Importing Libraries and dataset

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

```
df=pd.read_csv('/content/census-income .csv')
df
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	nati coun
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	Unit Sta
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	Unit Sta
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	Unit Sta
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	Unit Sta
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cu
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	Unit Sta
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	Unit Sta
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	Unit Sta
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	Unit Sta
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	Unit Sta

32561 rows × 15 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

EDA

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

```
np.int64(0)
```

```
df.duplicated().sum()
```

```
np.int64(24)
```

```
df.drop_duplicates(inplace=True)
df.duplicated().sum()
```

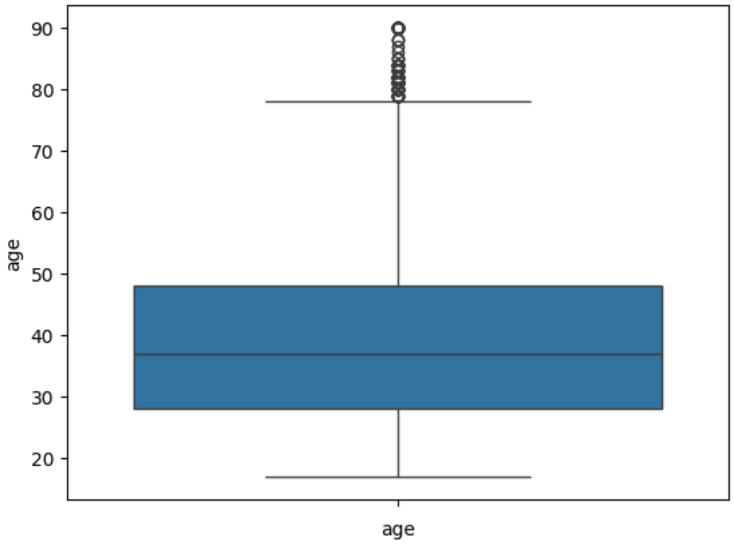
```
np.int64(0)
```

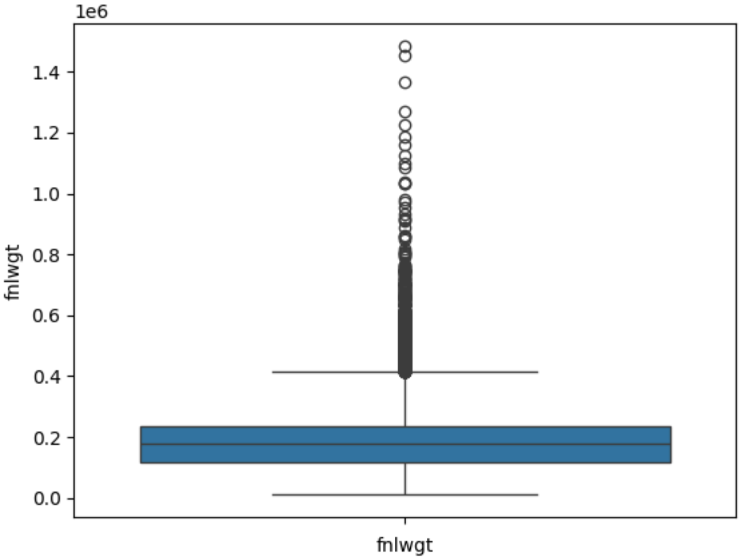
```
df.info()
```

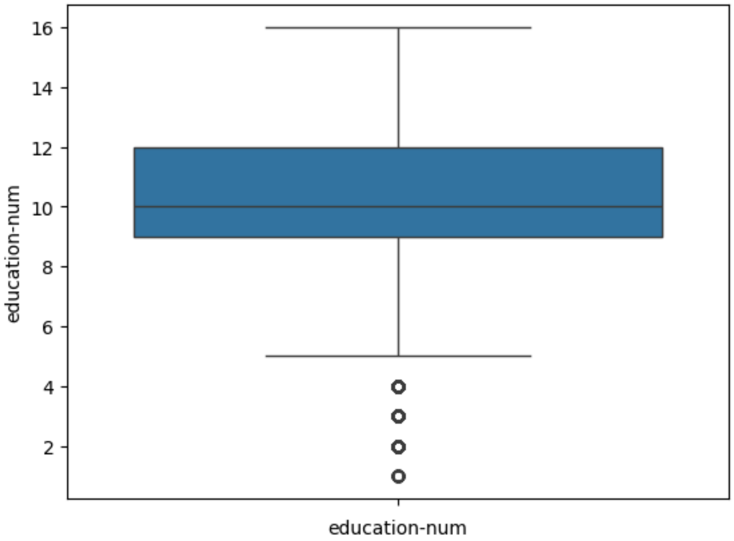
```
<class 'pandas.core.frame.DataFrame'>
Index: 32537 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32537 non-null  int64 
 1   workclass       32537 non-null  object
 2   fnlwgt          32537 non-null  int64 
 3   education       32537 non-null  object
 4   education-num   32537 non-null  int64 
 5   marital-status  32537 non-null  object
 6   occupation      32537 non-null  object
 7   relationship    32537 non-null  object
 8   race            32537 non-null  object
 9   sex             32537 non-null  object
 10  capital-gain    32537 non-null  int64 
 11  capital-loss    32537 non-null  int64 
 12  hours-per-week  32537 non-null  int64 
 13  native-country  32537 non-null  object
 14  annual_income   32537 non-null  object
dtypes: int64(6), object(9)
memory usage: 4.0+ MB
```

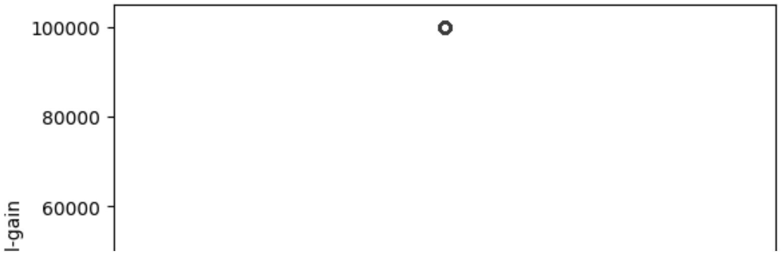
Outliers

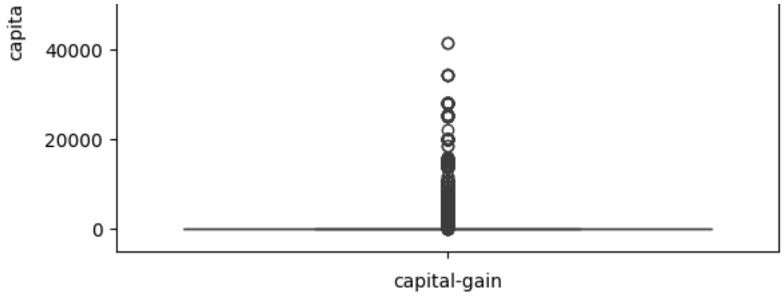
```
for x in df.columns:
  if df[x].dtypes!='object':
    sns.boxplot(df[x])
    plt.xlabel(x)
    plt.show()
```

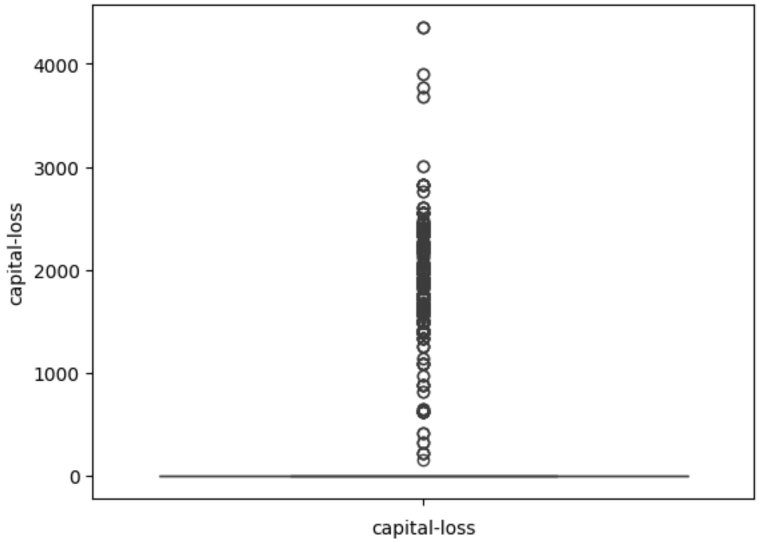

90
80
70
60
50
40
30
20
age
age

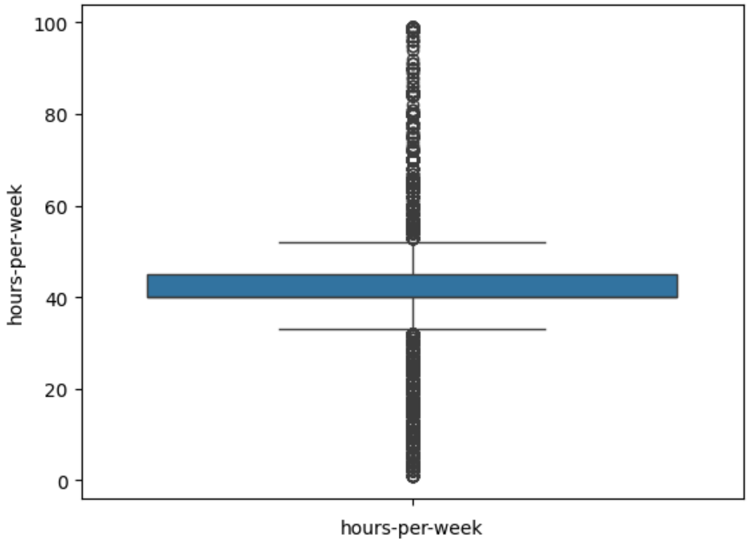

1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
fnlwgt
fnlwgt


16
14
12
10
8
6
4
2
education-num
education-num


100000
80000
60000
l-gain

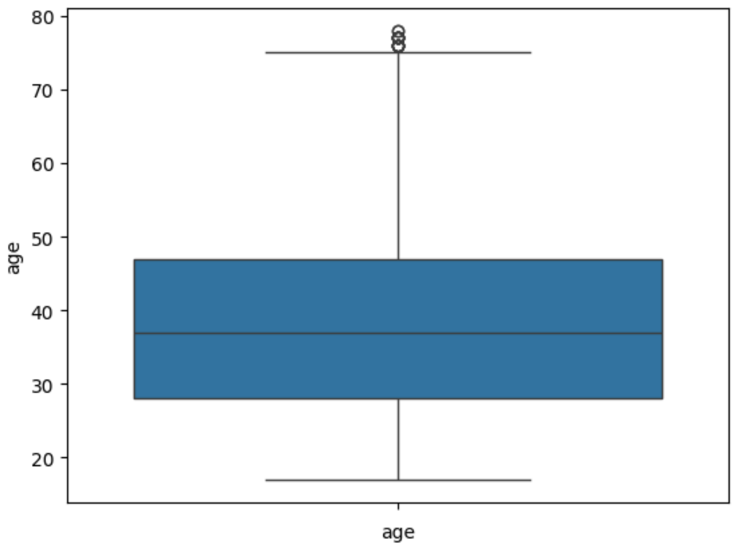

capita
40000
20000
0
capital-gain

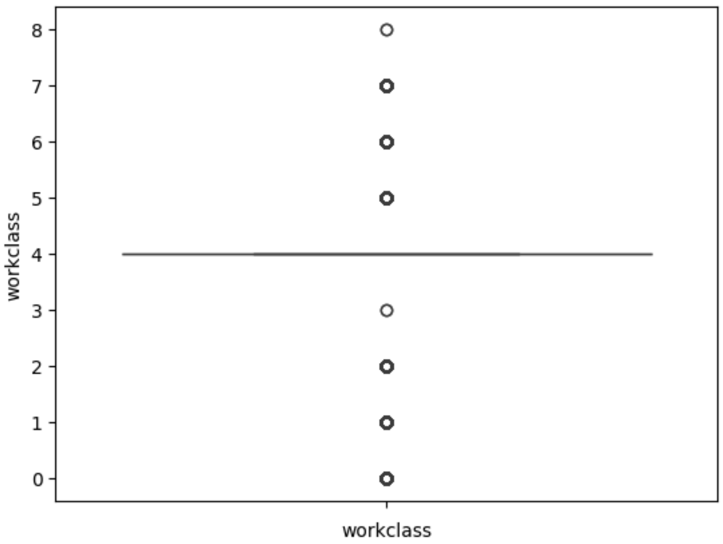

4000
3000
2000
1000
0
capital-loss
capital-loss

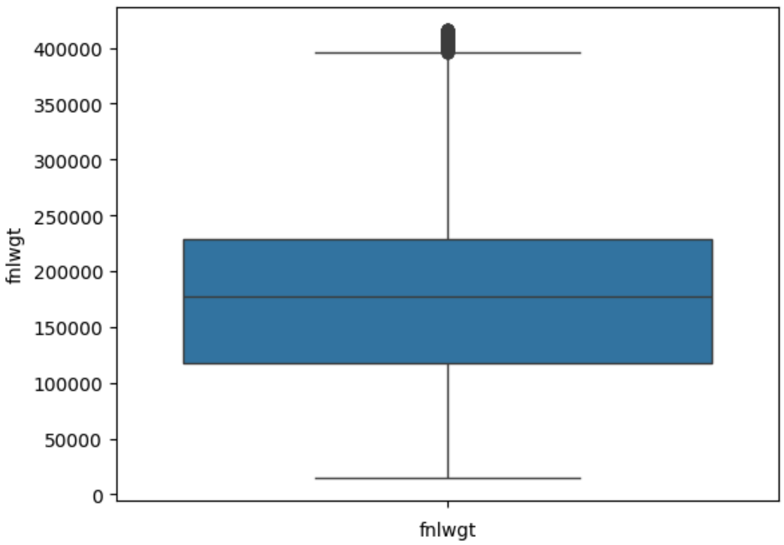

100
80
60
40
20
0
hours-per-week
hours-per-week

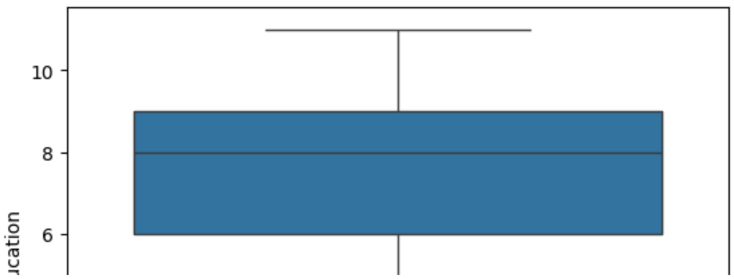
```
for x in df.columns:
  if df[x].dtypes!='object':
    q1=df[x].quantile(0.25)
    q3=df[x].quantile(0.75)
    IQR=q3-q1
    upper=q3+(1.5*IQR)
    lower=q1-(1.5*IQR)
    df=df[(df[x]>=lower) & (df[x]<=upper)]
```

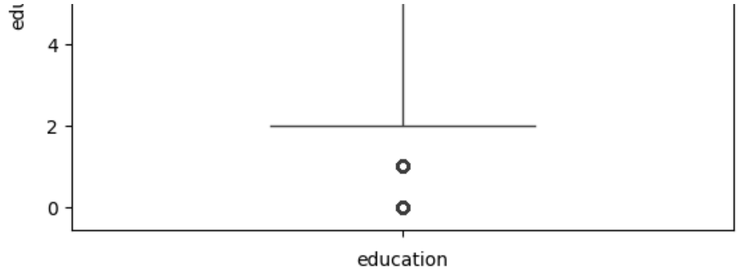
```
for x in df.columns:
  if df[x].dtypes!='object':
    sns.boxplot(df[x])
    plt.xlabel(x)
    plt.show()
```

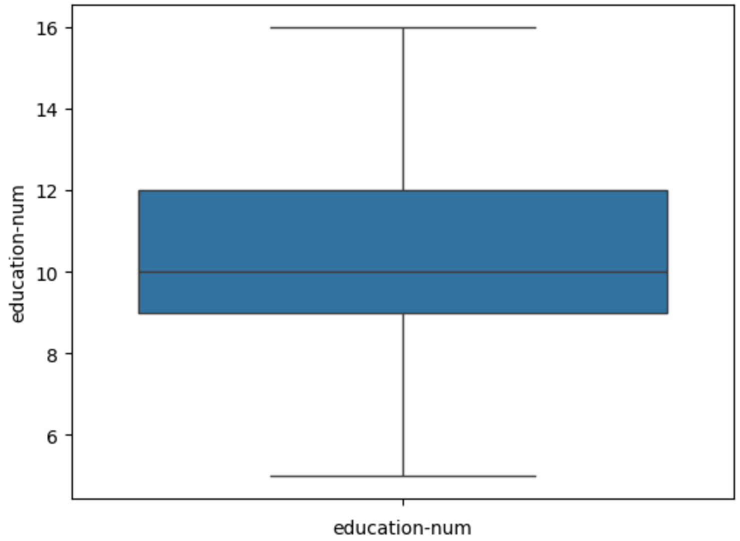

80
70
60
50
40
30
20
age
age

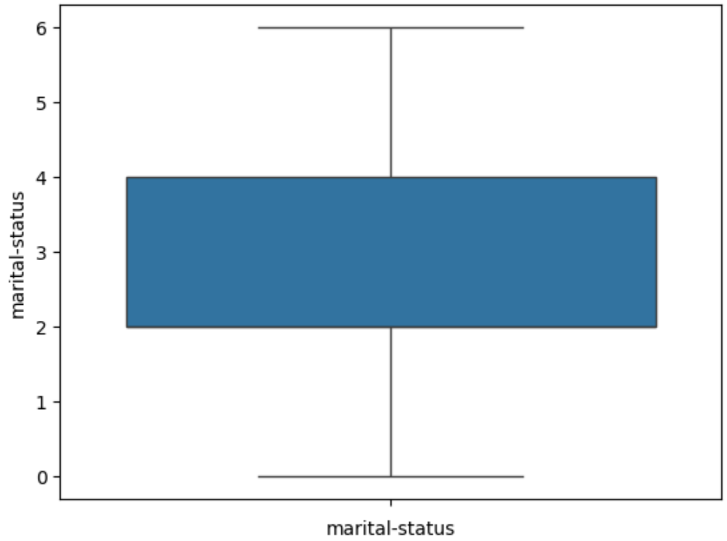

8
7
6
5
4
3
2
1
0
workclass
workclass

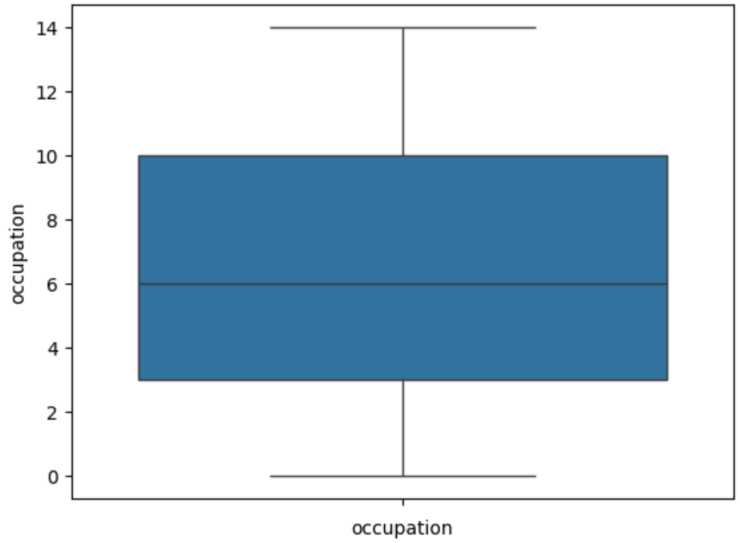

400000
350000
300000
250000
200000
150000
100000
50000
0
fnlwgt
fnlwgt


10
8
6
ucation


edu
4
2
0
education


16
14
12
10
8
6
education-num
education-num


6
5
4
3
2
1
0
marital-status
marital-status


14
12
10
8
6
4
2
0
occupation
occupation

LabelEncoding

```
from sklearn.preprocessing import LabelEncoder
```

```
le=LabelEncoder()
for x in df.columns:
  if df[x].dtypes=='object':
    df[x]=le.fit_transform(df[x])
```

```
df
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
2	38	4	215646	8	9	0	6	1	4	1	0	0	40	38
3	53	4	234721	1	7	2	6	0	2	1	0	0	40	38
4	28	4	338409	6	13	2	10	5	2	0	0	0	40	5
5	37	4	284582	9	14	2	4	5	4	0	0	0	40	38
7	52	6	209642	8	9	2	4	0	4	1	0	0	45	38
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32554	53	4	321865	9	14	2	4	0	4	1	0	0	40	38
32555	22	4	310152	11	10	4	11	1	4	1	0	0	40	38
32556	27	4	257302	4	12	2	13	5	4	0	0	0	38	38
32557	40	4	154374	8	9	2	7	0	4	1	0	0	40	38
32558	58	4	151910	8	9	6	1	4	4	0	0	0	40	38

Model Building

```
x=df.drop('annual_income',axis=1)
y=df['annual_income']
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=23)
```

1.Logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

```
model1=LogisticRegression()
model1.fit(x_train,y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
▾ LogisticRegression
LogisticRegression()
```

```
y_pred=model1.predict(x_test)
```

```
accuracy_score(y_test,y_pred)*100
```

```
80.38957620426427
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[2953,   78],
       [ 667,  101]])
```

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

```
              precision    recall  f1-score   support

           0       0.82      0.97      0.89      3031
           1       0.56      0.13      0.21       768

    accuracy                           0.80      3799
   macro avg       0.69      0.55      0.55      3799
weighted avg       0.76      0.80      0.75      3799
```

2.Decision Trees

```
from sklearn.tree import DecisionTreeClassifier
```

```
#Brute Force to find the max_depth

max_depth=[1,2,3,4,5,6,7,8,9,10]
for i in max_depth:
  model2=DecisionTreeClassifier(max_depth=i)
  model2.fit(x_train,y_train)
  y_pred=model2.predict(x_test)
  acc=accuracy_score(y_test,y_pred)
  print(f"for the max depth {i} the accuracy score is: {acc}")
```

```
for the max depth 1 the accuracy score is: 0.7978415372466439
for the max depth 2 the accuracy score is: 0.8341668860226376
for the max depth 3 the accuracy score is: 0.8381152934982891
for the max depth 4 the accuracy score is: 0.8373256120031587
for the max depth 5 the accuracy score is: 0.8349565675177678
for the max depth 6 the accuracy score is: 0.8383785206633324
for the max depth 7 the accuracy score is: 0.8336404316925506
for the max depth 8 the accuracy score is: 0.8304817057120295
for the max depth 9 the accuracy score is: 0.8304817057120295
for the max depth 10 the accuracy score is: 0.8267965254014215
```

```
model2=DecisionTreeClassifier(max_depth=4)
model2.fit(x_train,y_train)
```

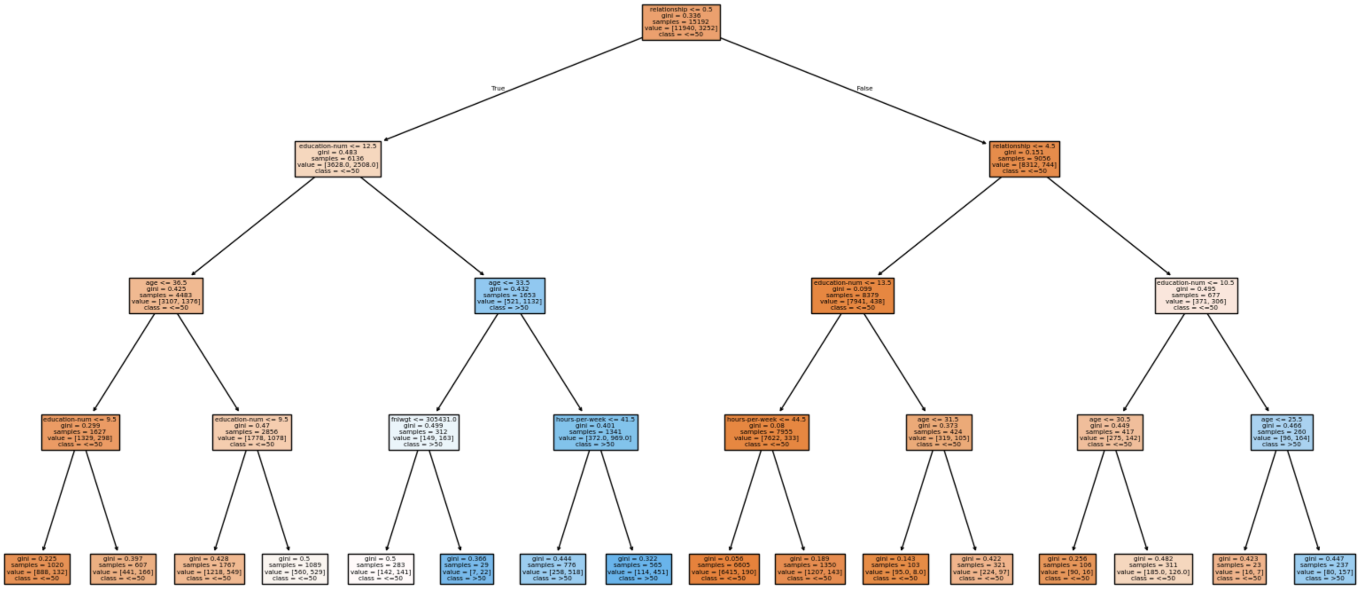
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4)
```

```
dt_y_pred=model2.predict(x_test)
```

```
accuracy_score(y_test,dt_y_pred)*100
```

```
83.73256120031587
```

```
from sklearn.tree import plot_tree
plt.figure(figsize=(20,10))
plot_tree(model2,filled=True,feature_names=x.columns,class_names=['<=50','>50'])
plt.show()
```



3.Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
base_model=RandomForestClassifier(random_state=23)
```

```
params_grid={
    'n_estimators':[100,200,300],
    'max_depth':[1,5,10],
    'min_samples_split':[2,5,7],
    'min_samples_leaf':[1,2,4],
    'criterion':['gini','entropy']
}
```

```
random_search=RandomizedSearchCV(estimator=base_model,param_distributions=params_grid)
```

```
random_search.fit(x_train,y_train)
```

```
▸ RandomizedSearchCV
▸ best_estimator_: RandomForestClassifier
  ▸ RandomForestClassifier
```

```
print(random_search.best_params_)
```

```
{'n_estimators': 300, 'min_samples_split': 7, 'min_samples_leaf': 2, 'max_depth': 5, 'criterion': 'entropy'}
```

```
model3=RandomForestClassifier(n_estimators=300,min_samples_split=7,min_samples_leaf=2,max_depth=5,criterion='entropy',random_state=23)
```

```
model3.fit(x_train,y_train)
```

```
▾                          RandomForestClassifier                          i ?
RandomForestClassifier(criterion='entropy', max_depth=5, min_samples_leaf=2,
                       min_samples_split=7, n_estimators=300, random_state=23)
```

```
rf_y_pred=model3.predict(x_test)
```

```
accuracy_score(y_test,rf_y_pred)*100
```

```
83.75888391682021
```