**Problem Definition:**

*The data was obtained from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics).* The goal is to train a classifier to predict whether a person makes over $50K a year *or not so it has two possible outcomes “>50K” and “<50K”. The data contain information such as “Age”,” Workclass”,” Fnlwgt”,” Education” etc. Dataset has 32560 instances and 15 attributes containing a blend of categorical and numerical values*.

**Data Analysis**:

Before moving to model building and evaluation phase, data analysis helps in understanding the data and deriving insights about dataset. Data analysis require cleaning, transforming and modelling of data to identify useful information from data and taking decision on basis of derived result.

First, import the required libraries

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

Now, Load the data into pandas DataFrame using read\_csv function.

df=pd.read\_csv("census\_income.csv")

df.shape

Shape attribute provide the information about the dataset containing 32500 rows and 15 columns

Info function provide concise summary of the DataFrame including the datatype

df.info ()

Data columns (total 15 columns):

Age 32560 non-null int64

Workclass 32560 non-null object

Fnlwgt 32560 non-null int64

Education 32560 non-null object

Education\_num 32560 non-null int64

Marital\_status 32560 non-null object

Occupation 32560 non-null object

Relationship 32560 non-null object

Race 32560 non-null object

Sex 32560 non-null object

Capital\_gain 32560 non-null int64

Capital\_loss 32560 non-null int64

Hours\_per\_week 32560 non-null int64

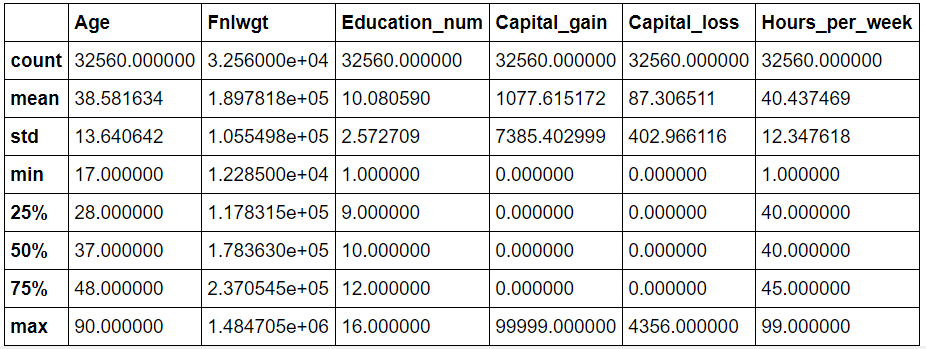
Native\_country 32560 non-null object

Income 32560 non-null object

It can be observed that no missing values can be identified with attributes a combination of numerical and categorical datatypes.

**Handling numerical columns:**

df.describe()



Describe function give us summary of statistics regarding the DataFrame columns. It displays the mean, std and IQR values for numeric columns.

It can be identified that Capital\_gain and Capital\_loss has median as 0 and huge difference between median and mean thus indicating skewness.

df["Workclass"].value\_counts()

df["Occupation"].value\_counts()

df["Native\_country"].value\_counts()

Using the value\_counts()function it can be noticed that "Workclass","Occupation",

"Native\_country" have “?” as the value which need to be replaced by “unknown”.

edit\_cols = ['Native\_country','Occupation','Workclass']

*# Replace ? with Unknown*

**for** col **in** edit\_cols:

df.loc[df[col] == '?', col] = 'unknown'

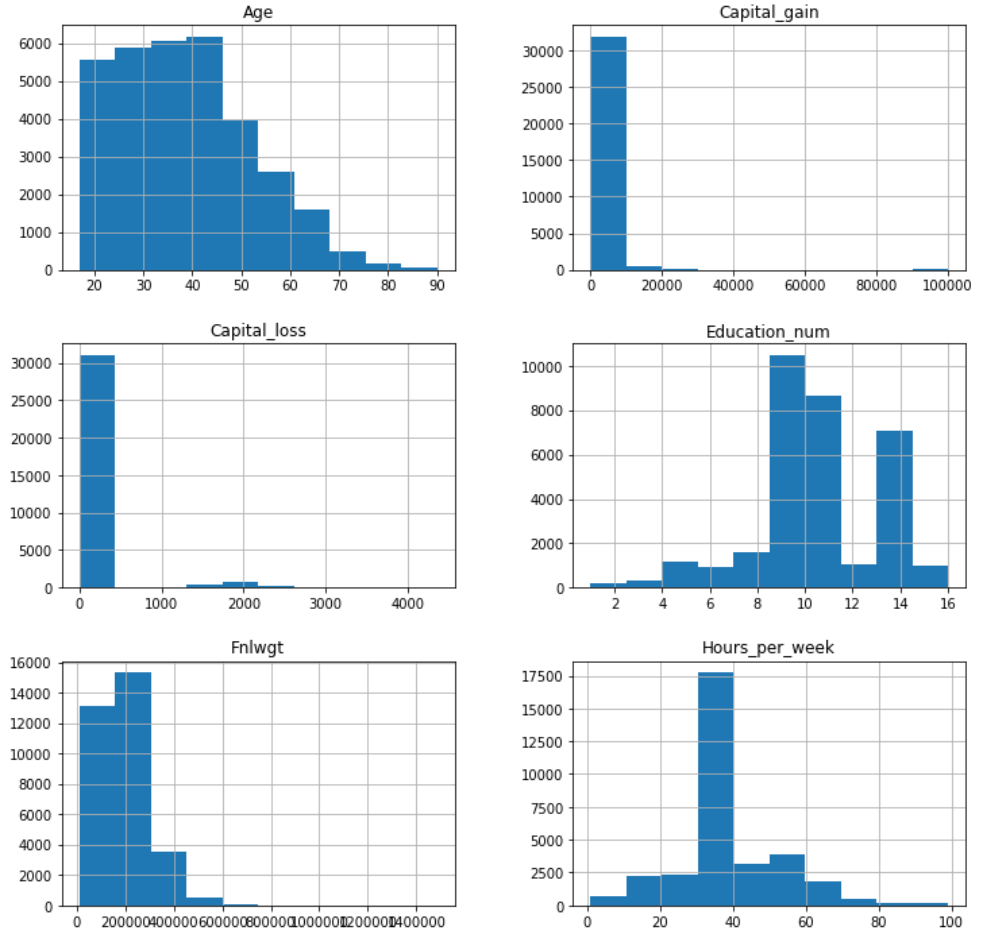
**Visualization of numerical columns**

*#Separate categorical and numerical columns*

cat\_set = df.dtypes[df. dtypes == 'object']

num\_set = df.dtypes[df.dtypes != 'object']

df[list(num\_set.index)].hist(figsize = (12,12))



Histogram generate bin of the ranges, then distributes the values into a series of intervals and count the values which fall into each of the intervals. It provides information about frequency distribution.

It can be deducted from the plot that for “Age”,” Capital\_gain”,” Capital\_loss”,” Education\_num”,” Fnlwgt”,” Hours\_per\_week” data is not uniformly distributed.

sns.distplot(df["Age"])

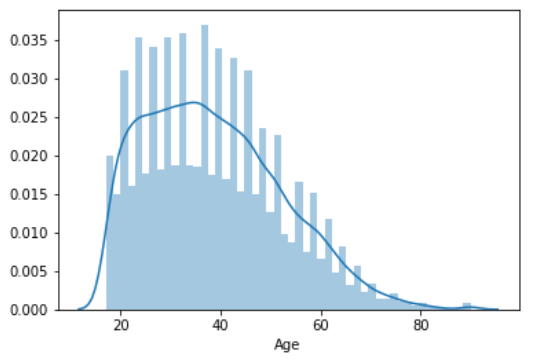
sns.distplot(df["Fnlwgt"])

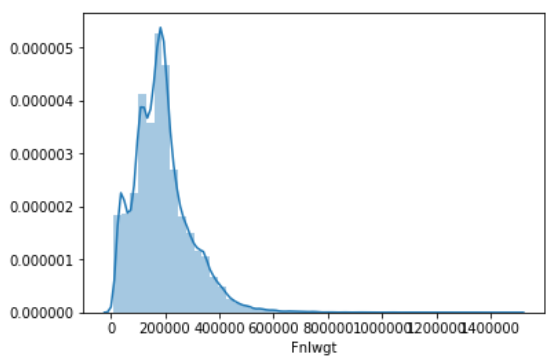
sns.distplot(df["Education\_num"])

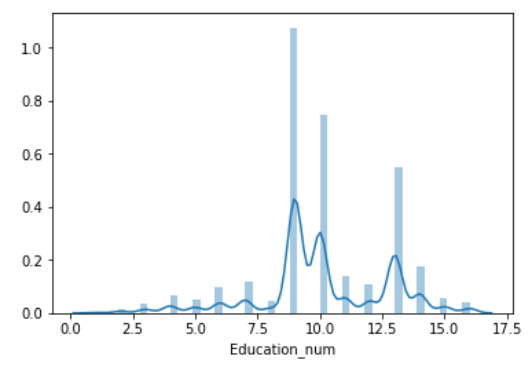
sns.distplot(df["Capital\_gain"])

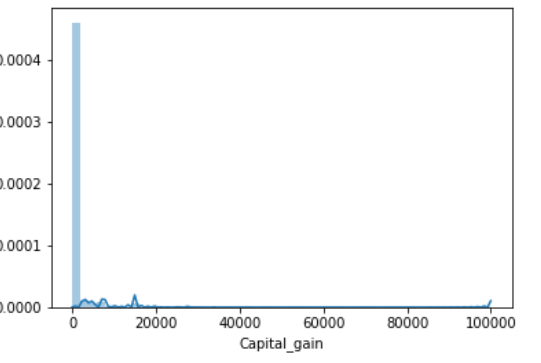
sns.distplot(df["Capital\_loss"])

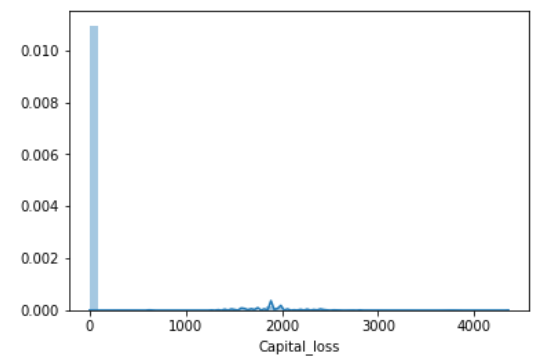
sns.distplot(df["Hours\_per\_week"])

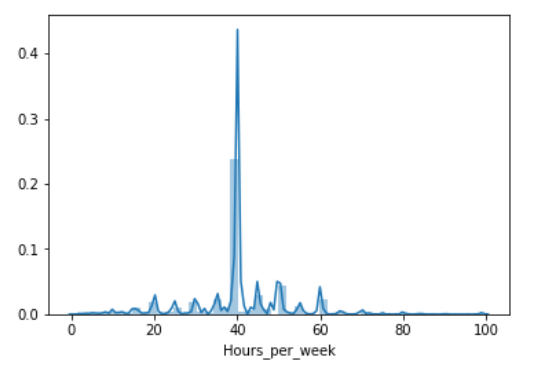












df.skew()

Age 0.558738

Fnlwgt 1.446972

Education\_num -0.311630

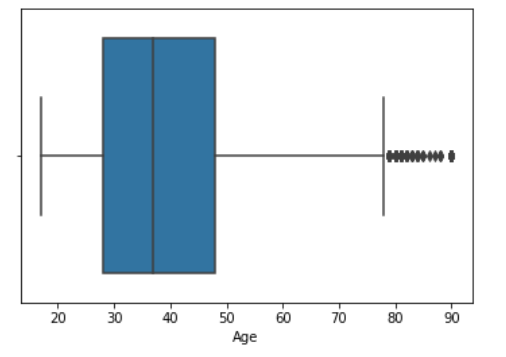
Capital\_gain 11.953690

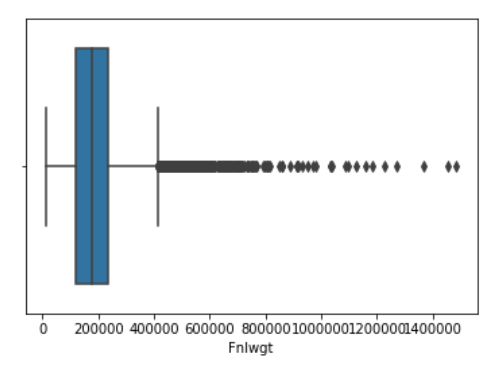
Capital\_loss 4.594549

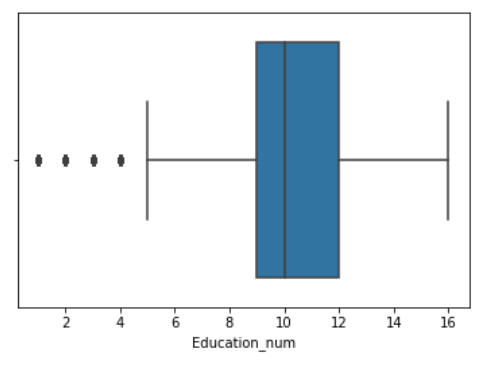
Hours\_per\_week 0.227636

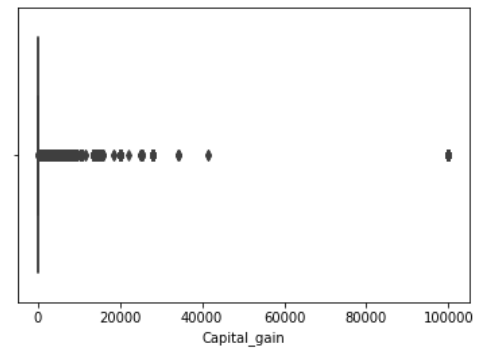
Using the distribution plot variation in data distribution can be identified. Also it helps us understanding the skewness in the data. It can be observed that “Age”,” Capital\_gain”,” Capital\_loss”,” Fnlwgt” have skewed data. Skewness can be reduced using Power Transform, Log Transform, Square root Transform, Box-Cox Transform.

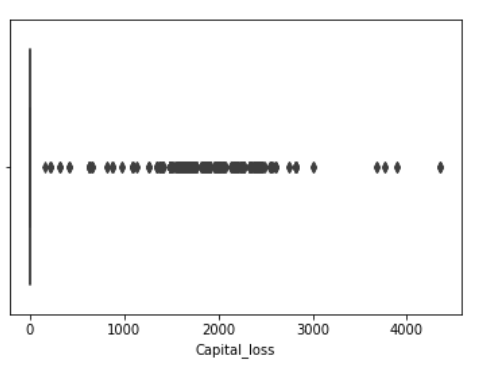
Along with skewness, outliers must also be considered so that model accuracy is not affected. To check the outliers, boxplot can be used and outliers can be removed using zscore.

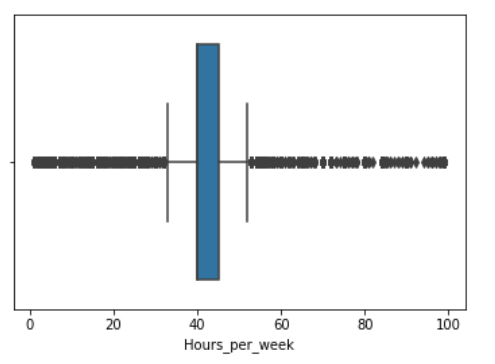












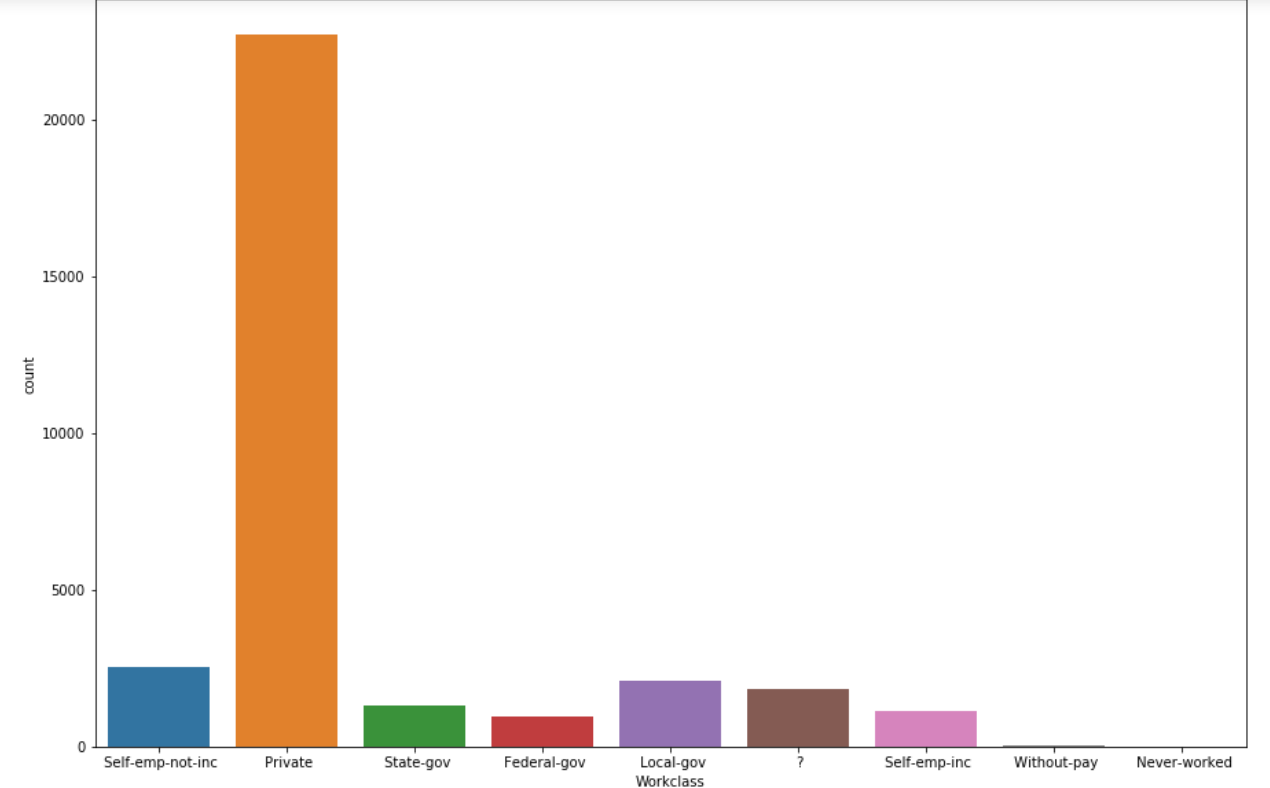
Looking at the boxplots, it can be identified that “Age”,” Capital\_gain”,” Capital\_loss”,” Education\_num”,” Fnlwgt”,” Hours\_per\_week” have outlier values.

**Handling Categorical columns**

Categorical columns can be visualized using countplot as it provides the counts of observations in each categorical bin using bars.

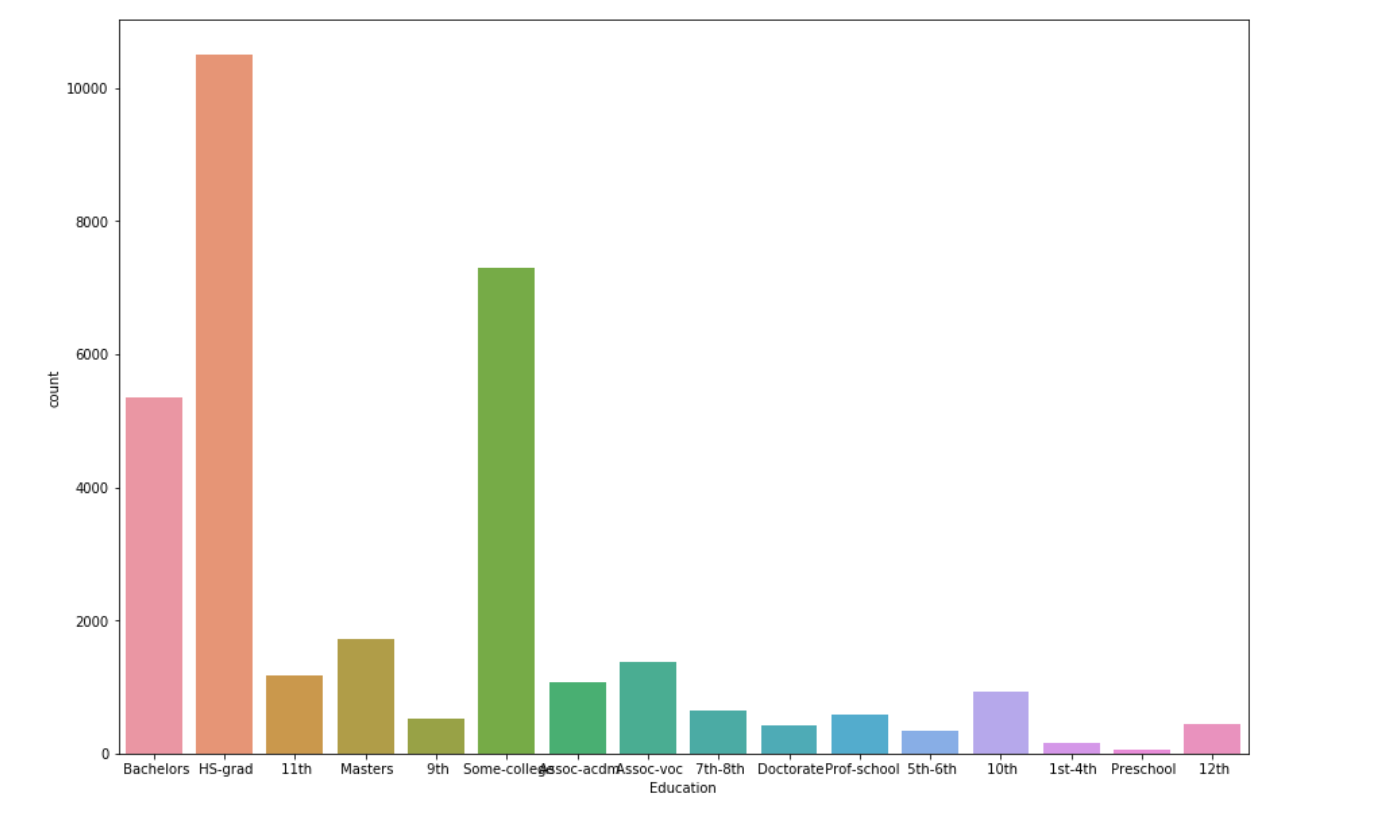
plt.figure(figsize=(15,10),facecolor="white")

sns.countplot(x="Workclass”, data=df)



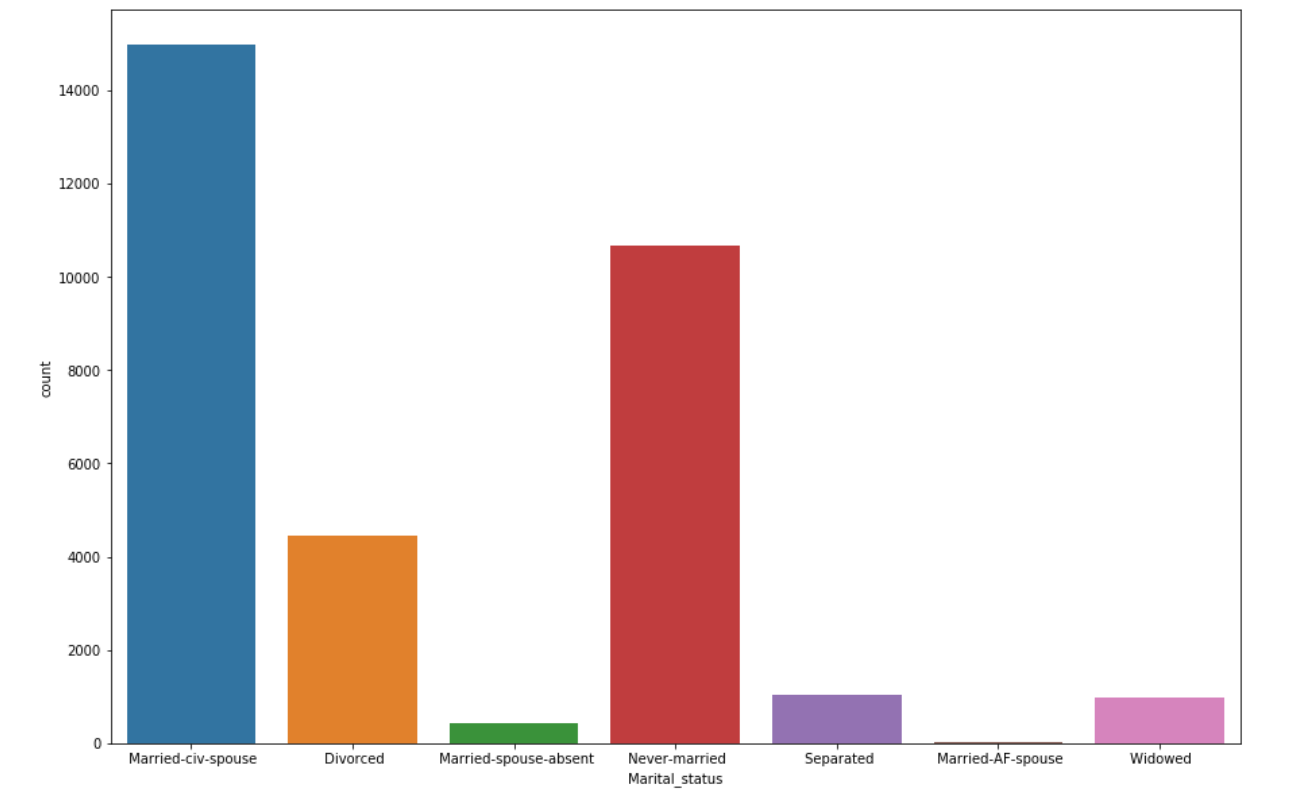
plt.figure(figsize=(15,10),facecolor="white")

sns.countplot(x="Education",data=df)



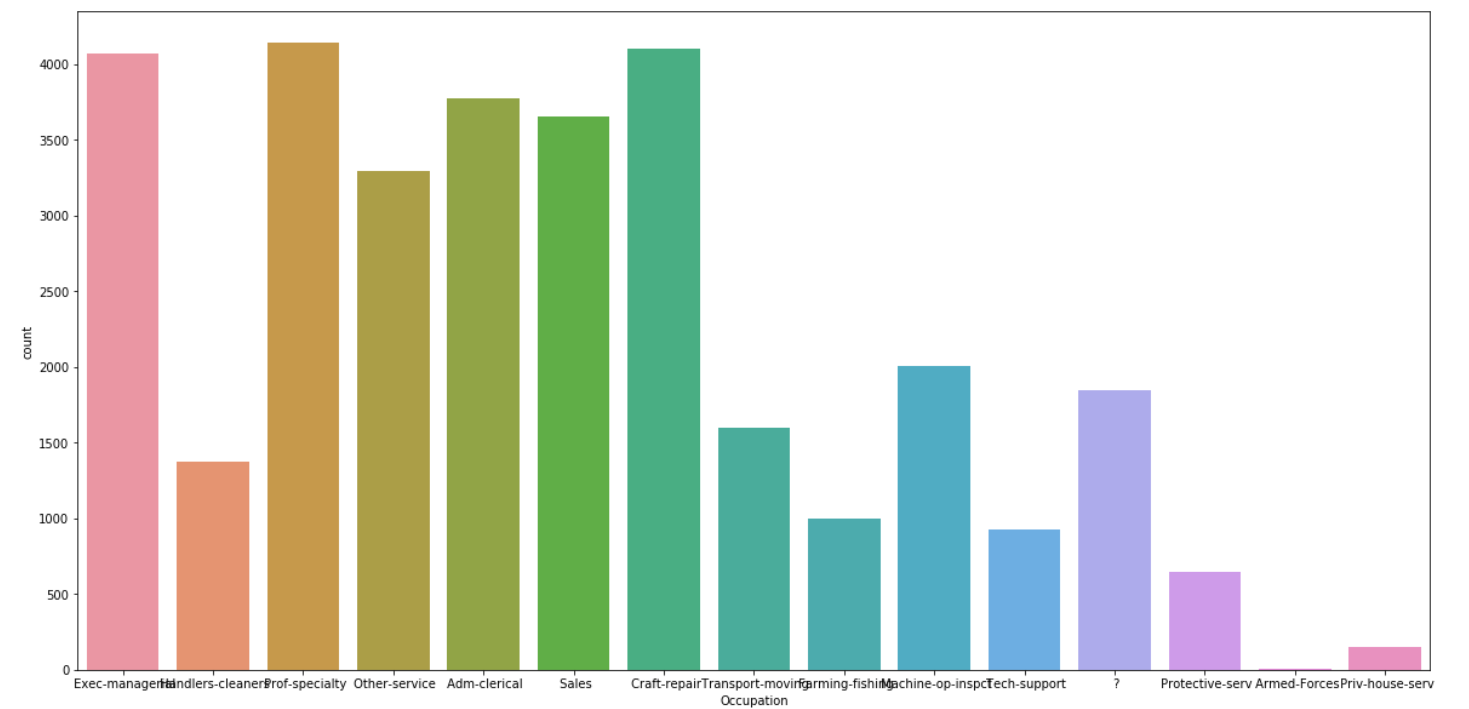
plt.figure(figsize=(15,10),facecolor="white")

sns.countplot(x="Marital\_status",data=df)



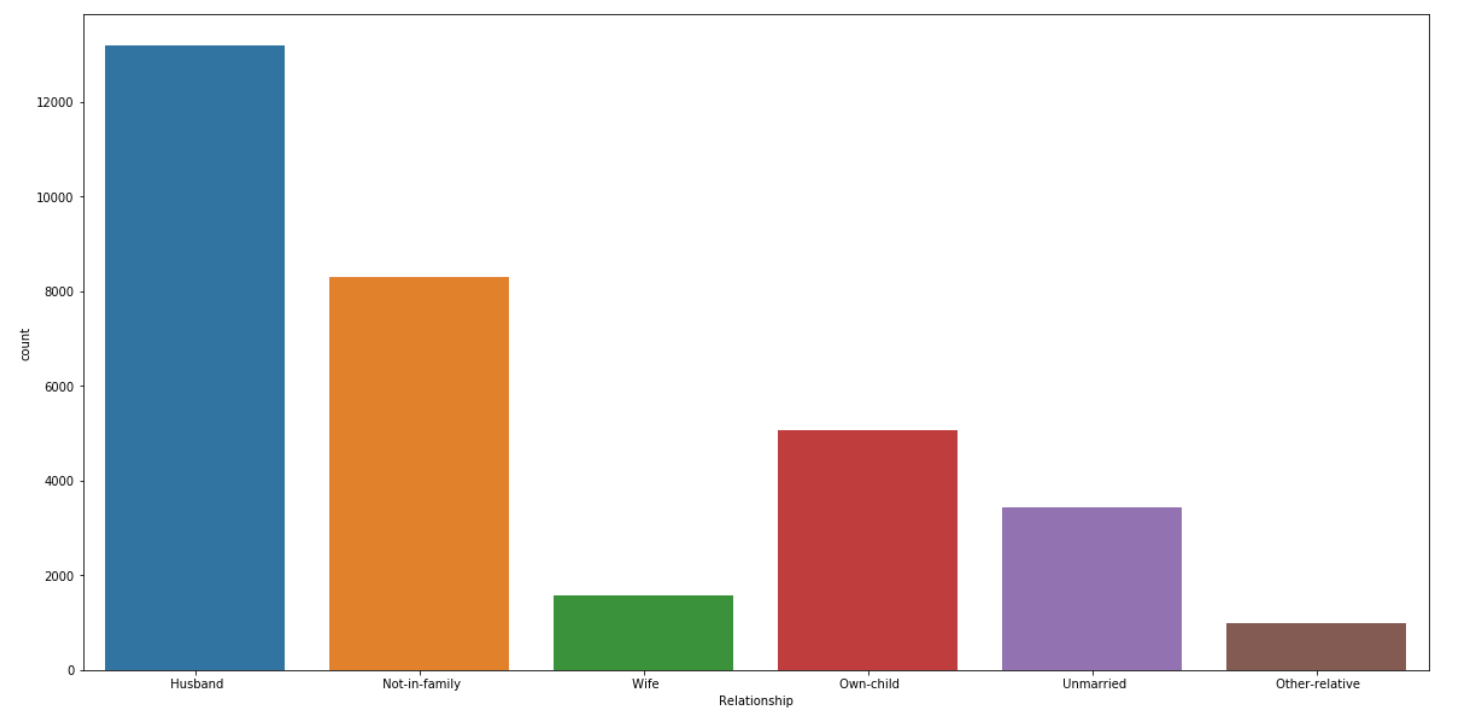
plt.figure(figsize=(20,10),facecolor="white")

sns.countplot(x="Occupation",data=df)



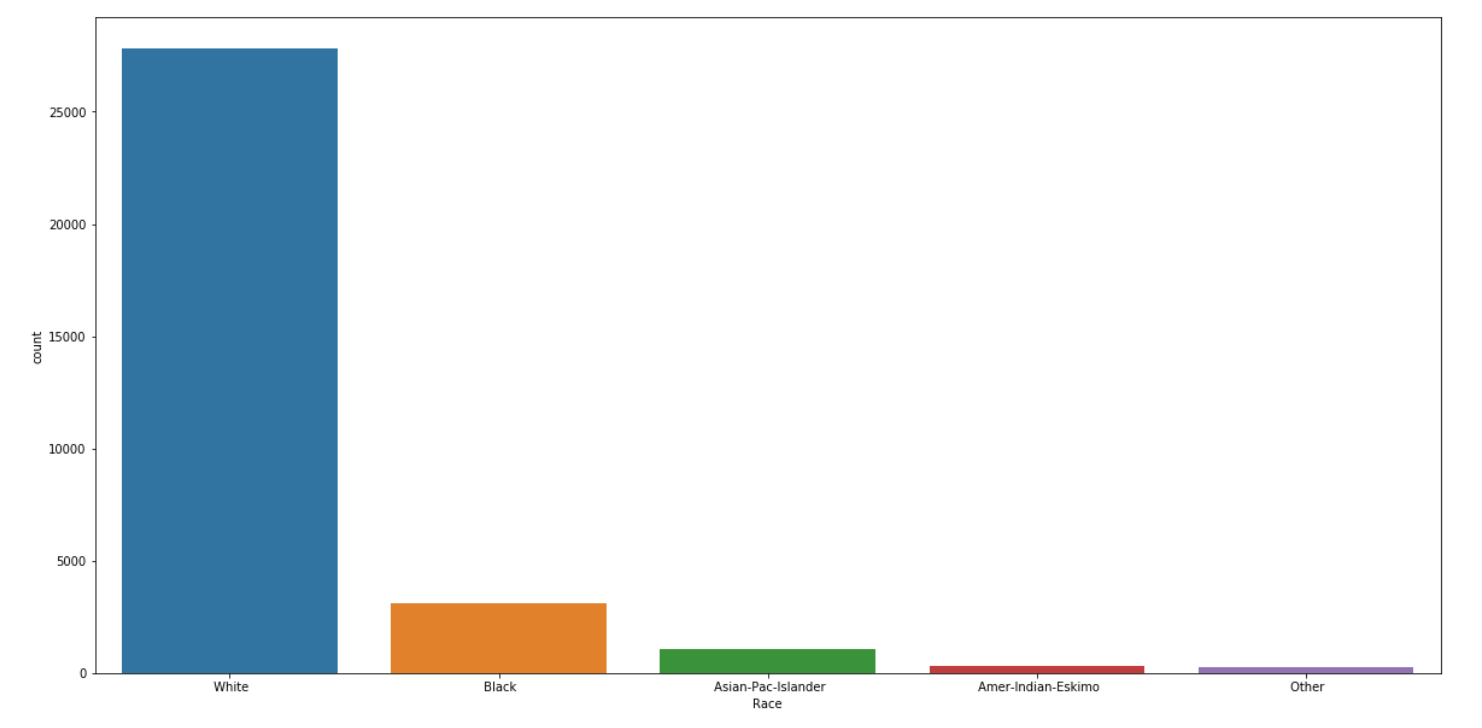
plt.figure(figsize=(20,10),facecolor="white")

sns.countplot(x="Relationship",data=df)



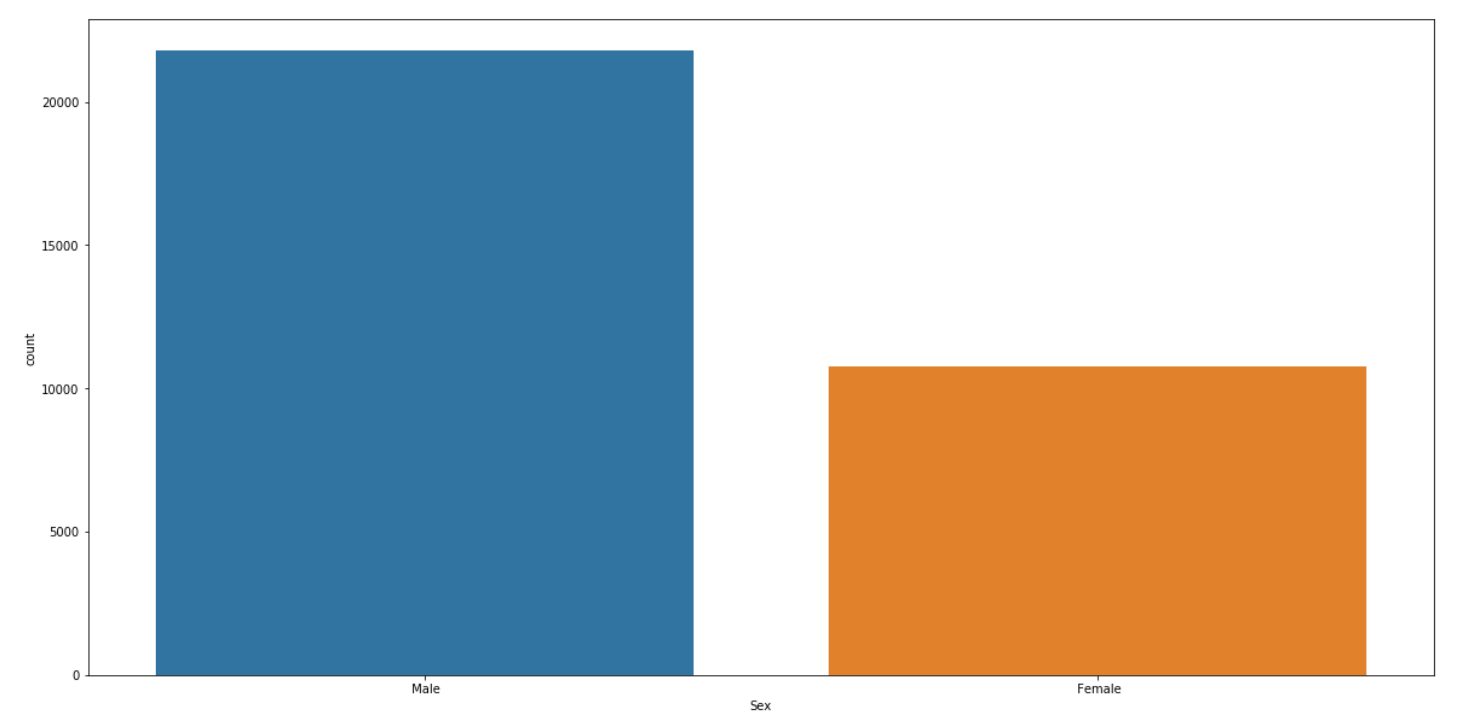
plt.figure(figsize=(20,10),facecolor="white")

sns.countplot(x="Race",data=df)



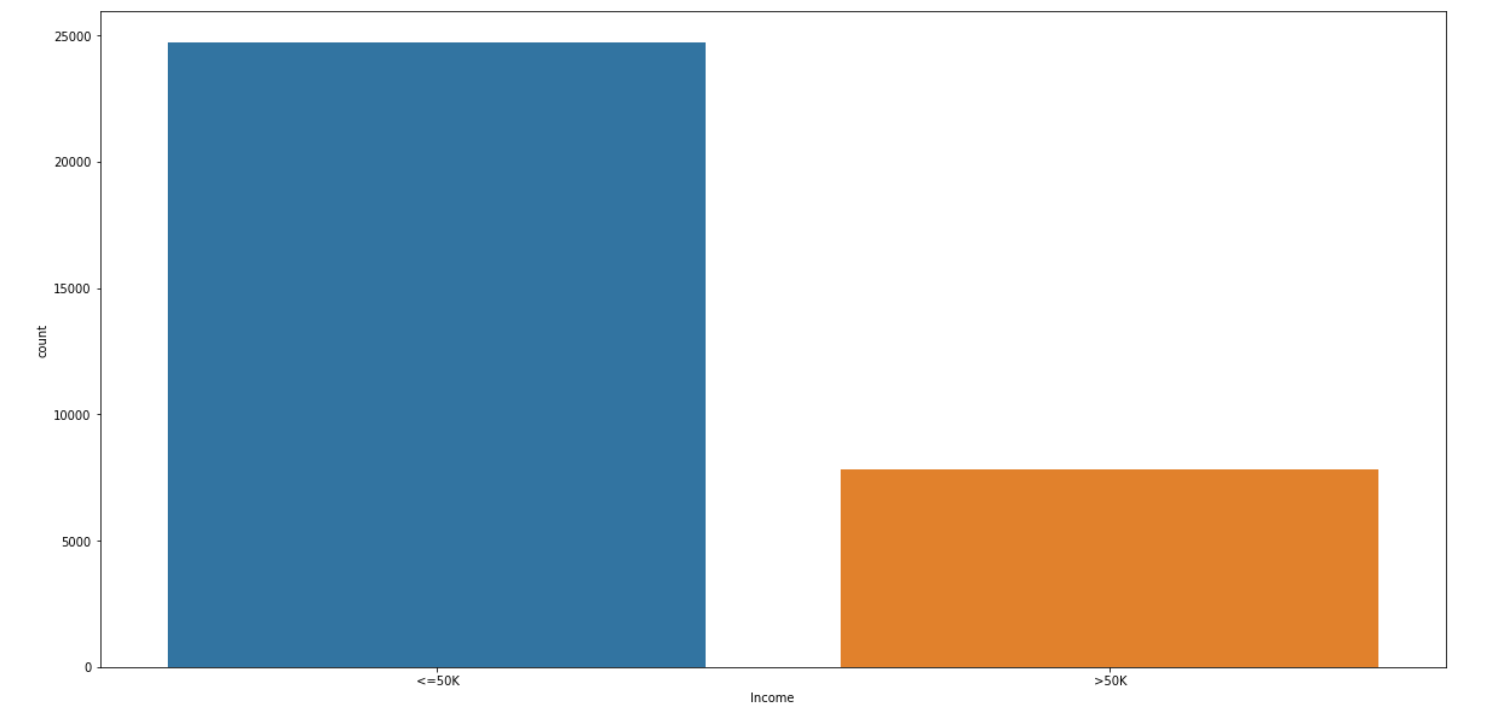
plt.figure(figsize=(20,10),facecolor="white")

sns.countplot(x="Sex",data=df)



plt.figure(figsize=(20,10),facecolor="white")

sns.countplot(x="Income",data=df)



Using the countplot it can be noticed that columns “Workclass”, "Education", "Marital\_status", "Occupation", "Relationship", "Race", "Sex" have class imbalance problem which can be resolved using undersampling or oversampling technique.

**Correlation**

Correlation is really important to identify which columns are affecting the target column positively or negatively. Through correlation it can also be identified that which columns are strongly related to target.

df.corr()

For plotting correlation heatmap can be used to understand the correlation more easily.

**import** **matplotlib.pyplot** **as** **plt**

plt.figure(figsize=(30,15))

sns.heatmap(df.corr(),annot=**True**,linewidths=0.1,linecolor="black",fmt=".2f")

**EDA Concluding Remark**:

* Dataset contains 32500 rows and 15 columns with no missing values
* Dataset comprise of column of categorical as well as numerical type.
* "Workclass”,” Occupation”, Native\_country" have “?” as the value which need to be

replaced by “unknown”.

* For columns “Age”,” Capital\_gain”,” Capital\_loss”,” Education\_num”,” Fnlwgt”,” Hours\_per\_week” data is not uniformly distributed.
* Capital\_gain and Capital\_loss has median as 0 and huge difference between median and mean thus indicating skewness. “Age”,” Capital\_gain”,” Capital\_loss”,” Fnlwgt” have skewed data. Power Transform can be used to remove skewness.
* It can be identified that “Age”,” Capital\_gain”,” Capital\_loss”,” Education\_num”,” Fnlwgt”,” Hours\_per\_week” have outlier values. Outliers can be removed using zscore.
* For columns “Workclass”, "Education", "Marital\_status", "Occupation", "Relationship", "Race", "Sex”, it can be noticed that columns have class imbalance problem which can be resolved using undersampling or oversampling technique
* MinMaxScaler can be used to normalize data column by column.

**Pre-Processing:**

To handle the numerical and categorical attributes differently. Numerical attributes need to be scaled, whereas for categorical columns missing values need to be filled and then encode the categorical values into numerical values.

We have lot of features which are categorical and for building a model, categorical column need to encoded. OrdicalEncoder can be used to encode the categorical columns

**from** **sklearn.preprocessing** **import** OrdinalEncoder

enc=OrdinalEncoder()

**for** i **in** df.columns:

**if** df[i].dtypes=="object":

df[i]=enc.fit\_transform(df[i].values.reshape(-1,1))

Before proceeding to model fitting, data transformation must be performed. Properly formatted and validated data improves data quality. Data transformation allow scaling each feature to a given range. For the given dataset MinMaxScaler can be used which scales all the data features in the range [0,1] or else in the range [-1,1].

Here, MinMaxScaler will be used to normalize the data column by column.

**from** **sklearn.preprocessing** **import** MinMaxScaler

scale=MinMaxScaler()

df["Age"]=scale.fit\_transform(df["Age"].values.reshape(-1,1))

**from** **sklearn.preprocessing** **import** MinMaxScaler

scale=MinMaxScaler()

df["Fnlwgt"]=scale.fit\_transform(df["Fnlwgt"].values.reshape(-1,1))

**from** **sklearn.preprocessing** **import** MinMaxScaler

scale=MinMaxScaler()

df["Education\_num"]=scale.fit\_transform(df["Education\_num"].values.reshape(-1,1))

**from** **sklearn.preprocessing** **import** MinMaxScaler

scale=MinMaxScaler()

df["Capital\_gain"]=scale.fit\_transform(df["Capital\_gain"].values.reshape(-1,1))

**from** **sklearn.preprocessing** **import** MinMaxScaler

scale=MinMaxScaler()

df["Capital\_loss"]=scale.fit\_transform(df["Capital\_loss"].values.reshape(-1,1))

**from** **sklearn.preprocessing** **import** MinMaxScaler

scale=MinMaxScaler()

df["Hours\_per\_week"]=scale.fit\_transform(df["Hours\_per\_week"].values.reshape(-1,1))

**Building Machine Learning Models:**

Model Building involves splitting of original dataset into input(x) and output (y) columns.

y=df["Income"]

x=df.drop("Income”, axis=1)

x and y will be used to divide the dataset into training dataset and testing dataset.

Training dataset can be used for model fitting and statistical method can be used to estimate the accuracy of the model using the unseen data. Testing set can be used for evaluating the model accuracy.

Loaded dataset will be spitted into two, 70% of which will be used to train and 30% will be used to test the model.

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score,classification\_report

from sklearn.metrics import confusion\_matrix, roc\_auc\_score

from sklearn.model\_selection import train\_test\_split

import warnings

warnings. filterwarnings("ignore")

Random state allows to generate a reproducible split. Scikit-learn use random permutations to generate the splits. Random state provided is a seed to generate random number and also ensure that generated numbers are in same order.

Maximum accuracy\_score can be used to identify the best random state from 1 to 200 and will proceed with same random state to fit the model using different algorithms.

Maxr2=0

BestRs=0

for i in range(1,200):

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=.30,random\_state=i)

lr=LogisticRegression()

lr.fit(xtrain,ytrain)

pred=lr.predict(xtest)

accu=accuracy\_score(pred,ytest)

if accu>Maxr2:

Maxr2=accu

BestRs=i

print("with random state as",BestRs,"max accuracy is",Maxr2)

with random state as 56 max accuracy is 0.8223791973791974

So maximum accuracy score is .82237 which mean 82.237% accuracy at random state 56.

Following are the metrics we’ll use to evaluate our predictive accuracy:

* Sensitivity = True Positive Rate (TP/TP+FN) – It says, ‘out of all the positive (majority class) values, how many have been predicted correctly’.
* Specificity = True Negative Rate (TN/TN +FP) – It says, ‘out of all the negative (minority class) values, how many have been predicted correctly’.
* Precision = (TP/TP+FP)
* Recall = Sensitivity
* F score = 2 \* (Precision \* Recall)/ (Precision + Recall) – It is the harmonic mean of precision and recall. It is used to compare several models side-by-side. Higher the better.

**Logistic Regression Model**

So we will fit logistic regression model with random state=56

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=.30,random\_state=56)

lr=LogisticRegression()

lr.fit(xtrain,ytrain)

pred=lr.predict(xtest)

accu=accuracy\_score(pred,ytest)

print(classification\_report(pred,ytest))

print(accu)

print(confusion\_matrix(pred,ytest))

precision recall f1-score support

0.0 0.94 0.85 0.89 8318

1.0 0.44 0.69 0.53 1450

accuracy 0.82 9768

macro avg 0.69 0.77 0.71 9768

weighted avg 0.86 0.82 0.84 9768

accuracy score = 0.8223791973791974

confusion matrix = [[7036 1282]

[ 453 997]]

Outcome of the logistic regression model gave us accuracy score (0.822) and f1-score as .82

**Random Forest Model**

So, fitting a Random Forest model with random state 56

from sklearn.ensemble import RandomForestClassifier

RFC=RandomForestClassifier()

RFC.fit(xtrain,ytrain)

pred=RFC.predict(xtest)

accu=accuracy\_score(pred,ytest)

print(classification\_report(pred,ytest))

print(accu)

print(confusion\_matrix(pred,ytest))

precision recall f1-score support

0.0 0.93 0.89 0.91 7764

1.0 0.64 0.73 0.68 2004

accuracy 0.86 9768

macro avg 0.78 0.81 0.80 9768

weighted avg 0.87 0.86 0.86 9768

accuracy score = 0.8606674856674856

confusion matrix = [[6946 818]

[ 543 1461]]

Outcome of the Random forest classifier model gave us accuracy score (0.86) and f1-score is .86

**Decision Tree Model**

So, fitting a Decision Tree model with random state 56

from sklearn.tree import DecisionTreeClassifier

dtc=DecisionTreeClassifier()

dtc.fit(xtrain,ytrain)

pred=dtc.predict(xtest)

accu=accuracy\_score(pred,ytest)

print(classification\_report(pred,ytest))

print(accu)

print(confusion\_matrix(pred,ytest))

precision recall f1-score support

0.0 0.87 0.89 0.88 7323

1.0 0.63 0.59 0.61 2445

accuracy 0.81 9768

macro avg 0.75 0.74 0.74 9768

weighted avg 0.81 0.81 0.81 9768

accuracy score = 0.812039312039312

confusion matrix = [[6488 835]

[1001 1444]]

Outcome of the Decision Tree Classifier model gave us accuracy score (0.812) and f1-score as .81

**Support Vector Machine Model**

So, fitting a support vector machine classifier model with random state 56

from sklearn.svm import SVC

svc=SVC()

svc.fit(xtrain,ytrain)

pred=svc.predict(xtest)

accu=accuracy\_score(pred,ytest)

print(classification\_report(pred,ytest))

print(accu)

print(confusion\_matrix(pred,ytest))

precision recall f1-score support

0.0 1.00 0.77 0.87 9768

1.0 0.00 0.00 0.00 0

accuracy 0.77 9768

macro avg 0.50 0.38 0.43 9768

weighted avg 1.00 0.77 0.87 9768

accuracy score = 0.7666871416871417

confusion matrix = [[7489 2279]

[ 0 0]]

Outcome of the support vector classifier model gave us accuracy score (0.766) and f1-score ( .77)

**Cross validation**

Cross Validation is a technique for assessing how the statistical analysis generalises to an independent dataset. It is a technique for evaluating machine learning models by training several models on subsets of the available input data and evaluating them on the complementary subset of the data

For a k fold cross-validation, k refers to the number of groups a given data sample is to be split into and the procedure is often called k-fold cross-validation. When a specific value for k is chosen, then it may be used such as k=5

from sklearn.model\_selection import cross\_val\_score

model=[lr,dtc,RFC,svc]

for model in model:

print(cross\_val\_score(model, x, y, cv=5).mean())

cross\_val\_score for logistic regression = 0.8126535626535626

cross\_val\_score for decision tree = 0.8073095823095823

cross\_val\_score for random forest = 0.8570331695331694

cross\_val\_score for support vector = 0.7591830466830467

After comparing the accuracy\_score and cross\_val\_score for logistic regression, decision tree classifier, random forest classifier and support vector classifier, it can be noticed that difference in accuracy\_score and cross\_val\_score for random forest classifier is lowest implying that random forest classifier model is more accurate for prediction.

So we will proceed with random forest classifier and will try to tune hyperparameter using GridSearchCV

**Hyperparameter tuning**

Hyperparameter is a parameter which control the learning rate. Hyperparameters tuning is crucial as they control the overall behavior of a machine learning model. Every machine learning models will have different hyperparameters that can be set. A hyperparameter is a parameter whose value is set before the learning process begins

Tuning the parameters of the Random Forest in order to obtain the best possible parameters for model building.

from sklearn.model\_selection import GridSearchCV

parameter={"n\_estimators":np.arange(10,100),"max\_depth":np.arange(2,10),"criterion":["gini", "entropy"],"max\_features":["auto","sqrt","log2"]}

grid=GridSearchCV(estimator=RFC,param\_grid=parameter,cv=5)

grid.fit(xtrain,ytrain)

print(grid.best\_score\_)

print(grid.best\_params\_)

Best score = 0.8548175493632748

Best parameter = {'criterion': 'gini', 'max\_depth': 9, 'max\_features': 'auto', 'n\_estimators': 34}

Using gridsearchcv best parameters are identified which can used to develop final random forest classifier model.Best score which is obtained using the best parameters is .85481.

So we will be developing final model using the best parameters.

{'criterion': 'gini', 'max\_depth': 9, 'max\_features': 'auto', 'n\_estimators': 34}

**Final Model**

finalrfc=RandomForestClassifier(n\_estimators=34,criterion='gini',max\_depth=9,max\_features='auto')

finalrfc.fit(xtrain,ytrain)

pred=finalrfc.predict(xtest)

accu=accuracy\_score(pred,ytest)

print(classification\_report(pred,ytest))

print(accu)

print(confusion\_matrix(pred,ytest))

precision recall f1-score support

0.0 0.96 0.88 0.92 8199

1.0 0.56 0.81 0.66 1569

accuracy 0.87 9768

macro avg 0.76 0.84 0.79 9768

weighted avg 0.89 0.87 0.87 9768

0.8654791154791155

[[7187 1012]

[ 302 1267]]

Final model which is finalrfc has an accuracy\_score of .8654 and f1\_score of .87. Now we will be saving the final model and 87% accuracy means that it can be predicted that a person has a salary above $50K or lower than $50 K with 87% accuracy.

**Saving the final model**

Saving the model using joblib library

import joblib

joblib.dump(finalrfc,"census\_income\_pred.obj")

**Concluding Remarks:**

Here by using sklearn, we have built a preliminary machine learning tool and we have used the accuracy\_score to select the random forest classifier model as the best model for predicting the person’s salary whether it is above $50 K or below $50 K. Our final model has an accuracy of 87% using the parameter tuning. We can further try to improve the model prediction by working on preprocessing and roc\_auc\_score. Of course, there is always tools and analysis you can do further in order to make it more accurate, and better to use.