Income Prediction Using Census Income Data Set

Problem Statement

Predict whether income exceeds $50K/yr based on census data. Also known as "Adult" dataset.

Data Collection

dataset: [https://archive.ics.uci.edu/ml/datasets/Census+Income (https://archive.ics.uci.edu/ml/datasets/Census+Income)](https://archive.ics.uci.edu/ml/datasets/Census+Income)

Attribute Information:

The Census Income dataset has 48,842 entries. Each entry contains the following information about an individual:

- salary (target feature/label): whether or not an individual makes more than $50,000 annually. (<= 50K, >50K)
- age: the age of an individual. (Integer greater than 0)
- workclass: a general term to represent the employment status of an individual. (Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked)

- fnlwgt: this is the number of people the census believes the entry represents. People with similar demographic characteristics should have similar weights. There is one important caveat to remember about this statement. That is that since the CPS sample is actually a collection of 51 state samples, each with its own probability of selection, the statement only applies within state.(Integer greater than 0)
- education: the highest level of education achieved by an individual. (Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.) education-num: the highest level of education achieved in numerical form. (Integer greater than 0)
- marital-status: marital status of an individual. Married-civ-spouse corresponds to a civilian spouse while Married-AF-spouse is a spouse in the Armed Forces. Married-spouse-absent includes married people living apart because either the husband or wife was employed and living at a considerable distance from home (Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse)
- occupation: the general type of occupation of an individual. (Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces) relationship: represents what this individual is relative to others. For example an individual could be a Husband. Each entry only has one relationship attribute. (Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried)
- race: Descriptions of an individual's race. (White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black)
- sex: the biological sex of the individual. (Male, female)
- capital-gain: capital gains for an individual. (Integer greater than or equal to 0)
- capital-loss: capital loss for an individual. (Integer greater than or equal to 0)
- hours-per-week: the hours an individual has reported to work per week. (continuous)
- native-country: country of origin for an individual (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, Trinadad&Tobago, Peru, Hong, Holand-Netherlands)

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,ConfusionMatrixDis
                            precision_score, recall_score, f1_score, roc_auc_score,r


from sklearn import metrics
from sklearn.model_selection import  train_test_split, RepeatedStratifiedKFold, cros
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer, KNNImputer
from xgboost import XGBClassifier
from sklearn.preprocessing import StandardScaler, MinMaxScaler,RobustScaler
from sklearn.compose import ColumnTransformer
from catboost import CatBoostClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
df=pd.read_csv('../data/adult.csv',names=['age','workclass','fnlwgt','education','ec
```

EDA

In [3]:

```
# Taking a first look
df.head()
```

Out[3]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	I
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	I
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	I
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	I
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Fe

In [4]:

```
df.shape
```

Out[4]:

```
(32561, 15)
```

In [5]:

```
test=pd.read_csv('../data/test.csv',names=['age','workclass','fnlwgt','education','e
```

In [6]:

```
test.drop(0,inplace=True)
test.head()
```

Out[6]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	
1	25	Private	226802.0	11th	7.0	Never-married	Machine-op-inspct	Own-child	Black	
2	38	Private	89814.0	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	White	
3	28	Local-gov	336951.0	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	White	
4	44	Private	160323.0	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	Black	
5	18	?	103497.0	Some-college	10.0	Never-married	?	Own-child	White	F

In [7]:

```
df = pd.concat([df,test],ignore_index=True)
```

In [8]:

```
#Final shape of data
df.shape
```

Out[8]:

```
(48842, 15)
```

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             48842 non-null  object 
 1   workclass       48842 non-null  object 
 2   fnlwgt          48842 non-null  float64
 3   education       48842 non-null  object 
 4   education-num   48842 non-null  float64
 5   marital-status  48842 non-null  object 
 6   occupation      48842 non-null  object 
 7   relationship    48842 non-null  object 
 8   race            48842 non-null  object 
 9   sex             48842 non-null  object 
 10  capital-gain    48842 non-null  float64
 11  capital-loss    48842 non-null  float64
 12  hours-per-week  48842 non-null  float64
 13  native-country  48842 non-null  object 
 14  salary          48842 non-null  object 
dtypes: float64(5), object(10)
memory usage: 5.6+ MB
```

In [10]:

```
df['age'] = df['age'].astype('int')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             48842 non-null  int64  
 1   workclass       48842 non-null  object 
 2   fnlwgt          48842 non-null  float64
 3   education       48842 non-null  object 
 4   education-num   48842 non-null  float64
 5   marital-status  48842 non-null  object 
 6   occupation      48842 non-null  object 
 7   relationship    48842 non-null  object 
 8   race            48842 non-null  object 
 9   sex             48842 non-null  object 
 10  capital-gain    48842 non-null  float64
 11  capital-loss    48842 non-null  float64
 12  hours-per-week  48842 non-null  float64
 13  native-country  48842 non-null  object 
 14  salary          48842 non-null  object 
dtypes: float64(5), int64(1), object(9)
memory usage: 5.6+ MB
```

In [11]:

```
df.describe(include='all').T
```

Out[11]:

	count	unique	top	freq	mean	std	min	25%
age	48842.0	NaN	NaN	NaN	38.643585	13.71051	17.0	28.0
workclass	48842	9	Private	33906	NaN	NaN	NaN	NaN
fnlwgt	48842.0	NaN	NaN	NaN	189664.134597	105604.025423	12285.0	117550.5
education	48842	16	HS-grad	15784	NaN	NaN	NaN	NaN
education-num	48842.0	NaN	NaN	NaN	10.078089	2.570973	1.0	9.0
marital-status	48842	7	Married-civ-spouse	22379	NaN	NaN	NaN	NaN
occupation	48842	15	Prof-specialty	6172	NaN	NaN	NaN	NaN
relationship	48842	6	Husband	19716	NaN	NaN	NaN	NaN
race	48842	5	White	41762	NaN	NaN	NaN	NaN
sex	48842	2	Male	32650	NaN	NaN	NaN	NaN
capital-gain	48842.0	NaN	NaN	NaN	1079.067626	7452.019058	0.0	0.0
capital-loss	48842.0	NaN	NaN	NaN	87.502314	403.004552	0.0	0.0
hours-per-week	48842.0	NaN	NaN	NaN	40.422382	12.391444	1.0	40.0
native-country	48842	42	United-States	43832	NaN	NaN	NaN	NaN
salary	48842	4	<=50K	24720	NaN	NaN	NaN	NaN

In [12]:

```
df.isna().sum()
```

Out[12]:

```
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
salary            0
dtype: int64
```

Report

- Great, No null values.
- But I can see ? in dataset. I think we can consider them as null.

In [13]:

```
df.replace(" ?",np.nan,inplace=True)
df.isnull().sum()
```

Out[13]:

```
age                  0
workclass         2799
fnlwgt               0
education            0
education-num        0
marital-status       0
occupation        2809
relationship         0
race                 0
sex                  0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     857
salary               0
dtype: int64
```

In [14]:

```
df.isnull().sum()/len(df)*100
```

Out[14]:

```
age               0.000000
workclass         5.730724
fnlwgt            0.000000
education         0.000000
education-num     0.000000
marital-status    0.000000
occupation        5.751198
relationship      0.000000
race              0.000000
sex               0.000000
capital-gain      0.000000
capital-loss      0.000000
hours-per-week    0.000000
native-country    1.754637
salary            0.000000
dtype: float64
```

In [15]:

```
df.duplicated().value_counts()
```

Out[15]:

```
False    48813
True        29
dtype: int64
```

In [16]:

```
# Drop Duplicates
df.drop(df[df.duplicated()].index, inplace=True)
```

In [17]:

```
df.shape
```

Out[17]:

```
(48813, 15)
```

Target column (salary)

In [18]:

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

Out[18]:

```
 <=50K     24698
 <=50K.    12430
 >50K       7839
 >50K.      3846
Name: salary, dtype: int64
```

In [19]:

```
df['salary'].replace(" <=50K.", " <=50K",inplace=True)
df['salary'].replace(" >50K.", " >50K",inplace=True)
```

In [20]:

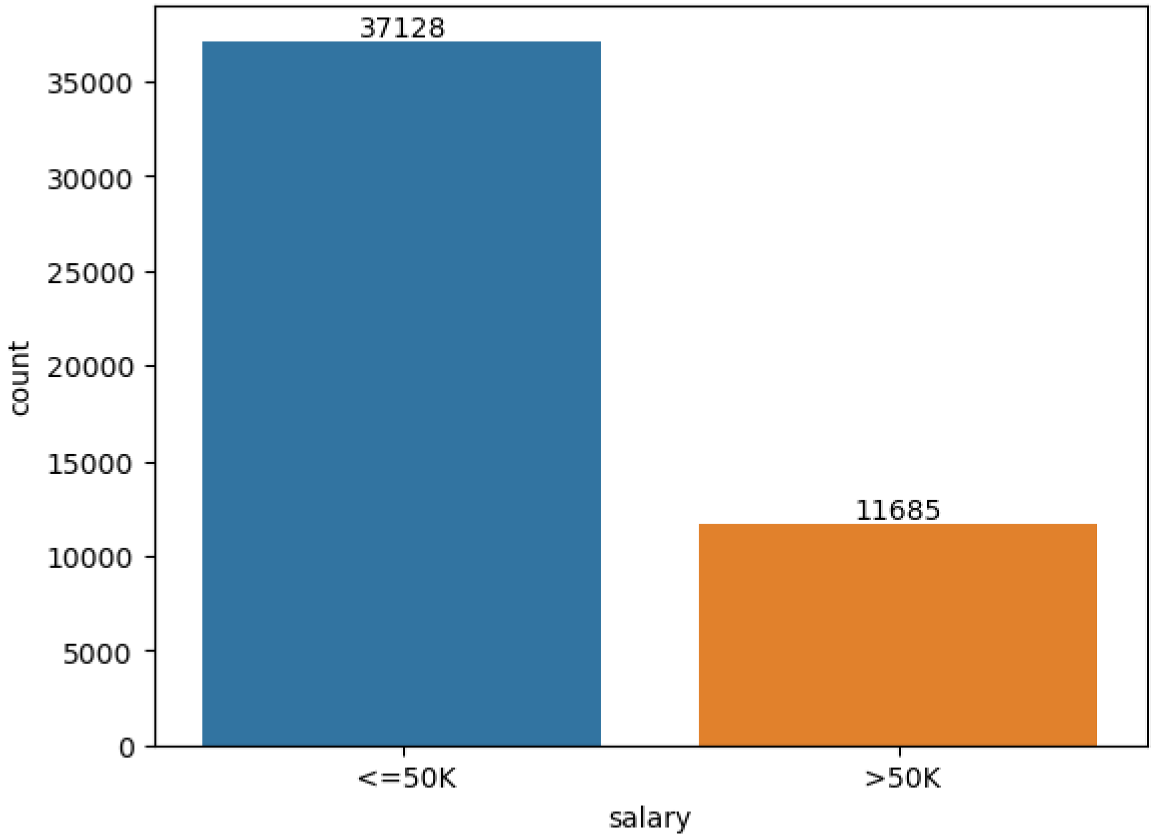
```
df['salary'].value_counts()
```

Out[20]:

```
 <=50K    37128
 >50K     11685
Name: salary, dtype: int64
```

In [21]:

```
img = sns.countplot(x=df['salary'])
for c in img.containers:
    img.bar_label(c)
```

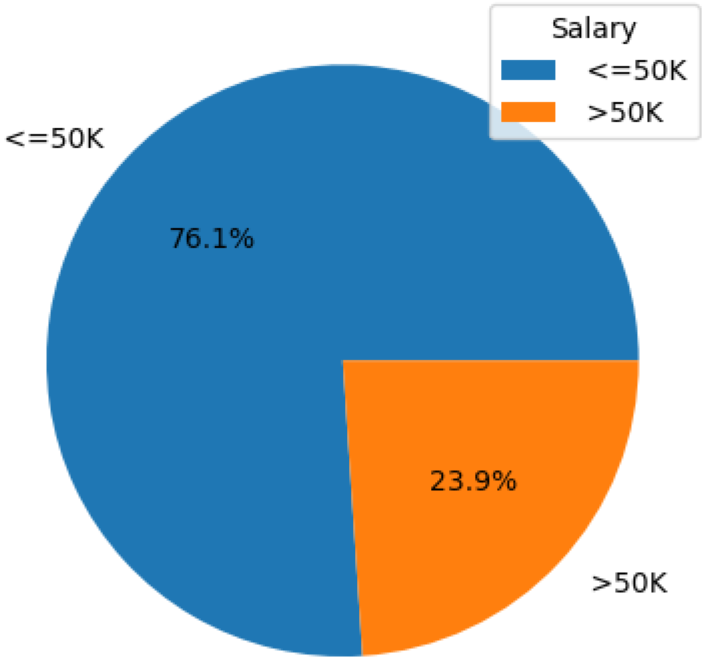


In [22]:

```
plt.pie(df.salary.value_counts(normalize=True),
        labels=df.salary.value_counts(normalize=True).index,
        autopct="%.1f%%")
plt.legend(title="Salary")
```

Out[22]:

```
<matplotlib.legend.Legend at 0x7fca88124ee0>
```



Report

- We'll try balance data by oversampling using SMOTE

In [23]:

```
#define numerical & categorical columns
numeric_features = [feature for feature in df.columns if df[feature].dtype != 'O']
categorical_features = [feature for feature in df.columns if df[feature].dtype == 'O
categorical_features.remove('salary')
# print columns
print('We have {} numerical features : {}'.format(len(numeric_features), numeric_fea
print('\nWe have {} categorical features : {}'.format(len(categorical_features), cat
```

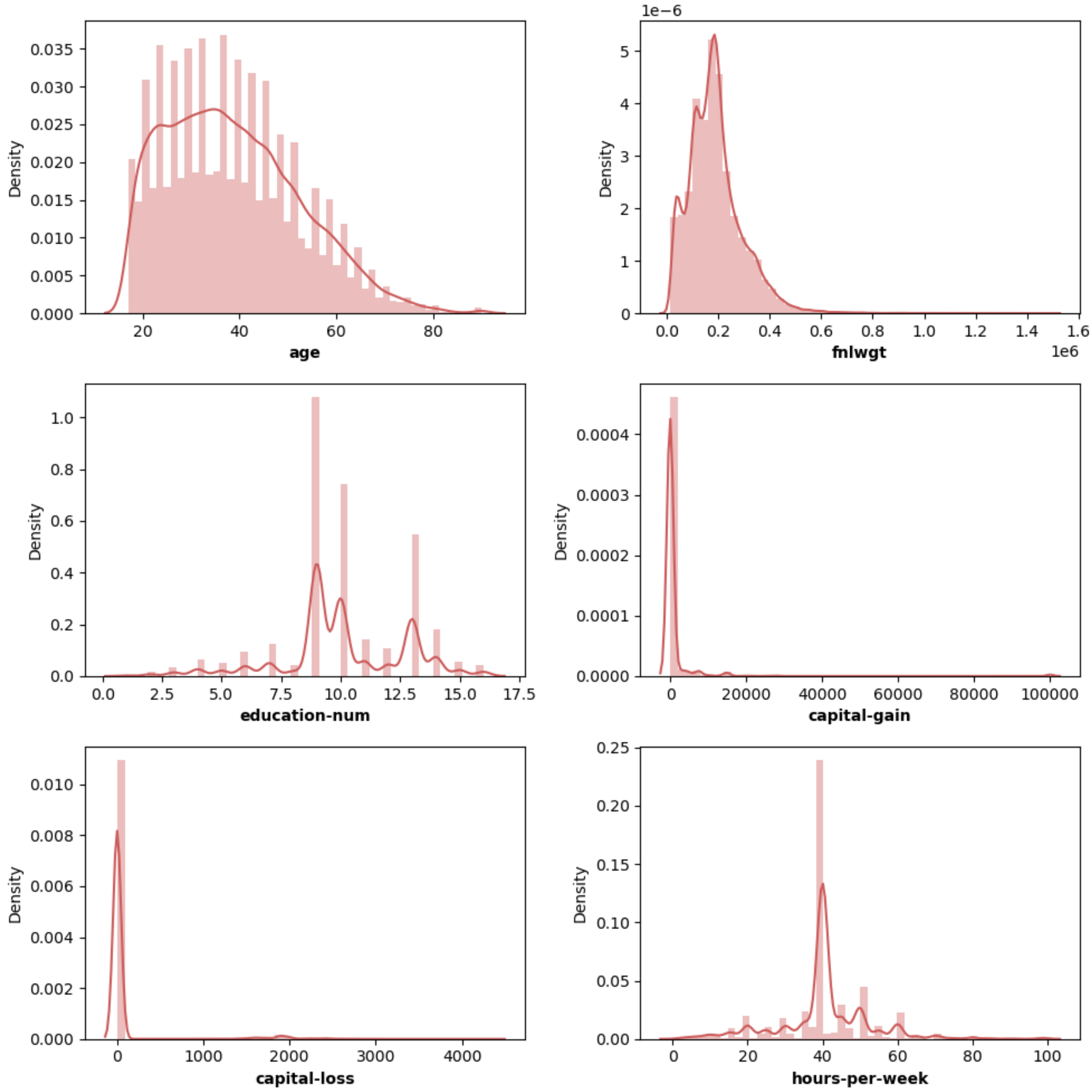
```
We have 6 numerical features : ['age', 'fnlwgt', 'education-num', 'cap
ital-gain', 'capital-loss', 'hours-per-week']

We have 8 categorical features : ['workclass', 'education', 'marital-s
tatus', 'occupation', 'relationship', 'race', 'sex', 'native-country']
```

Univariant Analysis on numerical features

Checking on distribution of numeric features

In [24]:

```
plt.figure(figsize=(10, 10))
for i, col in enumerate(numeric_features):
    plt.subplot(3, 2, i+1)
    sns.distplot(x=df[col], color='indianred')
    plt.xlabel(col, weight='bold')
    plt.tight_layout()
```



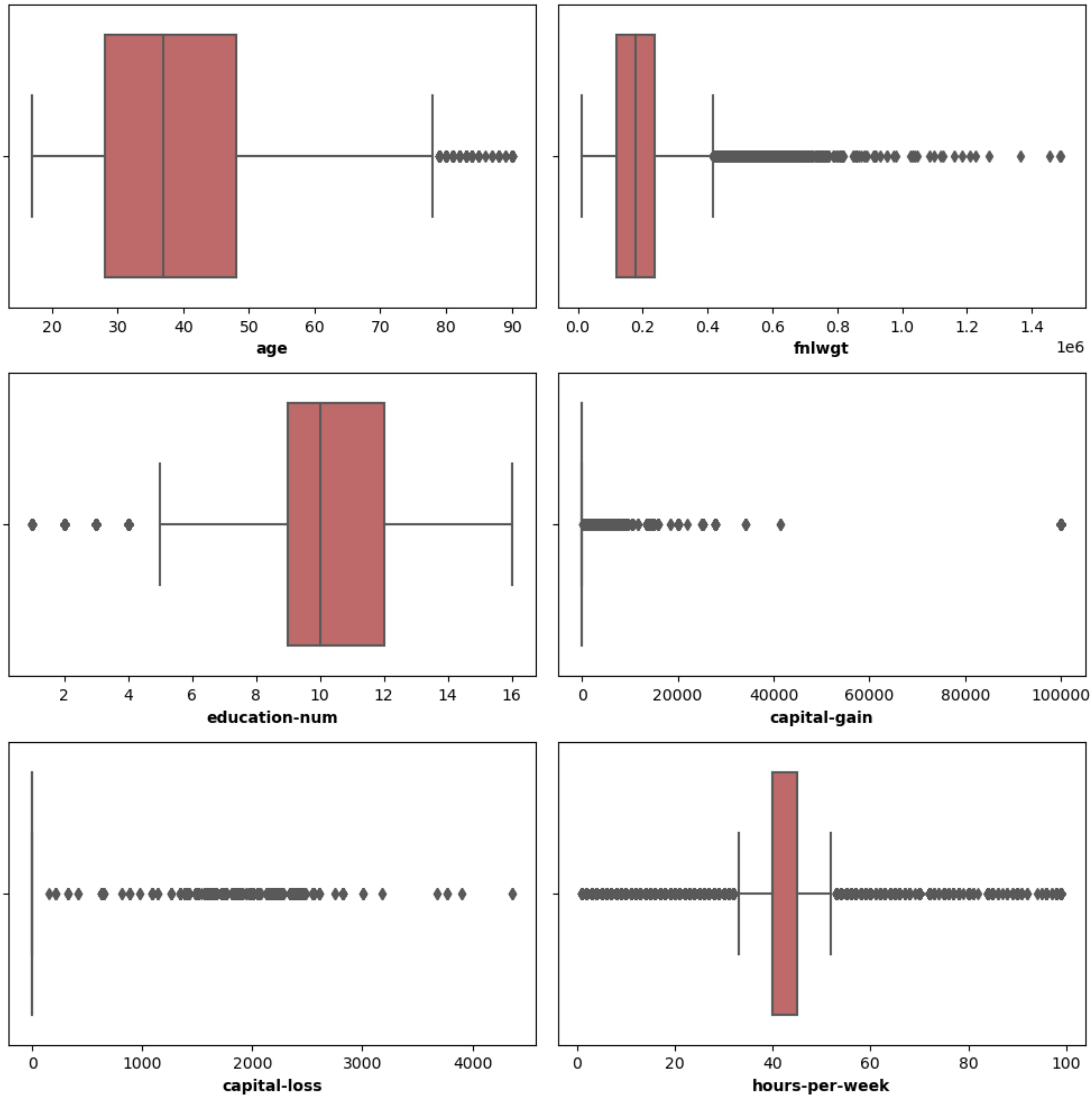
Report

- As per the above plot most of the features are not normally distributed.
- Transformation of data is not of prime importance since it is a classification problem.

Checking outliers on numberic features

In [25]:

```
plt.figure(figsize=(10, 10))
for i, col in enumerate(numeric_features):
    plt.subplot(3, 2, i+1)
    sns.boxplot(x=df[col], color='indianred')
    plt.xlabel(col, weight='bold')
    plt.tight_layout()
```



Report

- Lots of outliers in each features.
- We'll try remove outliers by using IQR method.

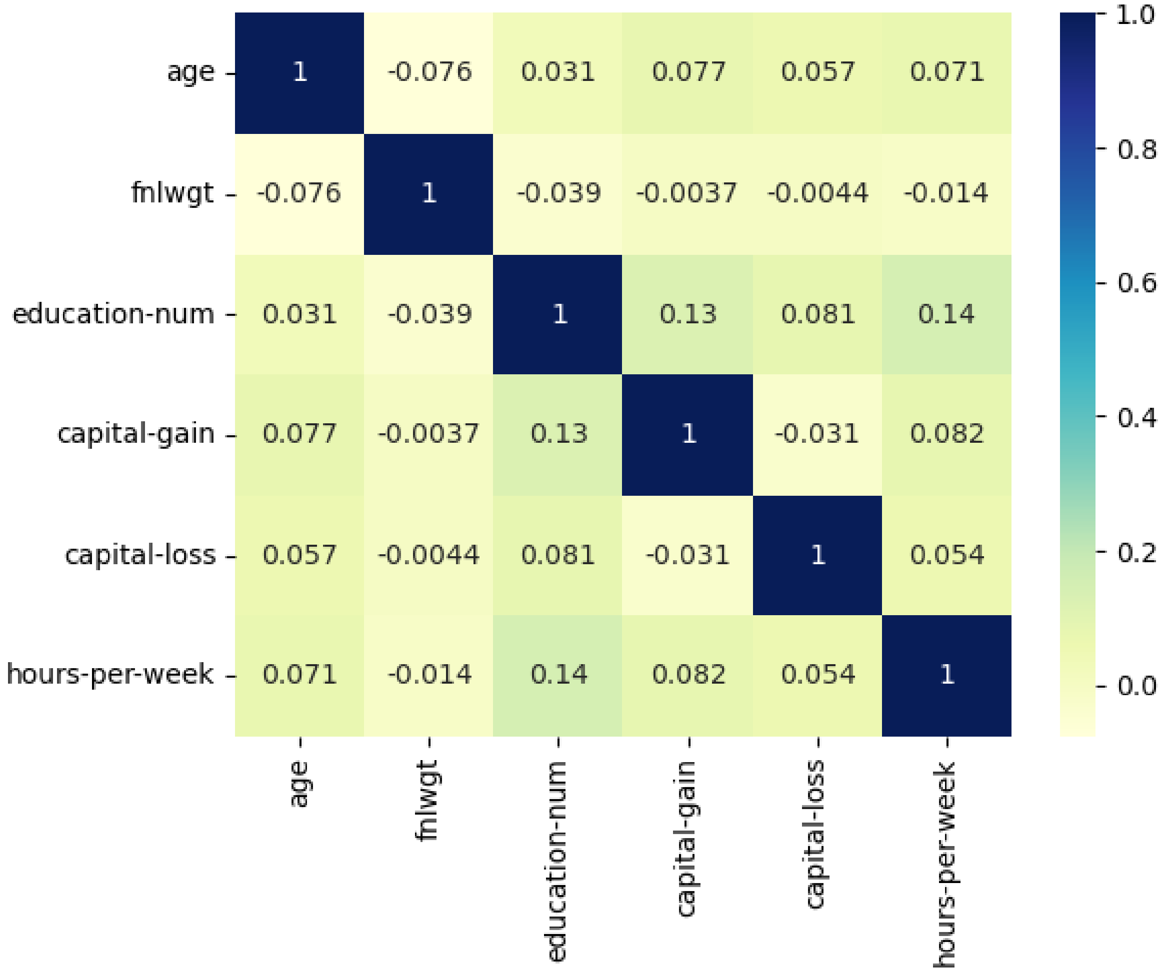
Checking correlation

In [26]:

```
sns.heatmap(df.corr(), annot=True, cmap="YlGnBu")
```

Out[26]:

<AxesSubplot:>



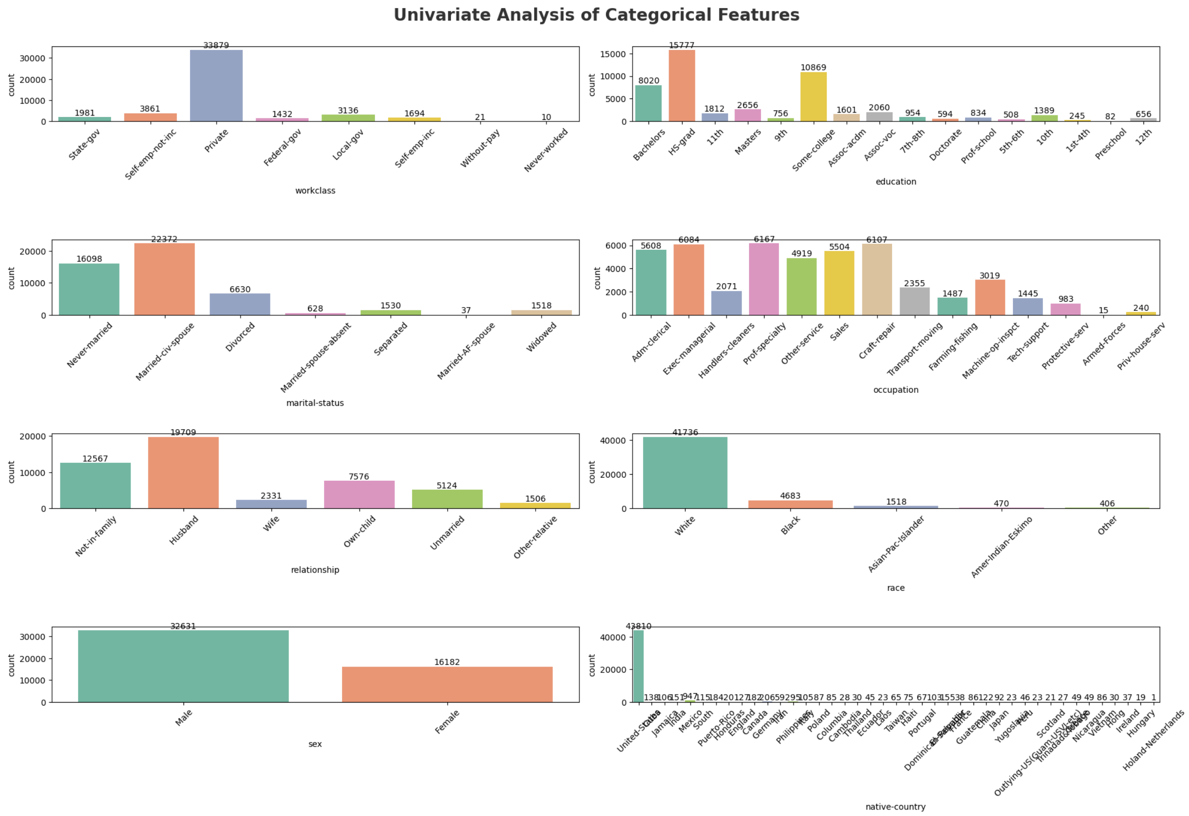
Report

- No correlation.

Checking distribution of categorical features

In [27]:

```
plt.figure(figsize=(20, 15))
plt.suptitle('Univariate Analysis of Categorical Features', fontsize=20, fontweight=
category = categorical_features.copy()
for i in range(0, len(category)):
    plt.subplot(5, 2, i+1)
    img = sns.countplot(x=df[category[i]],palette="Set2")
    plt.xlabel(category[i])
    plt.xticks(rotation=45)
    for c in img.containers:
        img.bar_label(c)
    plt.tight_layout()
```



Report

- We have majority of people working in private sector.
- Most of them are high school graduate.
- We can see most of them are US native. We'll to divide native - country into US and non-US.
- we can see relationship and marital-status are similar kind of column. Better to drop relationship column.

- We'll try to summarize marital-status into Married and Unmarried.
- We can see education and education-num are similar kind of column. Better to drop education-num column.
- We'll summarize marital status into Single and Married

In [28]:

```
df["native-country-summary"] = np.where(df["native-country"] == " United-States", "U
```

In [29]:

```
df["native-country-summary"].isna().sum()
```

Out[29]:

```
0
```

In [30]:

```
df['marital-status-summary'] = df['marital-status'].replace(' Never-married','Single
df['marital-status-summary'] = df['marital-status-summary'].replace(' Married-civ-sp
df['marital-status-summary'] = df['marital-status-summary'].replace( ' Divorced','Si
df['marital-status-summary'] = df['marital-status-summary'].replace(' Married-spouse
df['marital-status-summary'] = df['marital-status-summary'].replace(' Married-AF-spo
df['marital-status-summary'] = df['marital-status-summary'].replace(' Widowed','Sing
df['marital-status-summary'] = df['marital-status-summary'].replace(' Separated','Si
```

In [31]:

```
df['marital-status-summary'].isna().sum()
```

Out[31]:

```
0
```

In [32]:

```
df['marital-status-summary'].unique()
```

Out[32]:

```
array(['Single', 'Married'], dtype=object)
```

In [33]:

```
df['native-country-summary'].unique()
```

Out[33]:

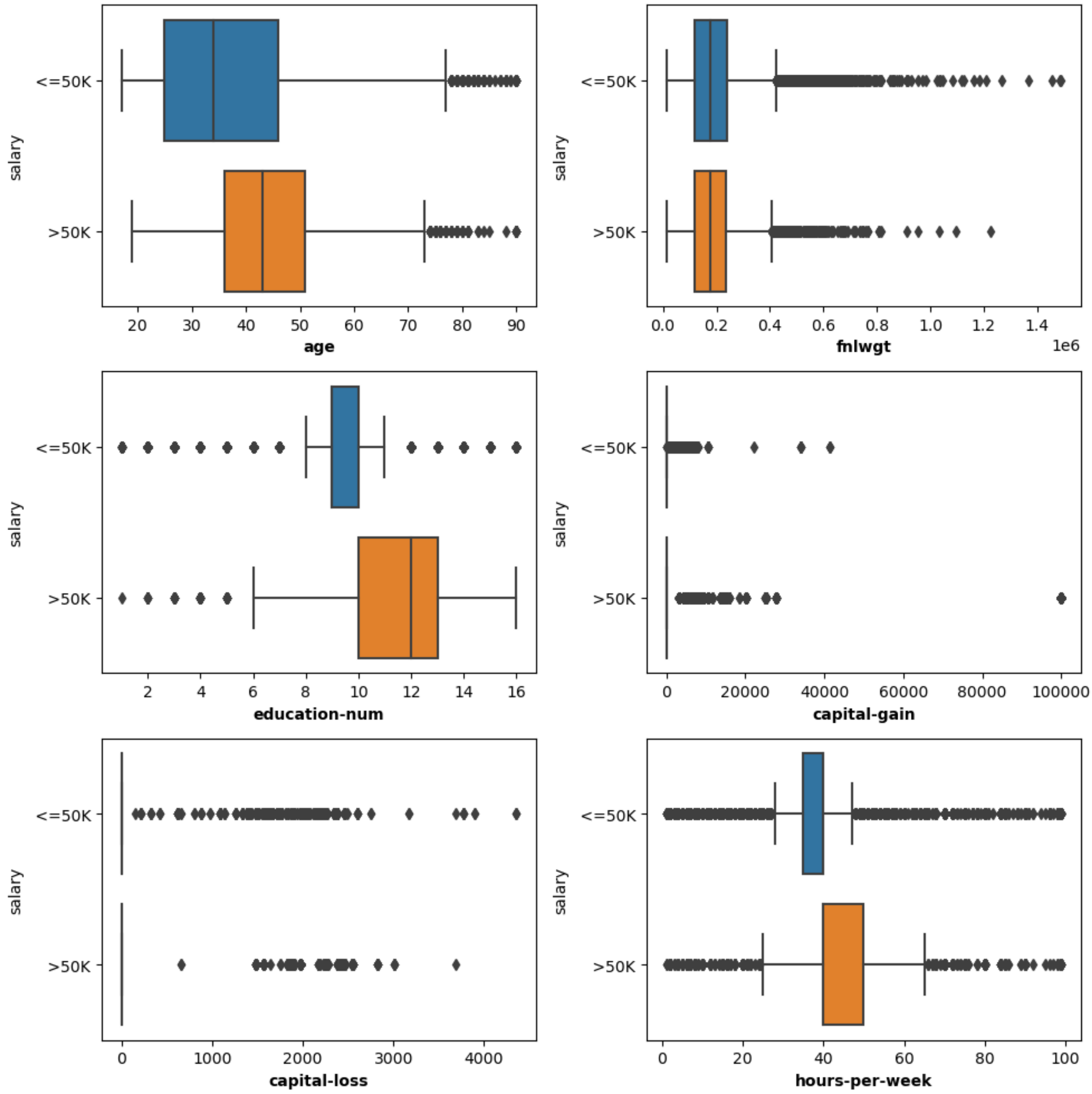
```
array(['US', 'Non-US'], dtype=object)
```

Bivariant Analysis of numerical features

Checking effect of numerical feature on salary

In [34]:

```
plt.figure(figsize=(10, 10))
for i, col in enumerate(numeric_features):
    plt.subplot(3, 2, i+1)
    sns.boxplot(y=df['salary'],x=df[col])
    plt.xlabel(col, weight='bold')
    plt.tight_layout()
```

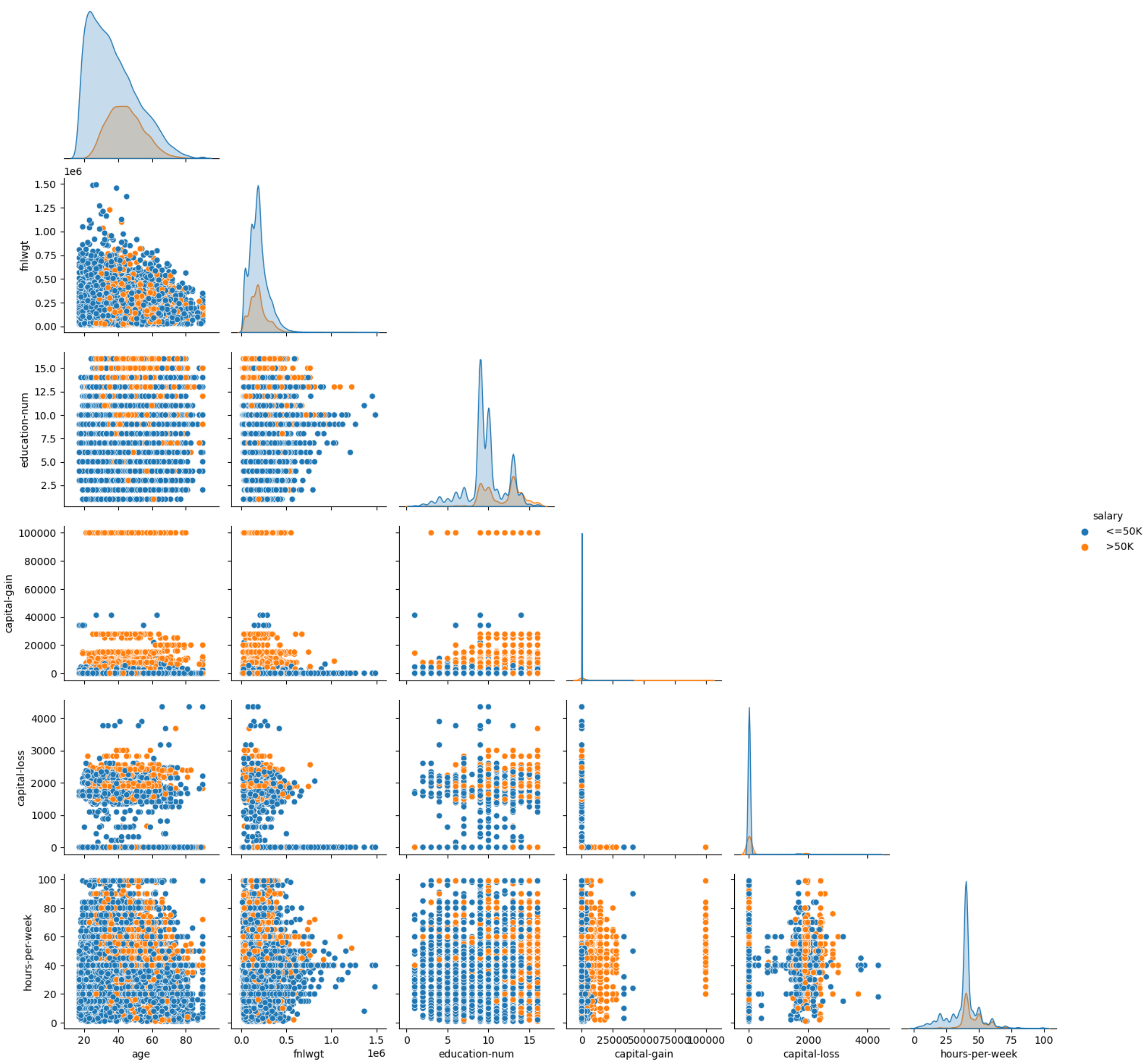


In [35]:

```
sns.pairplot(df, hue="salary", corner=True)
```

Out[35]:

<seaborn.axisgrid.PairGrid at 0x7fca7b81dd90>



Report

- Older population earns more than younger population.
- No effect of fnlwgt,capital gain and capital loss to salary.
- If your hours-per-week is more then your salary will be more.

Bivariant Analysis of Categorical features

In [36]:

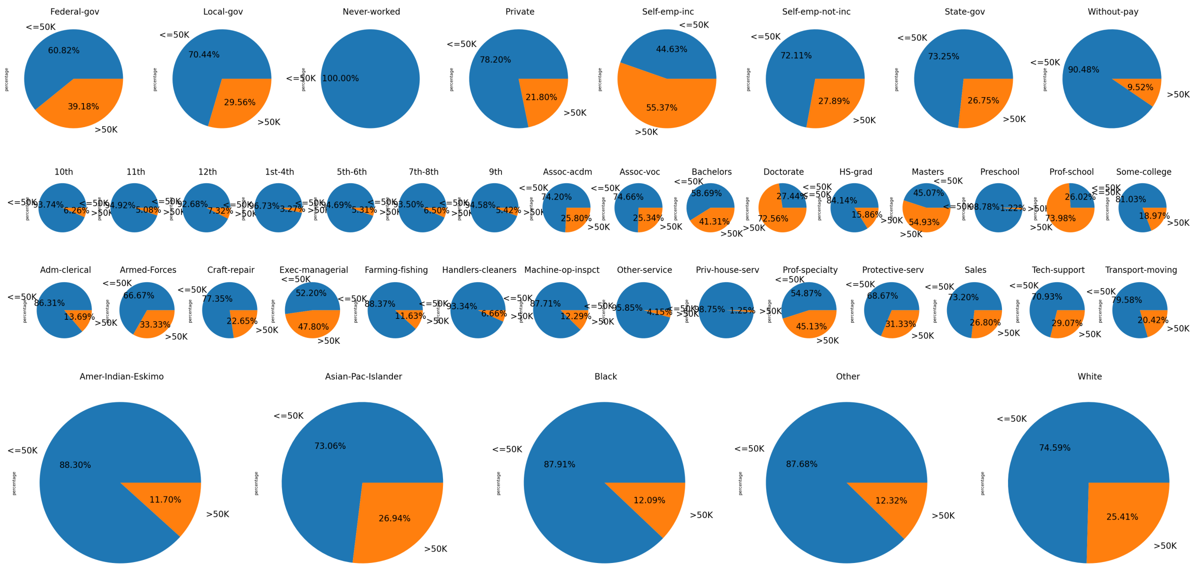
```
def analysis(col):
    count = df.groupby(col)["salary"].value_counts(normalize=True)
    count_df = pd.DataFrame(count)
    count_df.rename(columns = {"salary" : "percentage"}, inplace = True)
    count_df.reset_index(inplace = True)
    count_df.sort_values(by = ["salary", col], inplace=True)
    plt.figure(figsize = (50, 50))
    index = 1
    for c in list(count_df[col].unique()):
        plt.subplot(1,len(list(count_df[col].unique())),index)
        count_df.groupby(col)["percentage"].get_group(c).plot.pie(subplots=True,
                                                          labels=list(count_df['salary'].uniq
                                                          autopct="%.2f%%",
                                                          textprops={'fontsize': 20},
                                                          )
        plt.title(c, fontdict = {'fontsize': 20})
        index += 1
```

In [37]:

```
category = ['workclass',
 'education',
 'occupation',
 'race',
 'sex',
 'native-country-summary',
 'marital-status-summary']
```

In [38]:

```
for col in category:
    analysis(col)
```



Report

- If you are self employed then your chances of getting more than 50k salary.
- High Enducation(Doctorate,Master,Prof-school) brings more salary.
- Armed-force and Exec. Manager pay good salary.

Other Specific Question

1. What is the average age of males and females by income level?

In [39]:

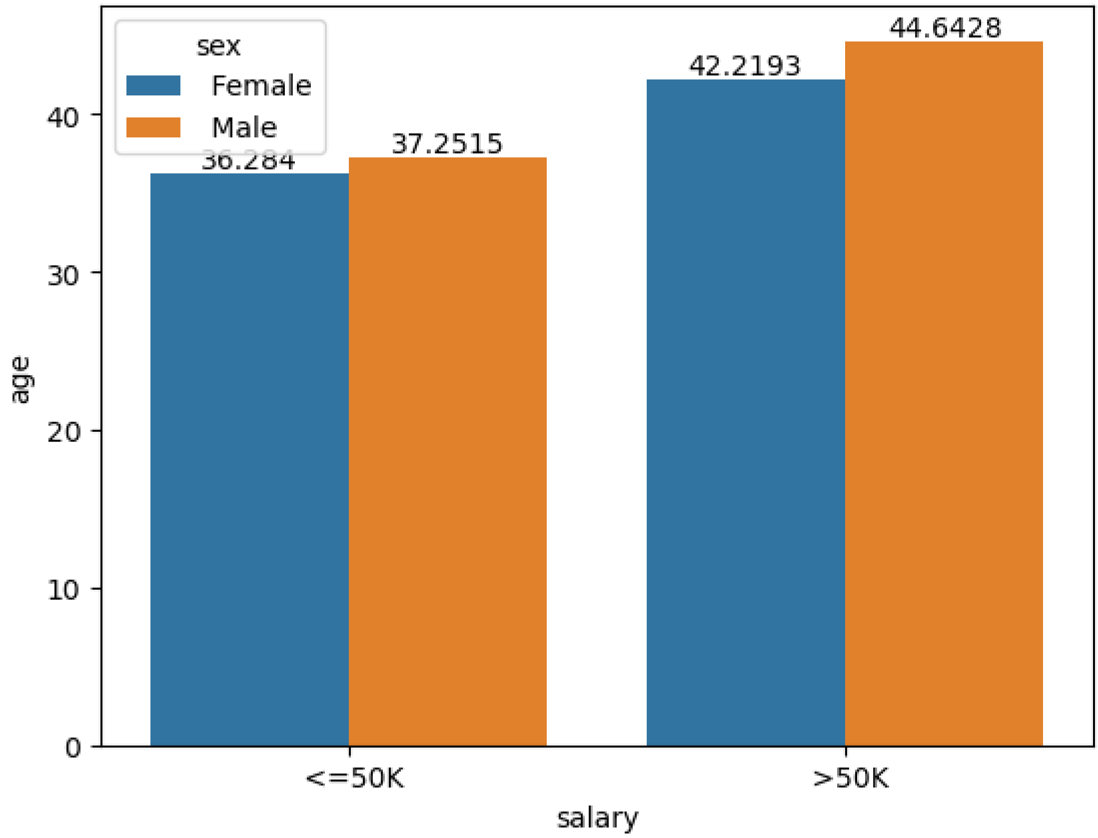
```
age = df.groupby(["salary", "sex"])[["age"]].mean().reset_index()
age
```

Out[39]:

	salary	sex	age
0	<=50K	Female	36.283980
1	<=50K	Male	37.251508
2	>50K	Female	42.219333
3	>50K	Male	44.642800

In [40]:

```
fig, ax = plt.subplots()

ax = sns.barplot(data=age, x="salary", y="age", hue="sex")

for container in ax.containers:
    ax.bar_label(container);
```



2. What is the workclass percentages of Americans in high-level income group?

In [41]:

```
workclass_US = df[(df.salary == " >50K") & (df['native-country-summary'] == "US")].w
workclass_US
```

Out[41]:

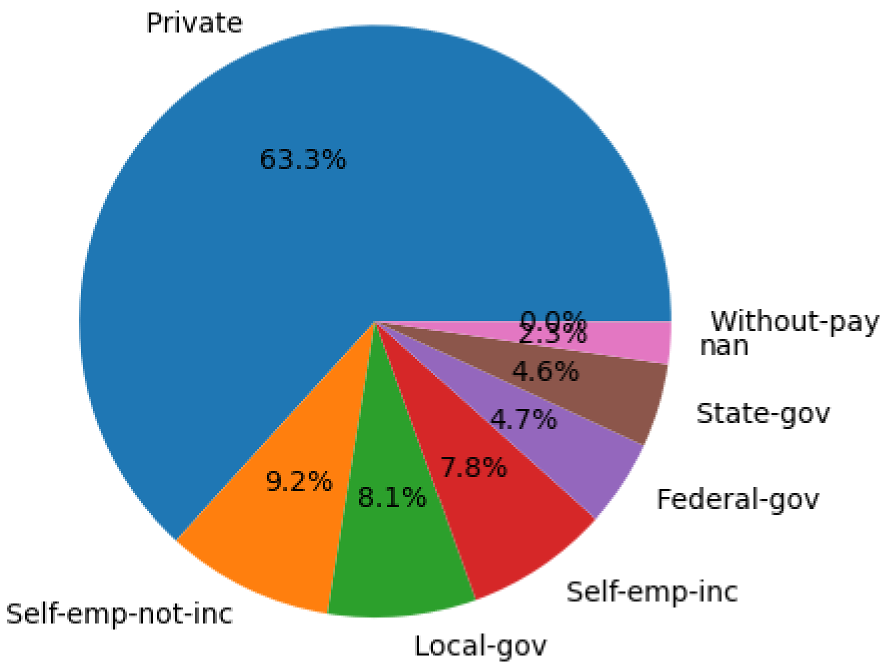
```
 Private             63.252899
 Self-emp-not-inc     9.165731
 Local-gov            8.099514
 Self-emp-inc         7.846988
 Federal-gov          4.732510
 State-gov            4.582866
NaN                   2.300786
 Without-pay          0.018706
Name: workclass, dtype: float64
```

In [42]:

```
plt.pie(workclass_US,
        labels=workclass_US.index,
        autopct="%.1f%%")
```

Out[42]:

```
([<matplotlib.patches.Wedge at 0x7fca7c190cd0>,
  <matplotlib.patches.Wedge at 0x7fca7c19e460>,
  <matplotlib.patches.Wedge at 0x7fca7c19eb80>,
  <matplotlib.patches.Wedge at 0x7fca7c1ab2e0>,
  <matplotlib.patches.Wedge at 0x7fca7c1aba00>,
  <matplotlib.patches.Wedge at 0x7fca7c1b7160>,
  <matplotlib.patches.Wedge at 0x7fca7c1b7880>,
  <matplotlib.patches.Wedge at 0x7fca7c1b7fa0>],
 [Text(-0.44486967643214986, 1.0060273211951822, ' Private'),
  Text(-0.4786027321318543, -0.9904238611806183, ' Self-emp-not-inc'),
  Text(0.10134392701527184, -1.0953216004704387, ' Local-gov'),
  Text(0.614951569620751, -0.9120496516204448, ' Self-emp-inc'),
  Text(0.9186813134701971, -0.6049997060170142, ' Federal-gov'),
  Text(1.054158569556669, -0.3142446661921846, ' State-gov'),
  Text(1.0970336136678516, -0.08072948954908042, 'nan'),
  Text(1.0999998101468191, -0.0006462793218057553, ' Without-pay')],
 [Text(-0.242656187144809, 0.5487421751973721, '63.3%'),
  Text(-0.2610560357082841, -0.5402311970076099, '9.2%'),
  Text(0.05527850564469373, -0.5974481457111483, '8.1%'),
  Text(0.335428128884046, -0.4974816281566O624, '7.8%'),
  Text(0.501098898256471, -0.3299998396456441, '4.7%'),
  Text(0.5749955833945467, -0.1714061815593734, '4.6%'),
  Text(0.5983819710915553, -0.044034267026771136, '2.3%'),
  Text(0.5999998964437195, -0.00035251599371223015, '0.0%')])
```



3. What is the occupation percentages of Americans who work as "Private" workclass in high-level income group?

In [43]:

```
Private_US = df[(df.workclass == " Private") & (df['native-country-summary'] == "US'
Private_US
```

Out[43]:

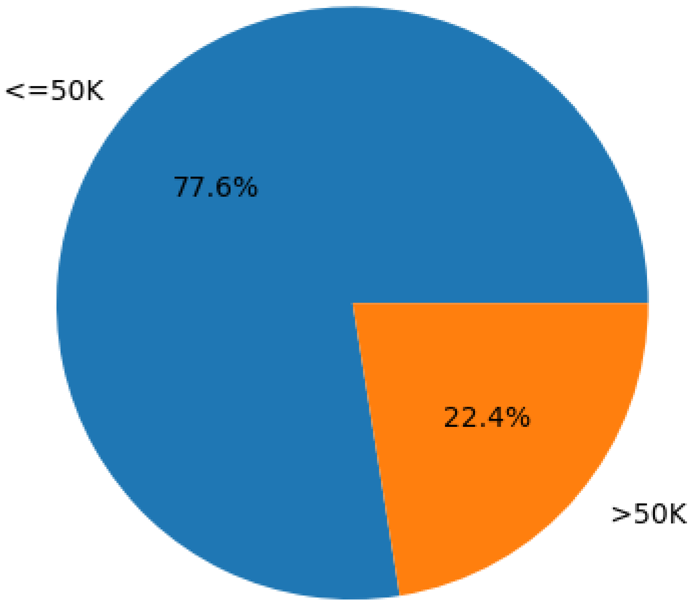
```
 <=50K    77.550207
 >50K     22.449793
Name: salary, dtype: float64
```

In [44]:

```
plt.pie(Private_US,
        labels=Private_US.index,
        autopct="%.1f%%")
```

Out[44]:

```
([<matplotlib.patches.Wedge at 0x7fca6beff130>,
  <matplotlib.patches.Wedge at 0x7fca6beff6d0>],
 [Text(-0.8375723677808423, 0.7130725970965325, ' <=50K'),
  Text(0.8375724011621788, -0.7130725578869395, ' >50K')],
 [Text(-0.45685765515318666, 0.3889486893253813, '77.6%'),
  Text(0.4568576733611883, -0.38894866793833055, '22.4%')])
```



4. What is the education level percentages of Asian-Pac-Islander race group in high-level income group?

In [45]:

```
Asian_Pac_Islander = df[(df.salary == " >50K") & (df.race == " Asian-Pac-Islander")]
Asian_Pac_Islander
```

Out[45]:

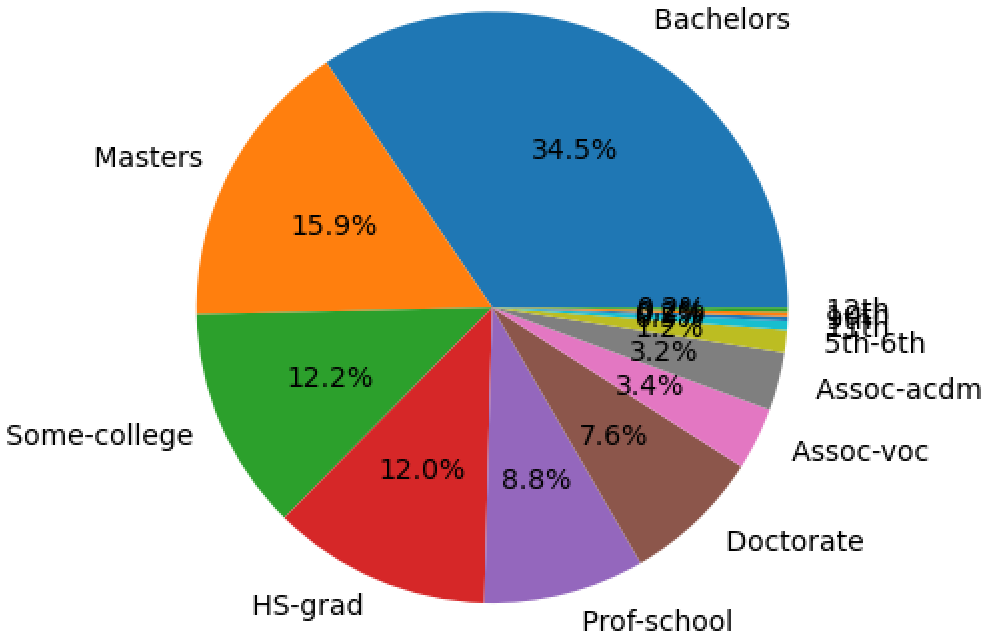
```
 Bachelors       34.474328
 Masters         15.892421
 Some-college    12.224939
 HS-grad         11.980440
 Prof-school      8.801956
 Doctorate        7.579462
 Assoc-voc        3.422983
 Assoc-acdm       3.178484
 5th-6th          1.222494
 11th             0.488998
 9th              0.244499
 10th             0.244499
 12th             0.244499
Name: education, dtype: float64
```

In [46]:

```
plt.pie(Asian_Pac_Islander,
        labels=Asian_Pac_Islander.index,
        autopct="%.1f%%")
```

Out[46]:

```
([<matplotlib.patches.Wedge at 0x7fca7bdd4b20>,
  <matplotlib.patches.Wedge at 0x7fca7bde4280>,
  <matplotlib.patches.Wedge at 0x7fca7bde49a0>,
  <matplotlib.patches.Wedge at 0x7fca7bdf0100>,
  <matplotlib.patches.Wedge at 0x7fca7bdf0820>,
  <matplotlib.patches.Wedge at 0x7fca7bdf0f40>,
  <matplotlib.patches.Wedge at 0x7fca7bdff6a0>,
  <matplotlib.patches.Wedge at 0x7fca7bdffdc0>,
  <matplotlib.patches.Wedge at 0x7fca7be09520>,
  <matplotlib.patches.Wedge at 0x7fca7be09c40>,
  <matplotlib.patches.Wedge at 0x7fca7bdd4af0>,
  <matplotlib.patches.Wedge at 0x7fca7be17a90>,
  <matplotlib.patches.Wedge at 0x7fca7be231f0>],
 [Text(0.5155067563608526, 0.9717267023944092, ' Bachelors'),
  Text(-0.977601544927056, 0.5042769272495357, ' Masters'),
  Text(-1.0100997863645464, -0.43554382280810483, ' Some-college'),
  Text(-0.4316617827167408, -1.0117648468600822, ' HS-grad'),
  Text(0.271757698930754, -1.0659023187290015, ' Prof-school'),
  Text(0.761214904264689, -0.7940729623437008, ' Doctorate'),
  Text(0.9852326537749447, -0.48919997744846644, ' Assoc-voc'),
  Text(1.0648505557794055, -0.27585012933890657, ' Assoc-acdm'),
  Text(1.0927067541798394, -0.12645927949249103, ' 5th-6th'),
  Text(1.0979238208137845, -0.06755208131257812, ' 11th'),
  Text(1.099188830529107, -0.04223641604177739, ' 9th'),
  Text(1.0997079518931416, -0.02534601631404266, ' 10th'),
  Text(1.0999675466430234, -0.008449635029308173, ' 12th')],
 [Text(0.28118550346955595, 0.5300327467605868, '34.5%'),
  Text(-0.5332372063238486, 0.2750601421361103, '15.9%'),
  Text(-0.5509635198352071, -0.23756935789532987, '12.2%'),
  Text(-0.2354518814818586, -0.5518717346509538, '12.0%'),
  Text(0.14823147214404764, -0.5814012647612735, '8.8%'),
  Text(0.4152081295989212, -0.4331307067329277, '7.6%'),
  Text(0.537399629331788, -0.26683635133552713, '3.4%'),
  Text(0.5808275758796756, -0.15046370691213082, '3.2%'),
  Text(0.596021865916276, -0.068977788814086, '1.2%'),
  Text(0.5988675386257005, -0.036846589806860786, '0.5%'),
  Text(0.5995575439249674, -0.02303804511369675 7, '0.2%'),
  Text(0.5998407010326227, -0.013825099807659632, '0.2%'),
  Text(0.5999822981689218, -0.004608891834168093 5, '0.2%')])
```



5. What is the occupation percentages of Asian-Pac-Islander race group who has a Bachelors degree in high-level income group?

In [47]:

```
Asian_Pac_Islander = df[(df.salary == " >50K") & (df.race == " Asian-Pac-Islander")
                        .occupation.value_counts(dropna = False, normalize = True) * 10
Asian_Pac_Islander
```

Out[47]:

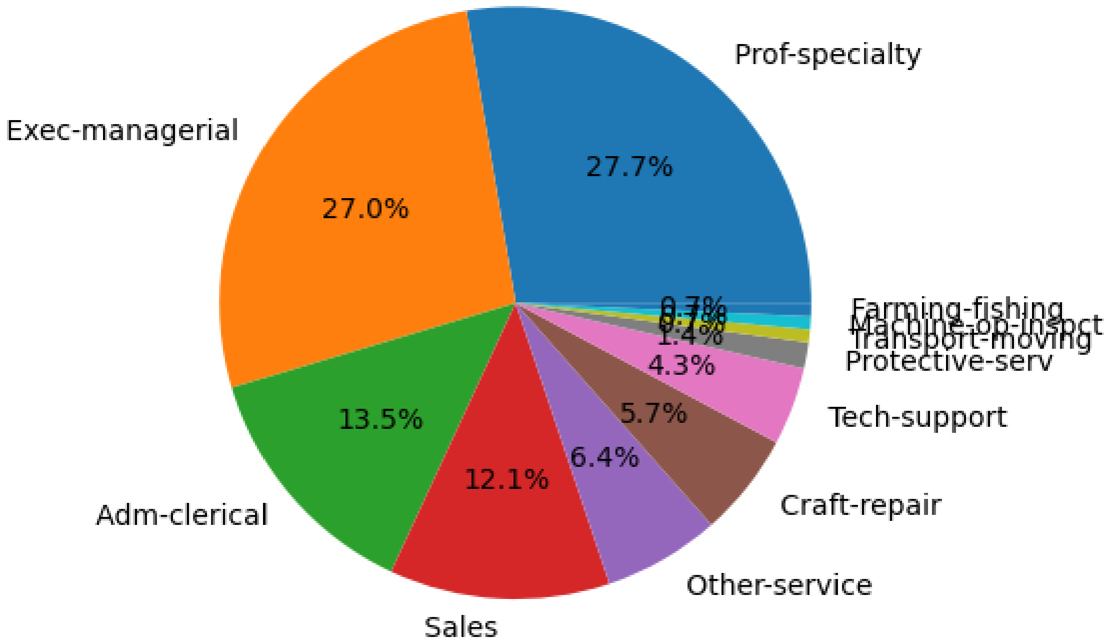
```
 Prof-specialty       27.659574
 Exec-managerial      26.950355
 Adm-clerical         13.475177
 Sales                12.056738
 Other-service         6.382979
 Craft-repair          5.673759
 Tech-support          4.255319
 Protective-serv       1.418440
 Transport-moving      0.709220
 Machine-op-inspct     0.709220
 Farming-fishing       0.709220
Name: occupation, dtype: float64
```

In [48]:

```
plt.pie(Asian_Pac_Islander,
        labels=Asian_Pac_Islander.index,
        autopct="%.1f%%")
```

Out[48]:

```
([<matplotlib.patches.Wedge at 0x7fca6c129610>,
  <matplotlib.patches.Wedge at 0x7fca6c129e20>,
  <matplotlib.patches.Wedge at 0x7fca6c135580>,
  <matplotlib.patches.Wedge at 0x7fca6beffe20>,
  <matplotlib.patches.Wedge at 0x7fca6c1431c0>,
  <matplotlib.patches.Wedge at 0x7fca6c1438b0>,
  <matplotlib.patches.Wedge at 0x7fca6c143fd0>,
  <matplotlib.patches.Wedge at 0x7fca6c150730>,
  <matplotlib.patches.Wedge at 0x7fca6c150e50>,
  <matplotlib.patches.Wedge at 0x7fca6c15c5b0>,
  <matplotlib.patches.Wedge at 0x7fca6c1295e0>],
 [Text(0.7101907237296053, 0.8400173426355075, ' Prof-specialty'),
  Text(-0.9337175819397948, 0.5815251303052198, ' Exec-managerial'),
  Text(-0.8320538646809795, -0.7195042503484231, ' Adm-clerical'),
  Text(-0.06124090394507291, -1.0982939277279058, ' Sales'),
  Text(0.5499996358014774, -0.9526281544329048, ' Other-service'),
  Text(0.8632790589894519, -0.681725213198314, ' Craft-repair'),
  Text(1.0308390664726683, -0.3838890712611629, ' Tech-support'),
  Text(1.0825716570558654, -0.19503488749277142, ' Protective-serv'),
  Text(1.0931810366955135, -0.12229154103748459, ' Transport-moving'),
  Text(1.097543534825798, -0.07347237005904077, ' Machine-op-inspct'),
  Text(1.099726961999124, -0.024507326500000626, ' Farming-fishing')],
 [Text(0.3873767583979665, 0.45819127780118585, '27.7%'),
  Text(-0.509300499239888, 0.317195525621029, '27.0%'),
  Text(-0.45384756255326153, -0.3924568638264126, '13.5%'),
  Text(-0.03340412942458522, -0.5990694151243122, '12.1%'),
  Text(0.29999980134626036, -0.5196153569634026, '6.4%'),
  Text(0.47087948672151914, -0.37185011628998943, '5.7%'),
  Text(0.5622758544396371, -0.20939403886972519, '4.3%'),
  Text(0.5904936311213811, -0.10638266590514804, '1.4%'),
  Text(0.59628056547028, -0.06670447692953704, '0.7%'),
  Text(0.5986601099049806, -0.04007583821402223, '0.7%'),
  Text(0.5998510701813402, -0.013367632636367052, '0.7%')])
```



Conclusion

- Data suggest High Education High Salary.
- Experiece brings high salary.
- if you are at high job role such as Exec-manager, you will get paid more.
- We have not seen any gender biases on salary.

Feature Engineering

Droping Uncessary Columns

In [49]:

```
df.drop(columns = ['marital-status','relationship','native-country'],inplace=True)
```

Handling missing values

In [50]:

```
df.isna().sum()
```

Out[50]:

```
age                          0
workclass                 2799
fnlwgt                       0
education                    0
education-num                0
occupation                2809
race                         0
sex                          0
capital-gain                 0
capital-loss                 0
hours-per-week               0
salary                       0
native-country-summary       0
marital-status-summary       0
dtype: int64
```

Since both workclass and occupation are categorical feature, we'll try to impute it with mode.

In [51]:

```
df = df.fillna(df.mode().iloc(0))
```

In [52]:

```
df.isna().sum()
```

Out[52]:

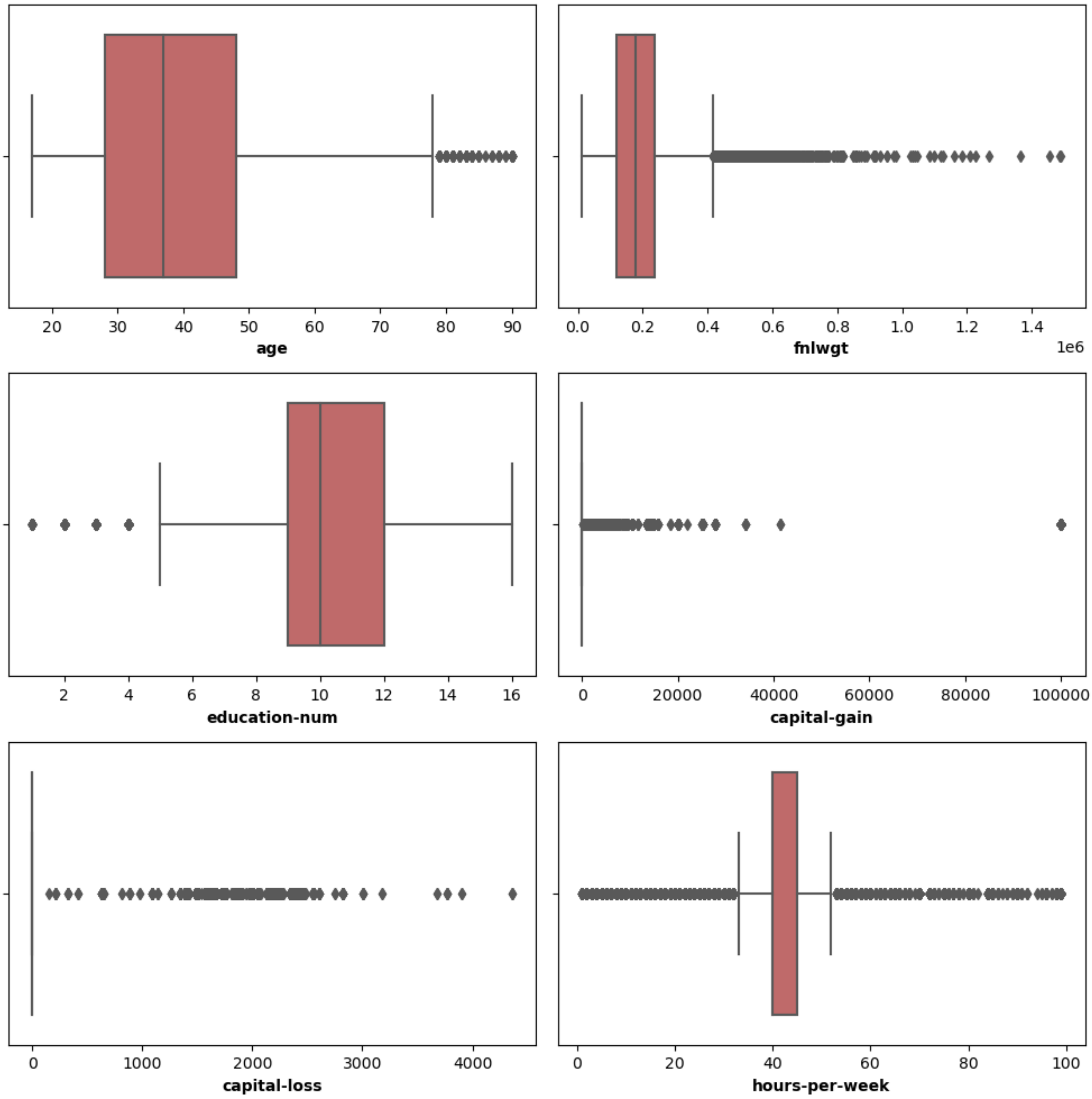
```
age                       0
workclass                 0
fnlwgt                    0
education                 0
education-num             0
occupation                0
race                      0
sex                       0
capital-gain              0
capital-loss              0
hours-per-week            0
salary                    0
native-country-summary    0
marital-status-summary    0
dtype: int64
```

Handling Outliers

Interquartile Range Method(IQR) method

In [53]:

```
plt.figure(figsize=(10, 10))
for i, col in enumerate(numeric_features):
    plt.subplot(3, 2, i+1)
    sns.boxplot(x=df[col], color='indianred')
    plt.xlabel(col, weight='bold')
    plt.tight_layout()
```



In [54]:

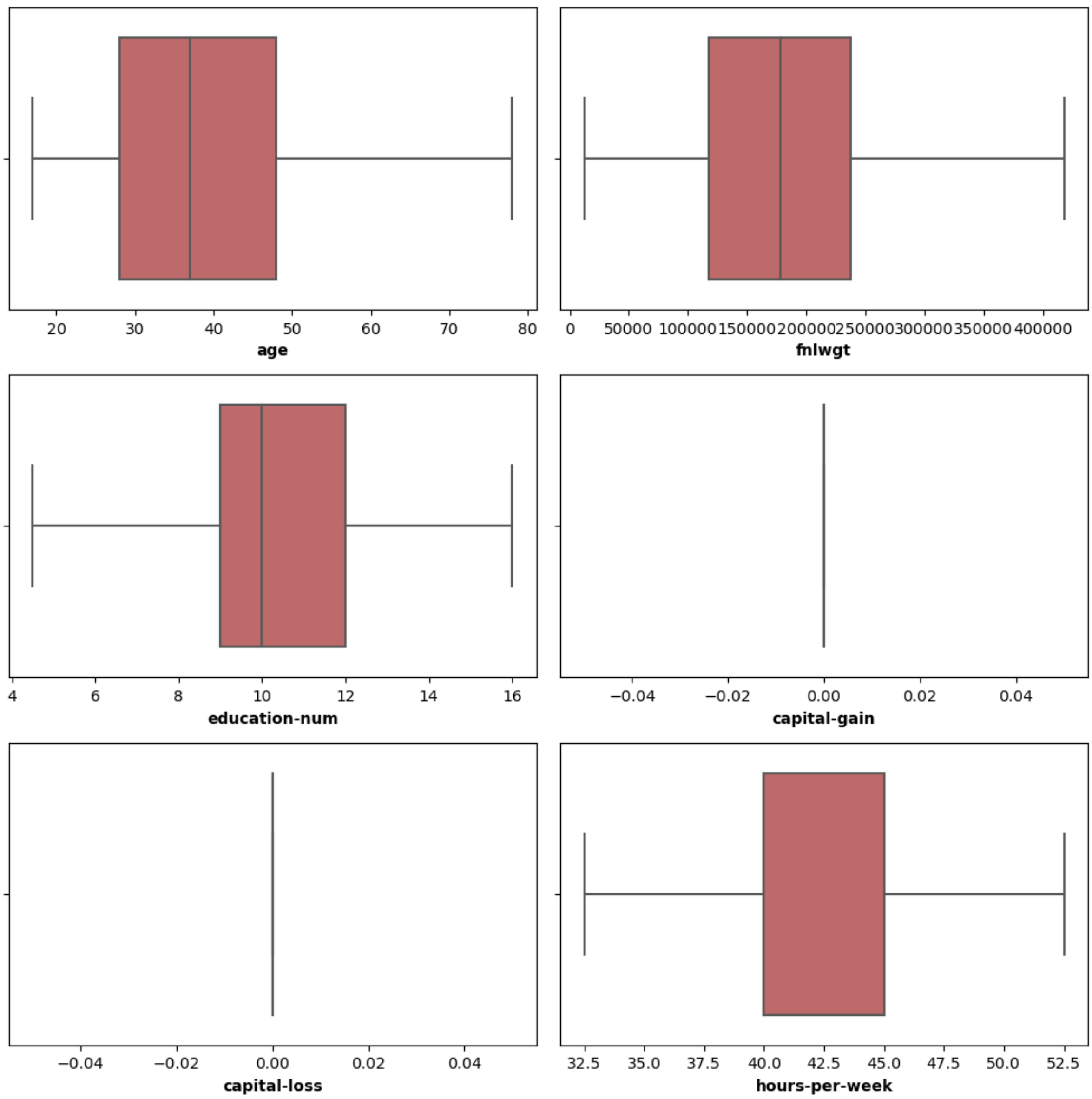
```
df1 = df.copy()
def remove_outliers_IQR(col):
    # Finding the IQR
    percentile25 = df1[col].quantile(0.25)
    percentile75 = df1[col].quantile(0.75)
    iqr = percentile75 - percentile25
    upper_limit = percentile75 + 1.5 * iqr
    lower_limit = percentile25 - 1.5 * iqr
    df1[col] = np.where(df1[col]>upper_limit, upper_limit, np.where(df1[col]<lower_l
    return df1[df1[col] > upper_limit]
```

In [55]:

```
for col in numeric_features:
    remove_outliers_IQR(col)
```

In [56]:

```
plt.figure(figsize=(10, 10))
for i, col in enumerate(numeric_features):
    plt.subplot(3, 2, i+1)
    sns.boxplot(x=df1[col], color='indianred')
    plt.xlabel(col, weight='bold')
    plt.tight_layout()
```



In [57]:

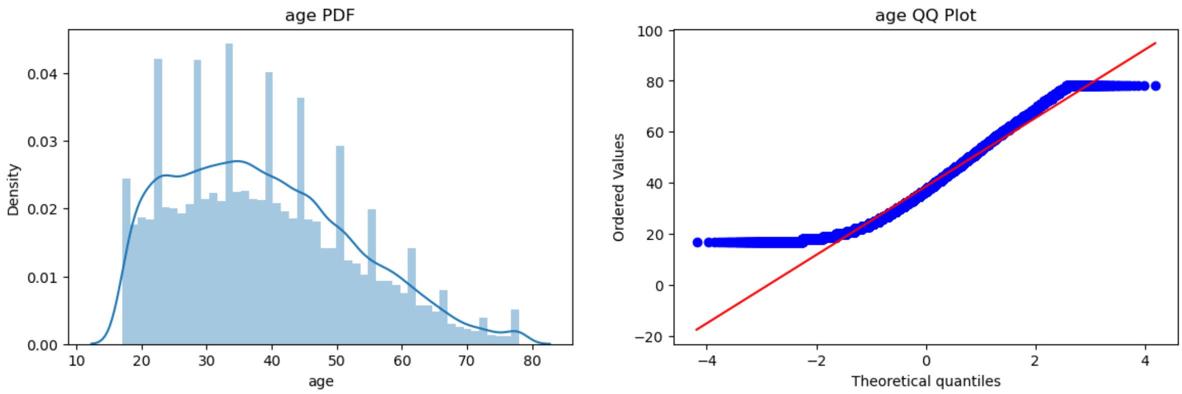
```
df = df1.copy()
```

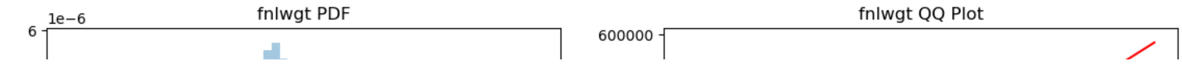
Checking normal distribution

QQ-plot

In [58]:

```
from scipy import stats
def plot_qq_plot(column):
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.distplot(df[column])
    plt.title("{} PDF".format(column))
    plt.subplot(122)
    stats.probplot(df[column], dist="norm", plot=plt)
    plt.title('{} QQ Plot'.format(column))
    plt.show()
for col in numeric_features:
    plot_qq_plot(col)
```



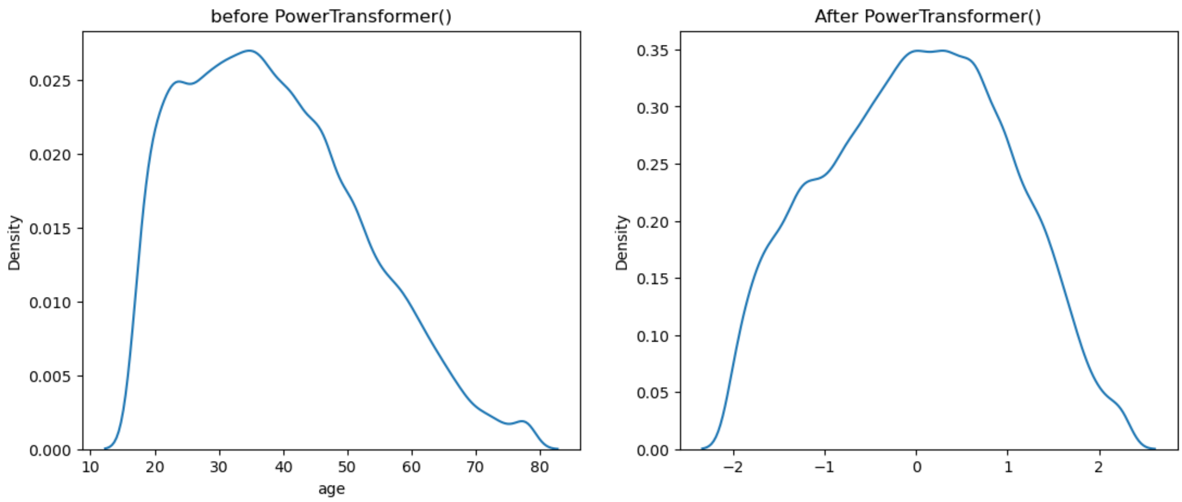


Report

- Some features are not normally distributed.

In [59]:

```
from sklearn.preprocessing import PowerTransformer
def power_plots(df,var,t):
    plt.figure(figsize=(13,5))
    plt.subplot(121)
    sns.kdeplot(df[var])
    plt.title('before ' + str(t))
    plt.subplot(122)
    p1 = t.fit_transform(df[[var]]).flatten()
    sns.kdeplot(p1)
    plt.title('After ' + str(t))
for col in numeric_features:
    power_plots(df,col,PowerTransformer())
```



In [60]:

```
data=pd.get_dummies(df,drop_first=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48813 entries, 0 to 48841
Data columns (total 51 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   age                           48813 non-null  float64
 1   fnlwgt                        48813 non-null  float64
 2   education-num                 48813 non-null  float64
 3   capital-gain                  48813 non-null  float64
 4   capital-loss                  48813 non-null  float64
 5   hours-per-week                48813 non-null  float64
 6   workclass_ Federal-gov        48813 non-null  uint8  
 7   workclass_ Local-gov          48813 non-null  uint8  
 8   workclass_ Never-worked       48813 non-null  uint8  
 9   workclass_ Private            48813 non-null  uint8  
 10  workclass_ Self-emp-inc       48813 non-null  uint8  
 11  workclass_ Self-emp-not-inc   48813 non-null  uint8  
 12  workclass_ State-gov          48813 non-null  uint8  
 13  workclass_ Without-pay        48813 non-null  uint8  
 14  education_ 11th               48813 non-null  uint8  
 15  education_ 12th               48813 non-null  uint8  
 16  education_ 1st-4th            48813 non-null  uint8  
 17  education_ 5th-6th            48813 non-null  uint8  
 18  education_ 7th-8th            48813 non-null  uint8  
 19  education_ 9th                48813 non-null  uint8  
 20  education_ Assoc-acdm         48813 non-null  uint8  
 21  education_ Assoc-voc          48813 non-null  uint8  
 22  education_ Bachelors          48813 non-null  uint8  
 23  education_ Doctorate          48813 non-null  uint8  
 24  education_ HS-grad            48813 non-null  uint8  
 25  education_ Masters            48813 non-null  uint8  
 26  education_ Preschool          48813 non-null  uint8  
 27  education_ Prof-school        48813 non-null  uint8  
 28  education_ Some-college       48813 non-null  uint8  
 29  occupation_ Adm-clerical      48813 non-null  uint8  
 30  occupation_ Armed-Forces      48813 non-null  uint8  
 31  occupation_ Craft-repair      48813 non-null  uint8  
 32  occupation_ Exec-managerial   48813 non-null  uint8  
 33  occupation_ Farming-fishing   48813 non-null  uint8  
 34  occupation_ Handlers-cleaners 48813 non-null  uint8  
 35  occupation_ Machine-op-inspct 48813 non-null  uint8  
 36  occupation_ Other-service     48813 non-null  uint8  
 37  occupation_ Priv-house-serv   48813 non-null  uint8  
 38  occupation_ Prof-specialty    48813 non-null  uint8  
 39  occupation_ Protective-serv   48813 non-null  uint8  
 40  occupation_ Sales             48813 non-null  uint8  
 41  occupation_ Tech-support      48813 non-null  uint8  
 42  occupation_ Transport-moving  48813 non-null  uint8  
 43  race_ Asian-Pac-Islander      48813 non-null  uint8  
 44  race_ Black                   48813 non-null  uint8  
 45  race_ Other                   48813 non-null  uint8  
 46  race_ White                   48813 non-null  uint8  
 47  sex_ Male                     48813 non-null  uint8  
 48  salary_ >50K                  48813 non-null  uint8  
 49  native-country-summary_US     48813 non-null  uint8  
 50  marital-status-summary_Single 48813 non-null  uint8  
```

```
dtypes: float64(6), uint8(45)
memory usage: 5.7 MB
```

In [61]:

```
X = data.drop('salary_ >50K',axis=1)
y = data['salary_ >50K']
```

Handling Imbalance Data

In [62]:

```
from imblearn.combine import SMOTETomek

# Resampling the minority class. The strategy can be changed as required.
smt = SMOTETomek(random_state=42,sampling_strategy='minority',n_jobs=-1)
# Fit the model to generate the data.
X_res, y_res = smt.fit_resample(X, y)
```

Model Training

In [63]:

```
def evaluate_clf(true, predicted):
    '''
    This function takes in true values and predicted values
    Returns: Accuracy, F1-Score, Precision, Recall, Roc-auc Score
    '''
    acc = accuracy_score(true, predicted) # Calculate Accuracy
    f1 = f1_score(true, predicted) # Calculate F1-score
    precision = precision_score(true, predicted) # Calculate Precision
    recall = recall_score(true, predicted)  # Calculate Recall
    roc_auc = roc_auc_score(true, predicted) #Calculate Roc
    return acc, f1 , precision, recall, roc_auc
```

In [64]:

```
# Create a function which can evaluate models and return a report
def evaluate_models(X, y, models):
    '''
    This function takes in X and y and models dictionary as input
    It splits the data into Train Test split
    Iterates through the given model dictionary and evaluates the metrics
    Returns: Dataframe which contains report of all models metrics with cost
    '''
    # separate dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_sta

    cost_list=[]
    models_list = []
    accuracy_list = []

    for i in range(len(list(models))):
        model = list(models.values())[i]
        model.fit(X_train, y_train) # Train model

        # Make predictions
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(X_test)

        # Training set performance
        model_train_accuracy, model_train_f1,model_train_precision,\
        model_train_recall,model_train_rocauc_score=evaluate_clf(y_train ,y_train_pr


        # Test set performance
        model_test_accuracy,model_test_f1,model_test_precision,\
        model_test_recall,model_test_rocauc_score=evaluate_clf(y_test, y_test_pred)

        print(list(models.keys())[i])
        models_list.append(list(models.keys())[i])
        print('Model performance for Training set')
        print("- Accuracy: {:.4f}".format(model_train_accuracy))
        print('- F1 score: {:.4f}'.format(model_train_f1))
        print('- Precision: {:.4f}'.format(model_train_precision))
        print('- Recall: {:.4f}'.format(model_train_recall))
        print('- Roc Auc Score: {:.4f}'.format(model_train_rocauc_score))

        print('----------------------------------')

        print('Model performance for Test set')
        print('- Accuracy: {:.4f}'.format(model_test_accuracy))
        print('- F1 score: {:.4f}'.format(model_test_f1))
        print('- Precision: {:.4f}'.format(model_test_precision))
        print('- Recall: {:.4f}'.format(model_test_recall))
        print('- Roc Auc Score: {:.4f}'.format(model_test_rocauc_score))
        print('='*35)
        print('\n')
```

In [65]:

```
# Dictionary which contains models for experiment
models = {
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "Logistic Regression": LogisticRegression(),
     "K-Neighbors Classifier": KNeighborsClassifier(),
    "XGBClassifier": XGBClassifier(),
     "CatBoosting Classifier": CatBoostClassifier(verbose=False),
    "AdaBoost Classifier": AdaBoostClassifier()
}
```

In [66]:

```
evaluate_models(X_res, y_res, models)
```

```
-----------------------------------


AdaBoost Classifier
Model performance for Training set
- Accuracy: 0.8925
- F1 score: 0.8933
- Precision: 0.8860
- Recall: 0.9007
- Roc Auc Score: 0.8925
----------------------------------
Model performance for Test set
- Accuracy: 0.8902
- F1 score: 0.8920
- Precision: 0.8811
- Recall: 0.9032
- Roc Auc Score: 0.8901
===================================
```

In [67]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.20, ra
```

In [68]:

```
xgb_model = XGBClassifier(
    objective= 'binary:logistic',
    nthread=4,
    seed=42
)

#brute force scan for all parameters, here are the tricks
#usually max_depth is 6,7,8
#learning rate is around 0.05, but small changes may make big diff
#tuning min_child_weight subsample colsample_bytree can have
#much fun of fighting against overfit
#n_estimators is how many round of boosting
#finally, ensemble xgboost with multiple seeds may reduce variance
parameters = {
    'max_depth': [6],
    'n_estimators': [5000],
    'learning_rate': [0.05]
}
skf=StratifiedKFold(n_splits=5,  shuffle=True)
clf = GridSearchCV(xgb_model, parameters, n_jobs=10,
                   cv=skf.get_n_splits(y_train),
                   scoring='roc_auc',
                   verbose=True, refit=True)

clf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

Out[68]:

```
▸ GridSearchCV
▸ estimator: XGBClassifier
  ▸ XGBClassifier
```

In [69]:

```
clf.best_params_
```

Out[69]:

```
{'learning_rate': 0.05, 'max_depth': 6, 'n_estimators': 5000}
```

In [70]:

```
final_model=clf.best_estimator_
y_pred = final_model.predict(X_test)
```

In [76]:

```
model_test_accuracy,model_test_f1,model_test_precision,model_test_recall,model_test_
```

In [77]:

```
print('Model performance for Test set')
print('- Accuracy: {:.4f}'.format(model_test_accuracy))
print('- F1 score: {:.4f}'.format(model_test_f1))
print('- Precision: {:.4f}'.format(model_test_precision))
print('- Recall: {:.4f}'.format(model_test_recall))
print('- Roc Auc Score: {:.4f}'.format(model_test_rocauc_score))
```

```
Model performance for Test set
- Accuracy: 0.9010
- F1 score: 0.8999
- Precision: 0.9086
- Recall: 0.8913
- Roc Auc Score: 0.9010
```

In [73]:

```
#Saving the model
import pickle
with open('../artifacts/model.pkl', 'wb') as files:
    pickle.dump(final_model, files)
```

In [ ]: