Presented by -

- Nidhi Warde - 17ucs099
- Sheetal Chaturvedi - 17ucs148
- Sivleen Kaur - 17ucc057
- Taruna Agarwal - 17ucs170

Contribution of Team Members -

17ucs099 & 17ucs170 - Code Implementation and Report Making

17ucs148 & 17ucc057 - Took screenshots of codes

Introduction to Data Science

Classification Project on Adult Dataset

6th December 2019

Overview

In this project, we develop accurate classifiers for data analysis of a census dataset composed of independent attributes and one target variable i.e., dependent variable and the outcome which tells us whether income exceeds $50,000/year based on census data.

Goal

To build a Machine Learning model that accurately predicts whether a person makes over $50K per year or not.

About dataset

The dataset was originally Extracted by Barry Becker from the 1994 Census database. For this project,it is collected from UCI MAchinelearning repository. The objective of the dataset is to predict whether a person makes over 50K a year or not based on certain attribute measurements included in the dataset.Several constraints were placed on the collection of these instances from a larger database.A set of reasonably clean records was extracted using the following conditions:

((Age>16) and (Final Weight>1) and (Hours per week > 0))

1.Dataset Description

The dataset consists of :

Data Set Characteristics:	Multivariate	**Number of Instances:**	48842	**Area:**	Social
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	14	**Date Donated**	1996-05-01
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	1665762

Dataset feature description:

Attributes	**Type of attributes**
age	Continuous
workclass	Categorical - nominal
fnlwgt	Continuous
education	Categorical
education-num	Continuous
marital-status	Categorical
occupation	Categorical - nominal
relationship	Categorical - nominal
race	Categorical - nominal
sex	Categorical - nominal
capital-gain	Continuous
capital-loss	Continuous
hours-per-week	Continuous
native-country	Categorical - nominal

Description of Categorical Attributes

- Workclass : Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
    - Individual work category
- Education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
    - Individual's highest education degree
- Marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
    - Individual marital status
- Occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-operation-inspector, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
    - Individual's occupation
- Relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
    - Individual's relation in a family
- Race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
    - Race of Individual
- Sex: Female, Male.
- Native-country: 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 andTobago, Peru, Hong, Holland, Netherlands.
    - Individual's native country

Description of Continuous Attributes

- Age: continuous.
    - Age of an individual

- Final Weight: final weight, continuous.
    - The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau.
- Capital-gain: continuous.
- Capital-loss: continuous.
- Hours-per-week: continuous.
    - Individual's working hour per week

2. Fetching Data

Importing Packages and Libraries

```
In [99]:
import numpy as np
import pandas as pd
import statistics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.decomposition import PCA as sklearnPCA
from sklearn import tree
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import precision_score,recall_score,confusion_matrix,f1_score
from sklearn import metrics
import itertools
import time
```

Fig1.Import Packages and libraries

Import Data using panda.csv() function.

```
In [14]: ds = pd.read_csv("C:/Users/TARUNA AGRAWAL/Desktop/ids project/adult.csv")
         print(ds.shape)
         ds.head()

         (48842, 15)
```

Out[14]:

	age	workclass	final weight	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K

Fig2. Top five row of dataset

At a first glance of our dataset, we see that missing values are present in the form of '?' in 'workclass' , 'occupation' , 'native-country'. Hence, we need to deal with the missing values before doing any further pre processing.

```
In [3]: ds.shape

Out[3]: (48842, 15)
```

The above figure shows that we have **48842** observation and **15** attributes including target attribute(income).

3. Data Cleaning

Data cleaning is the process of detection and correction of inaccurate records from a data set, table, or database and refers to identify incomplete, incorrect, inaccurate or irrelevant parts of the data and then replace, modife, or delete the dirty data.

Finding missing values

```
In [15]: #find number of instances with missing values in respective attributes
         (ds[['age','workclass','final weight','education','educational-num','marital-status','occupation','relationship','race'
```

```
Out[15]: age                0
         workclass       2799
         final weight       6
         education          0
         educational-num    0
         marital-status     0
         occupation      2809
         relationship       0
         race               0
         gender             0
         capital-gain       0
         capital-loss       0
         hours-per-week     0
         native-country   857
         dtype: int64
```

Figure 3: Count of missing values

From the above figure we observed that some of the values are missing. The attributes which have missing values in the dataset are: workclass, occupation and native country with the non zeros values written in their cells as shown in figure 3.

We counted the number of rows from above mentioned attributes with missing values and replaced them with NaN to make them identifiable, so that they can be dealt with, in future. Refer figure 4.

```
In [17]: #fill missing values of relevant attributes with NaNs to make them identifiable
         ds[['workclass','final weight','occupation','native-country']] = ds[['workclass','final weight','occupation','native-co
         ds.head()
```

Out[17]:

	age	workclass	final weight	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	NaN	103497	Some-college	10	Never-married	NaN	Own-child	White	Female	0	0	30	United-States	<=50K

Figure 4: Missing values of attributes are filled with NANs to make the identifiable.

Fixing the common NaN values

```
In [18]: #missing values of age attribute are set to the average of all values of that attribute
         ds['age'].fillna((ds['age'].mean()), inplace=True)

         #missing values of workclass attribute are set to the mode of all values of that attribute
         ds=ds.fillna(ds['workclass'].value_counts().index[0])

         #missing values of occupation attribute are set to the mode of all values of that attribute
         ds=ds.fillna(ds['occupation'].value_counts().index[0])

         #missing values of native-country attribute are set to the mode of all values of that attribute
         ds=ds.fillna(ds['native-country'].value_counts().index[0])
```

Figure 5: Code for replacing NaNs with the mean of all values of that attribute

Here we have replaced the missing values of the continuous attributes by their mean and the categorical attributes are replaced with the most repeated value corresponding to that attribute. Refer figure 5.

From figure 6, it can be seen that now none of the attributes have any of missing values. Hence, data cleaning is successful.

```
         #final check to see that none of the attributes have missing values
         (ds[['age','workclass','final weight','education','educational-num','marital-status','occupation','relationship','race'

Out[18]: age               0
         workclass         0
         final weight      0
         education         0
         educational-num   0
         marital-status    0
         occupation        0
         relationship      0
         race              0
         gender            0
         capital-gain      0
         capital-loss      0
         hours-per-week    0
         native-country    0
         dtype: int64
```

Figure 6:Again checking if all the missing values are replaced

Converting the target class attribute from categorical to binary attribute, so that further processing becomes simple and less cumbersome. Refer figure 7.

```
In [19]: #representing income class label of <=50 and >50 as 0 and 1 respectively
         ds['income']=ds['income'].map({'<=50K': 0, '>50K': 1, '<=50K.': 0, '>50K.': 1})
         ds.head()
```

Out[19]:

	age	workclass	final weight	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	0
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	0
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	1
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	1
4	18	Private	103497	Some-college	10	Never-married	Private	Own-child	White	Female	0	0	30	United-States	0

Figure 7: Replacing income <=50K as 0 and income>50K as 1

4. Dataset Statistics

Figure 8 gives the description of statistics like total count, mean and various percentiles of continuous attributes.

```
In [16]: #describe statistics of continuous attributes
         ds.describe()
```

Out[16]:

	age	educational-num	capital-gain	capital-loss	hours-per-week
count	48842.000000	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	10.078089	1079.067626	87.502314	40.422382
std	13.710510	2.570973	7452.019058	403.004552	12.391444
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	48.000000	12.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	4356.000000	99.000000

Figure 8: Basic statistics of the dataset

5. Data Visualization

```
In [20]: #visulaisation of target class (income label)
class_0=(ds[['income']]==0).sum()
class_1= (ds[['income']]==1).sum()
outcome = [int(class_0),int(class_1)]
label = ['Income <=50', 'Income>50']
colors = ['g', 'r']
plt.pie(outcome, labels=label, colors=colors, startangle=90, autopct='%.2f%%')
plt.show()
```

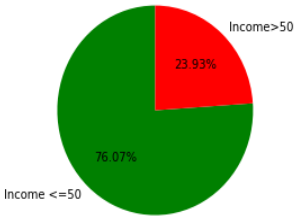


Figure 9: Pie chart for income <=50K and >50K

From figure 9, we can see that Income level less than 50K is more than 3 times of those above 50K, indicating that the dataset is somewhat skewed. However, since there are very very few records data on the upper limit of adult's income above 50K, it's premature to conclude that the total amount of wealth are skewed towards greater income group.

Exploratory Data Analysis

Univariate analysis

1. Age

```
In [21]: #histogram for age
ds['age'].hist(figsize=(10,5))
plt.show()
```

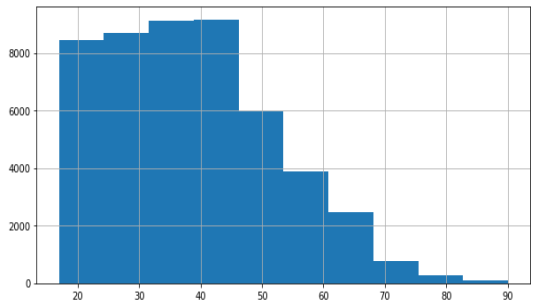


Figure 10: Histogram for age representation

The above histogram summarise that :

1. 'Age' attribute is non symmetric that is distribution is not normal.
2. It is right-skewed. But this is not a concern as younger adults earn not the aged ones.
3. Minimum and Maximum age of the people is 17 and 90 respectively.

4. Figure 11 concludes that this dataset has fewer observations (only 869 data records) of people's age, after a certain age of 70 years.

```
In [26]: #number of instances which are outliers with respect to age
         ds[ds['age']>70].shape

Out[26]: (868, 15)
```

Figure 11: Outliers with respect to age

Hours per week

```
In [23]: #histogram for hours per week
         ds['hours-per-week'].hist(figsize=(10,5))
         plt.show()
```

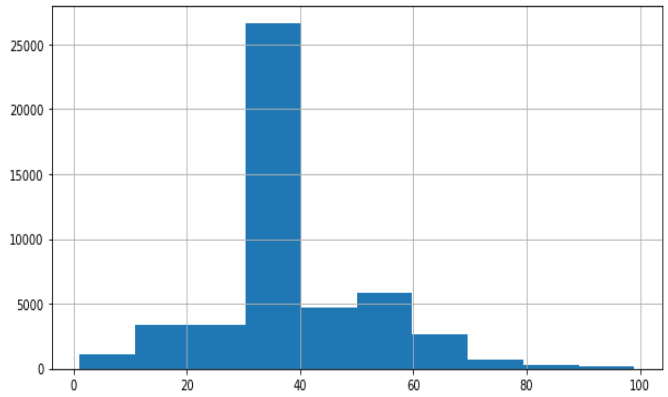


Figure 11: Histogram for hours per week

This histogram of "hours-per-week" summarises that:

1. This attribute varies within a range of 1 to 99.
2. Many people work in range 30-40 hours per week, they are around 27,500 people.
3. There are very few people who work around 80 hours per week and some less than 25 hours which is unusual.
4. Around ⅔ of people spend 40 or less working hours per week.

Capital-gain

```
In [24]: #histogram for capital gain
         ds['capital-gain'].hist(figsize=(10,5))
         plt.show()
```

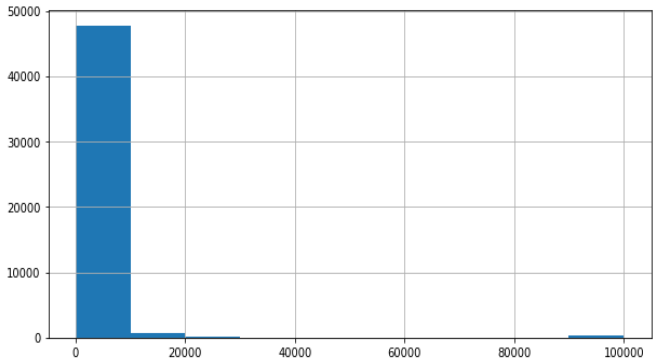


Figure 12: Histogram for capital gain

```
In [25]: #number of instances which are outliers with respect to capital gain
         ds[ds['capital-gain']>10000].shape

Out[25]: (1134, 15)
```

Figure 13: Outliers with respect to capital gain

1. This histogram shows that most of the "capital-gain" values are centered on 0 and very few between 10k and 99k.
2. capital-gain is concentrated on one particular value and other are spread with large standard deviation(7452.5).
3. Also it shows that either a person has no gain or has gain of very large amount(10k or 99k).
4. Figure 13. shows the outliers in capital gain, because only few records (1134) have capital gain less than 10000.

Capital-loss

1. This histogram shows that most of the "capital-loss" values are centered on 0 and only few are non zero(2280).
2. Most of the values are centered on 0 around 43000 instances.
3. Number of instances having negative values. The one’s facing loss in capital.

```
In [27]: #histogram for capital loss
ds['capital-loss'].hist(figsize=(10,5))
plt.show()
```

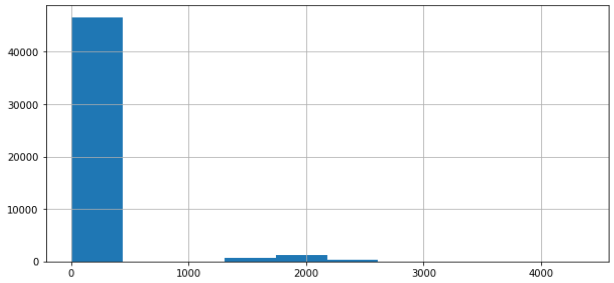


Figure 14: Histogram for capital loss

```
In [30]: #number of instances having non zero capital-loss values
ds[ds["capital-loss"]>0].shape
```

```
Out[30]: (2282, 15)
```

Figure 15: Number of values having non zero capital loss value

Relation between capital gain and capital loss:

```
In [33]: #relationship between gain and loss in capital
sns.relplot('capital-gain','capital-loss', data= ds)
plt.xlabel('Gain in capital')
plt.ylabel('Loss in capital')
plt.show()
```

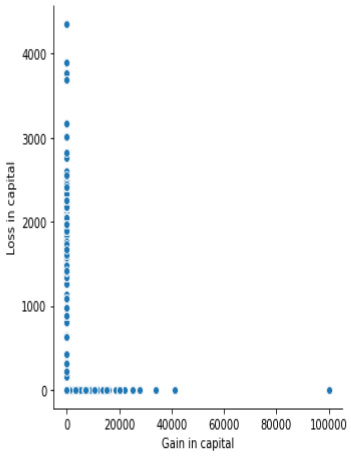


Figure 16: Relationship between gain and loss in capital

Summary of relationship between capital gain and capital loss:

1. Both attributes can be zero.

2. They are the reverse of each other if one is high, the other is low.
3. There is a possibility that capital gain having high value or above near zero.

5. Gender

```
#distribution for gender
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,3))
sns.countplot(y="gender", data=ds)
```

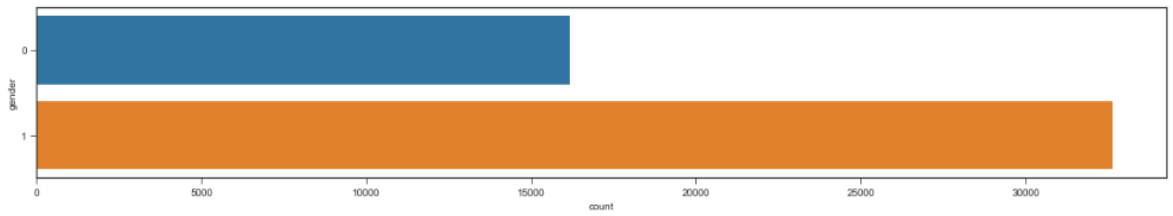


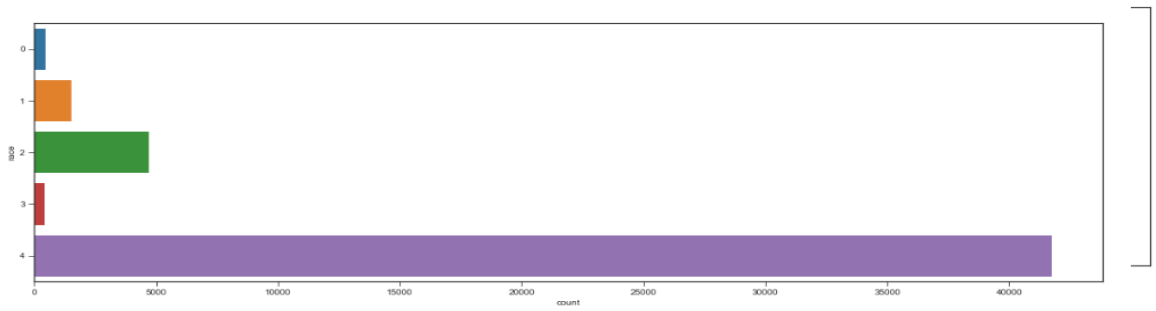
Figure 19: Bar plot for gender

This distribution explains that:

1. Gender has 2 categories male and female
2. The frequency of male is higher than the female..
3. This dataset is skewed toward the male with nearly 69%.

6. Race

```
#distribution for race
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,6))
sns.countplot(y="race", data=ds)
```



This distribution explains that:

1. There are 5 categories in the race attribute.
2. Majority of people are "white" which is roughly 85%.
3. Bias toward the "white" race.
4. 9.59% categories are of “black” race.

7. Native-country

```
#distribution for Native country
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,10))
sns.countplot(y="native-country", data=ds)
```

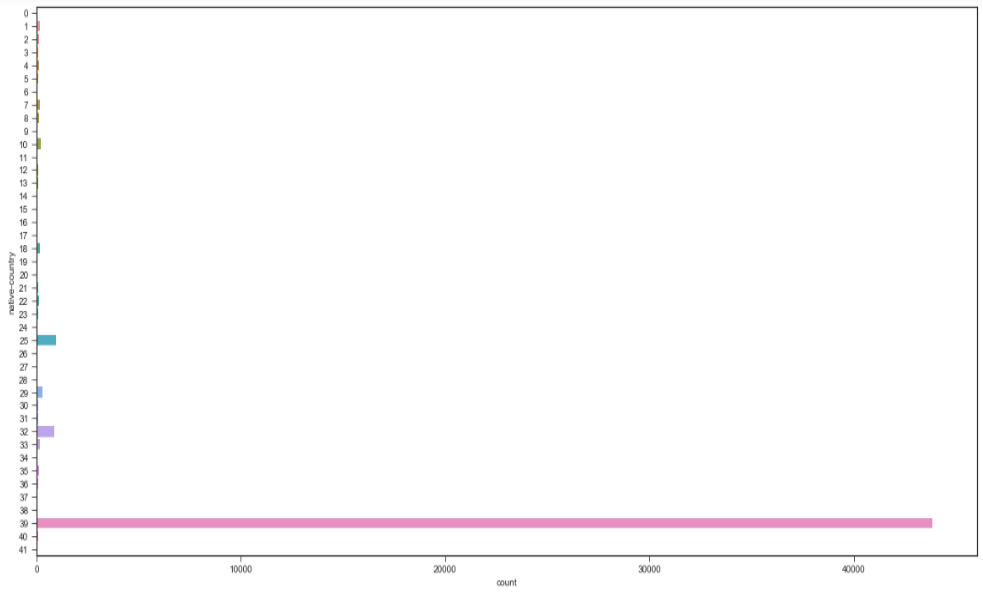


Figure 20: Native country

This distribution explains that:

1. This dataset is taken from the United States.
2. Majority of people have native country America while others are immigrants.

8. Workclass

```
#distribution for work class
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,4))
sns.countplot(y="workclass", data=ds)
```

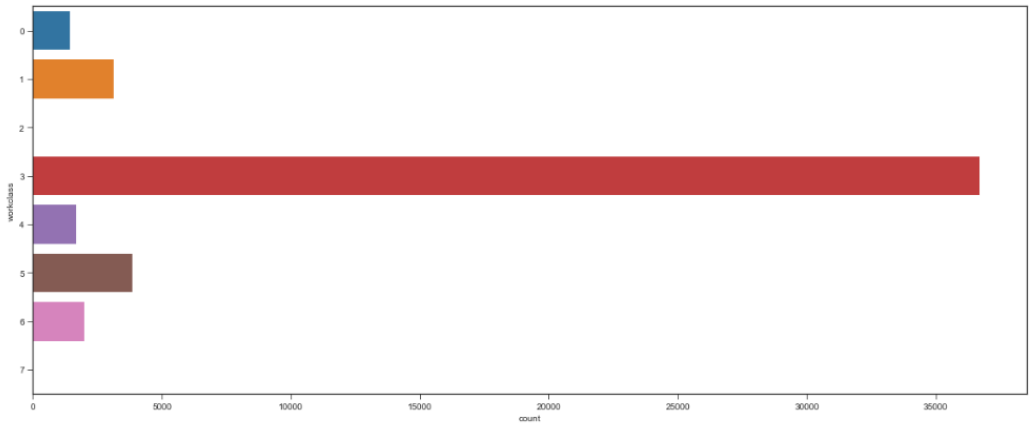


Figure 21: Work class

Distribution summarise that:

1. There are 8 unique categories present in the worclass attribute.
2. Most people belong to the private workclass(36705) around 75.15%.
3. without-pay and never-worked has minor count in workclass attribute around less than 1%.
4. There is a high imbalance among categories of workclass attribute.

9. Education

```
#distribution for education
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,8))
sns.countplot(y="education", data=ds)
```

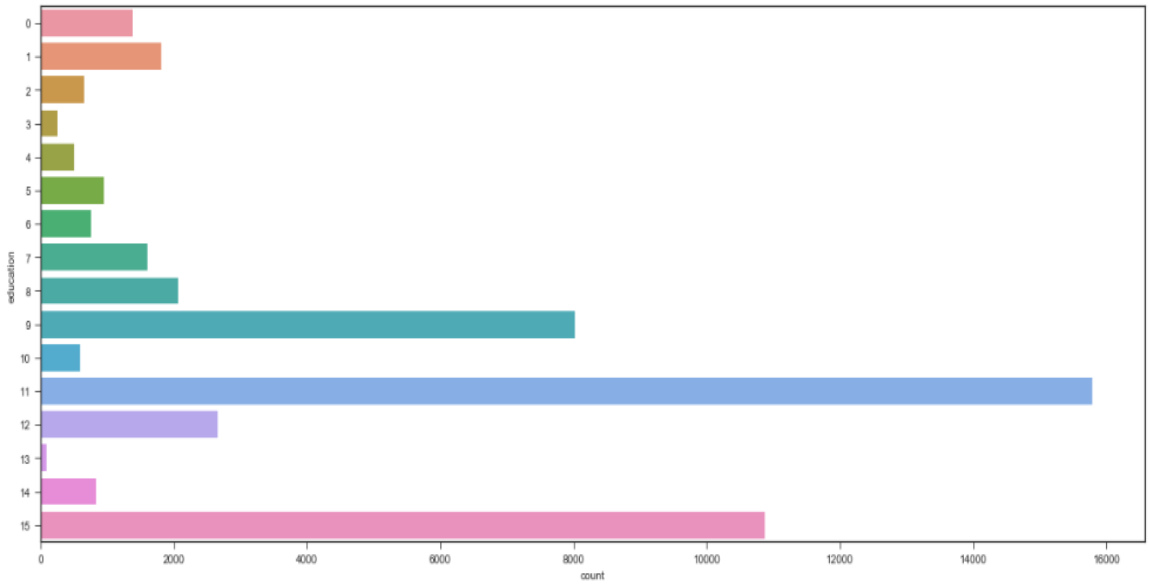
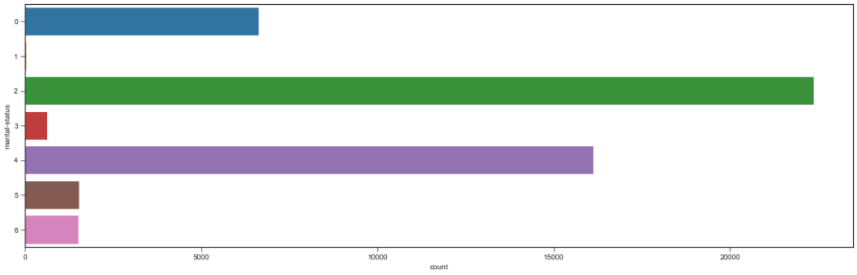


Figure 22: Education

1. There are 16 categories present in the **education** attribute.
2. Hs-grad are 31% of all education attribute.
3. HS-grad (15780) has the maximum number of instances followed by some-college(10978) and Bachelors(8024).

10. Marital-status

```
#distribution for marital status
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,6))
sns.countplot(y="marital-status", data=ds)
```



Summary of distribution:

1. This attribute attribute has 7 unique categories.

2. Two of them are dominant over other categories(these are Never-married(32%) and married-civ-spouse(45.80%).
3. Married-civ-spouse has maximum number of instances.
4. Married-AF-spouse has minimum number of observations.

Multivariate analysis

1.Multivariate analysis between "income", "age", "gender"

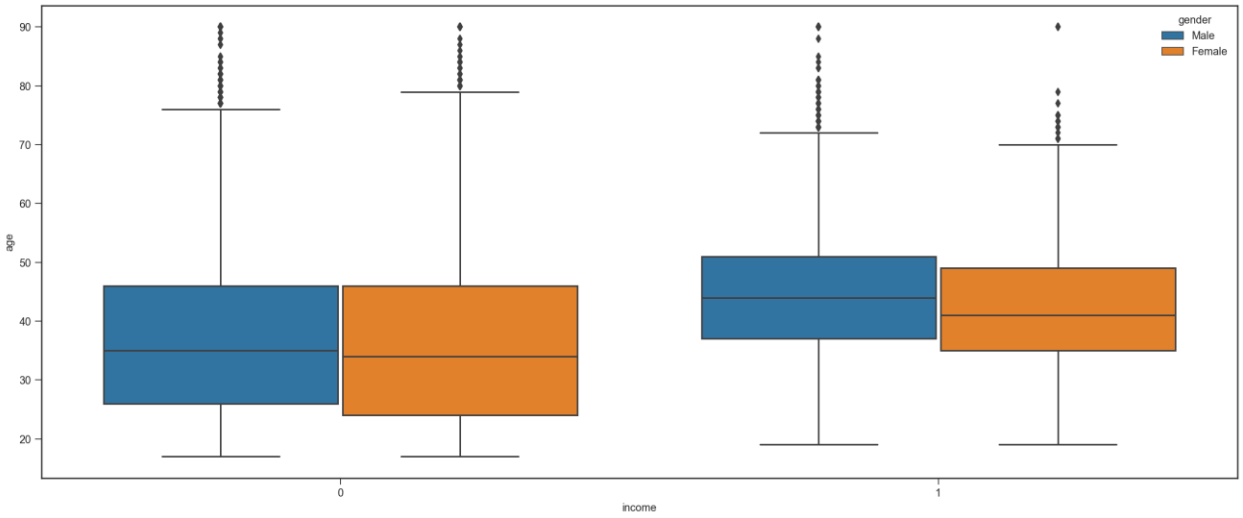


Figure 24: Multivariate analysis between income, age and gender

Multivariate analysis between attributes "income", "age", "gender" shows that:

1. Median "age" of Females who earn less than 51k has very minor difference than the Median "age" of males who earn less than 51k.
2. But the Median "age" of females who earn greater than 51k has age difference of 2-3years than the Median "age" of males who earn greater than 51k.

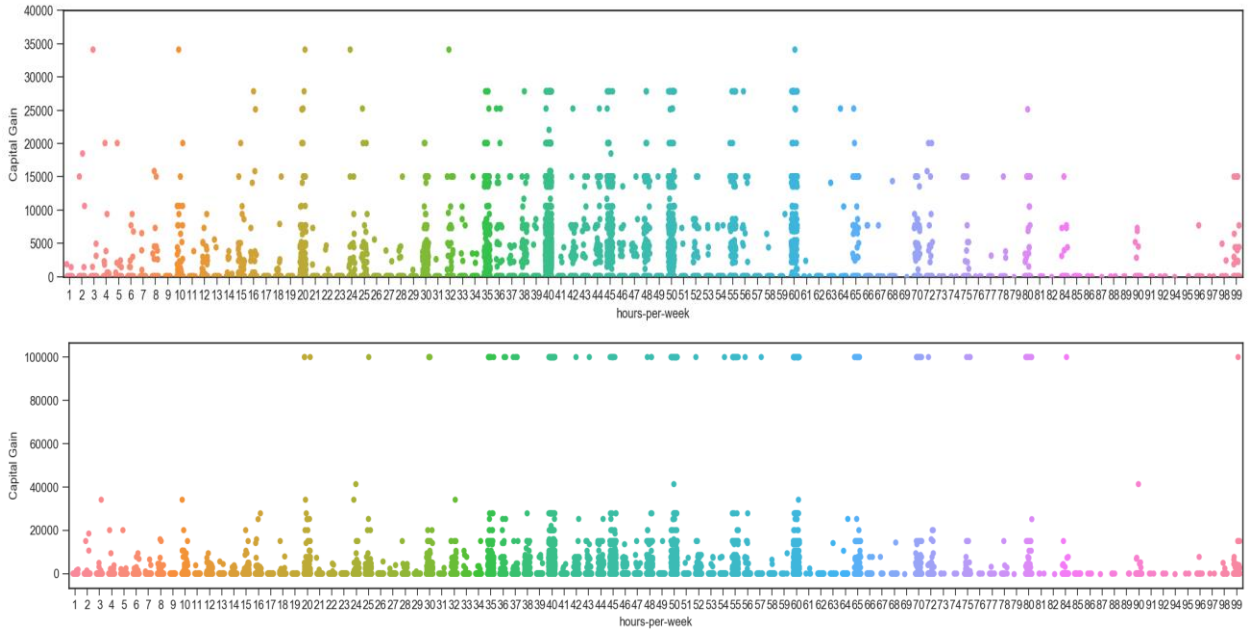
2.Other multivariate analysis

```
Out[18]:

In [20]: #multivariate anakysis between capital-gain and hours per week
figure = plt.figure(figsize = (20,8))
x = figure.add_subplot(2,1,2)
sns.stripplot('hours-per-week', 'capital-gain', data = ds,jitter = 0.25);
plt.xlabel('hours-per-week',fontsize = 10);
plt.ylabel('Capital Gain',fontsize = 10);
plt.ylim(0,40000);

figure = plt.figure(figsize = (20,8))
x = figure.add_subplot(2,1,1)
sns.stripplot('hours-per-week', 'capital-gain', data = ds,jitter = 0.25,ax = x);
plt.xlabel('hours-per-week',fontsize = 10);
plt.ylabel('Capital Gain',fontsize = 10);
```

Figure 25:Multivariate analysis between capital-gain and ga



Summary shows that:

1. Between age 29 and 63 capital gain is upto 15500 and after that it decreases and then increases at age 90.
2. Age 80 doesn't follow the pattern.
3. Capital gain of 99998 is clearly an outlier .

7. Data transformation

To fit the data into predictive model, we need to convert categorical values to numerical values.Before that, we will evaluate if any transformation on categorical columns are required. Discretisation is a common method to make categorical data more tidy and meaningful.Here, we apply discretisation to attributemarital_status. Now we can convert categorical columns to numerical ones.

```
In [21]: #convert categorical attributes to numerical
         # Feature Selection and Encoding
         ds = ds.apply(LabelEncoder().fit_transform)
         ds.head(10)
```

Out[21]:

	age	workclass	final weight	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	8	3	14385	1	6	4	6	3	2	1	0	0	39	39	0
1	21	3	27674	11	8	2	4	0	4	1	0	0	49	39	0
2	11	1	21161	7	11	2	11	0	4	1	0	0	39	39	1
3	27	3	6173	15	9	2	6	0	2	1	98	0	39	39	1

```
In [22]: ds.head()
```

Figure 26: Converting categorical attributes into numerical

Correlation between Attributes-

```
In [93]: #Correlation between attributes
         corr = ds.corr()
         plt.figure(figsize=(20, 10))
         sns.heatmap(corr, annot=True)
         plt.show()
```

The below figure represents the plot between each and every pair of attributes. By analyzing it, we conclude that they are not correlated with one another and thus they are independent.

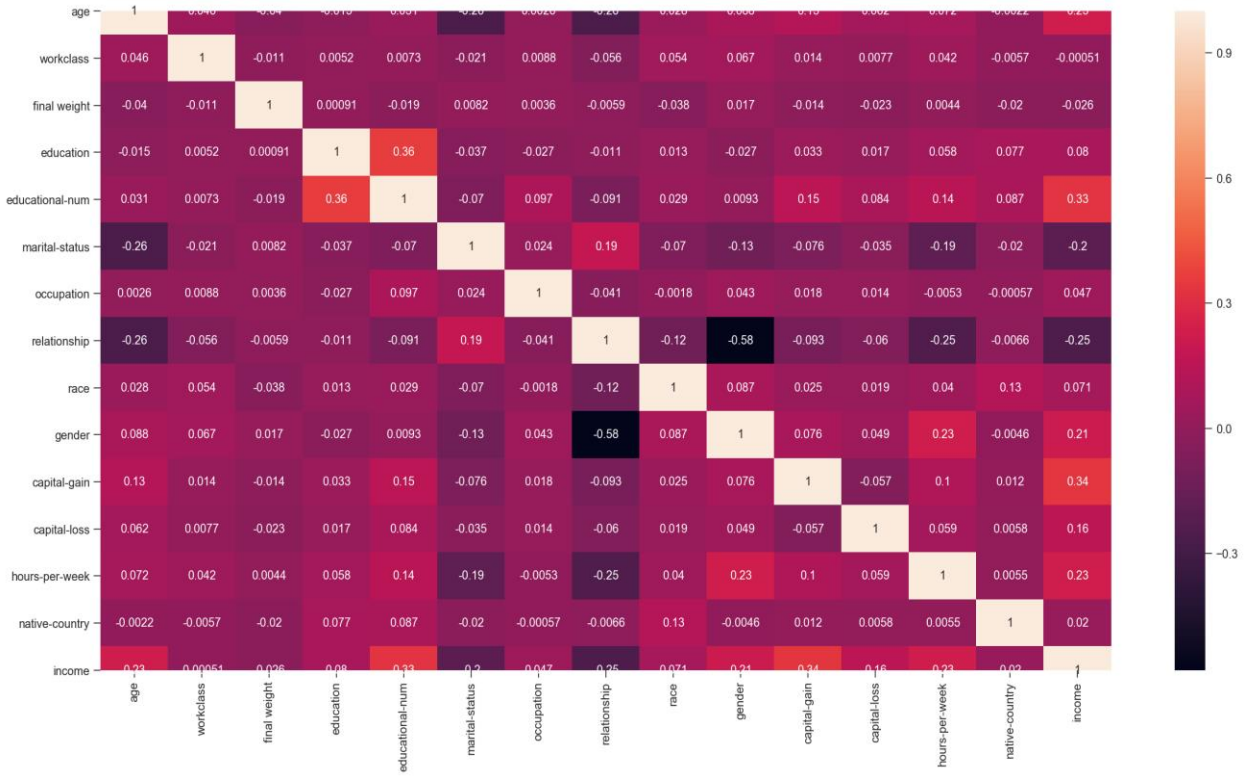


Figure 27: Correlation between the attributes

Splitting the data set into training and test set-

Wenow split the dataset into training set and test set by giving one-fifth of the data to the test set remaining to the training set.

Do this before you start scaling, normalizing the data. It's cheating otherwise since the normalization and encodings would know about values and classes in the test dataset before validation.

Normalisation-

Normalization method used in scaling the data of an attribute so that it falls in a smaller range, such as either -1.0 to 1.0 or 0.0 to 1.0. It is generally userin classification algorithms.

Normalization is needed when we deal with attributes on a different scale, otherwise, it may lead to a dilution in effectiveness of an equally important attribute(on lower scale) because other attributes might having values on larger scale.

In other words, we can say that when multiple attributes are there but attributes have values on different scales, this might lead to poor data models while performing data mining operations. So they are needed to be normalized to bring all the attributes on the same scale.

Out[23]:

age	workclass	final weigh	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gair	capital-loss	hours-per-week	native-country	income

In [24]:
```
#Splitting data into training and test set
X = ds[['age','workclass','final weight','education','educational-num','marital-status','occupation','relationship','race','gender','capital-
gain','capital-loss','hours-per-week','native-country']]
y = ds['income']
#normalisation of values
x_norm = preprocessing.normalize(X)
X_train, x_test, y_train, y_test = train_test_split(x_norm, y, test_size=0.2, random_state=2)
X_train.shape
```

Out[24]: (39073, 14)

Figure 28: Splitting dataset and data normalisation

In [127]:
```
#pie chart for income distribution in training data
income = [y_train[y_train==1].count(),y_train[y_train==0].count()]
label = ['Income <=50', 'Income >50']
colors = ['g', 'r']
plt.pie(income, labels=label, colors=colors, startangle=90, autopct='%.1f%%')
plt.show()
```

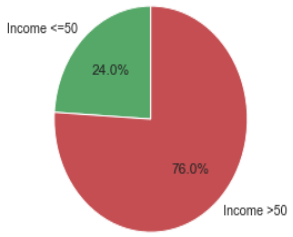


Figure 29: Income distribution in training dataset

In [27]:
```
#principal component analysis
pca = sklearnPCA(n_components=2)
pca_new = pd.DataFrame(pca.fit_transform(x_norm))
xx = pca_new[0]
yy = pca_new[1]
plt.scatter(xx[y==1], yy[y==1], label='Income <=50', c='green')
plt.scatter(xx[y==0], yy[y==0], label='Income >50', c='red')
plt.show()
```

1.969 seconds 25

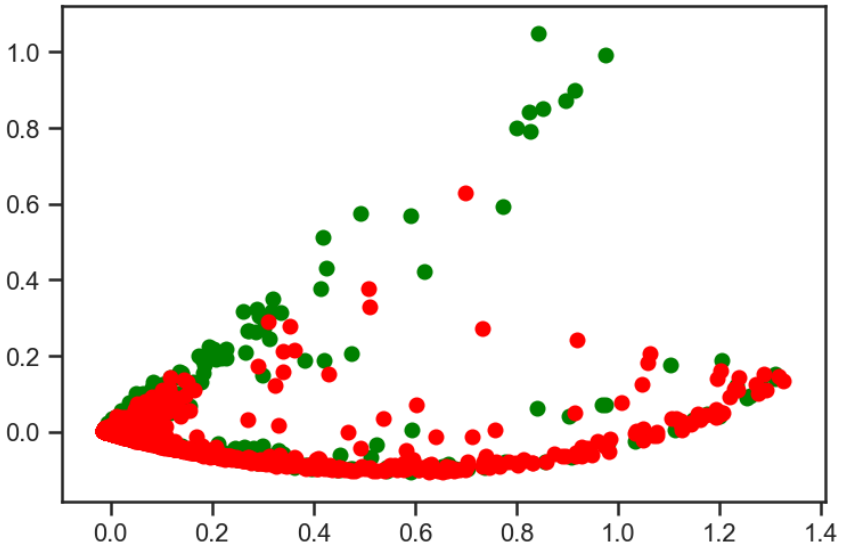


Figure 30: Principal Component Analysis

8.CLASSIFIERS AND EVALUATION MEASURES

Over the course of the last few weeks as we have been exploring the dataset, we realized that our initial assumptions had to be modified a bit in order to deal with the realities of the dataset itself. Several secondary questions emerged such with regard to effective dimensionality reduction via PCA. In the end, the process had multiple stages, including data pre-processing, sampling of training and test data and now the classification of that dataset.

PERFORMANCE AND EVALUATION MEASURES:

Confusion matrix: A confusion matrix is a form of table used to describe the performance of a classification model on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm.

The Confusion matrix corresponding to our project is as follows :

	Predicted(Income<=50)	Predicted(Income>50)
Actual(Income<=50)	True Positive(TP)	FalseNegative(FN)
Actual(Income>50)	False Positive(FP)	True Negative(TN)

TP : If income is <=50(actual) and is classified as 0 by the classifier.

TN : If income is >50(actual) and is classified as 1 by the classifier.

FP : If income is >50 (actual) and is classified as 0 by the classifier.

FN : If income is <=50(actual) and is classified as 1 by the classifier

Accuracy is one metric for evaluating classification models .

Accuracy is the fraction of predictions which are right in a model.

Accuracy=number of correct predictions/total number of predictions.

For binary classification, accuracy can be calculated in terms of positives and negatives as follows:

Accuracy=(TP+TN)/(TP+TN+FP+FN)

Our success metric for this project is to try and maximize the f-measure score:

The F1 score, commonly used in information retrieval, measures accuracy using the statistics precision (P) and recall(R). Precision is the ratio of true positives (TP) to all predicted positives (TP+FP)). Recall is the ratio of true positives to all actual positives (TP+FN). The F1 score is given by:

F1 = 2P· R/(P+R) where P=TP / (TP+FP), R=(TP / TP+FN)

The F1 metric weights recall and precision equally, and a good algorithm will maximize both precision and recall simultaneously. Thus, moderately good performance on both will be favoured over extremely good performance on one and poor performance on the other.

In [40]:
```
def plot_confusion_matrix(cm, classes, title='CONFUSION MATRIX', cmap=plt.cm.Reds):
    plt.figure()
    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)

    fmt = 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Actual Label')
    plt.xlabel('Predicted Label')
```
0.006 seconds 26

Figure 32. Structure for plotting Confusion matrix

The classifier we used in this project for classification are:

Support Vector Machine SVM is a discriminative classifier which is defined by a separating hyperplanes, given labeled training data (in supervised learning), this algorithm results an optimal hyperplane which

categorizes new examples. Since we have a 2-D plane so it gives a line which divides the plane into two parts where each class lay in either side. Figure 32. is showing the data validation by SVM classifier.

```
In [15]: start_time = time.time()
         clf1 = SVC(kernel='rbf', gamma=0.1, C=1)
         clf1.fit(X_train, y_train)
         pred1 = clf1.predict(x_test)
         sc1 = clf1.score(x_test,y_test)
         f1 = f1_score(y_test,pred1)
         print('Accuracy: '+str(sc1))
         print('F-score '+str(f1))
         cm = confusion_matrix(y_test,pred1,labels=[0,1])
         plot_confusion_matrix(cm,classes=[0,1])
         print("--- %s seconds ---" % (time.time() - start_time))
```

```
Accuracy: 0.765277919950865
F-score 0.005206073752711497
[[7470    4]
 [2289    6]]
--- 85.05261635780334 seconds ---
```

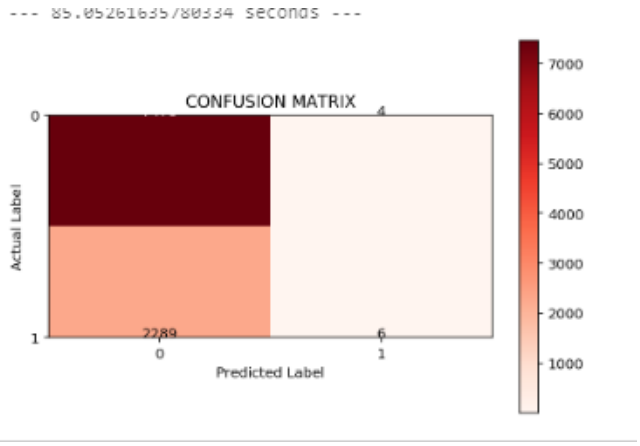


Figure 32. Code and Confusion matrix for SVM classifier

Naive Bayes Classifier It is a supervised machine-learning algorithm that uses Bayes' Theorem, which assumes that attributes are statistically independent. This theorem relies on the naive assumption that attributes are independent of each other, so there is no way to know anything about other variables when given an additional variable. Regardless of this assumption, it has proven itself to be a classifier with good results. Refer Figure 33. For results-

```
In [17]: clf2 = GaussianNB()
         clf2.fit(X_train, y_train)
         pred2 = clf2.predict(x_test)
         sc2 = clf2.score(x_test,y_test)
         f2 = f1_score(y_test,pred2)
         print('Accuracy: '+str(sc2))
         print('F-score '+str(f2))
         cm = confusion_matrix(y_test,pred2,labels=[0,1])
         plot_confusion_matrix(cm,classes=[0,1])
         print("--- %s seconds ---" % (time.time() - start_time))
```

```
Accuracy: 0.7631282628723514
F-score 0.09254901960784315
[[7337  137]
 [2177  118]]
--- 302.62344336509705 seconds ---
```

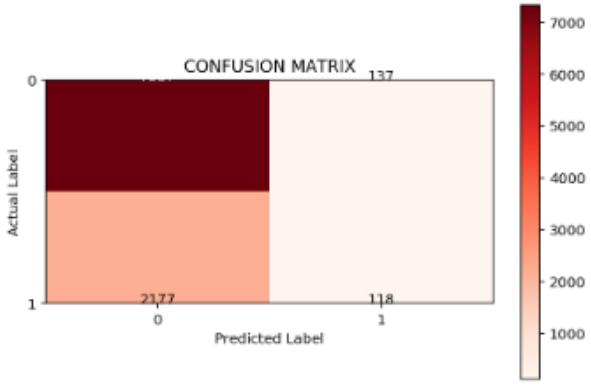


Figure 33. Code and Confusion matrix for Naive Bayes classifier

Random Forest Classifier-It is an ensemble algorithm. It creates a set of decision trees from randomly selected subset of training sets. Afterwards it aggregates the votes from different decision trees to decide the final class of the test record.

```
In [18]: start_time = time.time()
clf3 = RandomForestClassifier(max_depth=6, n_estimators=8, random_
clf3.fit(X_train,y_train)
pred3 = clf3.predict(x_test)
sc3 = clf3.score(x_test,y_test)
f3 = f1_score(y_test,pred3)
print('Accuracy: '+str(sc3))
print('F-score '+str(f3))
cm = confusion_matrix(y_test,pred3,labels=[0,1])
plot_confusion_matrix(cm,classes=[0,1])
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Accuracy: 0.8102159893540792
F-score 0.37826961770623746
[[7351  123]
 [1731  564]]
--- 0.6420607566833496 seconds ---
```

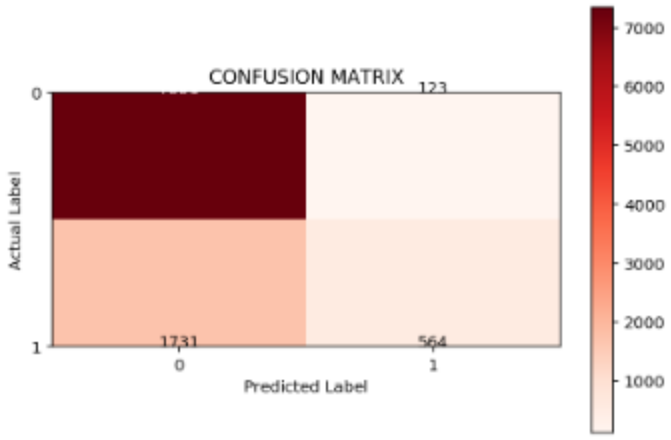


Figure 34. Code Confusion Matrix for Random Forests Classifier

Comparison Between the classifiers w.r.t. their Accuracy and F score:

	SVM	NAIVE BAYES	RANDOM FOREST
Accuracy	76.522%	76.312%	81.02%
F score	0.005	0.0925	0.3782
Time of Execution (in milisecond)	85.026	302.65	0.6420

Accuracy : Random Forest > SVM > Naive Bayes

F-score : Naive Bayes > Random Forest > SVM

Majority Voting

After classifying the dataset we have applied maximum voting(Hard).The simplest case of majority voting is hard voting. Here, we have predicted the class label through majority voting of each classifier :

We combined the three classifiers that classify our training sample as follows:

● SVM (classifier 1) - Income <= 50K (class 0)

● Naive Bayes (classifier 2) - Income <=50K (class 0)

● Random Forest (classifier 3) - Income >50K (class 1)

Via majority vote, we would classify as Income <=50K.

```
In [19]: #max_voting
         start_time = time.time()
         y_test1 = np.array(y_test)
         count=0
         final_pred = np.array([])
         for i in range(0,len(x_test)):
             final_pred = np.append(final_pred, statistics.mode([pred1[i],pred2[i],pred3[i]]))
         for i in range(len(final_pred)):
             if final_pred[i]==y_test1[i]:
                 count+=1
         sc=count/(i)
         print('Accuracy: '+str(sc))
         f = f1_score(y_test,final_pred)
         print('F-measure: '+str(f))
         cm = confusion_matrix(y_test,final_pred,labels=[0,1])
         plot_confusion_matrix(cm,classes=[0,1])
         print("--- %s seconds ---" % (time.time() - start_time))
```

```
Accuracy: 0.8102159893540792
F-score 0.37826961770623746
[[7351  123]
 [1731  564]]
--- 0.6420607566833496 seconds ---
```

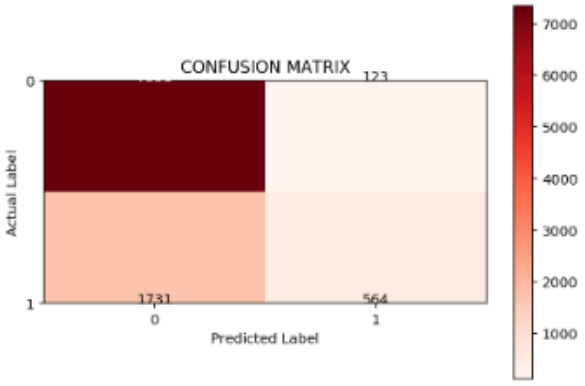


Figure 35. Max Voting for the above three classifiers

9.Final analysis

In the final analysis, we have learned a great deal about the dataset that we collect for this project. We had a few questions at the start of the project, of which the most important was: “Can we build a machine learning model to accurately predict whether the income exceeds $50K/yr based on census data or not”.

The answer to this question is Yes, provided that class attribute is balanced, if we have class imbalance problem then we need to do stratified cross validation(sampling) for training and test data at the time of splitting the dataset and then apply the classifiers.

This project, rather than advancing new theories about data analysis, was more about maximizing a particular success metric in a competitive context. Nevertheless, we gained a significant insight into the nature of different classification schemes and are now well-equipped to continue tackling related challenges in the domain of classification in future.

10.Code for reference :

```
import numpy as np

import pandas as pd

import statistics

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.decomposition import PCA as sklearnPCA

from sklearn import tree

import warnings

warnings.filterwarnings("ignore")

from sklearn.metrics import precision_score,recall_score,confusion_matrix,f1_score

from sklearn import metrics

import itertools
```

```
import time

ds = pd.read_csv("C:/Users/TARUNA AGRAWAL/Desktop/ids project/adult.csv")

print(ds.shape)

ds.head()


#find number of instances with missing values in respective attributes

(ds[['age','workclass','final weight','education','educational-num','marital-
status','occupation','relationship','race','gender','capital-gain','capital-loss','hours-per-week','native-
country']]=='?').sum()

#describe statistics of continuous attributes

ds.describe()


# frequency for categorical attributes

columns =['workclass', 'education','marital-status','relationship', 'race','gender', 'native-country','occupation',
'income']

for i in columns:

        print (i)

        print (ds[i].value_counts())


#fill missing values of relevant attributes with NaNs to make them identifiable

ds[['workclass','final weight','occupation','native-country']] = ds[['workclass','final weight','occupation','native-
country']].replace('?',np.NaN)

ds.head()


#missing values of age attribute are set to the average of all values of that attribute

ds['age'].fillna((ds['age'].mean()), inplace=True)
```

```
#missing values of workclass attribute are set to the mode of all values of that attribute

ds=ds.fillna(ds['workclass'].value_counts().index[0])


#missing values of occupation attribute are set to the mode of all values of that attribute

ds=ds.fillna(ds['occupation'].value_counts().index[0])


#missing values of native-country attribute are set to the mode of all values of that attribute

ds=ds.fillna(ds['native-country'].value_counts().index[0])


#final check to see that none of the attributes have missing values

(ds[['age','workclass','final weight','education','educational-num','marital-
status','occupation','relationship','race','gender','capital-gain','capital-loss','hours-per-week','native-
country']]==np.NaN).sum()


#representing income class label of <=50 and >50 as 0 and 1 respectively

ds['income']=ds['income'].map({'<=50K': 0, '>50K': 1, '<=50K.': 0, '>50K.': 1})

ds.head()


#visualisation of target class (income label)

class_0=(ds[['income']]==0).sum()

class_1= (ds[['income']]==1).sum()

outcome = [int(class_0),int(class_1)]

label = ['Income <=50', 'Income>50']

colors = ['g', 'r']

plt.pie(outcome, labels=label, colors=colors, startangle=90, autopct='%.2f%%')

plt.show()
```

```
#histogram for age

ds['age'].hist(figsize=(10,5))

plt.show()


#number of instances which are outliers with respect to age

ds[ds['age']>70].shape


#histogram for hours per week

ds['hours-per-week'].hist(figsize=(10,5))

plt.show()


#histogram for capital gain

ds['capital-gain'].hist(figsize=(10,5))

plt.show()


#number of instances which are outliers with respect to capital gain

ds[ds['capital-gain']>10000].shape


#histogram for capital loss

ds['capital-loss'].hist(figsize=(10,5))

plt.show()


#number of instances having non zero capital-loss values
```

```
ds[ds["capital-loss"]>0].shape


#relationship between gain and loss in capital
sns.relplot('capital-gain','capital-loss', data= ds)
plt.xlabel('Gain in capital')
plt.ylabel('Loss in capital')
plt.show()



#distribution for relationship
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,6))
sns.countplot(y="relationship", data=ds)

#distribution for race
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,6))
sns.countplot(y="race", data=ds)

#distribution for gender
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,3))
sns.countplot(y="gender", data=ds)

#distribution for Native country
```

```
plt.style.use('seaborn-ticks')

plt.figure(figsize=(20,10))

sns.countplot(y="native-country", data=ds)


#distribution for work class

plt.style.use('seaborn-ticks')

plt.figure(figsize=(20,4))

sns.countplot(y="workclass", data=ds)


#distribution for education

plt.style.use('seaborn-ticks')

plt.figure(figsize=(20,10))

sns.countplot(y="education", data=ds)


#distribution for marital status

plt.style.use('seaborn-ticks')

plt.figure(figsize=(20,10))

sns.countplot(y="marital-status", data=ds)


#distribution for work class

plt.style.use('seaborn-ticks')

plt.figure(figsize=(20,8))

sns.countplot(y="workclass", data=ds)


#distribution for education
```

```
plt.style.use('seaborn-ticks')

plt.figure(figsize=(20,8))

sns.countplot(y="education", data=ds)


#distribution for marital status

plt.style.use('seaborn-ticks')

plt.figure(figsize=(20,6))

sns.countplot(y="marital-status", data=ds)


#convert categorical attributes to numerical

from sklearn.preprocessing import LabelEncoder

lb = LabelEncoder()

columns = ['workclass', 'education','marital-status','relationship', 'race','gender', 'native-country','occupation',
'income']

for i in columns:

        ds[i] = lb.fit_transform(ds[i])

ds.head(10)


#Correlation between attributes

corr = ds.corr()

plt.figure(figsize=(20, 10))

sns.heatmap(corr, annot=True)

plt.show()


#Splitting data into training and test set
```

```
X = ds[['age','workclass','final weight','education','educational-num','marital-status','occupation','relationship','race','gender','capital-gain','capital-loss','hours-per-week','native-country']]

y = ds['income']

x_norm=preprocessing.normalize(X)

X_train, x_test, y_train, y_test = train_test_split(x_norm, y, test_size=0.2, random_state=2)

X_train.shape


#pie chart for income distribution in training data

income = [y_train[y_train==1].count(),y_train[y_train==0].count()]

label = ['Income <=50', 'Income >50']

colors = ['g', 'r']

plt.pie(income, labels=label, colors=colors, startangle=90, autopct='%.1f%%')

plt.show()

plt.figure

#principal component analysis of attributes

sns.set(style='ticks')

sns.pairplot(ds, hue='income')

plt.show()

pca = sklearnPCA(n_components=2) #2-dimensional PCA

pca_transformed = pd.DataFrame(pca.fit_transform(X_norm))

xax = pca_transformed[0]

yax = pca_transformed[1]

plt.scatter(xax[y==1], yax[y==1], label='Diabetic', c='red')

plt.scatter(xax[y==0], yax[y==0], label='Non-diabetic', c='blue')

plt.show()
```

```
#structure for confusion matrix plotting
def plot_confusion_matrix(cm, classes, title='CONFUSION MATRIX', cmap=plt.cm.Reds):
    plt.figure()
    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)



 #SVM classifier
 fmt = 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
     plt.tight_layout()
    plt.ylabel('Actual Label')
    plt.xlabel('Predicted Label')

#Naive Bayes classifier
clf2 = GaussianNB()
```

```
clf2.fit(X_train, y_train)

pred2 = clf2.predict(x_test)

sc2 = clf2.score(x_test,y_test)

f2 = f1_score(y_test,pred2)

print('Accuracy: '+str(sc2))

print('F-score '+str(f2))

cm = confusion_matrix(y_test,pred2,labels=[0,1])

plot_confusion_matrix(cm,classes=[0,1])

print("--- %s seconds ---" % (time.time() - start_time))


#Random forest Classifier

start_time = time.time()

clf3 = RandomForestClassifier(max_depth=6, n_estimators=8, random_state=0)

clf3.fit(X_train,y_train)

pred3 = clf3.predict(x_test)

sc3 = clf3.score(x_test,y_test)

f3 = f1_score(y_test,pred3)

print('Accuracy: '+str(sc3))

print('F-score '+str(f3))

cm = confusion_matrix(y_test,pred3,labels=[0,1])

plot_confusion_matrix(cm,classes=[0,1])

print("--- %s seconds ---" % (time.time() - start_time))


#max_voting

start_time = time.time()
```

```
y_test1 = np.array(y_test)

count=0

final_pred = np.array([])

for i in range(0,len(x_test)):

    final_pred = np.append(final_pred, statistics.mode([pred1[i],pred2[i],pred3[i]]))

for i in range(len(final_pred)):

    if final_pred[i]==y_test1[i]:

        count+=1

sc=count/(i)

print('Accuracy: '+str(sc))

f = f1_score(y_test,final_pred)

print('F-measure: '+str(f))

cm = confusion_matrix(y_test,final_pred,labels=[0,1])

plot_confusion_matrix(cm,classes=[0,1])

print("--- %s seconds ---" % (time.time() - start_time))
```