HR Salary Dashboard - Train the Dataset and Predict Salary

Student's Name: Ketaki Desale.

Internship Project Title: TCS iON RIO-125: HR Salary Dashboard - Train the Dataset and Predict Salary.

Organization: TCS iON.

Industry Mentor: Debashis Roy.

Institute: B. K. Birla College of Arts, Science & Commerce (Autonomous), Kalyan.

Importing Necessary Libraries For Data Wrangling

In [1]:
```
import pandas as pd
import numpy as np
```

Importing Necessary Libraries For Data Visualization

In [2]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Remove unnecessary warnings

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

Importing and Reading Dataset

In [4]:
```
data=pd.read_csv(r"C:\Users\91779\Downloads\salarydata.csv")
data.head()
```

Out[4]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	sex
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female

Print The Last 5 Lines Of The Dataset

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

Out[5]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s
32556	27	Private	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Fem
32557	40	Private	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	M
32558	58	Private	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Fem
32559	22	Private	HS-grad	9	Never-married	Adm-clerical	Own-child	White	M
32560	52	Self-emp-inc	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Fem

Display The Full Summary Of The Dataframe

In [6]:
```
data.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  int64 
 1   workclass       32561 non-null  object
 2   education       32561 non-null  object
 3   education-num   32561 non-null  int64 
 4   marital-status  32561 non-null  object
 5   occupation      32561 non-null  object
 6   relationship    32561 non-null  object
 7   race            32561 non-null  object
 8   sex             32561 non-null  object
 9   capital-gain    32561 non-null  int64 
 10  capital-loss    32561 non-null  int64 
 11  hours-per-week  32561 non-null  int64 
 12  native-country  32561 non-null  object
 13  salary          32561 non-null  object
dtypes: int64(5), object(9)
memory usage: 3.5+ MB
```

Shape Of The Dataset

In [7]:
```
data.shape
```

Out[7]:
```
(32561, 14)
```

Finding Null Values In Dataset

In [8]:
```
data.isna().sum()
```

Out[8]:
```
age               0
workclass         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
salary            0
dtype: int64
```

Statistics Summary Of The Dataframe

In [9]:
```
data.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
age	32561.0	38.581647	13.640433	17.0	28.0	37.0	48.0	90.0
education-num	32561.0	10.080679	2.572720	1.0	9.0	10.0	12.0	16.0
capital-gain	32561.0	1077.648844	7385.292085	0.0	0.0	0.0	0.0	99999.0
capital-loss	32561.0	87.303830	402.960219	0.0	0.0	0.0	0.0	4356.0
hours-per-week	32561.0	40.437456	12.347429	1.0	40.0	40.0	45.0	99.0

Statistics Summary Of The Dataframe Of Object Datatype

In [10]:
```
data.describe(include='object').T
```

Out[10]:

	count	unique	top	freq
workclass	32561	9	Private	22696
education	32561	16	HS-grad	10501
marital-status	32561	7	Married-civ-spouse	14976
occupation	32561	15	Prof-specialty	4140
relationship	32561	6	Husband	13193
race	32561	5	White	27816
sex	32561	2	Male	21790
native-country	32561	42	United-States	29170
salary	32561	2	<=50K	24720

Check Datatypes

In [11]:
```
data.dtypes
```

Out[11]:
```
age               int64
workclass        object
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
salary           object
dtype: object
```

Display Columns Of The Dataset

In [12]:
```
data.columns
```

Out[12]:
```
Index(['age', 'workclass', 'education', 'education-num', 'marital-status',
       'occupation', 'relationship', 'race', 'sex', 'capital-gain',
       'capital-loss', 'hours-per-week', 'native-country', 'salary'],
      dtype='object')
```

In [13]:
```
data.drop(['capital-gain','capital-loss','education-num'], axis = 1,inplace
data.head()
```

Out[13]:

	age	workclass	education	marital-status	occupation	relationship	race	sex	hours-per-week	nat cour
0	39	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	40	Uni Sta
1	50	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	13	Uni Sta
2	38	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	40	Uni Sta
3	53	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	40	Uni Sta
4	28	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	40	C

1)Upon scrutinizing the dataset, it becomes evident that variables like capital-gain and capital-loss hold negligible influence on salary prediction.

2)Hence, it is prudent to discard these columns. Additionally, we opt to remove the education-num column as it essentially duplicates the information present in the education column in numerical form.

Unique Occurance Of Each Variable

In [14]:
```
for i in data.columns:
    print(i,':',data[i].nunique(),'\n')
```

```
age : 73

workclass : 9

education : 16

marital-status : 7

occupation : 15

relationship : 6

race : 5

sex : 2

hours-per-week : 94

native-country : 42

salary : 2
```

Exploratory Data Analysis EDA

In [15]:
```
cat_var = []

for column in data:
    if data[column].dtype == 'O':
        cat_var.append(column)
cat_var
```

Out[15]:
```
['workclass',
 'education',
 'marital-status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native-country',
 'salary']
```

Finding the unique values in each categorical variables

In [16]:
```
for col in cat_var:
    print('\n', col, '\n', data[col].unique(), '\n', '---' * 40)
```

```
 workclass
 ['State-gov' 'Self-emp-not-inc' 'Private' 'Federal-gov' 'Local-gov' '?'
 'Self-emp-inc' 'Without-pay' 'Never-worked']
 ------------------------------------------------------------------------
-----------------------------------------------

 education
 ['Bachelors' 'HS-grad' '11th' 'Masters' '9th' 'Some-college' 'Assoc-acdm'
 'Assoc-voc' '7th-8th' 'Doctorate' 'Prof-school' '5th-6th' '10th'
 '1st-4th' 'Preschool' '12th']
 ------------------------------------------------------------------------
-----------------------------------------------

 marital-status
 ['Never-married' 'Married-civ-spouse' 'Divorced' 'Married-spouse-absent'
 'Separated' 'Married-AF-spouse' 'Widowed']
 ------------------------------------------------------------------------
-----------------------------------------------

 occupation
 ['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']
 ------------------------------------------------------------------------
-----------------------------------------------

 relationship
 ['Not-in-family' 'Husband' 'Wife' 'Own-child' 'Unmarried' 'Other-relativ
e']
 ------------------------------------------------------------------------
-----------------------------------------------

 race
 ['White' 'Black' 'Asian-Pac-Islander' 'Amer-Indian-Eskimo' 'Other']
 ------------------------------------------------------------------------
-----------------------------------------------

 sex
 ['Male' 'Female']
 ------------------------------------------------------------------------
-----------------------------------------------

 native-country
 ['United-States' 'Cuba' 'Jamaica' 'India' '?' 'Mexico' 'South'
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand' 'Ecuador'
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-Salvador'
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'
 'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinadad&Tobago' 'Greece'
 'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary' 'Holand-Netherlands']
 ------------------------------------------------------------------------
-----------------------------------------------

 salary
 ['<=50K' '>50K']
 ------------------------------------------------------------------------
-----------------------------------------------
```

Replace '?' row with nan to apply the changes to whole dataframe

In [17]:
```
data.replace('?', np.NaN, inplace=True)
```

Finding the unique values in each categorical variables

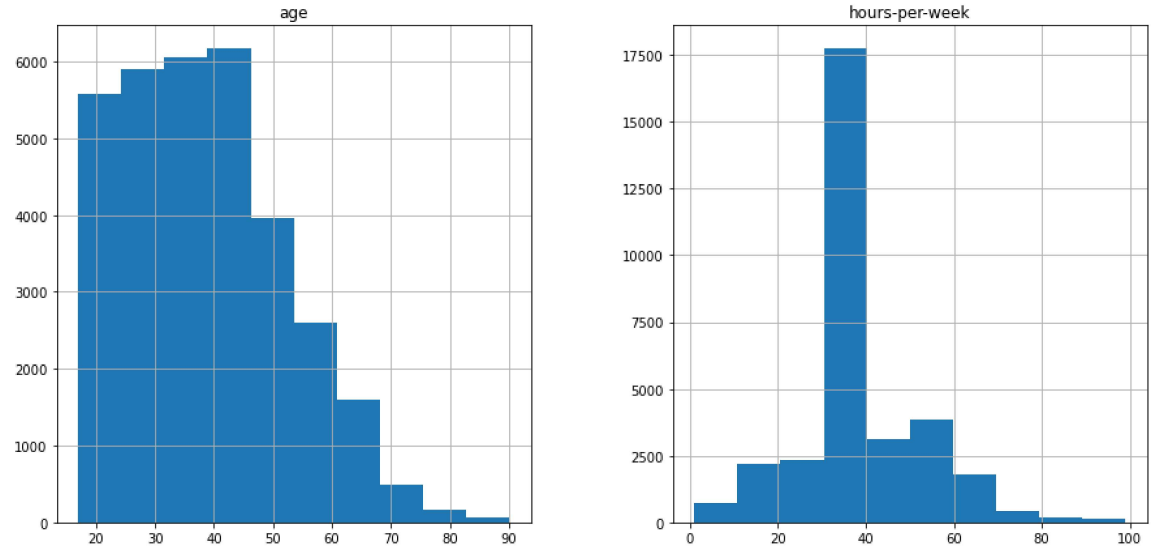
In [18]:
```
for col in cat_var:
    print('\n', col, '\n', data[col].unique(), '\n', '---' * 40)
```

```
 workclass
 ['State-gov' 'Self-emp-not-inc' 'Private' 'Federal-gov' 'Local-gov' nan
 'Self-emp-inc' 'Without-pay' 'Never-worked']
 ------------------------------------------------------------------------
-----------------------------------------------

 education
 ['Bachelors' 'HS-grad' '11th' 'Masters' '9th' 'Some-college' 'Assoc-acdm'
 'Assoc-voc' '7th-8th' 'Doctorate' 'Prof-school' '5th-6th' '10th'
 '1st-4th' 'Preschool' '12th']
 ------------------------------------------------------------------------
-----------------------------------------------

 marital-status
 ['Never-married' 'Married-civ-spouse' 'Divorced' 'Married-spouse-absent'
 'Separated' 'Married-AF-spouse' 'Widowed']
 ------------------------------------------------------------------------
-----------------------------------------------

 occupation
 ['Adm-clerical' 'Exec-managerial' 'Handlers-cleaners' 'Prof-specialty'
 'Other-service' 'Sales' 'Craft-repair' 'Transport-moving'
 'Farming-fishing' 'Machine-op-inspct' 'Tech-support' nan
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv']
 ------------------------------------------------------------------------
-----------------------------------------------

 relationship
 ['Not-in-family' 'Husband' 'Wife' 'Own-child' 'Unmarried' 'Other-relativ
e']
 ------------------------------------------------------------------------
-----------------------------------------------

 race
 ['White' 'Black' 'Asian-Pac-Islander' 'Amer-Indian-Eskimo' 'Other']
 ------------------------------------------------------------------------
-----------------------------------------------

 sex
 ['Male' 'Female']
 ------------------------------------------------------------------------
-----------------------------------------------

 native-country
 ['United-States' 'Cuba' 'Jamaica' 'India' nan 'Mexico' 'South'
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand' 'Ecuador'
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-Salvador'
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'
 'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinadad&Tobago' 'Greece'
 'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary' 'Holand-Netherlands']
 ------------------------------------------------------------------------
-----------------------------------------------

 salary
 ['<=50K' '>50K']
 ------------------------------------------------------------------------
-----------------------------------------------
```

In [19]:
```
data.dtypes
```

Out[19]:
```
age                int64
workclass         object
education         object
marital-status    object
occupation        object
relationship      object
race              object
sex               object
hours-per-week     int64
native-country    object
salary            object
dtype: object
```

In [20]:
```
freqgraph = data.select_dtypes(include = ['int'])
freqgraph.hist(figsize =(15,7))
plt.show()
```



Filling missing values using mode

In [21]:
```
for i in ['workclass', 'occupation', 'native-country']:
    data[i]=data[i].fillna(data[i].mode()[0])
```

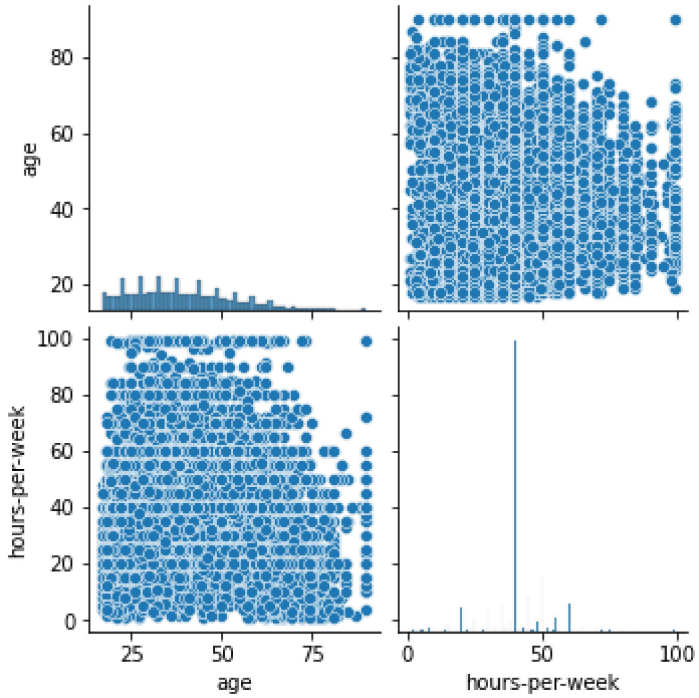
In [22]:
```
num_cols =  data.select_dtypes(["int"])
#get the valuecounts
for i in num_cols.columns:
    print(num_cols[i].value_counts())
    print("-"*50)
    print("\n")
```

```
36    898
31    888
34    886
23    877
35    876
     ...
83      6
88      3
85      3
86      1
87      1
Name: age, Length: 73, dtype: int64
--------------------------------------------------


40    15217
50     2819
45     1824
60     1475
35     1297
      ...
82        1
92        1
87        1
74        1
94        1
Name: hours-per-week, Length: 94, dtype: int64
--------------------------------------------------
```

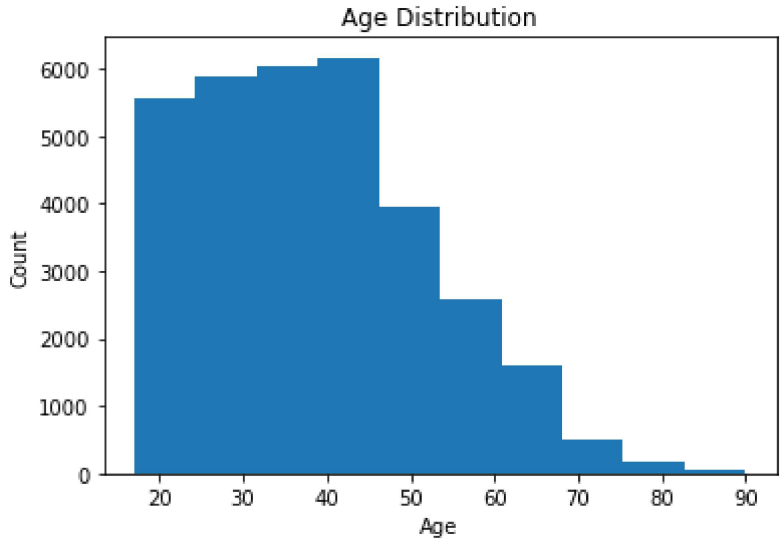
In [23]:
```
cat_cols =  data.select_dtypes(["object"])
#get the valuecounts
for i in cat_cols.columns:
    print(cat_cols[i].value_counts())
    print("-"*50)
    print("\n")
```

```
Private             24532
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
--------------------------------------------------


HS-grad         10501
Some-college     7291
Bachelors        5355
Masters          1723
Assoc-voc        1382
11th             1175
Assoc-acdm       1067
```

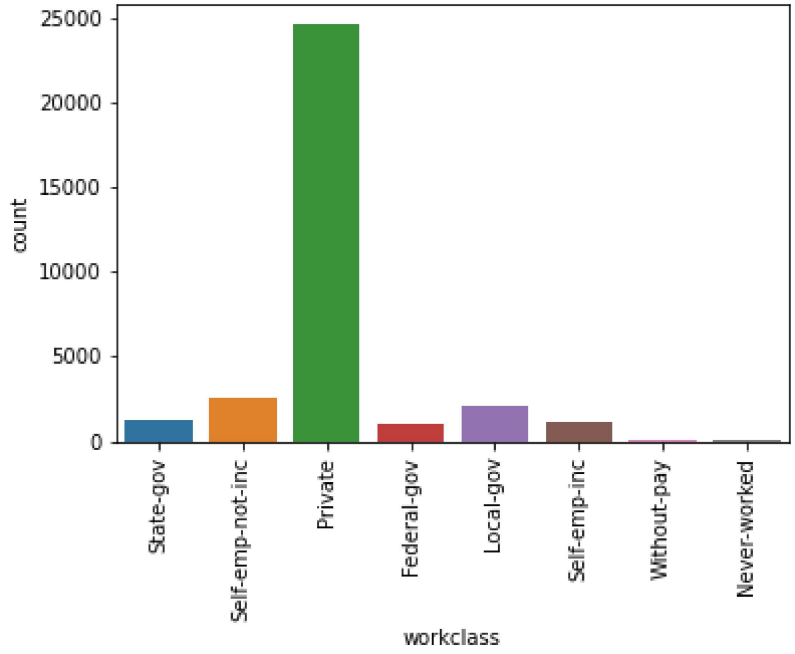
In [24]:
```
sns.pairplot(data)
plt.show()
```



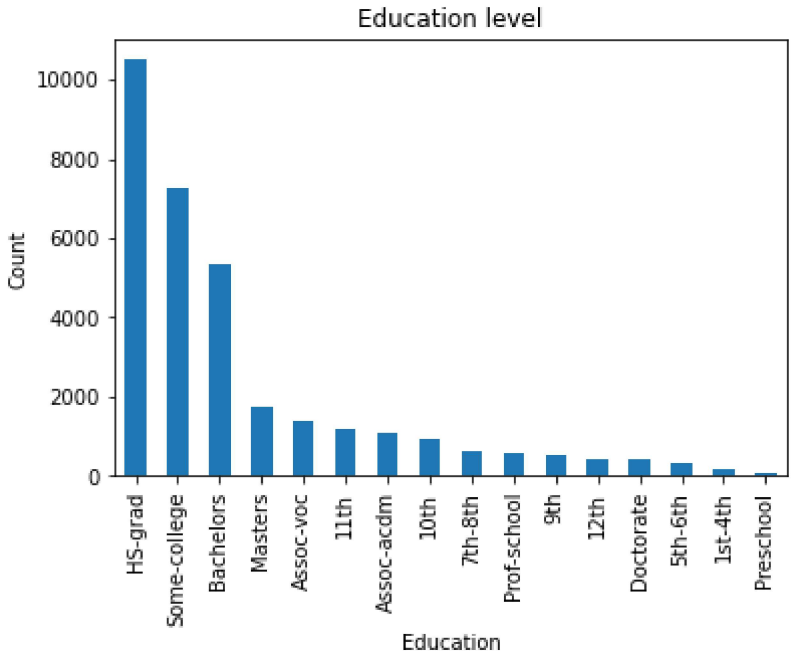
In [25]:
```
#checking age

plt.hist(data['age'])
plt.xlabel('Age')
plt.ylabel('Count')
# plt.xticks(np.arange(20,100,5))
# plt.rcParams['figure.figsize'] = (8,8)
plt.title('Age Distribution')
plt.show()
```



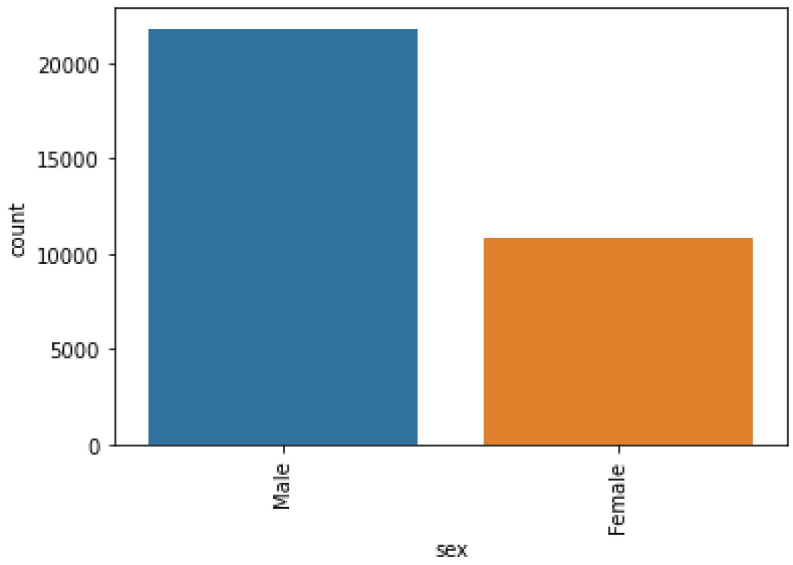
In [26]:
```
#checking workclass

sns.countplot(x = data['workclass'], data = data)
plt.xticks(rotation = 90)
plt.show()
```



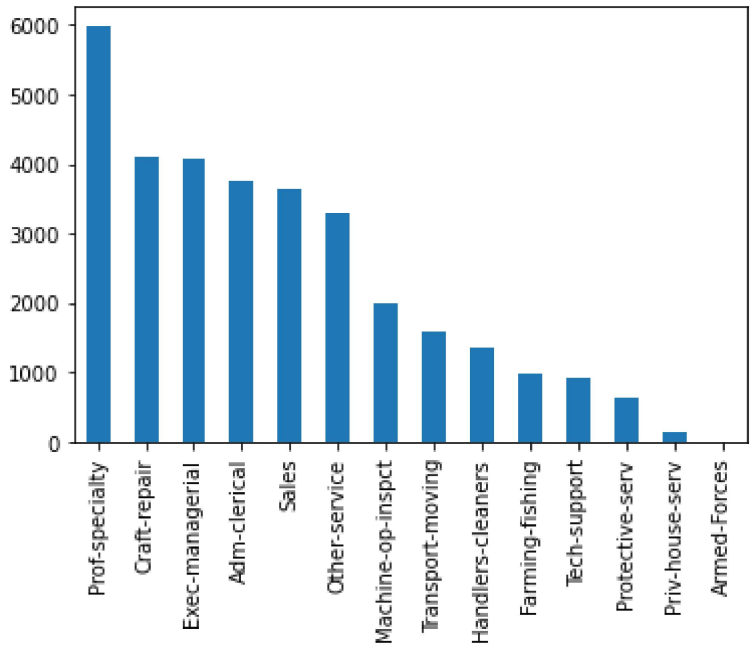
In [27]:
```
#checking education

data['education'].value_counts().plot(kind = 'bar')
plt.xlabel('Education')
plt.ylabel('Count')
plt.title('Education level')
plt.show()
```



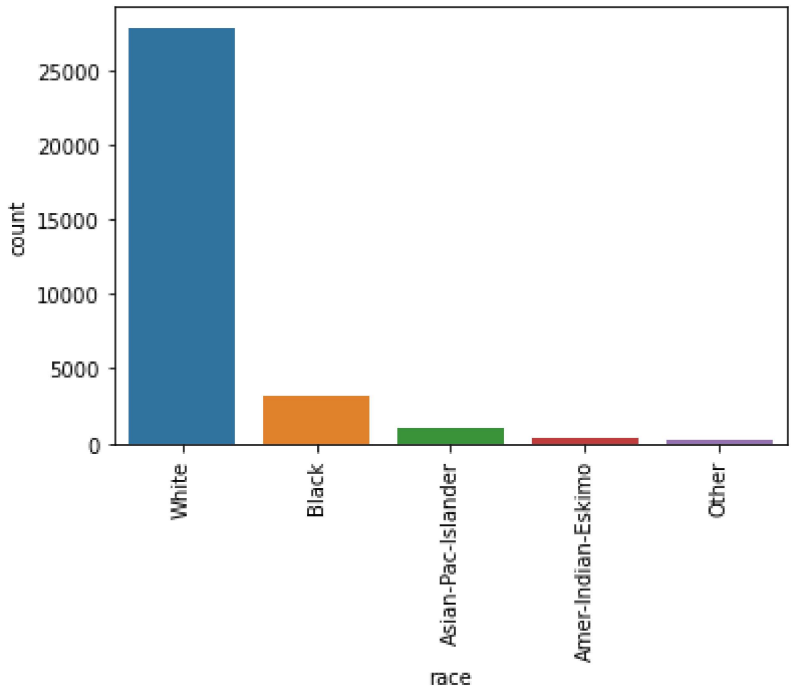
In [28]:
```
#checking sex

sns.countplot(x = data['sex'], data = data)
plt.xticks(rotation = 90)
plt.show()
```



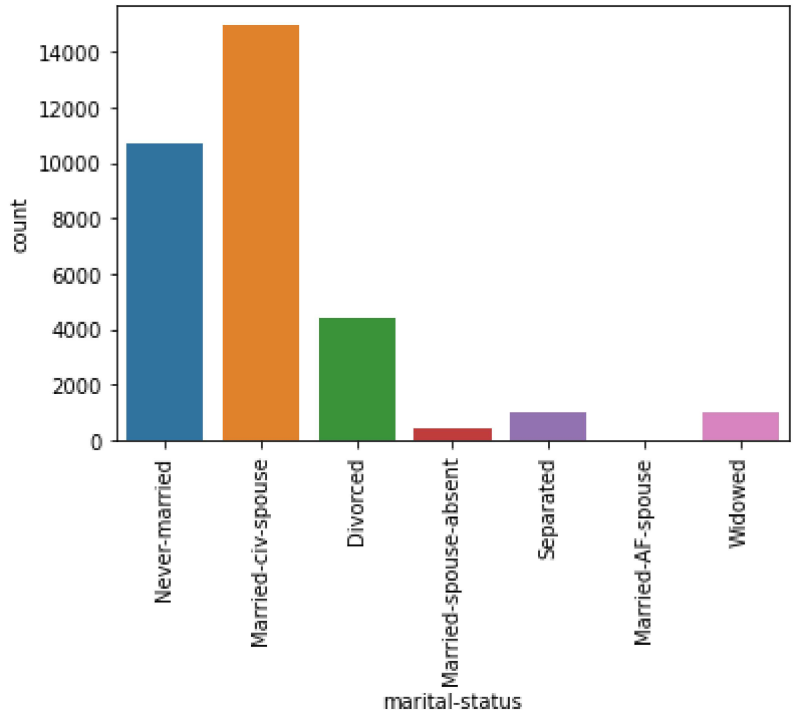
In [29]:
```
#checking occupation

data['occupation'].value_counts().plot(kind = 'bar')
plt.show()
```



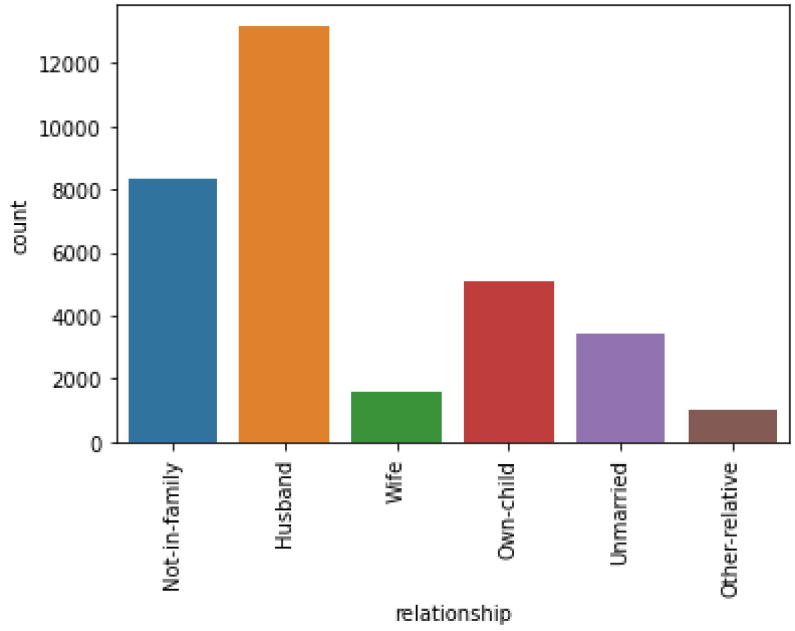
In [30]:
```
#Checking race

sns.countplot(x = data['race'], data = data)
plt.xticks(rotation = 90)
plt.show()
```



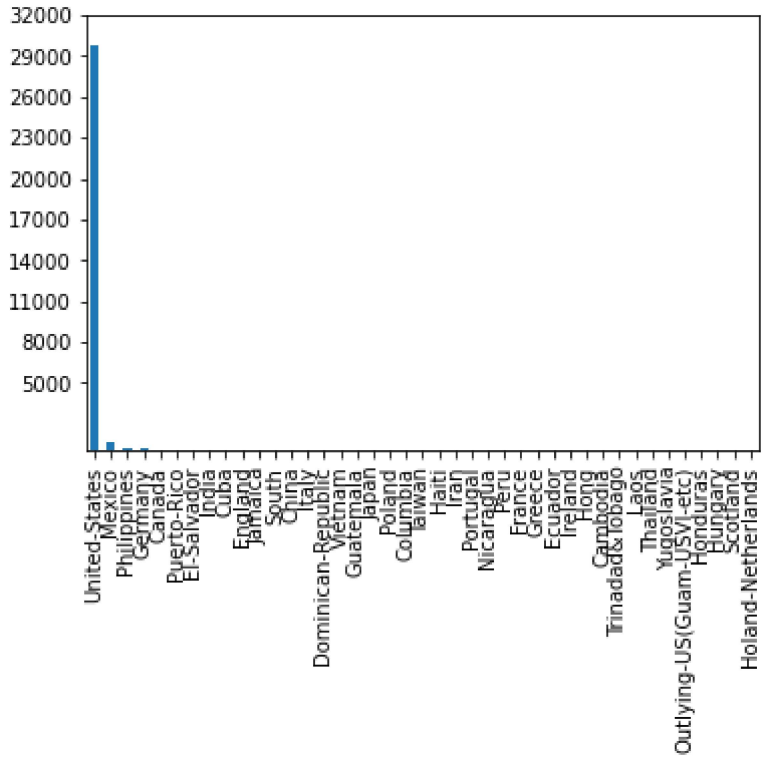
In [31]:
```
#checking marital-status

sns.countplot(x = data['marital-status'], data = data)
plt.xticks(rotation = 90)
plt.show()
```



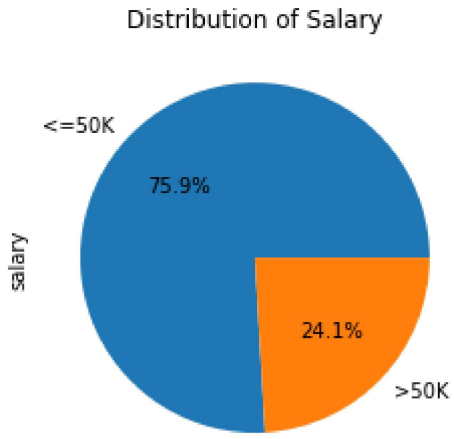
In [32]:
```
#checking relationship

sns.countplot(x = data['relationship'], data = data)
plt.xticks(rotation = 90)
plt.show()
```



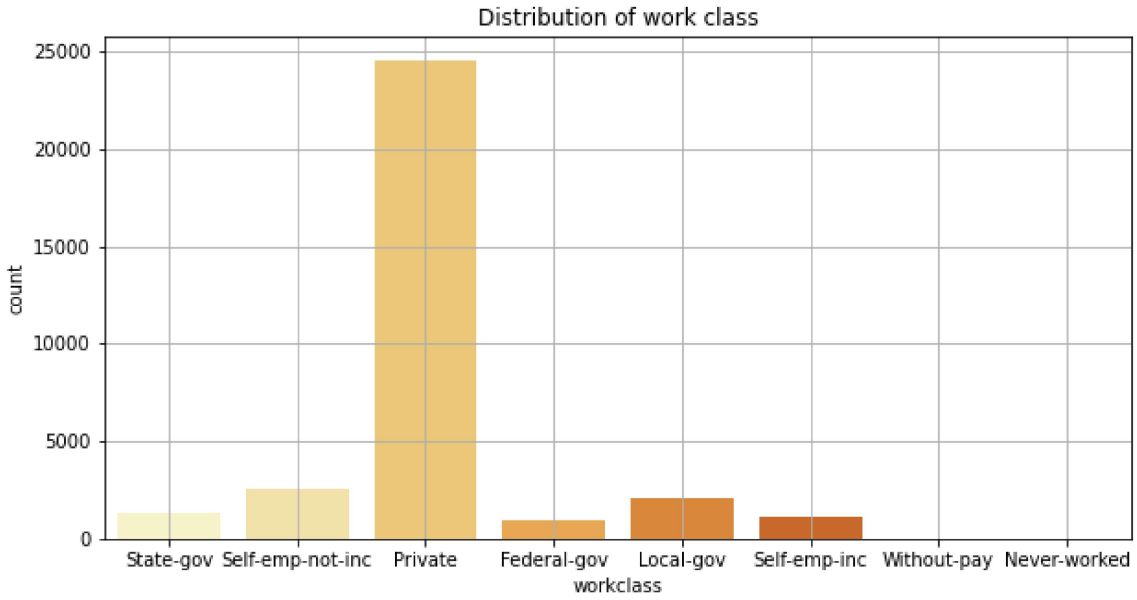
In [33]:
```
#checking native-country

data['native-country'].value_counts().plot(kind = 'bar')
plt.yticks(np.arange(5000,33000,3000))
plt.show()
```



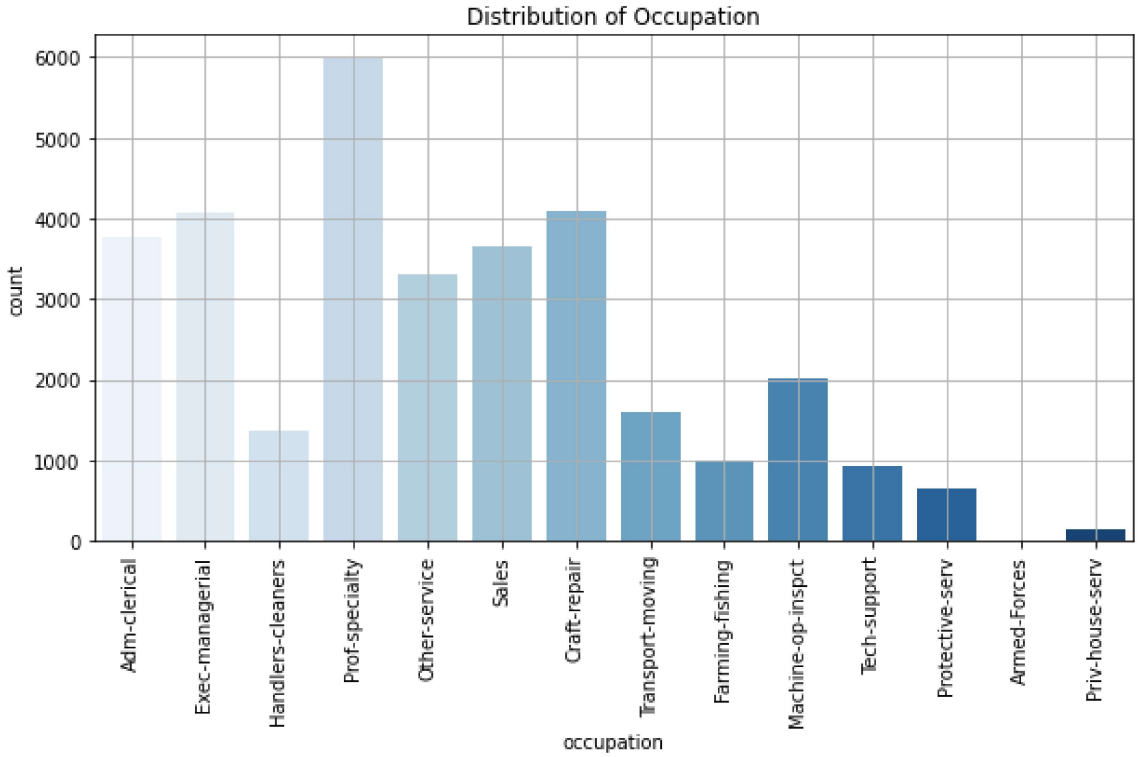
In [34]:
```
data['salary'].value_counts().plot.pie(autopct='%.1f%%')
plt.title("Distribution of Salary")
plt.rcParams['figure.figsize'] = (10,5)
plt.show()
```



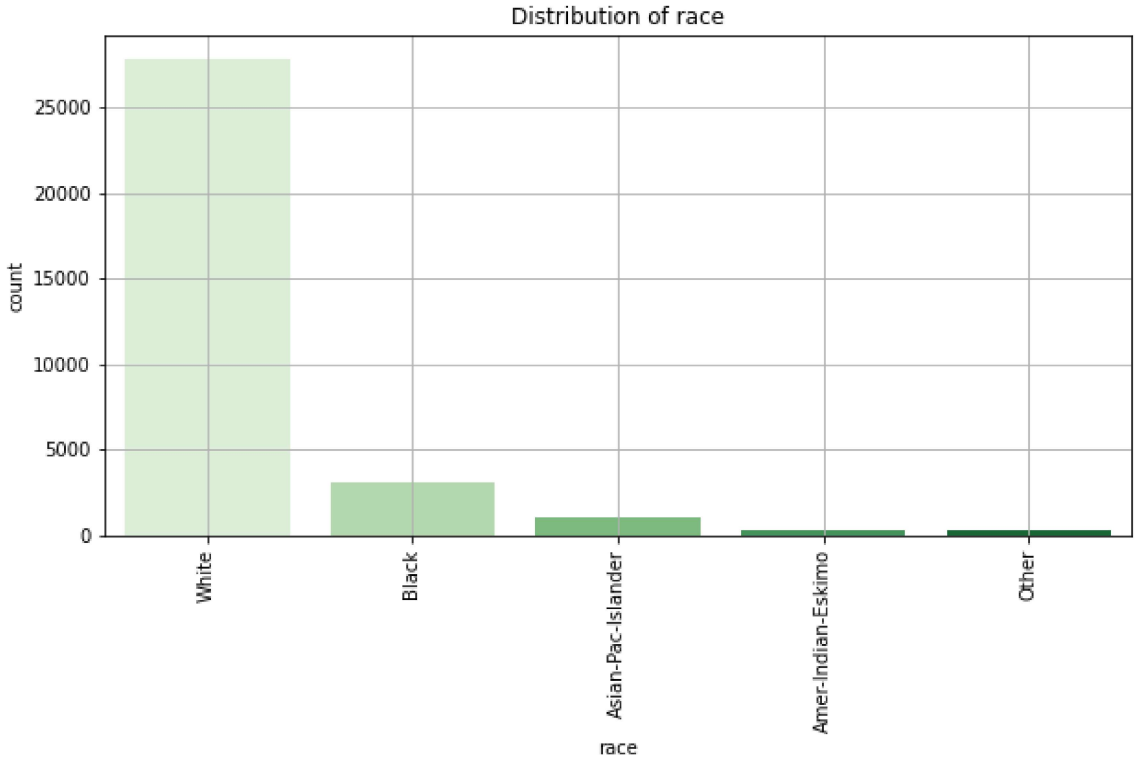
In [35]:
```
sns.countplot(x= data['workclass'],palette="YlOrBr")
plt.title('Distribution of work class')
plt.grid()
plt.show()
```



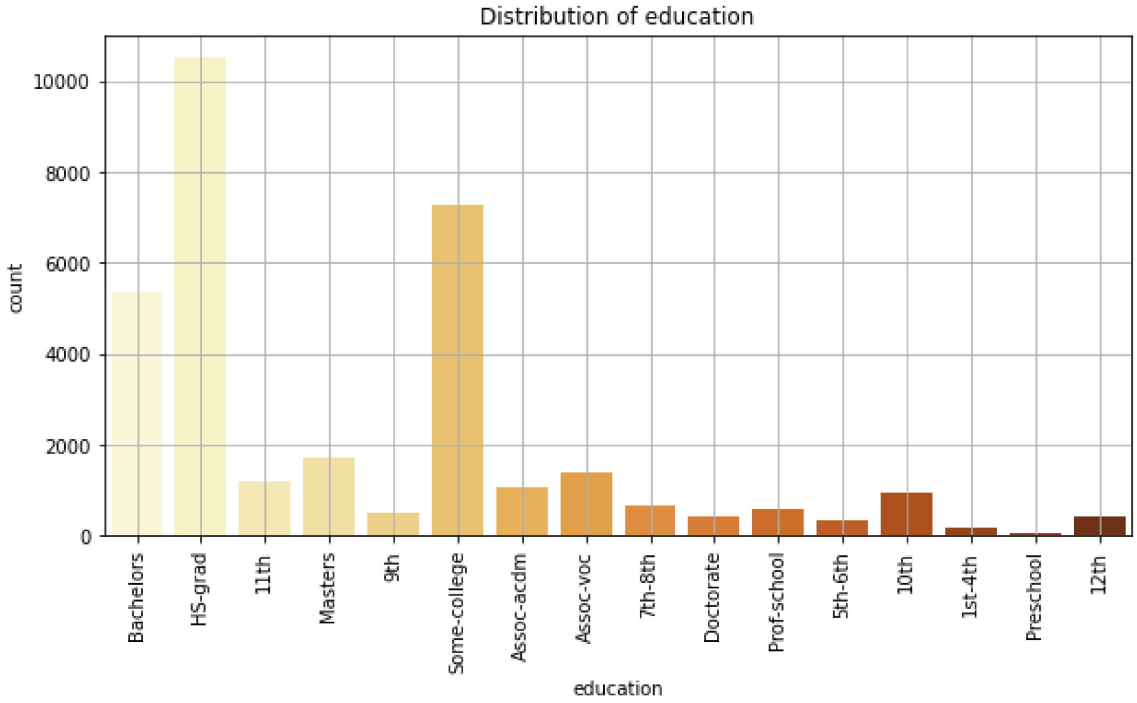
In [36]:
```
sns.countplot(x= data['occupation'],palette="Blues")
plt.title('Distribution of Occupation')
plt.xticks(rotation=90)
plt.grid()
plt.show()
```



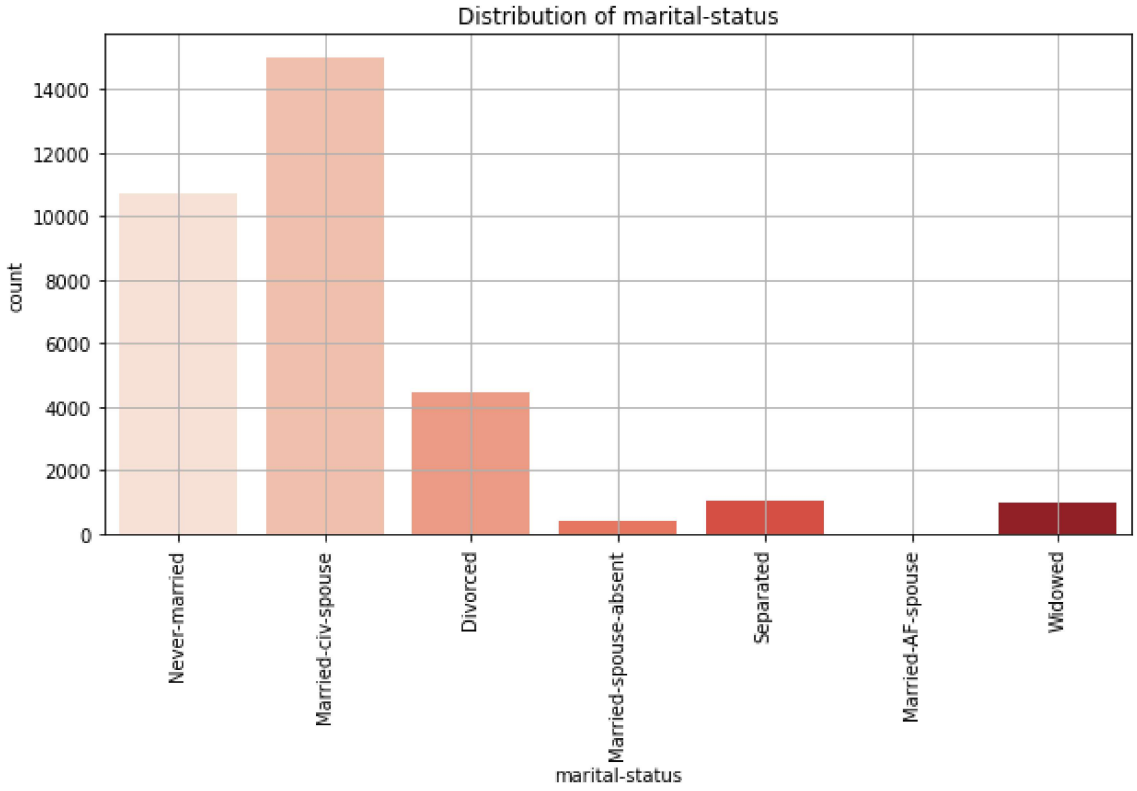
In [37]:
```
sns.countplot(x= data['race'],palette="Greens")
plt.title('Distribution of race')
plt.xticks(rotation=90)
plt.grid()
plt.show()
```



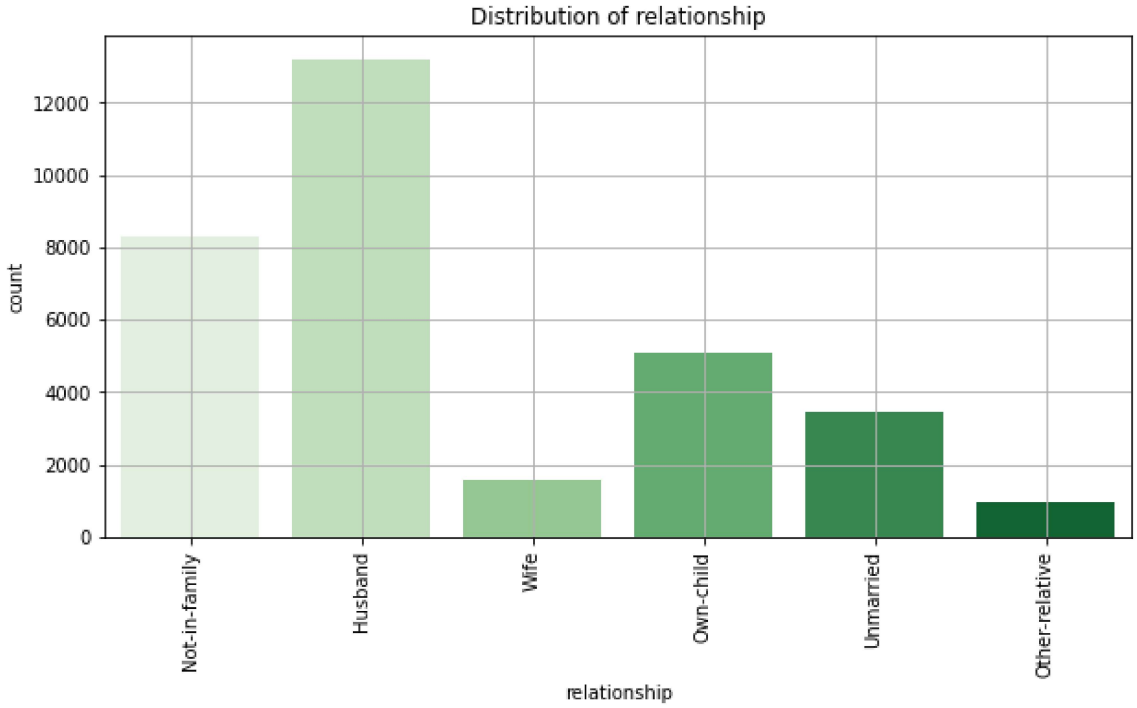
In [38]:
```
sns.countplot(x= data['education'],palette="YlOrBr")
plt.title('Distribution of education')
plt.xticks(rotation = 90)
plt.grid()
plt.show()
```



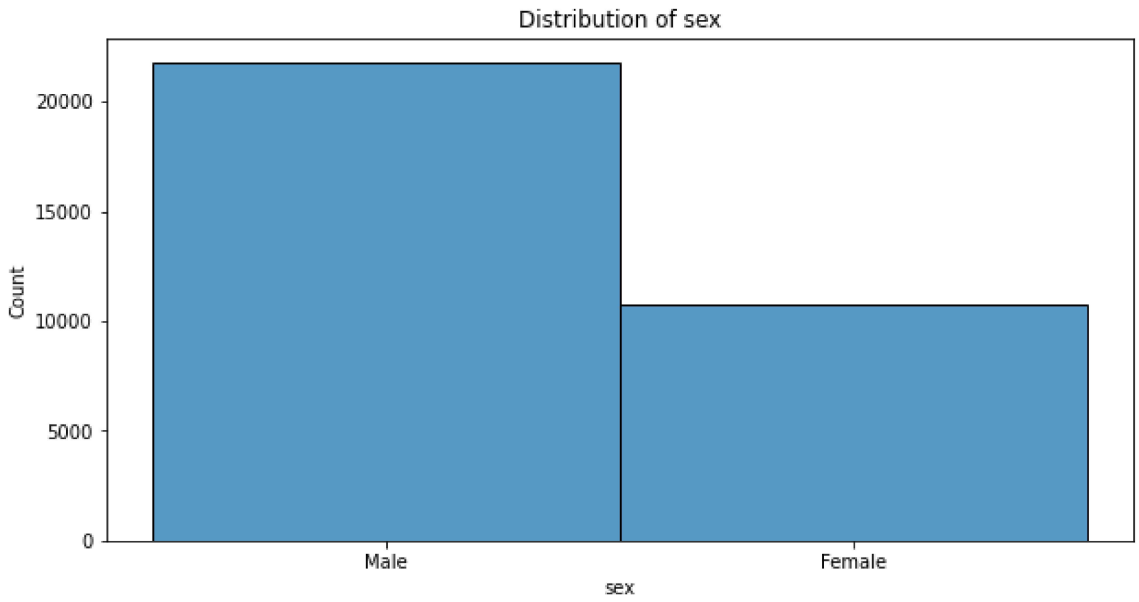
In [39]:
```
sns.countplot(x= data['marital-status'],palette="Reds")
plt.title('Distribution of marital-status')
plt.xticks(rotation = 90)
plt.grid()
plt.show()
```



In [40]:
```
sns.countplot(x= data['relationship'],palette="Greens")
plt.title('Distribution of relationship')
plt.xticks(rotation = 90)
plt.grid()
plt.show()
```

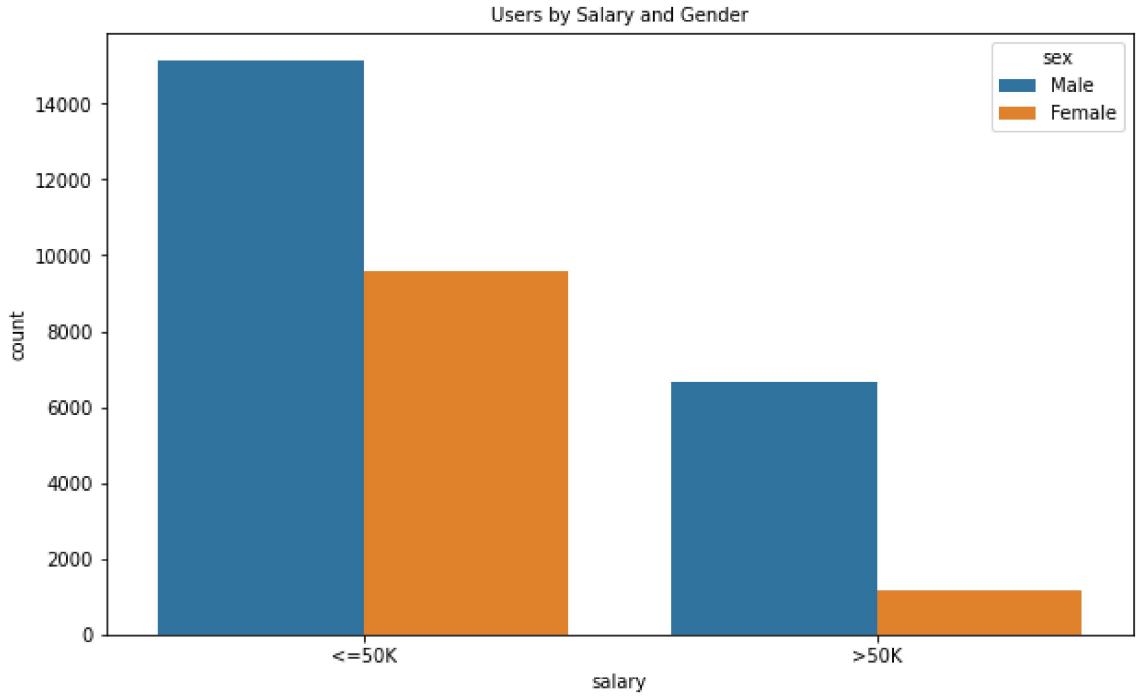


In [41]:
```
sns.histplot(data['sex'])
plt.title('Distribution of sex')
plt.show()
```

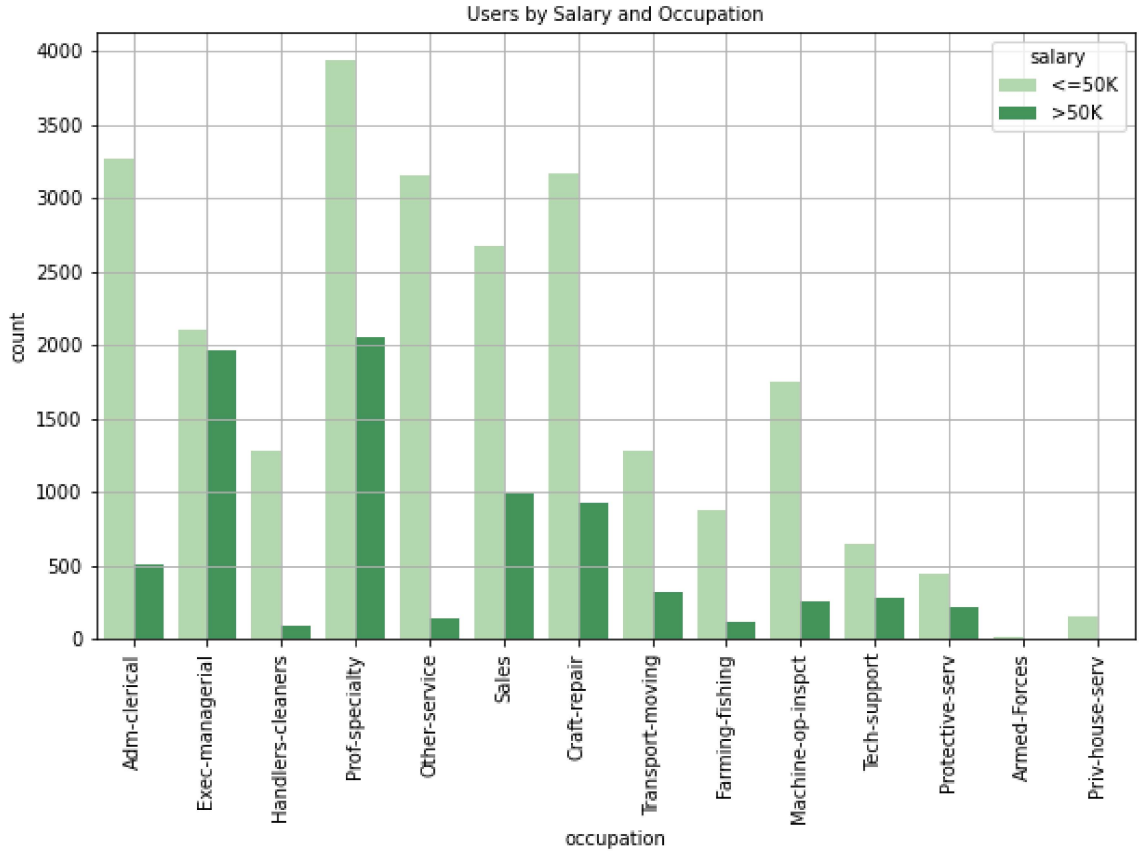


In [42]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='salary',hue='sex',data=data)
plt.title("Users by Salary and Gender", fontsize = 10)
```

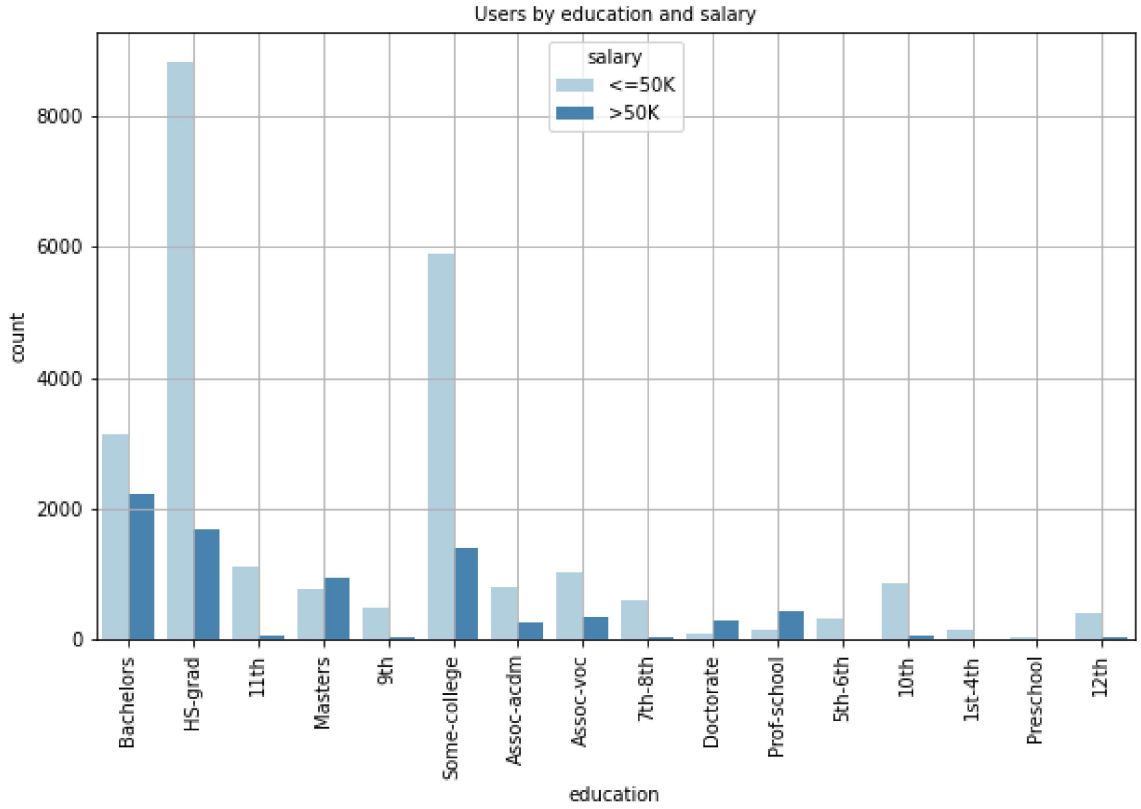
Out[42]:
```
Text(0.5, 1.0, 'Users by Salary and Gender')
```



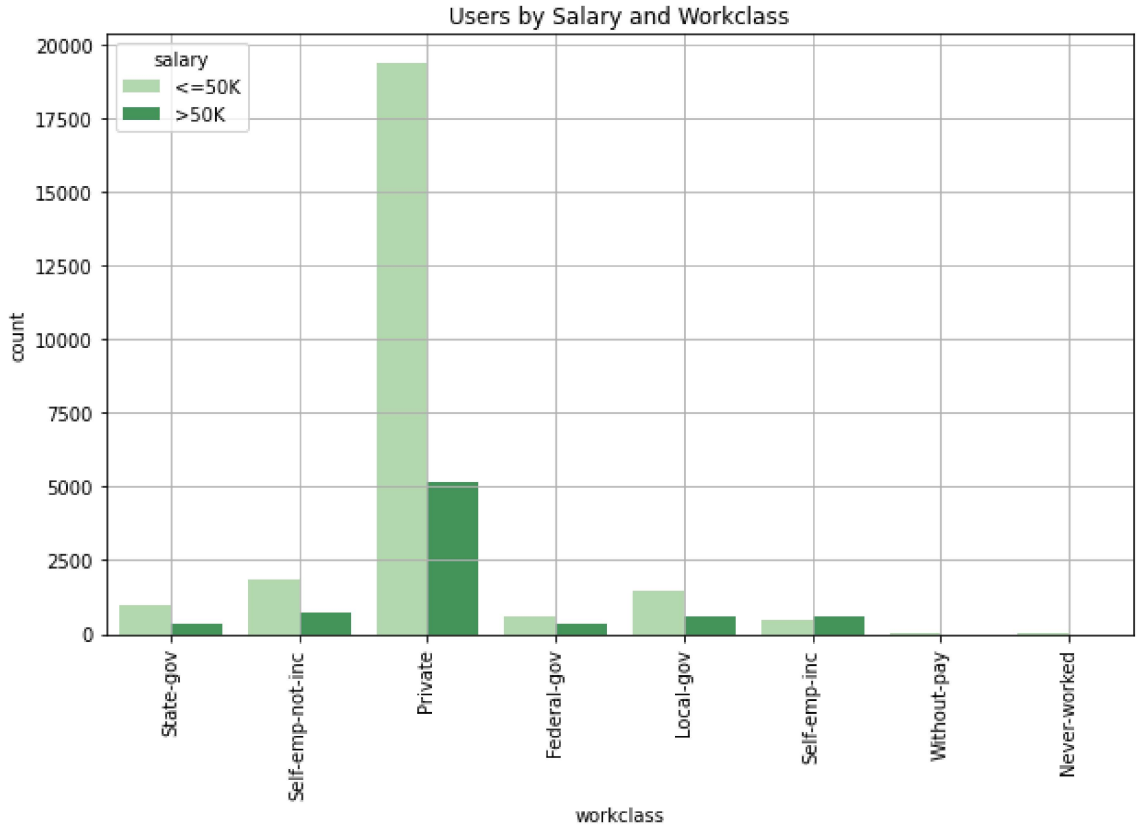
In [43]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='occupation',hue='salary',data=data,palette="Greens")
plt.title("Users by Salary and Occupation", fontsize = 10)
plt.xticks(rotation = 90)
plt.grid()
```



In [44]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='education',hue='salary',data=data,palette="Blues")
plt.title("Users by education and salary", fontsize = 10)
plt.xticks(rotation = 90)
plt.grid()
```

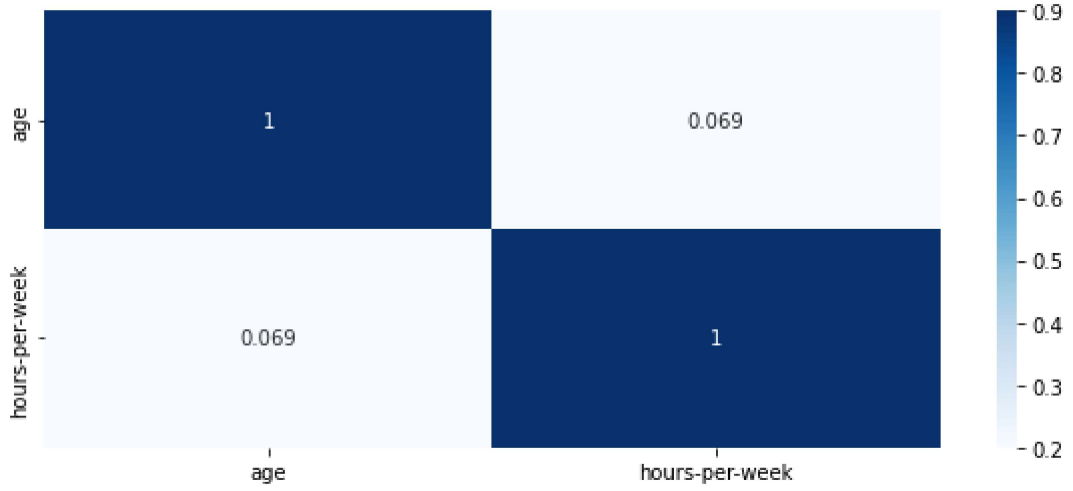


In [45]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='workclass',hue='salary',data=data,palette="Greens")
plt.title("Users by Salary and Workclass", fontsize = 12)
plt.xticks(rotation = 90)
plt.grid()
```

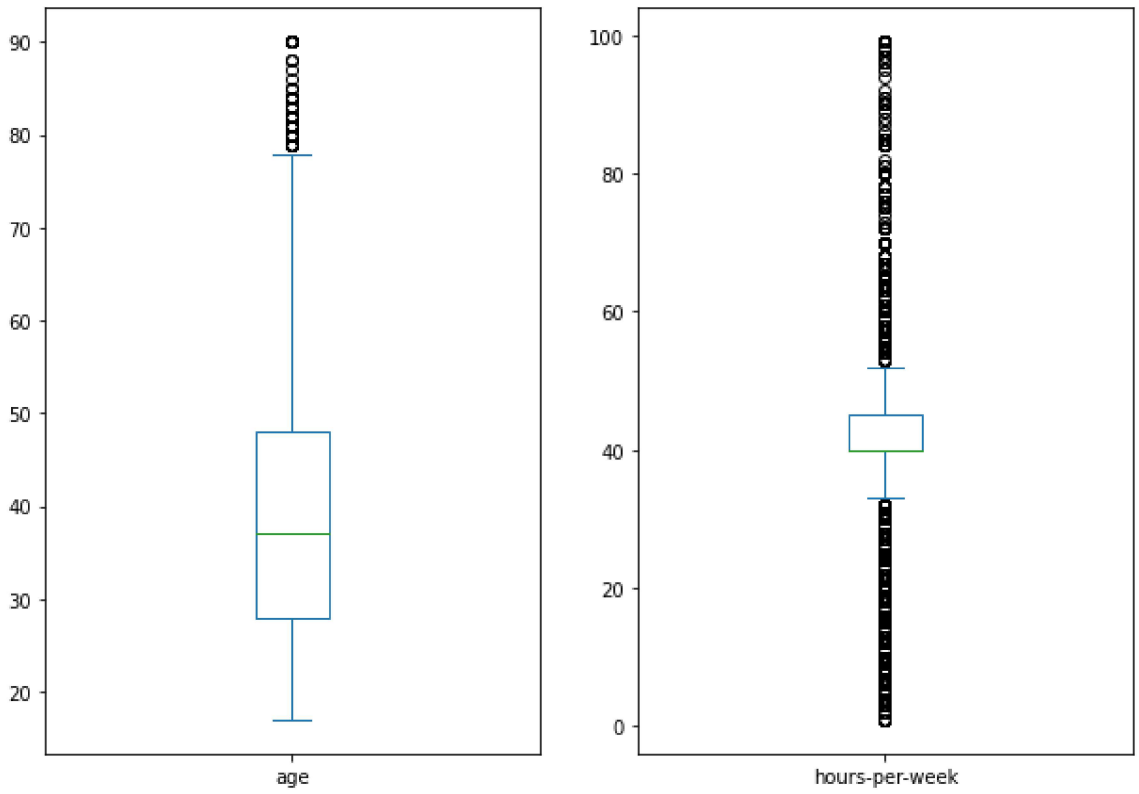


In [46]:
```
corrmatrix = data.corr()
plt.subplots(figsize=(10,4))
sns.heatmap(corrmatrix, vmin=0.2, vmax=0.9, annot=True, cmap="Blues")
```

Out[46]: <AxesSubplot:>



In [47]:
```
data.plot(kind='box',subplots=True,layout=(2,3),figsize=(16,16));
```

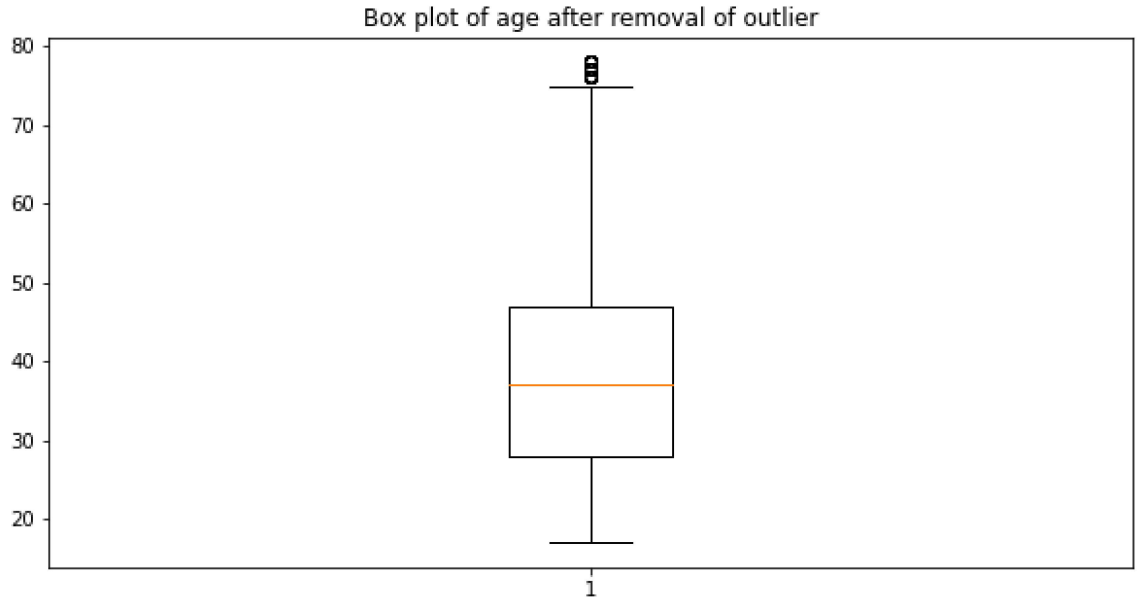


In [48]:
```
#Outlier handling of age

Q1=np.percentile(data['age'],25,interpolation='midpoint')
Q2=np.percentile(data['age'],50,interpolation='midpoint')
Q3=np.percentile(data['age'],75,interpolation='midpoint')
IQR=Q3-Q1
print(Q1,Q2,Q3)
low_limit=Q1-1.5*IQR
up_limit=Q3+1.5*IQR
print('low_limit=',low_limit)
print('up_limit=',up_limit)
outlier=[]
for x in data['age']:
  if((x>up_limit) or (x<low_limit)):
    outlier.append(x)

indA=data['age']>up_limit
indA1=data.loc[indA].index
indB=data['age']<low_limit
indB1=data.loc[indB].index

data.drop(indA1,inplace=True)
data.drop(indB1,inplace=True)
```

```
28.0 37.0 48.0
low_limit= -2.0
up_limit= 78.0
```

In [49]:
```
plt.boxplot(data['age'])
plt.title('Box plot of age after removal of outlier')
```

Out[49]:
```
Text(0.5, 1.0, 'Box plot of age after removal of outlier')
```

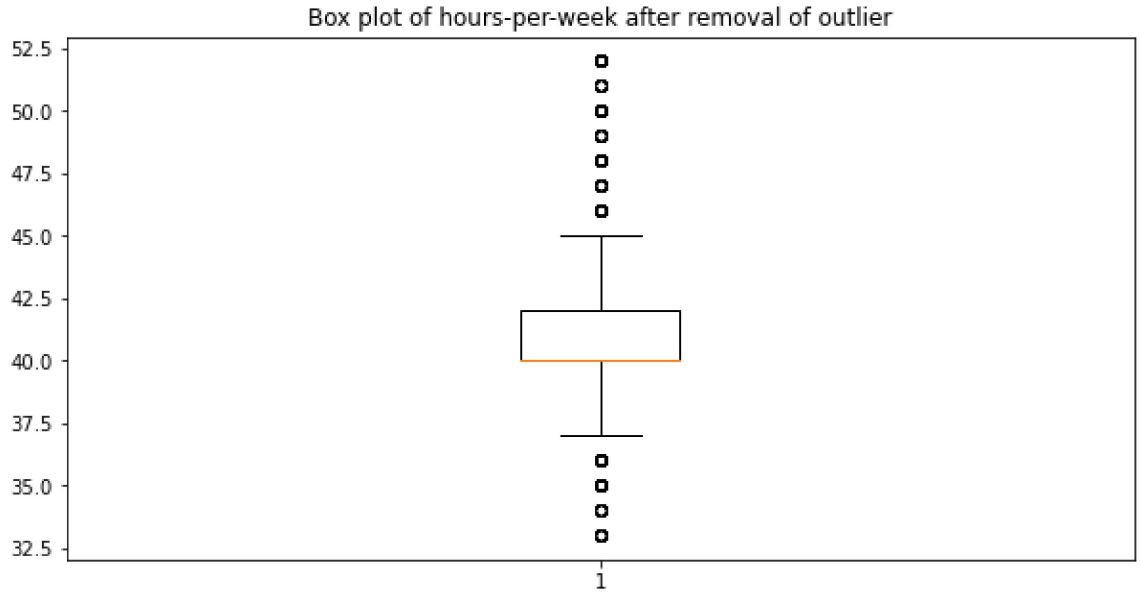


In [50]:
```
#Outlier handling of hours-per-week

Q1=np.percentile(data['hours-per-week'],25,interpolation='midpoint')
Q2=np.percentile(data['hours-per-week'],50,interpolation='midpoint')
Q3=np.percentile(data['hours-per-week'],75,interpolation='midpoint')
IQR=Q3-Q1
print(Q1,Q2,Q3)
low_limit=Q1-1.5*IQR
up_limit=Q3+1.5*IQR
print('low_limit=',low_limit)
print('up_limit=',up_limit)
outlier=[]
for x in data['hours-per-week']:
  if((x>up_limit) or (x<low_limit)):
    outlier.append(x)

indA=data['hours-per-week']>up_limit
indA1=data.loc[indA].index
indB=data['hours-per-week']<low_limit
indB1=data.loc[indB].index

data.drop(indA1,inplace=True)
data.drop(indB1,inplace=True)
```

```
40.0 40.0 45.0
low_limit= 32.5
up_limit= 52.5
```

In [51]:
```
plt.boxplot(data['hours-per-week'])
plt.title('Box plot of hours-per-week after removal of outlier')
```

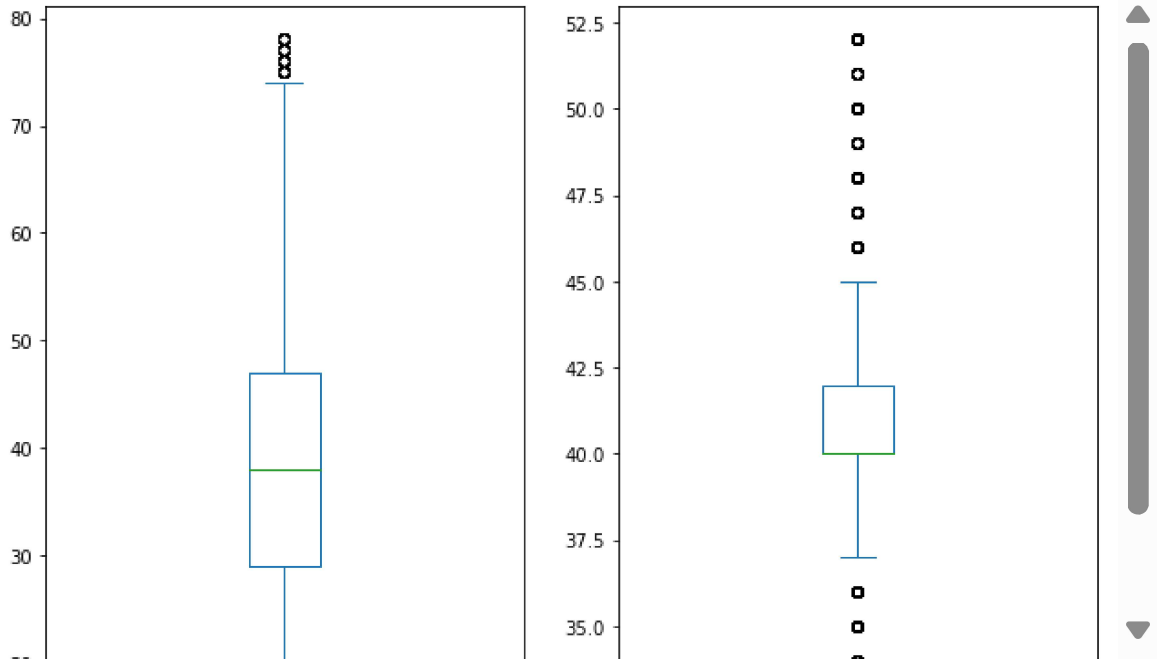
Out[51]:
```
Text(0.5, 1.0, 'Box plot of hours-per-week after removal of outlier')
```



In [52]:
```
data.shape
```

Out[52]:
```
(23499, 11)
```

In [53]:
```
data.plot(kind='box',subplots=True,layout=(2,3),figsize=(16,16));
```



Outliers have been addressed, resulting in a streamlined dataset with dimensions reduced from (32561,14) to (23499,11).

Using groupby condition

In [54]:
```
data.groupby('education')['salary'].value_counts()\
.unstack().style.background_gradient(cmap='gist_heat_r')
```

Out[54]:

salary	<=50K	>50K
education		
10th	559.000000	50.000000
11th	602.000000	49.000000
12th	248.000000	28.000000
1st-4th	110.000000	4.000000
5th-6th	247.000000	12.000000
7th-8th	418.000000	26.000000
9th	368.000000	22.000000
Assoc-acdm	565.000000	216.000000
Assoc-voc	795.000000	292.000000
Bachelors	2326.000000	1716.000000
Doctorate	63.000000	194.000000
HS-grad	6663.000000	1331.000000
Masters	541.000000	721.000000
Preschool	33.000000	nan
Prof-school	85.000000	260.000000
Some-college	3883.000000	1072.000000

Create a pivot table

In [55]:
```
pivot_table=data.pivot_table(columns='workclass',index='education',values='
pivot_table.style.background_gradient(cmap='cividis_r')
```

Out[55]:

workclass	Federal-gov	Local-gov	Never-worked	Private	Self-emp-inc	Self-emp not-in
education						
10th	197.000000	805.000000	40.000000	21135.000000	510.000000	1560.00000
11th	200.000000	770.000000	nan	23734.000000	255.000000	1218.00000
12th	120.000000	559.000000	nan	9667.000000	125.000000	432.00000
1st-4th	nan	80.000000	nan	4369.000000	nan	160.00000
5th-6th	40.000000	285.000000	nan	9549.000000	170.000000	350.00000
7th-8th	40.000000	868.000000	35.000000	14516.000000	442.000000	2013.00000
9th	120.000000	765.000000	nan	13693.000000	299.000000	885.00000
Assoc-acdm	2025.000000	2812.000000	nan	23557.000000	1212.000000	1735.00000
Assoc-	1390.000000	2756.000000	nan	35984.000000	811.000000	2260.00000

After analyzing the data,Some intriguing inquiries emerged from the dataset.

In [56]:
```
print('The most demanding education is',data['education'].value_counts().id
print('\n the least demanding  education is',data['education'].value_counts
print('\n The highest working hours in the data is',data['hours-per-week'].
print('\n The less working hours in the data is',data['hours-per-week'].valu
print('\n Most dominate race is',data['race'].value_counts().idxmax())
print('\n Less dominate race is ',data['race'].value_counts().idxmin())
print('\n Most dominate occupation is',data['occupation'].value_counts().id
print('\n Less dominate race is',data['occupation'].value_counts().idxmin()
```

```
The most demanding education is HS-grad

 the least demanding  education is Preschool

 The highest working hours in the data is 40

 The less working hours in the data is 51

 Most dominate race is White

 Less dominate race is  Other

 Most dominate occupation is Prof-specialty

 Less dominate race is Armed-Forces
```

Label Encoding

In [57]:
```
# Importing LabelEncoder
from sklearn.preprocessing import LabelEncoder
label= LabelEncoder()
```

In [58]:
```
data['workclass']=label.fit_transform(data['workclass'])
data['education']=label.fit_transform(data['education'])
data['occupation']=label.fit_transform(data['occupation'])
data['sex']=label.fit_transform(data['sex'])
data['salary']=label.fit_transform(data['salary'])
data['race']=label.fit_transform(data['race'])
data['native-country']=label.fit_transform(data['native-country'])
data['marital-status']=label.fit_transform(data['marital-status'])
data['relationship']=label.fit_transform(data['relationship'])
```

In [59]:
```
data
```

Out[59]:

	age	workclass	education	marital-status	occupation	relationship	race	sex	hours-per-week	nati coun
0	39	6	9	4	0	1	4	1	40	
2	38	3	11	0	5	1	4	1	40	
3	53	3	1	2	5	0	2	1	40	
4	28	3	9	2	9	5	2	0	40	
5	37	3	12	2	3	5	4	0	40	
...	...	...	...	...	...	...	...	...	...	
32555	22	3	15	4	10	1	4	1	40	
32556	27	3	7	2	12	5	4	0	38	
32557	40	3	11	2	6	0	4	1	40	
32558	58	3	11	6	0	4	4	0	40	
32560	52	4	11	2	3	5	4	0	40	

23499 rows × 11 columns

Standardization

Setting the feature and targert variables

In [60]:
```
X=data.drop(columns=['salary'],axis=1)
y=data['salary']
X.head()
```

Out[60]:

	age	workclass	education	marital-status	occupation	relationship	race	sex	hours-per-week	native-country
0	39	6	9	4	0	1	4	1	40	38
2	38	3	11	0	5	1	4	1	40	38
3	53	3	1	2	5	0	2	1	40	38
4	28	3	9	2	9	5	2	0	40	4
5	37	3	12	2	3	5	4	0	40	38

Standardizing the feature using StandardScaler

In [61]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

Out[61]:
```
array([[ 0.02129829,  2.65135489, -0.36291919, ...,  0.67403711,
        -0.39054377,  0.26699709],
       [-0.06073556, -0.02804943,  0.17988309, ...,  0.67403711,
        -0.39054377,  0.26699709],
       [ 1.16977222, -0.02804943, -2.53412829, ...,  0.67403711,
        -0.39054377,  0.26699709],
       ...,
       [ 0.10333214, -0.02804943,  0.17988309, ...,  0.67403711,
        -0.39054377,  0.26699709],
       [ 1.57994149, -0.02804943,  0.17988309, ..., -1.48359783,
        -0.39054377,  0.26699709],
       [ 1.08773837,  0.86508535,  0.17988309, ..., -1.48359783,
        -0.39054377,  0.26699709]])
```

Splitting the data into training and testing data

In [62]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size =
```

In [63]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[63]:
```
((16449, 10), (7050, 10), (16449,), (7050,))
```

Modeling

In [64]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score,prec
```

1. Logistic Regression

Fitting the training data to Logostic Regression Model

In [65]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression(max_iter=2000)
lr.fit(X_train,y_train)
```

Out[65]:
```
LogisticRegression(max_iter=2000)
```

Predicting using test data

In [66]:
```
pred1=lr.predict(X_test)
pred1
```

Out[66]:
```
array([0, 0, 0, ..., 0, 0, 1])
```

Checking the confusion matrix and accuracy of the model

In [67]:
```
con_lr=confusion_matrix(y_test,pred1)
print("The confusion matrix of logistic regression is \n",con_lr)

ac_lr=accuracy_score(y_test,pred1)
print('Accuracy:',ac_lr*100)
```

```
The confusion matrix of logistic regression is
 [[4930  326]
 [1384  410]]
Accuracy: 75.74468085106383
```

Classification Report

In [68]:
```
print(classification_report(y_test, pred1))
```

```
              precision    recall  f1-score   support

           0       0.78      0.94      0.85      5256
           1       0.56      0.23      0.32      1794

    accuracy                           0.76      7050
   macro avg       0.67      0.58      0.59      7050
weighted avg       0.72      0.76      0.72      7050
```

2. K Nearest Neighbour Classifier (KNN Classifier)

Fitting the training data to KNNClassifier Model

In [69]:
```
from sklearn.neighbors import KNeighborsClassifier
acc_values=[]
neighbors=np.arange(70,90)
for k in neighbors:
    knn=KNeighborsClassifier(n_neighbors=k, metric='minkowski')
    knn.fit(X_train, y_train)
    pred2=knn.predict(X_test)
    ac_knn=accuracy_score(y_test,pred2)
    acc_values.append(ac_knn)
```

In [70]:
```
acc_values
```

Out[70]:
```
[0.7973049645390071,
 0.7971631205673759,
 0.7963120567375886,
 0.7963120567375886,
 0.7943262411347518,
 0.794468085106383,
 0.7937588652482269,
 0.7950354609929078,
 0.7948936170212766,
 0.7957446808510639,
 0.7920567375886525,
 0.7948936170212766,
 0.7947517730496454,
 0.7956028368794327,
 0.7934751773049645,
 0.7950354609929078,
 0.794468085106383,
 0.7943262411347518,
 0.7933333333333333,
 0.7943262411347518]
```

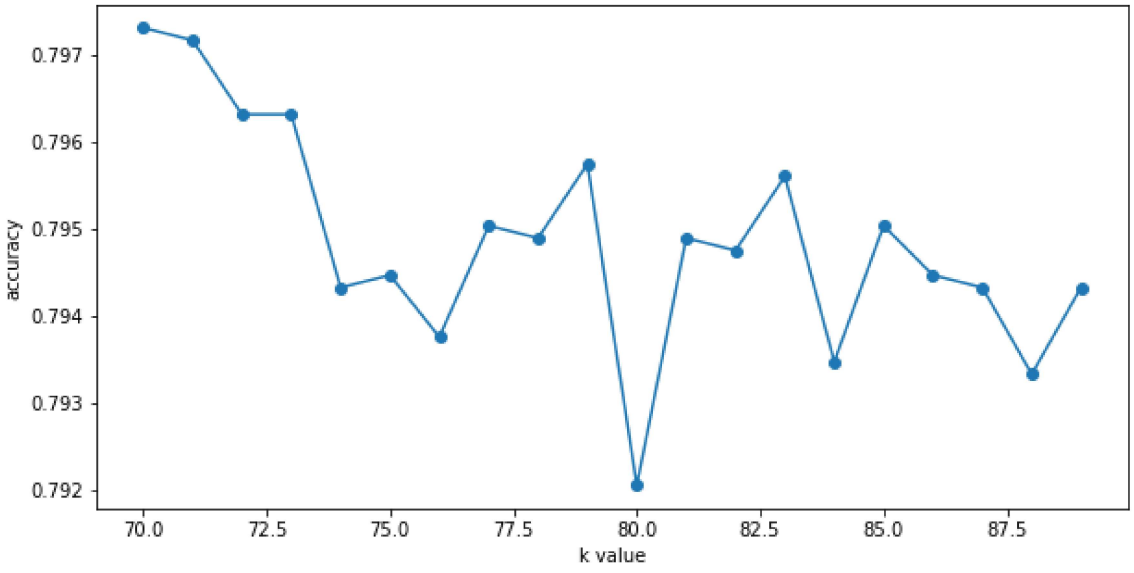
Checking the average accuracy of the model

In [71]:
```
avg_acc = np.array(acc_values).mean()
print('Accuracy: ', avg_acc * 100)
```

```
Accuracy:  79.48794326241135
```

In [72]:
```
plt.plot(neighbors,acc_values,'o-')
plt.xlabel('k value')
plt.ylabel('accuracy')
```

Out[72]:
```
Text(0, 0.5, 'accuracy')
```



Classification Report

In [73]:
```
print(classification_report(y_test, pred2))
```

```
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      5256
           1       0.61      0.53      0.57      1794

    accuracy                           0.79      7050
   macro avg       0.73      0.71      0.72      7050
weighted avg       0.79      0.79      0.79      7050
```

3. Support Vector Machine

Fitting the training data to SVC Model

In [74]:
```
from sklearn.svm import SVC
svc=SVC()
svc.fit(X_train,y_train)
```

Out[74]:
```
SVC()
```

Predicting using test data

In [75]:
```
pred3=svc.predict(X_test)
pred3
```

Out[75]:
```
array([0, 0, 1, ..., 0, 0, 0])
```

Checking the confusion matrix and accuracy of the model

In [76]:
```
con_svc=confusion_matrix(y_test,pred3)
print("The confusion matrix of decision tree is \n",con_svc)

ac_svc=accuracy_score(y_test,pred3)
print('Accuracy:',ac_svc*100)
```

```
The confusion matrix of decision tree is
 [[4713  543]
 [ 939  855]]
Accuracy: 78.97872340425532
```

Classification Report

In [77]:
```
print(classification_report(y_test, pred3))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      5256
           1       0.61      0.48      0.54      1794

    accuracy                           0.79      7050
   macro avg       0.72      0.69      0.70      7050
weighted avg       0.78      0.79      0.78      7050
```

4.Decision Tree classifier

Fitting the training data to Decision Tree Model

In [78]:
```
from sklearn.tree import DecisionTreeClassifier
dt_clf=DecisionTreeClassifier()
dt_clf.fit(X_train,y_train)
```

Out[78]:
```
DecisionTreeClassifier()
```

Predicting using test data

In [79]:
```
pred4=dt_clf.predict(X_test)
pred4
```

Out[79]:
```
array([0, 0, 1, ..., 0, 1, 1])
```

Checking the confusion matrix and accuracy of the model

In [80]:
```
con_dtr=confusion_matrix(y_test,pred4)
print("The confusion matrix of decision tree is \n",con_dtr)

ac_dt=accuracy_score(y_test,pred4)
print('Accuracy:',ac_dt*100)
```

```
The confusion matrix of decision tree is
 [[4454  802]
 [ 880  914]]
Accuracy: 76.1418439716312
```

Classification Report

In [81]:
```
print(classification_report(y_test, pred4))
```

```
              precision    recall  f1-score   support

           0       0.84      0.85      0.84      5256
           1       0.53      0.51      0.52      1794

    accuracy                           0.76      7050
   macro avg       0.68      0.68      0.68      7050
weighted avg       0.76      0.76      0.76      7050
```

5.Random Forest Classifier

Fitting the training data to Random Forest Model

In [82]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
```

Out[82]:
```
RandomForestClassifier()
```

Predicting using test data

In [83]:
```
pred5=rf.predict(X_test)
pred5
```

Out[83]:
```
array([0, 0, 1, ..., 0, 1, 1])
```

Checking the confusion matrix and accuracy of the model

In [84]:
```
con_rf=confusion_matrix(y_test,pred5)
print("The confusion matrix of random forest is \n",con_rf)

ac_rf=accuracy_score(y_test,pred5)
print('Accuracy:',ac_rf*100)
```

```
The confusion matrix of random forest is
 [[4653  603]
 [ 819  975]]
Accuracy: 79.82978723404256
```

Classification Report

In [85]:
```
print(classification_report(y_test, pred5))
```

```
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      5256
           1       0.62      0.54      0.58      1794

    accuracy                           0.80      7050
   macro avg       0.73      0.71      0.72      7050
weighted avg       0.79      0.80      0.79      7050
```

Summary of accuracies of all models

Tabulating the accuracies of different models

In [86]:
```
from tabulate import tabulate
table = [['Model', 'Accuracy'], ['Logistic Regression',ac_lr], ['KNN',avg_a
print(tabulate(table,headers='firstrow',tablefmt='fancy_grid'))
```

Model	Accuracy
Logistic Regression	0.757447
KNN	0.794879
Decision tree	0.761418
SVM	0.789787
Random Forest	0.798298

From the table, it is clear that Random forest have better accuracy compared to others.So Random forest is taken as our model to predict the salary.So we can tune this to check whether the performance is improving.

Hyper parameter tuning of Random Forest Model

Fittin the training set

In [87]:
```
rf=RandomForestClassifier(criterion='gini',max_depth=10,n_estimators=600)
rf.fit(X_train,y_train)
```

Out[87]:
```
RandomForestClassifier(max_depth=10, n_estimators=600)
```

Passing the test set

In [88]:
```
y_pr=rf.predict(X_test)
y_pr
```

Out[88]:
```
array([0, 0, 1, ..., 0, 0, 0])
```

Checking the accuracy

In [89]:
```
acc_sc=accuracy_score(y_test,y_pr)*100
print('Accuracy: ', acc_sc)
```

```
Accuracy:  82.56737588652481
```

Hyper parameter tuning improved the accuracy of Random forest modeling to 82.57%.

So we can take random forest classifier to build our model.

In [ ]: