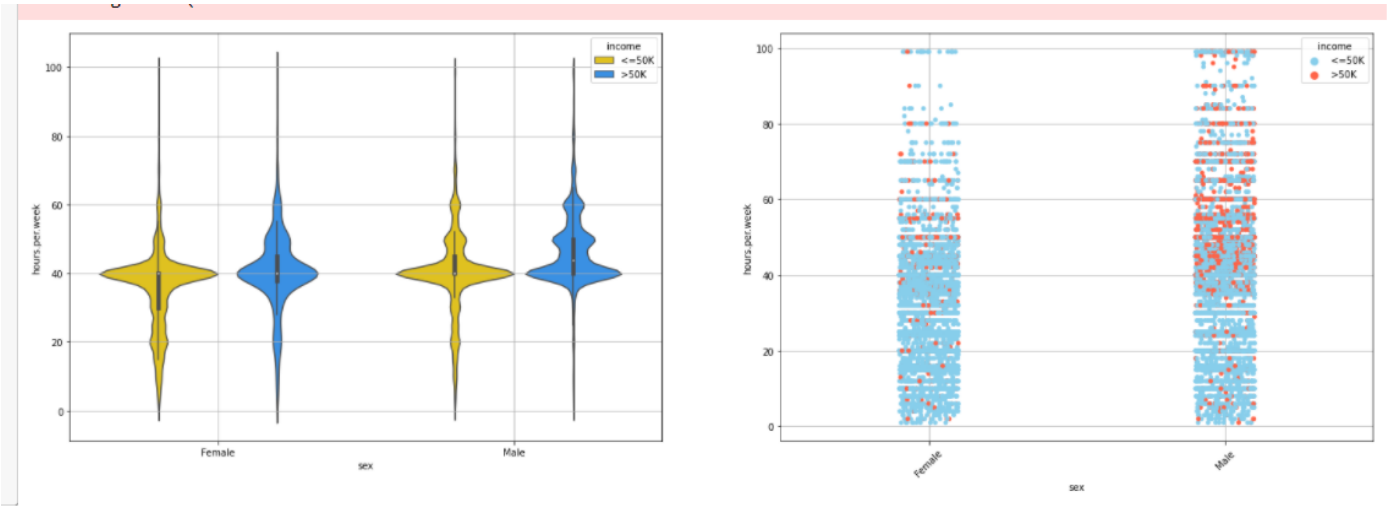
```
In [33]: fig, axs = plt.subplots(1, 2, figsize=(20, 10))
         plt.figure(figsize=(20, 10))
         sns.countplot(data['workclass'], hue=data['sex'], ax=axs[1], palette=['pink', 'dodgerblue'], order=data[data['sex'] == 'Female']
         sns.countplot(data['occupation'], hue=data['sex'], ax=axs[0], palette=['pink', 'dodgerblue'], order=data[data['sex'] == 'Female']
         plt.setp(axs[0].xaxis.get_majorticklabels(), rotation=45)
         plt.setp(axs[1].xaxis.get_majorticklabels(), rotation=45)
         plt.show()
```



Observations:- Most women occupy the jobs related to clerical administration, cleaning services and other services, but jobs related to professor speciality, business and sales, engineering, technology, transport, protection service and primary sector are mostly occupied by men. It's also interesting to see that most gender gap in private sector and self employment is bigger than in other sectors.

4) Gender, Hours Per Week and Income:

```
In [34]: fig, ax = plt.subplots(1, 2, figsize=(25, 8))
         plt.xticks(rotation=45)
         sns.violinplot(data['sex'], data['hours.per.week'], hue=data['income'], palette=['gold', 'dodgerblue'], ax=ax[0])
         sns.stripplot(data['sex'], data['hours.per.week'], hue=data['income'], palette=['skyblue', 'tomato'], ax=ax[1])
         ax[0].grid(True)
         ax[1].grid(True)
```



Observations:- The charts show that men work more hours than women. The left chart shows that, regardless of the income, there are more women working for less than men and the men chart is more distributed above 40 hours per week. The right chart shows that men working more hours tend to earn more than 50,000$. We see a concentration of red dots among the 40 and 60 hours/week interval. On the other hand, this concentration doesn't appear on the women's side. Even though the hours per week gap between men and women is not so big, it's clear that there's no correlation between hours per week and income when it comes to women.

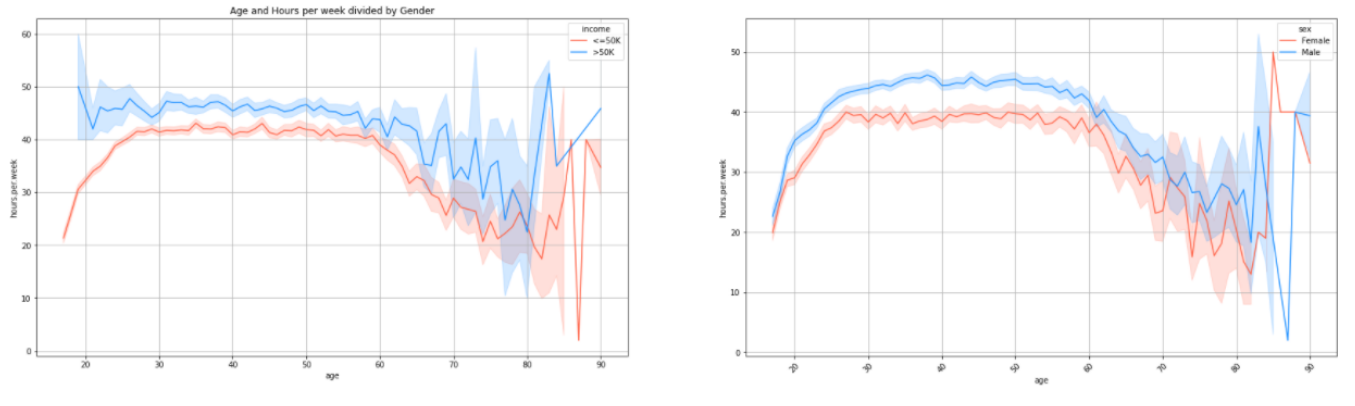
Reasons to use Violin Plot:

Here we have used a violin plot because they are easy to read. Unlike bar graphs with means and error bars, violin plots contain all data points. This makes them an excellent tool to visualize samples of small sizes. Violin plots are perfectly appropriate even if data do not conform to normal distribution. They work well to visualize both quantitative and qualitative data. The dot in the middle is the median and the box presents the interquartile range.

5) Age, Gender and Hours Per Week:

In [35]:
```
fig, ax = plt.subplots(1, 2, figsize=(30, 8))
plt.xticks(rotation=45)
sns.lineplot(data['age'], data['hours.per.week'], hue=data['income'], palette=['tomato', 'dodgerblue'], ax=ax[0])
sns.lineplot(data['age'], data['hours.per.week'], hue=data['sex'], palette=['tomato', 'dodgerblue'], ax=ax[1])
ax[0].grid(True)
ax[0].title.set_text("Age and Hours per week divided by Income")
ax[1].grid(True)
ax[0].title.set_text("Age and Hours per week divided by Gender")
```

```
C:\Users\shail\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword ar
gs: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\shail\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword ar
gs: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```



Observations:- We see a very interesting trend in the chart above. Let's take a loot at the left chart first. As the age grows, there are more people earning more than 50,000$ but working for more hours. In both cases, as the age reaches the 60 year old, people tend to work for less hours but the number of people earning more than 50K increases. What's funny is that people who earn a lot start working for more hours when they start turning 80.

The right chart shows very similar line paths. Men tend work for more hours than women, but as they get closer the standard retirement age, men and women work for the similar number of hours. What's very bizarre, is that women who are 80 and 90 are the one working for more hours than the rest of ages.

Reasons to use Line Plot:
Line graphs are common and effective charts because they are simple and easy to understand and efficient. They are great for comparing lots of data at once, showing changes and trends over time, displaying forcat data and uncertainty and also for highlighting anomalies.

This concludes the Data Visualization of the dataset. We analyzed and explored some of the important features that gave us insight on how our data is distributed and we discovered some interesting facts about various features and how they might influence our model training. Bulletin some of the import observations:-

Workclass and Occupation:

- The 55% of self employed people work are self-employed
- The 63.3% of the total people in the census earning more than $50K work in the private sector and the 71% of the total people in the census earning under $50K work in the private sector too.
- If we focus only on the private sector, the 26% earn more than $50K.
- The jobs where we can find more people earning above $50K are executive managers, protection services, college professors, engineering and jobs related to technology who are mostly occupied by men.

Education:

- It's interesting that 73% of the Professors, 74% of PhDs, the 55% of people owning a Master Degree and the 40% of Bachelors bachelors earn above $50K.
- With this information we can conclude that owning at least a college degree will increase your probabilities to earn $50K/Y.

Gender, Marital Status and Relationship:

- 85% of total people in the census earning more than $50K are married.
- 44% of people who are married earn more than $50K.
- 44% of husbands earn more than $50K.
- 47% of Wifes earn more than $50K.
- According to this info, being married increases the probability of earning above $50K.

Other information:

- The salary is directly related to age. The older people get, the more they surpass the $50K line.
- Men work for more hours than women of all ages but as they both get closer to the 60's they tend to work for similar amounts of hours per week.
- People earning more than $50K per year tend to work for more hours too.
- Men working for more than 40 hours per week tend to earn above $50K but women don't follow this trend and there's no correlation between hours per week and income when it comes to females.

5) Data Pre-Processing:

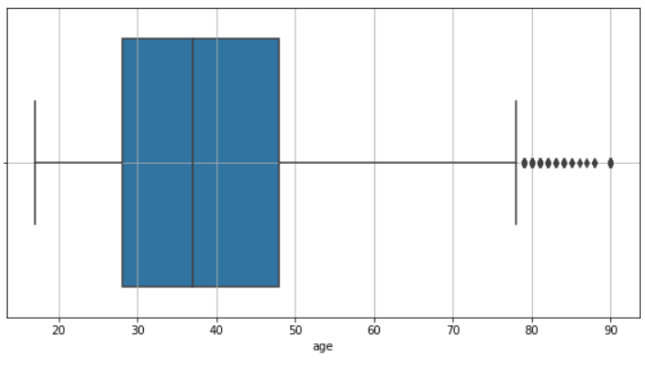
Removing Outliers:

Outliers can be very harmful for our learning models and can cause noise that can create distortions in our predictions. We'll create an auxiliary function to erase the outliers in each numerical feature.

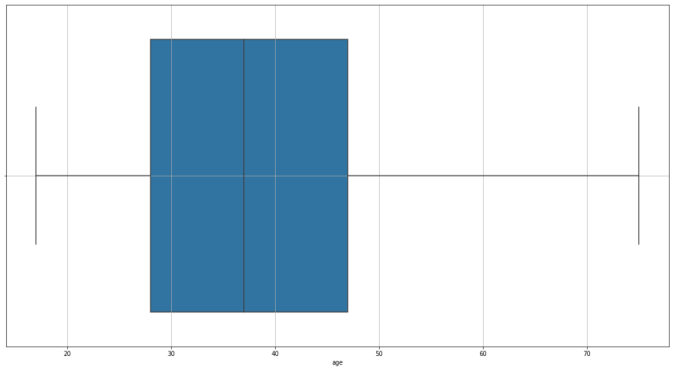
```
In [37]: def treat_outliers(data, column, upper=False, lower=False):
             Q1=adult_income_prep[column].quantile(0.25)
             Q3=adult_income_prep[column].quantile(0.75)
             IQR=Q3-Q1
             print(Q1)
             print(Q3)
             print(IQR)
             U_threshold = Q3+1.5*IQR
             #print(L_threshold, U_threshold)
             if upper:
                 adult_income_prep[column] = adult_income_prep[adult_income_prep[column] < U_threshold]
             if lower:
                 adult_income_prep[column] = adult_income_prep[adult_income_prep[column] >= U_threshold]
```

Now we will check outliers for all the numeric values and show the box and whiskers plot before and after removing the outliers.

1) Age:

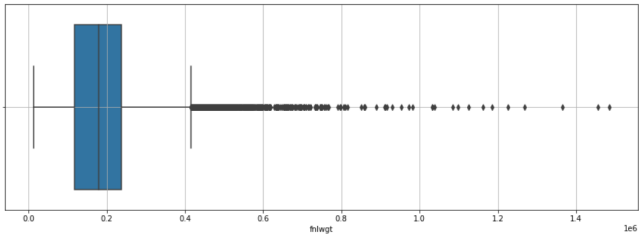


Before

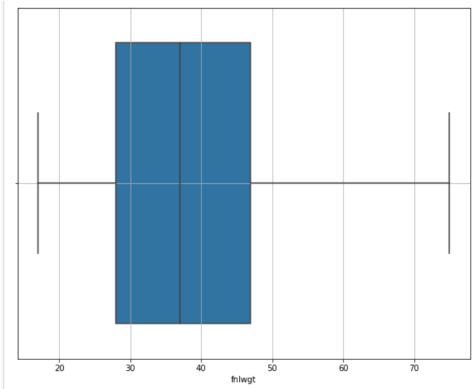


After

2) fnlwgt

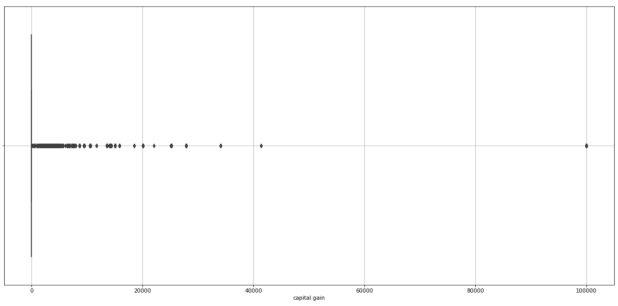


Before

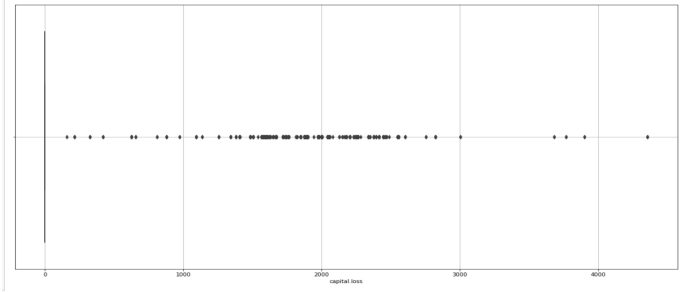


After

3) Capital Gain and Loss



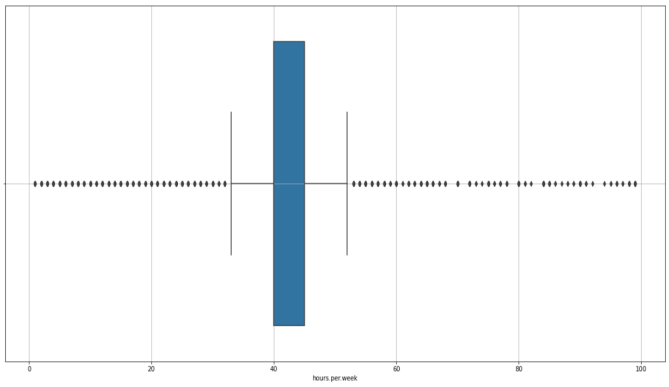
Capital Gain



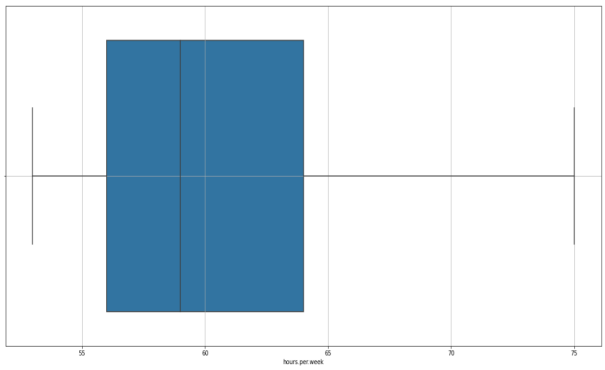
Capital Loss

We realize capital.gain and capital.loss will disturb our learning process as they don't give any useful information either.

4) Hours Per Week:



Before



After

Analyzing the data and moving towards final training

After removing all the outliers, we found some null values that need to be accounted for.

In [52]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
adult_income_num = adult_income_prep[['age', 'fnlwgt', 'hours.per.week']]
adult_income_num.head()
```

Out[52]:

	age	fnlwgt	hours.per.week
0	NaN	NaN	NaN
1	NaN	NaN	NaN
2	66.0	66.0	66.0
3	54.0	54.0	54.0
4	41.0	41.0	NaN

We separated the the features which contain null values, the method we are choosing to fill the null values in case of numeric data is to use the median rather than removing the patterns as it helps us to preserve the total columns for better training of the model.

In [53]:
```
imputer.fit(adult_income_num)
X = imputer.transform(adult_income_num)
adult_tr = pd.DataFrame(X, columns=adult_income_num.columns)
adult_income_prep['age'] = adult_tr['age']
adult_income_prep['fnlwgt'] = adult_tr['fnlwgt']
adult_income_prep['hours.per.week'] = adult_tr['hours.per.week']

adult_income_prep.head()
```

Out[53]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.co
0	37.0	Private	37.0	HS-grad	9	Widowed	Prof-specialty	Not-in-family	White	Female	0	4356	59.0	United-S
1	37.0	Private	37.0	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	59.0	United-S
2	66.0	Private	66.0	Some-college	10	Widowed	Prof-specialty	Unmarried	Black	Female	0	4356	66.0	United-S
3	54.0	Private	54.0	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	54.0	United-S
4	41.0	Private	41.0	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	59.0	United-S

Some additional pre processing:

Changing income values by 0 and 1

```
In [55]: adult_income_prep['income'] = adult_income_prep['income'].replace('<=50K', 0)
         adult_income_prep['income'] = adult_income_prep['income'].replace('>50K', 1)
```

Since education and education.num is same, we will go ahead and remove education

```
In [56]: adult_income_prep = adult_income_prep.drop(columns='education')
```

Categorical Encoding

During our learning process, we can not use non-numerical values, so it's better to encode our non-numerical features.

```
In [57]: adult_income_prep.workclass = adult_income_prep.workclass.astype('category').cat.codes
         adult_income_prep['marital.status'] = adult_income_prep['marital.status'].astype('category').cat.codes
         adult_income_prep['occupation'] = adult_income_prep['occupation'].astype('category').cat.codes
         adult_income_prep['relationship'] = adult_income_prep['relationship'].astype('category').cat.codes
         adult_income_prep['race'] = adult_income_prep['race'].astype('category').cat.codes
         adult_income_prep['sex'] = adult_income_prep['sex'].astype('category').cat.codes
         adult_income_prep['native.country'] = adult_income_prep['native.country'].astype('category').cat.codes
```

```
In [58]: adult_income_prep.head()
```

Out[58]:

	age	workclass	fnlwgt	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	37.0	3	37.0	9	6	9	1	4	0	0	4356	59.0	38	0
1	37.0	3	37.0	9	6	3	1	4	0	0	4356	59.0	38	0
2	66.0	3	66.0	10	6	9	4	2	0	0	4356	66.0	38	0
3	54.0	3	54.0	4	0	6	4	4	0	0	3900	54.0	38	0
4	41.0	3	41.0	10	5	9	3	4	0	0	3900	59.0	38	0

In [59]:
```
adult_income_prep.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             32561 non-null  float64
 1   workclass       32561 non-null  int8   
 2   fnlwgt          32561 non-null  float64
 3   education.num   32561 non-null  int64  
 4   marital.status  32561 non-null  int8   
 5   occupation      32561 non-null  int8   
 6   relationship    32561 non-null  int8   
 7   race            32561 non-null  int8   
 8   sex             32561 non-null  int8   
 9   capital.gain    32561 non-null  int64  
 10  capital.loss    32561 non-null  int64  
 11  hours.per.week  32561 non-null  float64
 12  native.country  32561 non-null  int8   
 13  income          32561 non-null  int64  
dtypes: float64(3), int64(4), int8(7)
memory usage: 2.0 MB
```

Observations:- As we can see that we have no categorical features remaining. Now we can move towards training the model.

6) Classification(Training):

Before training the model, we need to separate the label from the data. That is separating X and y matrices.
In the next step we will split the dataset into training and testing data using sklearn.model_selection.train_test_split.

In [60]:
```
y = adult_income_prep['income']
X_prepared = adult_income_prep.drop(columns='income')
```

In [61]:
```
from sklearn.model_selection import train_test_split
train_X, val_X, train_y, val_y = train_test_split(X_prepared, y, random_state = 0)
```

Creating dictionaries containing the Mean Absolute Error and the Accuracy Value of each algorithm/classifier used.

```
In [62]: from sklearn.model_selection import cross_val_score
         MAE = dict()
         Acc = dict()
```

1) Logistic Regression:

The main reason why we are using this algorithm is to extract features who have a better or worse influence on prediction/accuracy.

```
In [63]: from sklearn.linear_model import LogisticRegression
         log_model = LogisticRegression()
         score = cross_val_score(log_model, X_prepared, y, scoring="neg_mean_absolute_error", cv=10)
```

```
In [65]: from sklearn.model_selection import GridSearchCV
         param_grid = [
             {'C': [0.001,0.01,0.1,1,10,100]},
         ]
         grid_search = GridSearchCV(log_model, param_grid, cv=5, scoring='neg_mean_squared_error')
         grid_search.fit(train_X, train_y)
```

We import Logistic Regressor from sklearn.model_selection and apply Hyperparameter Tuning GridSearchCV. This helps in finding the best hyperparameters suitable for our dataset.

```
In [66]: grid_search.best_params_ # for finding the best paramateres
Out[66]: {'C': 100}
```

```
In [67]: log_model = LogisticRegression(C=100, random_state=0)
         log_model.fit(train_X, train_y)
```

After fitting the model on our dataset we list out all the features and their coefficients. This is useful as it shows how different features are affecting our dataset and which features have most impact on the dataset.

In [68]:
```
val_predictions = log_model.predict(val_X)
columns = adult_income_prep.drop(columns='income').columns
coefs = log_model.coef_[0]
print("Features - Coefs")
for index in range(len(coefs)):
    print(columns[index], ":", coefs[index])
```

```
Features - Coefs
age : 0.04330804447265914
workclass : -0.010959162209384782
fnlwgt : -0.002044799514673439
education.num : 0.35985708083748985
marital.status : -0.16117246735455082
occupation : -0.029070492698821945
relationship : -0.22961726914238184
race : 0.03943669402528584
sex : 0.08449507328136217
capital.gain : 0.0003214770526594809
capital.loss : 0.0007024727384440753
hours.per.week : -0.11150012948972016
native.country : 0.009176629856042704
```

In [69]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error

lm_mae = mean_absolute_error(val_y, val_predictions)
lm_acc = accuracy_score(val_y, val_predictions)
MAE['lm'] = lm_mae
Acc['lm'] = lm_acc
```

In [70]:
```
print("The mae is", lm_mae)
```

```
The mae is 0.17909347745977153
```

In [71]:
```
print("The accuracy is", lm_acc * 100, "%")
```

```
The accuracy is 82.09065225402284 %
```

As we can see the accuracy of Logistic Regression is 82%.

Printing classification report and confusion matrix to print out **TP,FP,TN,FN** and **accuracy score.**

In [140]:
```
confusion_matrix(val_y,val_predictions)
```

Out[140]:
```
array([[5904,  289],
       [1310,  638]], dtype=int64)
```

In [139]:
```
print(classification_report(val_y,val_predictions))
```

```
              precision    recall  f1-score   support

           0       0.82      0.95      0.88      6193
           1       0.69      0.33      0.44      1948

    accuracy                           0.80      8141
   macro avg       0.75      0.64      0.66      8141
weighted avg       0.79      0.80      0.78      8141
```

2) Random Forest:

In [72]:
```
from sklearn.ensemble import RandomForestClassifier
param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]}
]
# This param grid is made for hyperparameter tuning(in order to find the best parameters for our model)

forest_model = RandomForestClassifier()
grid_search = GridSearchCV(forest_model, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(train_X, train_y)
```

Out[72]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             scoring='neg_mean_squared_error')
```

In [73]:
```
grid_search.best_params_
```

Out[73]:
```
{'max_features': 4, 'n_estimators': 30}
```

In [74]:
```
rf_model = RandomForestClassifier(max_features=2, n_estimators=30, random_state=0)
rf_model.fit(train_X, train_y)
```

Out[74]:
```
RandomForestClassifier(max_features=2, n_estimators=30, random_state=0)
```

In [75]:
```
val_predictions = rf_model.predict(val_X)
rf_mae = mean_absolute_error(val_y, val_predictions)
rf_mae
```

Out[75]:
```
0.15268394546124556
```

In [76]:
```
rf_acc = accuracy_score(val_y, val_predictions)
rf_acc
```

Out[76]:
```
0.8473160545387545
```

In [77]:
```
MAE['rf'] = rf_mae
Acc['rf'] = rf_acc
```

Observations:- As we can see that we are getting better accuracy than traditional Logistic Regression. Now we will print the confusion matrix and classification.

In [149]:
```
print('CONFUSION MATRIX \n\n',confusion_matrix(val_y,val_predictions))
print('\n\n')
print('CLASSIFICATION REPORT\n\n',classification_report(val_y,val_predictions))
```

```
CONFUSION MATRIX

 [[5904  289]
 [1310  638]]



CLASSIFICATION REPORT

              precision    recall  f1-score   support

           0       0.82      0.95      0.88      6193
           1       0.69      0.33      0.44      1948

    accuracy                           0.80      8141
   macro avg       0.75      0.64      0.66      8141
weighted avg       0.79      0.80      0.78      8141
```

3) K Nearest Neighbour:

In [78]:
```
from sklearn.neighbors import KNeighborsClassifier as KNN
```

In [79]:
```
knn_model = KNN()

param_grid = {'n_neighbors':range(5,10,1)}

grid_search = GridSearchCV(knn_model, param_grid, cv=5, scoring='neg_mean_squared_error')

grid_search.fit(train_X, train_y)
```

Out[79]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': range(5, 10)},
             scoring='neg_mean_squared_error')
```

In [80]:
```
knn_params = grid_search.best_params_
knn_params
```

Out[80]:
```
{'n_neighbors': 8}
```

In [81]:
```
knn_model = KNN(n_neighbors=8)
```

In [82]:
```
knn_model.fit(train_X, train_y)
```

Out[82]:
```
KNeighborsClassifier(n_neighbors=8)
```

In [83]:
```
val_predictions = knn_model.predict(val_X)
knn_mae = mean_absolute_error(val_y, val_predictions)
knn_mae
```

Out[83]:
```
0.1486303893870532
```

In [84]:
```
knn_acc = accuracy_score(val_y, val_predictions)
knn_acc
```

Out[84]:
```
0.8513696106129468
```

Now we will print confusion matrix and classification report

In [165]:
```
print('CONFUSION MATRIX \n\n',confusion_matrix(val_y,val_predictions))
print('\n\n')
print('CLASSIFICATION REPORT\n\n',classification_report(val_y,val_predictions))
```

```
CONFUSION MATRIX

 [[5875  318]
 [ 892 1056]]



CLASSIFICATION REPORT

              precision    recall  f1-score   support

           0       0.87      0.95      0.91      6193
           1       0.77      0.54      0.64      1948

    accuracy                           0.85      8141
   macro avg       0.82      0.75      0.77      8141
weighted avg       0.84      0.85      0.84      8141
```

4) Naive Bayes:

3.) Naive Bayes

In [86]:
```
from sklearn.naive_bayes import GaussianNB
GNB = GaussianNB()
GNB.fit(train_X, train_y)
```

Out[86]:
```
GaussianNB()
```

In [87]:
```
val_predictions = GNB.predict(val_X)
```

In [88]:
```
GNB_mae = mean_absolute_error(val_y, val_predictions)
```

In [89]:
```
GNB_mae
```

Out[89]:
```
0.19641321704950251
```

In [90]:
```
GNB_acc = accuracy_score(val_y, val_predictions)
GNB_acc
```

Out[90]:
```
0.8035867829504975
```

In [91]:
```
MAE['gnb'] = GNB_mae
Acc['gnb'] = GNB_acc
```

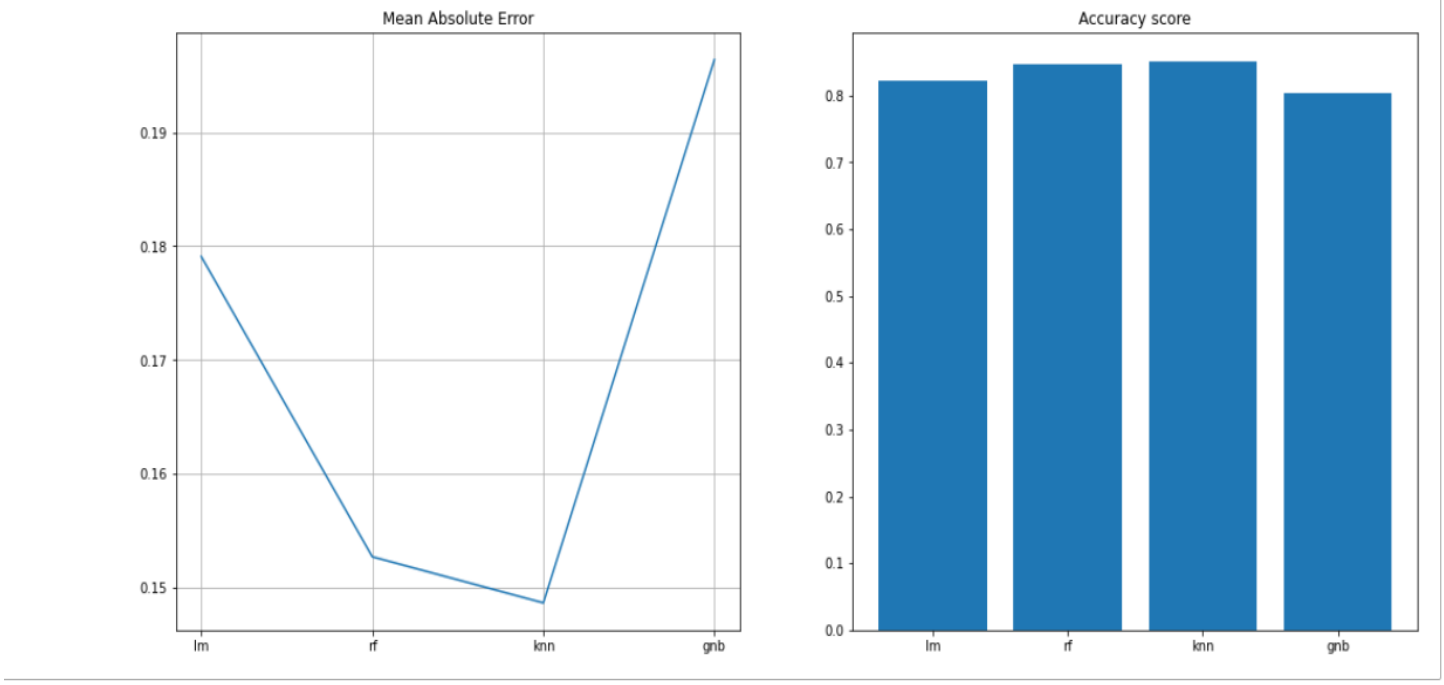
In [172]:
```
print('CONFUSION MATRIX \n\n',confusion_matrix(val_y,val_predictions))
print('\n\n')
print('CLASSIFICATION REPORT\n\n',classification_report(val_y,val_predictions))
```

```
CONFUSION MATRIX

 [[5904  289]
 [1310  638]]



CLASSIFICATION REPORT

              precision    recall  f1-score   support

           0       0.82      0.95      0.88      6193
           1       0.69      0.33      0.44      1948

    accuracy                           0.80      8141
   macro avg       0.75      0.64      0.66      8141
weighted avg       0.79      0.80      0.78      8141
```

Now let's summarise all the models that we have used in order to find which model will give us better classification results. Or in other sense maximum accuracy and minimum mean absolute error.

In [92]:
```
f,ax=plt.subplots(1,2,figsize=(18,8))
ax[0].plot(list(MAE.keys()), list(MAE.values()))
ax[0].set_title("Mean Absolute Error")
ax[0].grid()
ax[1].bar(list(Acc.keys()), list(Acc.values()))
ax[1].set_title("Accuracy score")
```

Out[92]:
```
Text(0.5, 1.0, 'Accuracy score')
```



7) Conclusion:

After successfully training the model on different classifiers we can see that Random Forest gives the best accuracy compared to Naive Bayes, KNN and Logistic Regression. The main aim behind using Logistic Regression is to find the weights of each feature and see how closely related they are to each other. Even after high accuracy, we choose over any other model because we know that Naive Bayes have the least misclassification where accuracy is not the correct measure to predict which algorithm to be used. Here we must consider the precision and recall. Although in our case Naive Bayes and KNN give almost the same precision and recall.

In Naive Bayes, we have more True Positives i.e. the number of people who would most likely to earn more than $50K.

Some other inferences that we can draw are:-

Accuracy might not be the only thing that we need to consider while predicting on new unseen data.

It is important to always visualize the data in every way possible. This helps in figuring out relationships between various or all features.

We also need to take care of categorical data(string) as we need to convert them to numeric values before model training.

Also take care of the missing values by either removing the pattern or filling them with mode or median depending on the feature.