Import libraries

```python
import pandas as pd
from scipy import stats
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.combine import SMOTEENN
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
import pickle
import warnings
warnings.filterwarnings('ignore')
import matplotlib
matplotlib.rcParams["figure.dpi"] = 80
```

Loading the data

```python
data_train = pd.read_csv("adult.csv", header=None, names=["age","workclass","final_weight","education","education-num","marital-status","occupation","relationship","race","sex","capital-gain","capital-loss","hours-per-week","native-country","income"])
data_test = pd.read_csv("adult.test", header=None, names=["age","workclass","final_weight","education","education-num","marital-status","occupation","relationship","race","sex","capital-gain","capital-loss","hours-per-week","native-country","income"])
```

EDA

What is the dimension of data?

```python
def row_col(data):
    print("Total rows present in dataset    - {}".format(data.shape[0]))
    print("Total columns present in dataset - {}".format(data.shape[1]))

print("Training Data:")
row_col(data_train)
print()
print("Testing Data:")
row_col(data_test)
```

```
Training Data:
Total rows present in dataset    - 32561
Total columns present in dataset - 15

Testing Data:
Total rows present in dataset    - 16281
Total columns present in dataset - 15
```

Categorical features and numerical features

```python
def get_cat_num(data):

    categorical = []
    numerical = []

    for column in data.drop("income", axis=1).columns:
        if data[column].dtype == "O":
            categorical.append(column)
        else:
            numerical.append(column)
    return categorical, numerical

categorical, numerical = get_cat_num(data_train)
print("There are {} categorical features.".format(len(categorical)))
print("Categorical features: {}".format(categorical))
print()
print("There are {} numerical features.".format(len(numerical)))
print("Numerical features: {}".format(numerical))
```

```
There are 8 categorical features.
Categorical features: ['workclass', 'education', 'marital-status', '
occupation', 'relationship', 'race', 'sex', 'native-country']

There are 6 numerical features.
Numerical features: ['age', 'final_weight', 'education-num', 'capita
l-gain', 'capital-loss', 'hours-per-week']
```

```python
def strip_data(data):
    for cat_f in categorical:
        data[cat_f] = data[cat_f].str.strip()

strip_data(data_train)
strip_data(data_test)
```

Is there any duplicated rows?

```python
if data_train.duplicated().sum():
    print("Yes! There are", data_train.duplicated().sum(), "duplicate rows.")
    data_train.drop_duplicates(inplace=True)
```

```
Yes! There are 24 duplicate rows.
```

Replace '?' with 'Unknown'

```python
data_train = data_train.replace('?', 'Unknown')
data_test = data_test.replace('?', 'Unknown')
```

Analysis of Categorical Data

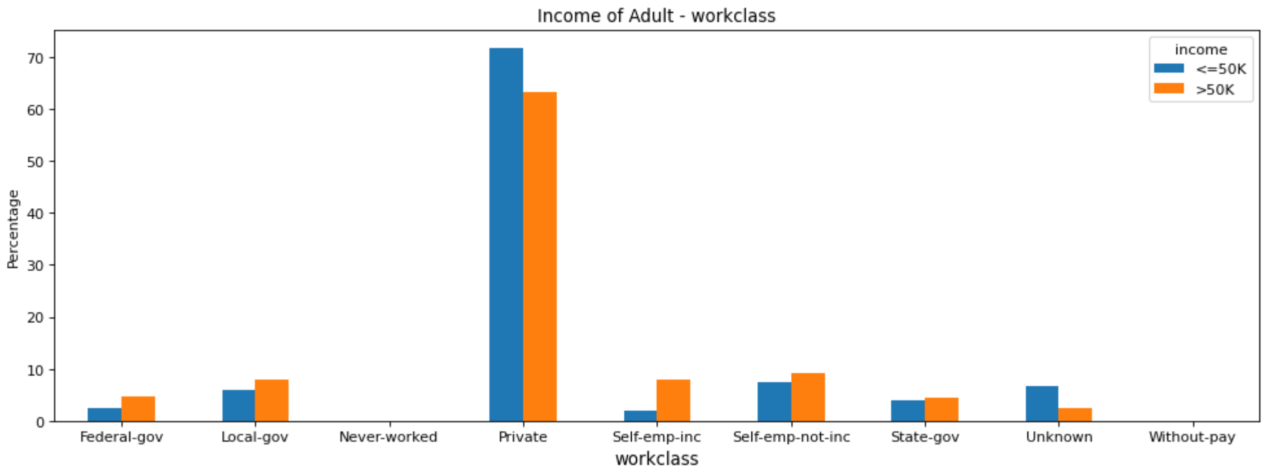
```python
summary = pd.DataFrame(columns=["Feature", "Relationship", "Strength"])

def get_plot(feature, rot=0):
    crossTable = pd.crosstab(data_train[feature],data_train["income"])
```

```
    percTable = crossTable.div(crossTable.sum(axis=0), axis=1)*100
    ax = percTable.plot(kind="bar", figsize=(15,5), title="Income of Adult - "+feature, rot=rot)
    ax.set_ylabel("Percentage", fontsize=10)
    ax.set_xlabel(feature, fontsize=12)
    plt.show()

def cramers_v_test(feature):
    global summary
    crossTable = pd.crosstab(data_train[feature],data_train["income"])
    chiVal, pVal, df, exp = stats.chi2_contingency(crossTable)
    print("Chi-square value :", round(chiVal,4))
    print("P Value          :", pVal)
    print()

    if pVal < 0.05:
        r = len(crossTable.index)
        c = len(crossTable.columns)
        n = crossTable.to_numpy().sum()
        v = np.sqrt(chiVal / (n * (min(r,c)-1)))

        print("Income is dependent on", feature, end=". ")
        print("Cramér’s V: ", round(v,2))

        if df == 1:
            if v < 0.10:
                relationship = "Negligible"
                print("Relationship: negligible")
            elif v < 0.30:
                print("Relationship: small")
            elif v < 0.50:
                print("Relationship: medium")
            else:
                print("Relationship: large")

        elif df == 2:
            if v < 0.07:
                relationship = "Negligible"
                print("Relationship: negligible")
            elif v < 0.21:
                relationship = "Small"
                print("Relationship: small")
            elif v < 0.35:
                relationship = "Medium"
                print("Relationship: medium")
            else:
                relationship = "Large"
                print("Relationship: large")

        elif df == 3:
            if v < 0.06:
                relationship = "Negligible"
                print("Relationship: negligible")
            elif v < 0.17:
                relationship = "Small"
                print("Relationship: small")
            elif v < 0.29:
                relationship = "Medium"
                print("Relationship: medium")
            else:
                relationship = "Large"
                print("Relationship: large")

        elif df == 4:
            if v < 0.05:
                relationship = "Negligible"
```

```python
                print("Relationship: negligible")
            elif v < 0.15:
                relationship = "Small"
                print("Relationship: small")
            elif v < 0.25:
                relationship = "Medium"
                print("Relationship: medium")
            else:
                relationship = "Large"
                print("Relationship: large")

        else:
            if v < 0.05:
                relationship = "Negligible"
                print("Relationship: negligible")
            elif v < 0.13:
                relationship = "Small"
                print("Relationship: small")
            elif v < 0.22:
                relationship = "Medium"
                print("Relationship: medium")
            else:
                relationship = "Large"
                print("Relationship: large")

        summary = summary.append({"Feature":feature, "Relationship":relationship, 
"Strength":round(v,2)}, ignore_index=True)
    else:
        print("Income is not dependent on",feature)
```

Workclass

```python
get_plot("workclass")
```



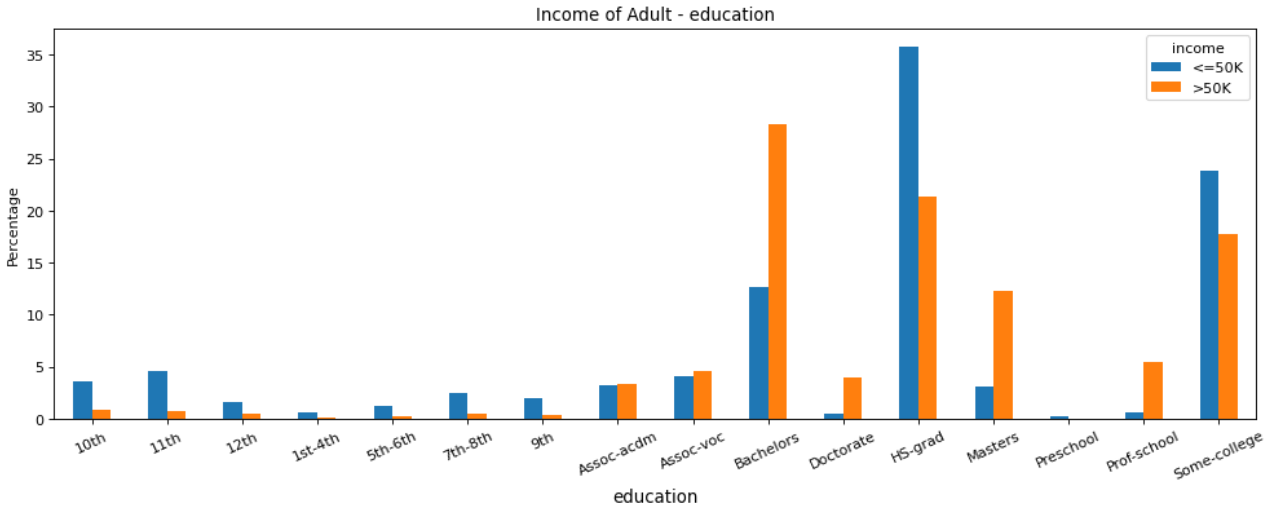

```python
cramers_v_test("workclass")
```

```
Chi-square value : 1044.6962
P Value          : 3.352256069028484e-220

Income is dependent on workclass. Cramér’s V:  0.18
```

```
Relationship: medium
```

Education

```
1.  get_plot("education", 25)
```



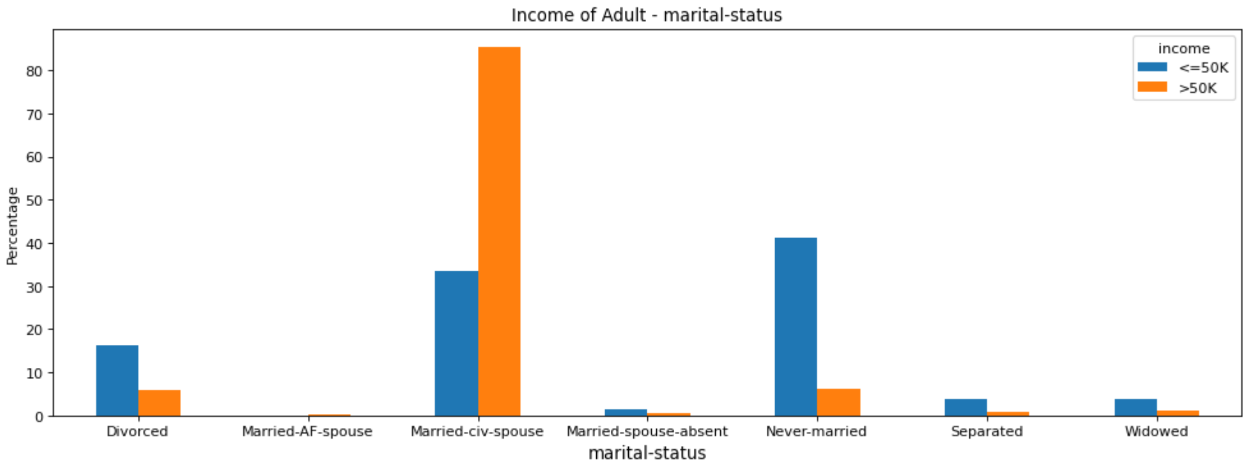

```
1.  data_train["education"] = np.where(data_train["education"].isin(["Preschool","1st-4th","5th-6th","7th-8th","9th","10th","11th","12th"]),"School",data_train["education"])
2.  data_train['education'] = data_train['education'].map({'School':0,'HS-grad':1,'Some-college':2,'Assoc-voc':3,'Assoc-acdm':4,'Bachelors':5,'Masters':6,'Prof-school':7,'Doctorate':8})
3.  data_test["education"] = np.where(data_test["education"].isin(["Preschool","1st-4th","5th-6th","7th-8th","9th","10th","11th","12th"]),"School",data_test["education"])
4.  data_test['education'] = data_test['education'].map({'School':0,'HS-grad':1,'Some-college':2,'Assoc-voc':3,'Assoc-acdm':4,'Bachelors':5,'Masters':6,'Prof-school':7,'Doctorate':8})
```

```
1.  cramers_v_test("education")
```

```
Chi-square value : 4425.2842
P Value          : 0.0

Income is dependent on education. Cramér’s V:  0.37
Relationship: large
```

Marital-status

```
1.  get_plot("marital-status")
```

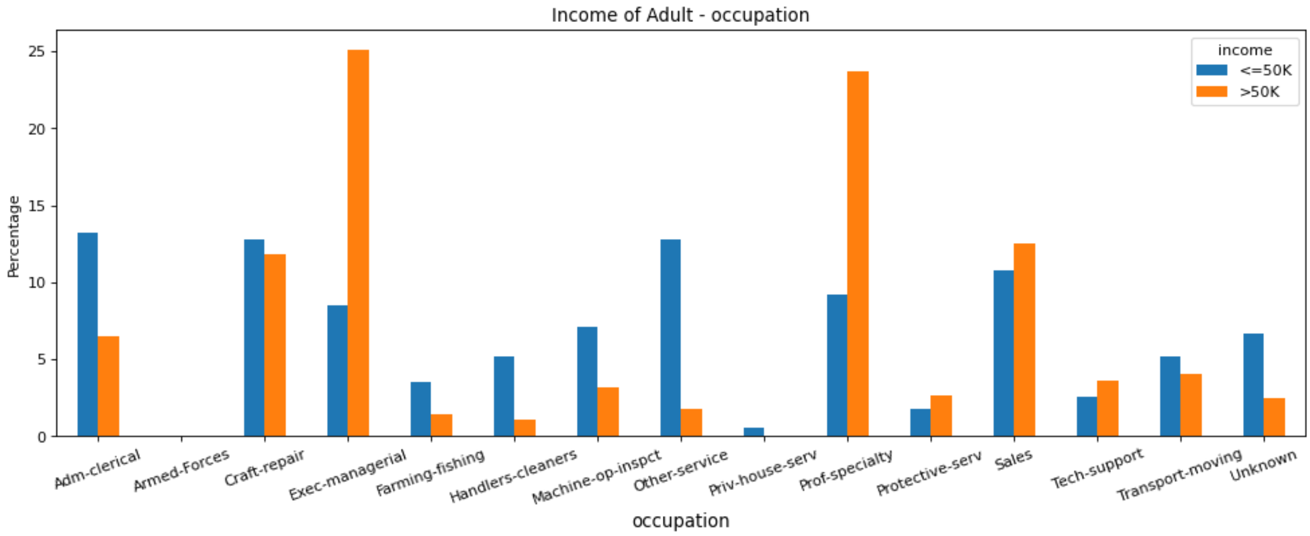



```
1. cramers_v_test("marital-status")
```

```
Chi-square value : 6510.3321
P Value          : 0.0

Income is dependent on marital-status. Cramér's V:  0.45
Relationship: large
```

Occupation

```
1. get_plot("occupation", 20)
```

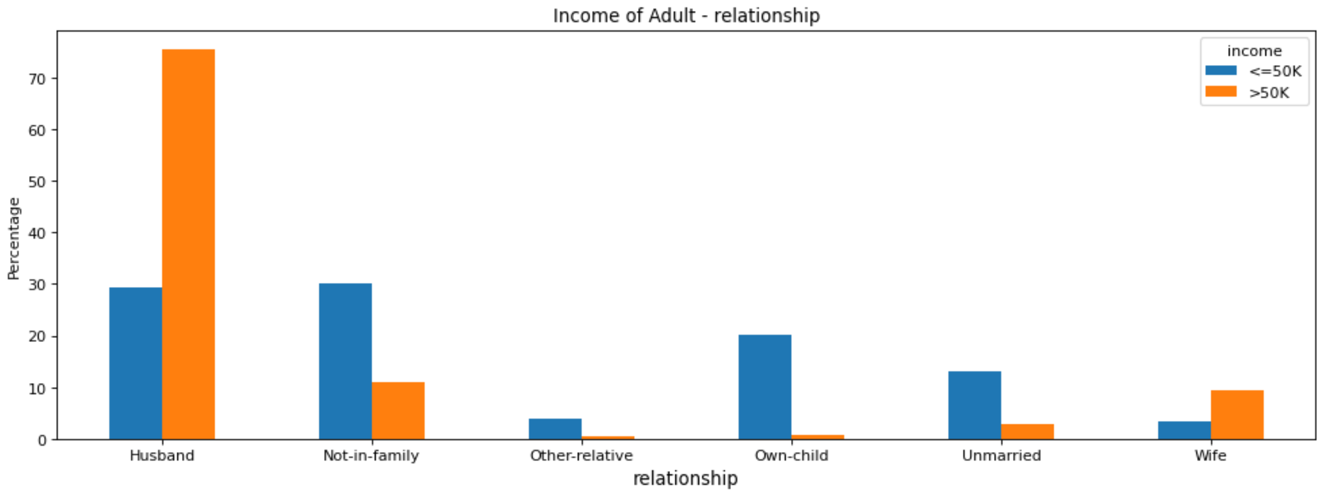



```
1. cramers_v_test("occupation")
```

```
Chi-square value : 4030.2092
P Value          : 0.0

Income is dependent on occupation. Cramér’s V:  0.35
Relationship: large
```

Relationship

```
1. get_plot("relationship")
```

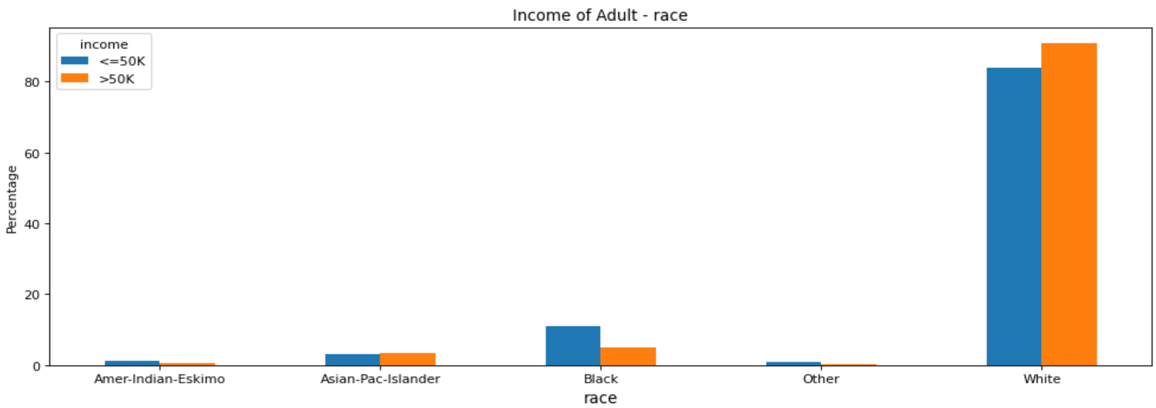



```
1. cramers_v_test("relationship")
```

```
Chi-square value : 6692.0988
P Value          : 0.0

Income is dependent on relationship. Cramér’s V:  0.45
Relationship: large
```

Race

```
1. get_plot("race")
```

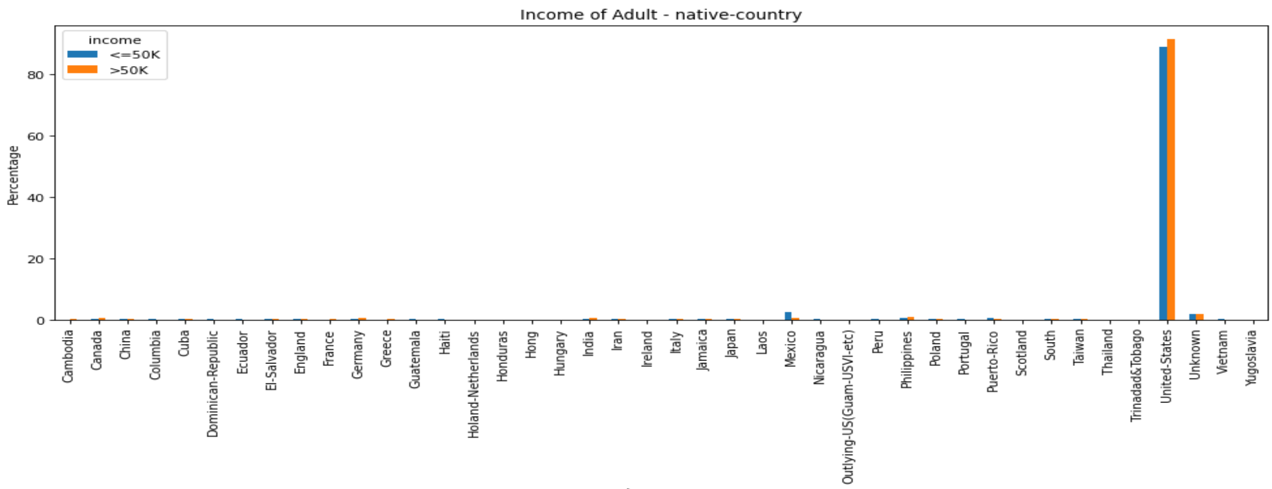


```
1. cramers_v_test("race")
```

```
Chi-square value : 330.9434
P Value          : 2.2797874171824478e-70

Income is dependent on race. Cramér's V:  0.1
Relationship: small
```

Native-country

```
1. get_plot("native-country", 90)
```




```
1. cramers_v_test("native-country")
```

```
Chi-square value : 315.4485
P Value          : 4.833085519399296e-44

Income is dependent on native-country. Cramér's V:  0.1
Relationship: small
```

```
1. summary
```

	Feature	Relationship	Strength
0	workclass	Medium	0.18
1	education	Large	0.37
2	marital-status	Large	0.45
3	occupation	Large	0.35
4	relationship	Large	0.45
5	race	Small	0.10
6	native-country	Small	0.10

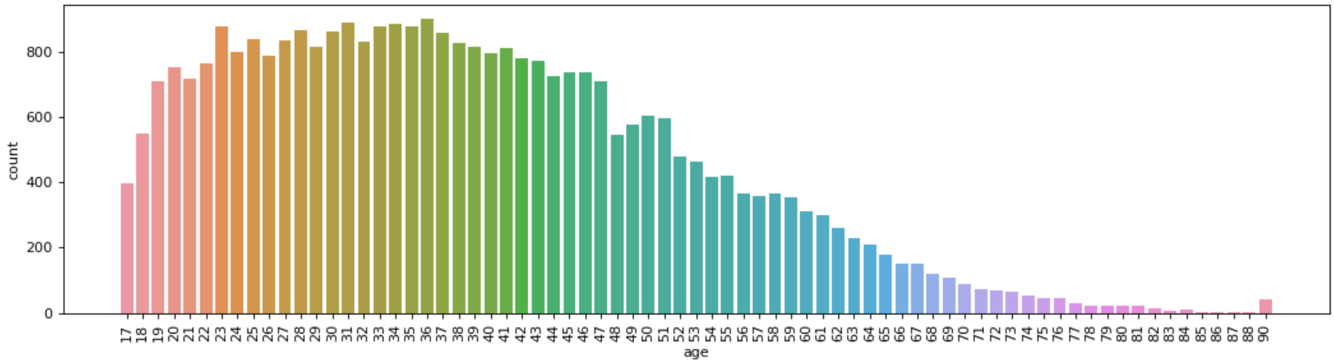
Okay! You can clearly see **race** and **native-country** both can be dropped as they have small relationship with dependent variable.

```
data_train.drop(["race", "native-country"], axis=1, inplace=True)
data_test.drop(["race", "native-country"], axis=1, inplace=True)
```

Analysis of Numerical Data

Age

```
plt.figure(figsize=(16,4))
sns.countplot(data_train["age"])
plt.xticks(rotation=90)
plt.plot()
```



Education-num

```
data_train['education-num'].unique()
```

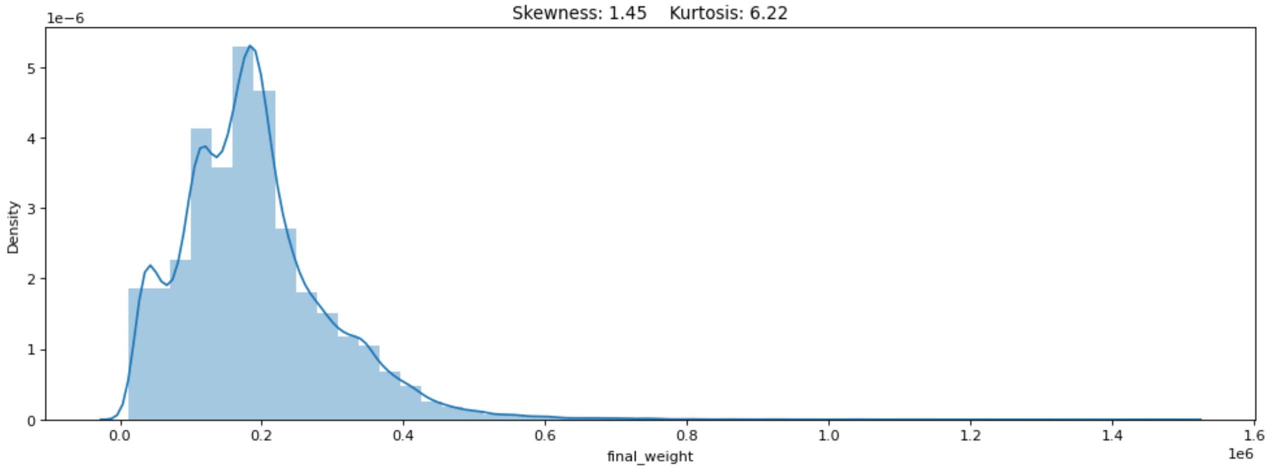
```
array([13,  9,  7, 14,  5, 10, 12, 11,  4, 16, 15,  3,  6,  2,  1,  8],
      dtype=int64)
```

Seems like **education** and **education-num** are same features. education-num is just numerical representation of education. So, **We'll drop education-num**.

```
data_train.drop("education-num", axis=1, inplace=True)
data_test.drop("education-num", axis=1, inplace=True)
```

Final Weight

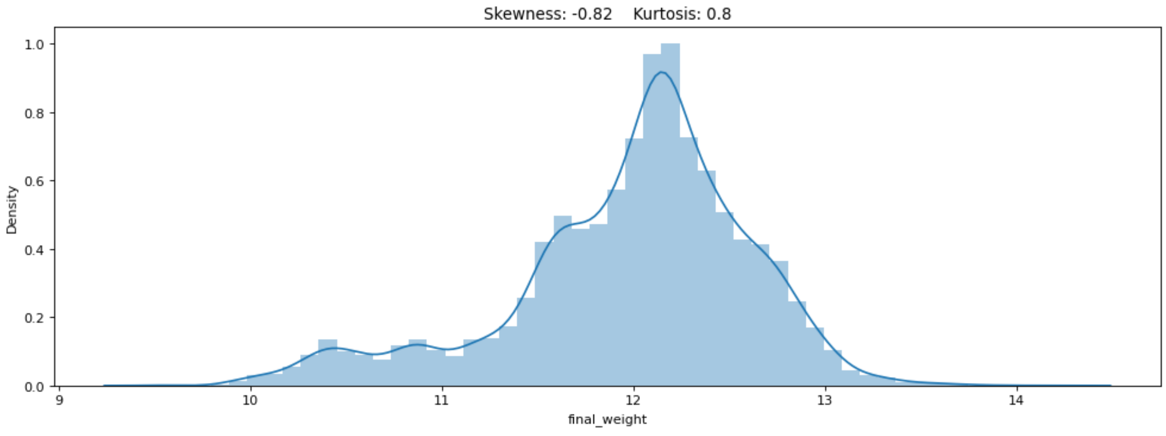
```
plt.figure(figsize=(15,5))
sns.distplot(data_train['final_weight'])
plt.title("Skewness: "+ str(round(data_train['final_weight'].skew(),2)) + "    " +
"Kurtosis: " + str(round(data_train['final_weight'].kurtosis(),2)))
plt.show()
```



Final weight is right skewed. We need to make it normal for further tasks.

```
1.  data_train['final_weight'] = np.log1p(data_train[['final_weight']])
2.  data_test['final_weight'] = np.log1p(data_test[['final_weight']])
```

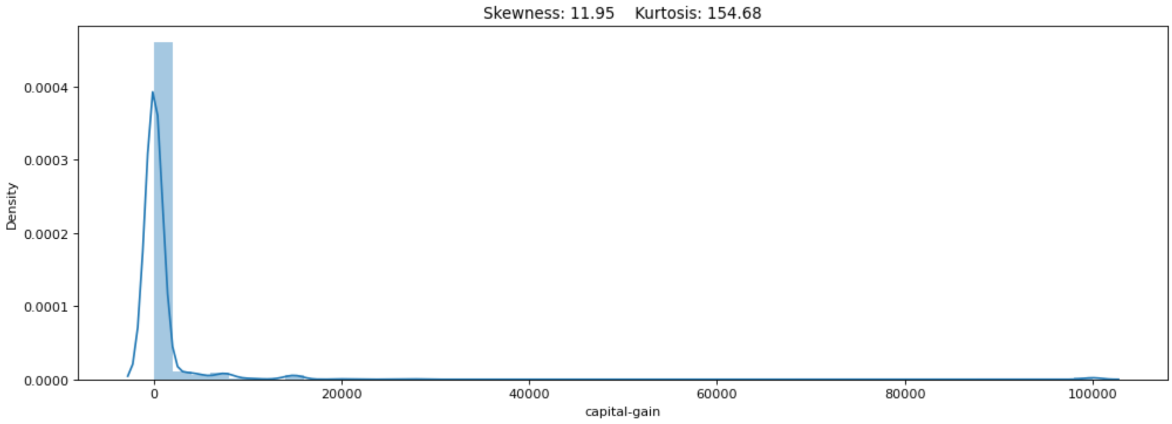
```
1.  plt.figure(figsize=(15,5))
2.  sns.distplot(data_test['final_weight'])
3.  plt.title("Skewness: "+ str(round(data_test['final_weight'].skew(),2)) + "     " +
    "Kurtosis: " + str(round(data_test['final_weight'].kurtosis(),2)))
4.  plt.show()
```



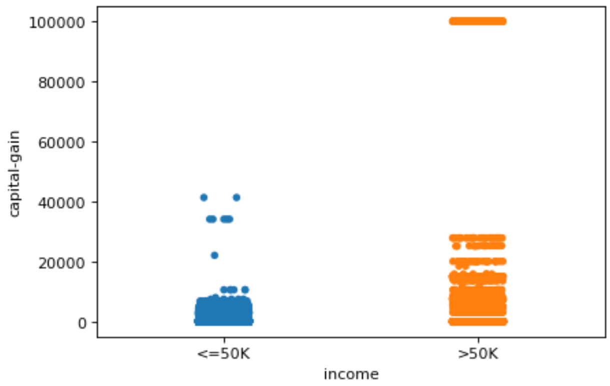
Capital Gain

```
1.  plt.figure(figsize=(15,5))
2.  sns.distplot(data_train['capital-gain'])
```

```
3.  plt.title("Skewness: "+ str(round(data_train['capital-gain'].skew(),2)) + "     " + 
    "Kurtosis: " + str(round(data_train['capital-gain'].kurtosis(),2)))
4.  plt.show()
```



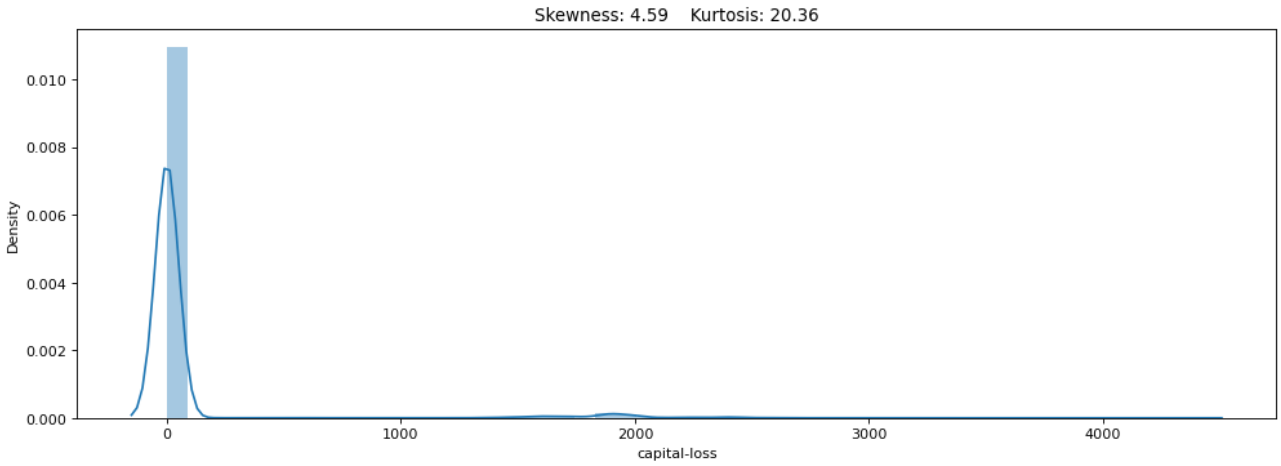

```
1.  sns.stripplot(data=data_train, x='income', y='capital-gain')
```




```
1.  data_train["capital-gain"] = np.where(data_train["capital-gain"]>=7000, 1, 0)
2.  data_test["capital-gain"] = np.where(data_test["capital-gain"]>=7000, 1, 0)
```

Capital Loss

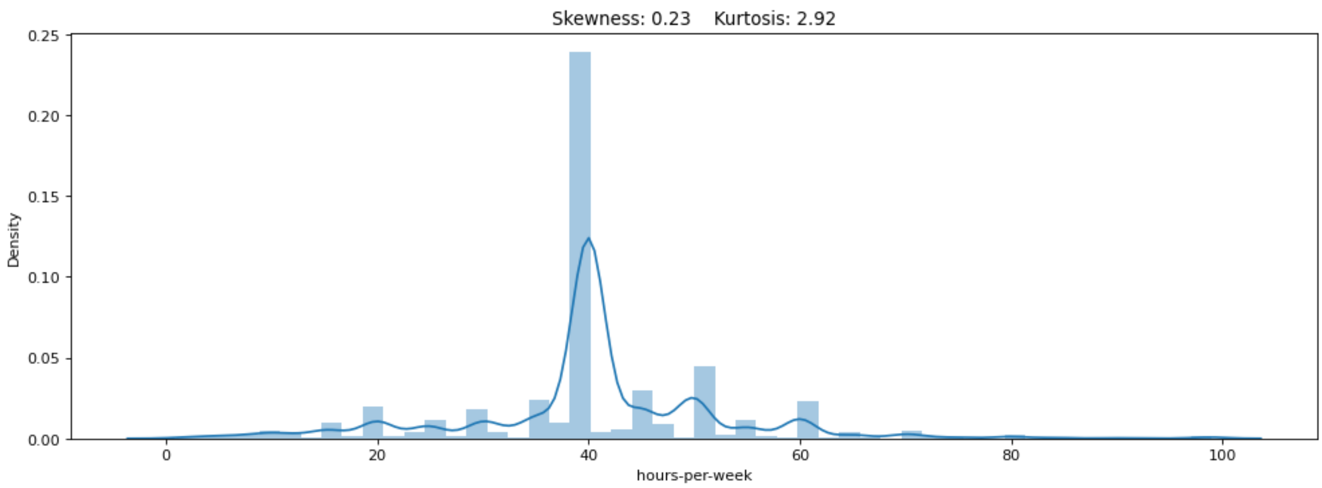
```
1.  plt.figure(figsize=(15,5))
2.  sns.distplot(data_train['capital-loss'])
3.  plt.title("Skewness: "+ str(round(data_train['capital-loss'].skew(),2)) + "     " + 
    "Kurtosis: " + str(round(data_train['capital-loss'].kurtosis(),2)))
4.  plt.show()
```




```
data_train["capital-loss"] = data_train["capital-loss"].apply(lambda x: x ** (1/2))
data_test["capital-loss"] = data_test["capital-loss"].apply(lambda x: x ** (1/2))
```

Hours-per-week

```
plt.figure(figsize=(15,5))
sns.distplot(data_train['hours-per-week'])
plt.title("Skewness: "+ str(round(data_train['hours-per-week'].skew(),2)) + "     " + "Kurtosis: " + str(round(data_train['hours-per-week'].kurtosis(),2)))
plt.show()
```

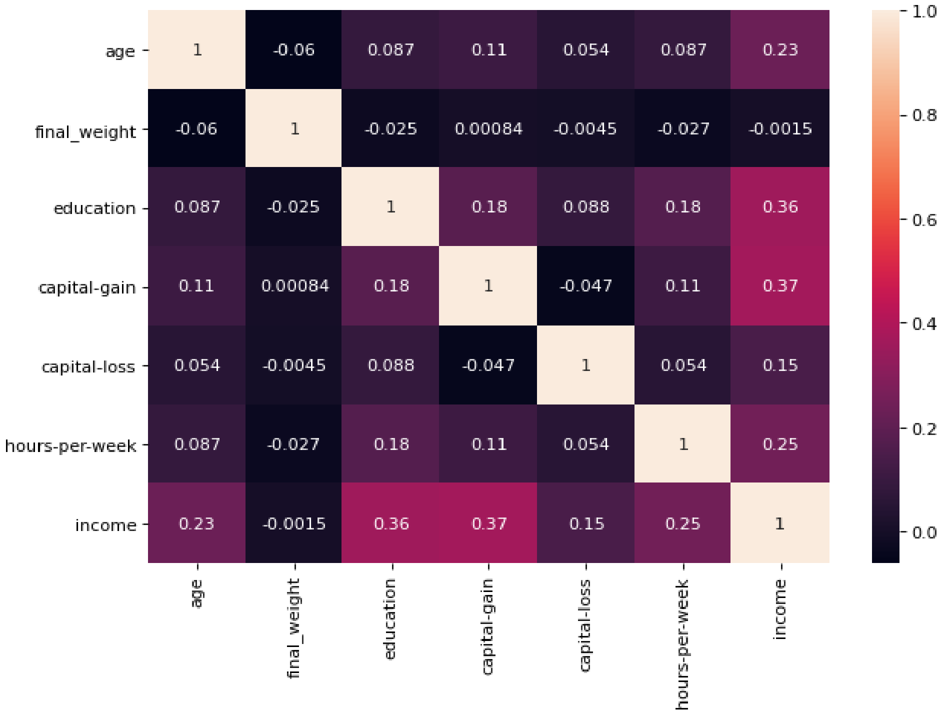



```
data_train['hours-per-week'] = np.where(data_train['hours-per-week']<35, 0, np.where(data_train['hours-per-week']<50, 1, np.where(data_train['hours-per-week']<72, 2, 3)))
data_test['hours-per-week'] = np.where(data_test['hours-per-week']<35, 0, np.where(data_test['hours-per-week']<50, 1, np.where(data_test['hours-per-week']<72, 2, 3)))
```

```
data_train['income'] = data_train['income'].map({"<=50K":0, ">50K":1})
data_test['income'] = data_test['income'].map({" <=50K.":0, " >50K.":1})
```

```
plt.figure(figsize=(9,6))
```

```
sns.heatmap(data_train.corr(), annot=True)
plt.plot()
```



Here, we can see that **final_weight** is not correlated with dependent variable. So, we can drop it.

```
data_train.drop("final_weight", axis=1, inplace=True)
data_test.drop("final_weight", axis=1, inplace=True)
```

```
data_train.head()
```

	age	workclass	education	education-num	marital-status	occupation	relationship	sex	capital-gain	capital-loss	hours-per-week	income
0	39	State-gov	5	13	Never-married	Adm-clerical	Not-in-family	Male	0	0.0	1	0
1	50	Self-emp-not-inc	5	13	Married-civ-spouse	Exec-managerial	Husband	Male	0	0.0	0	0
2	38	Private	1	9	Divorced	Handlers-cleaners	Not-in-family	Male	0	0.0	1	0
3	53	Private	0	7	Married-civ-spouse	Handlers-cleaners	Husband	Male	0	0.0	1	0
4	28	Private	5	13	Married-civ-spouse	Prof-specialty	Wife	Female	0	0.0	1	0

```
X_train = data_train.drop('income', axis=1)
y_train = data_train['income']

X_test = data_test.drop('income', axis=1)
y_test = data_test['income']
```

```
X_train = pd.get_dummies(X_train, drop_first=True)
X_test = pd.get_dummies(X_test, drop_first=True)
```

Handing Imbalanced Data

```
sme = SMOTEENN()
X_train, y_train = sme.fit_resample(X_train, y_train)
```

```
y_train.value_counts()
```

```
1    18088
0    16811
Name: income, dtype: int64
```

Selection of Model

```
model_params = {

    'decision_tree' : {
        'model' : DecisionTreeClassifier(),
        'params' : {
            'criterion': ['gini', 'entropy'],
            'max_depth': [18, 20, 22],
            'min_samples_leaf': [2, 5, 8]
        }
    },

    'random_forest' : {
        'model' : RandomForestClassifier(),
        'params' : {
            'n_estimators': [130, 140, 145],
            'max_depth': [90, 100, 110]
        }
    },

    'xgboost_classifier' : {
        'model' : XGBClassifier(verbosity = 0),
        'params' : {
            'max_depth': [4, 5, 6] ,
            'learning_rate': [0.3, 0.4, 0.5],
            'subsample': [1, 2],
            'gamma' : [5, 6, 7]
        }
    },

    'logistic_regression' : {
        'model' : LogisticRegression(solver='liblinear'),
        'params' : {
            'penalty' : ['l2'],
            'C': [0.5, 1.0, 1.5],
            'max_iter' : [40, 50, 60]
        }
    }
}
```

Hyperparameter Tuning

```
scores = []
best_estimators = {}
for algo, mp in model_params.items():
    clf = GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
    clf.fit(X_train, y_train)
    scores.append({
```

```
            'model':algo,
            'best_score':clf.best_score_,
            'best_params':clf.best_params_
        })
        best_estimators[algo] = clf.best_estimator_
```

```
df = pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])
df
```

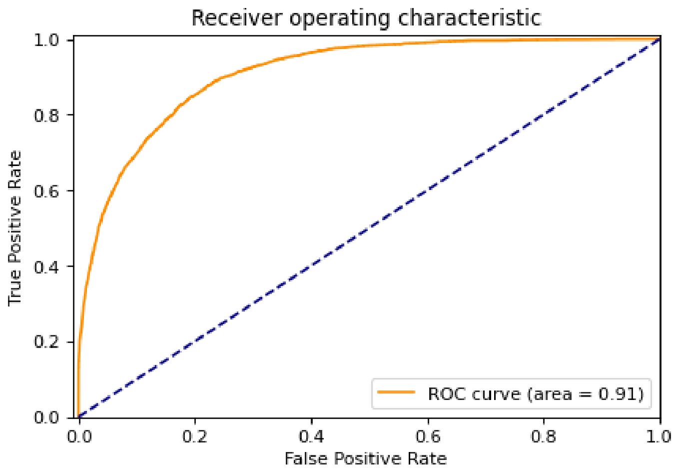
	model	best_score	best_params
0	decision_tree	0.956274	{'criterion': 'entropy', 'max_depth': 22, 'min...
1	random_forest	0.974297	{'max_depth': 110, 'n_estimators': 140}
2	xgboost_classifier	0.954612	{'gamma': 5, 'learning_rate': 0.4, 'max_depth'...
3	logistic_regression	0.932720	{'C': 1.5, 'max_iter': 40, 'penalty': 'l2'}

We'll select **XGBClassifier** as a final model.

```
clf = best_estimators['xgboost_classifier']
accuracy = clf.score(X_test, y_test)
prediction = clf.predict(X_test)
probs = clf.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, probs)
print("Accuracy Score :", round(accuracy, 4))
print()
print("ROU AUC Score  :", round(roc_auc, 4))
print()
print("Classification Report:\n")
print(classification_report(y_test, prediction))
print()
print("Note: As this is an imbalanced dataset accuracy can give us false assumptions regarding performance. So, It's better to rely on ROC metrices.")
print()
fpr, tpr, thresholds = roc_curve(y_test, probs)
plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([-0.01, 1.0])
plt.ylim([0.0, 1.01])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
```

```
Accuracy Score : 0.8039

ROU AUC Score  : 0.9104

Classification Report:

              precision    recall  f1-score   support

           0       0.95      0.78      0.86     12435
           1       0.55      0.87      0.68      3846

    accuracy                           0.80     16281
   macro avg       0.75      0.83      0.77     16281
weighted avg       0.86      0.80      0.82     16281
```

```
Note: As this is an imbalanced dataset accuracy can give us false assump
tions regarding performance. So, It's better to rely on ROC metrices.
```



Now, save this model so that we can use this same model in future.

```
1. pickle.dump(clf, open('final_model', 'wb'))
```