Adult Income Prediction notebook

Description:

> **In this notebook, we are going to predict whether a person's income is above 50k or below 50k using various features like age, education, and occupation.**

> **The dataset we are going to use is the Adult census income dataset from Kaggle which contains about 48842 rows and 15 features that can be downloaded here(https://www.kaggle.com/uciml/adult-census-income)**

> **The dataset contains the labels which we have to predict and the labels are discrete and binary. So the problem we have is a Supervised Classification type.**

Step 1: Load libraries and dataset

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

```
data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/DL/Adult_data/adult.csv")
```

```
data.head()
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationsh
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-ch
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husba
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husba
3	44	Private	160323	Some-college	10	Married-civ-	Machine-op-inspct	Husba

Step 2: Descriptive analysis

```
data.shape
```

```
(48842, 15)
```

```
data.tail()
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relati
48837	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	
48838	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	H
48839	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Un

```
data.info()
```

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

```
# Shape of dataset
print('Total dataframe  Rows: {} '.format(data.shape[0]))
print('Total dataframe Columns: {}'.format(data.shape[1]))
```

```
Total dataframe  Rows: 48842
```

```
Total dataframe Columns: 15
```

```
data1=data.copy()
```

```
data1.head()
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationsh
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-ch
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husba
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husba
3	44	Private	160323	Some-college	10	Married-civ-	Machine-op-inspct	Husba

Summary statistics for numeric attribute

```
data1.describe()
```

	age	fnlwgt	educational-num	capital-gain	capital-loss	hours-p w
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.422
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.391
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.000

Observations:

For Age :

> **1.The mean value is 38 i.e. on an average the value of age attribute is 38.**
> **2.Age is having the standerd deviation 13.71 which indicates the deviation of an observation from the mean.**
> **3.The value of Age attribute varies from 17 to 90.**
> **4.The 1st quartile is 28 i.e. 25% of the observations lies below 28.**
> **5.3rd quartile is 48 which indicates that in 75% of the observations the value of age is less than 48.**
> **6.The difference between 1st quartile and the minimum is lesser than the difference between 3rd quartile and the maximum which is showing that the data is more dispersed after the value 48.**
> **7.The difference between mean & median is not significantly high but the difference between 3rd quartile & maximum made the distribution right skewed.**

For fnlwgt :

> **1.This is the sampling weight corresponding to the observations.**
> **2.finalweight seems to be rightly skewed since there is very large distance between median & maximum value as compared to minimum & median value.**

For capital-gain :

> **1.or capital-gain, the mean is 1079.01 and median is 0, which indicates that the distribution is highly right skewed.**
> **2.From the qurtiles it is clearly visible that 75% observations are having capital gain zero.**
> **3.capital-gain is concentrated on the one particular value i.e. zero and other are spread after 3rd quartile which results as the large standard deviation(7452.01).**
> **4.capital-gain shows that either a person has no gain or has gain of very large amount(10k or 99k).**

For capital-loss :

> **1.This attribute is similar to the capital-gain i.e. most of the values are centered on 0(this can be told using the summary statistic as minimum is 0 and values lie under 75 percentile is also zero.**
> **2.Mean is 87 but median is 0(i.e. mean is greater than median this tells us that it is right skewed distribution).**

For hours-per-week :

> 1.This attribute means number of working hours spend by an individual in a week.
> 2.In this data the hours per week atrribute varies within the range of 1 to 99.
> 3.75 percentage of the people spend 45 or less working hours per week.
> 4.The IQR is very less i.e. [40-45] which indicates that 50% of the observations are concentrated between 40 & 45.
> 5.Observations are very sparse below 25th percentile and after 75th percentile.
> 6.Using quartiles we can say that data is approximately symmetric.
> 7Minimum is 1 hour per week & maximum value is 99 hours per week means person spending 99 working hours per week are very rare events. We will later analyze that which workclass they belong.

Summary categorical attribute

```
data1.describe(include=["O"])
```

	workclass	education	marital-status	occupation	relationship	race	gender	nat: cour
count	48842	48842	48842	48842	48842	48842	48842	4ᵋ
unique	9	16	7	15	6	5	2	
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	Un St

Observations:

> 1.Native-country has maximum number of unique categories i.e. 41 categories.

> 2.But the native-country is highly biased toward the US which has frequency of 44689 out of total 48842(nearly 91%).

> 3.Occupation has more or less uniform distribution of categories as comparerd to the other attributes.

> 4.Race is also biased to the white race category(41762) with 85.5%.

> **The top category in workclass is Private having frequency(36705) and percentage(75.5%).**

```
print(f"Target: 'Income'\nUnique Values in Income: {data1.income.unique()}\nNumber of uniq
```

```
Target: 'Income'
Unique Values in Income: ['<=50K' '>50K']
Number of unique values: 2
```

In the problem, we have 'Income' as the Target variable. we see that we have only two values which are to be predicted, either the income is greater than 50K, which is Yes, or the income is less than or equal to 50K, which is No. We will label encode the target variable.

```
# Check for '?' in dataset
round((data1.isin(['?']).sum() / data1.shape[0])
      * 100, 2).astype(str) + ' %'
```

```
age                 0.0 %
workclass          5.73 %
fnlwgt              0.0 %
education           0.0 %
educational-num     0.0 %
marital-status      0.0 %
occupation         5.75 %
relationship        0.0 %
race                0.0 %
gender              0.0 %
capital-gain        0.0 %
capital-loss        0.0 %
hours-per-week      0.0 %
native-country     1.75 %
income              0.0 %
dtype: object
```

```
# Checking the counts of label categories
income = data1['income'].value_counts(normalize=True)
round(income * 100, 2).astype('str') + ' %'
```

```
<=50K    76.07 %
>50K     23.93 %
Name: income, dtype: object
```

Observations:

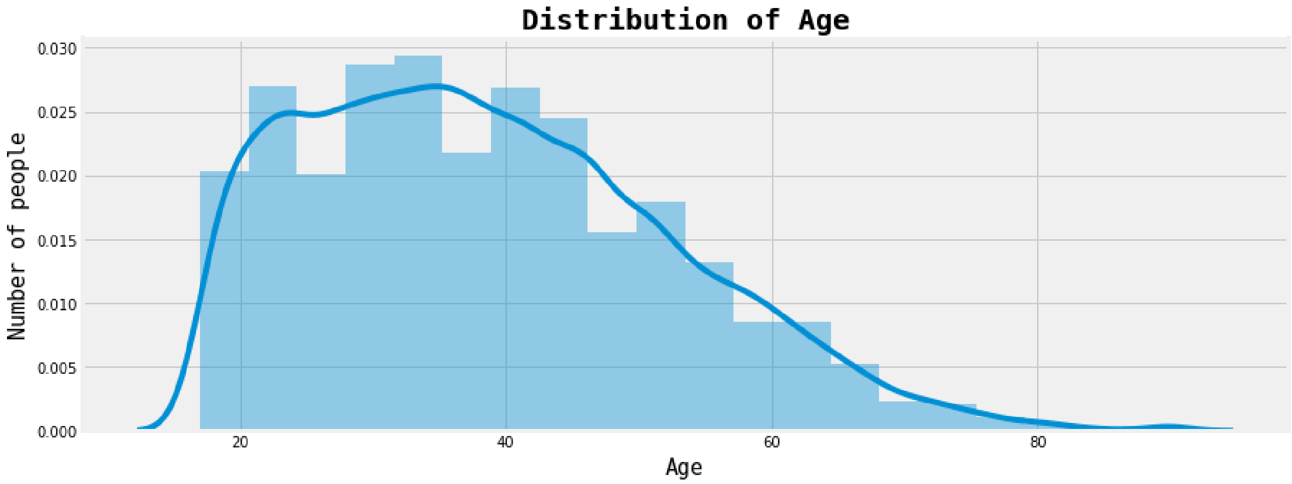
- **The dataset doesn't have any null values, but it contains missing values in the form of '?' which needs to be preprocessed.**

- **The dataset is unbalanced, as the dependent feature 'income' contains 76.7% values have**

Univariate Analysis

```
# Creating a distribution plot for 'Age'
age = data1['age'].value_counts()

plt.figure(figsize=(15, 5))
plt.style.use('fivethirtyeight')
sns.distplot(data1['age'], bins=20)
plt.title('Distribution of Age', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Age', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=10)
plt.show()
```

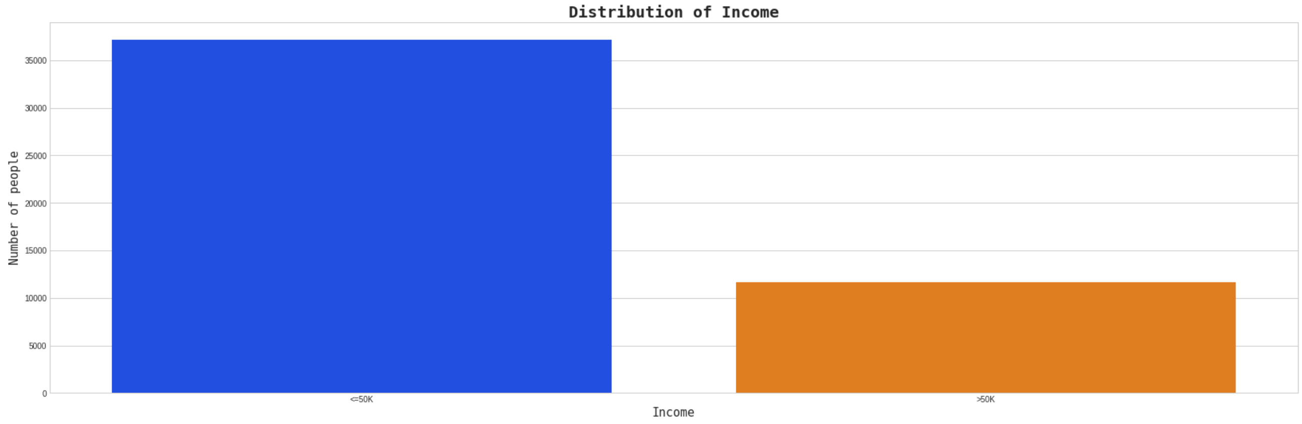
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:
  warnings.warn(msg, FutureWarning)
```



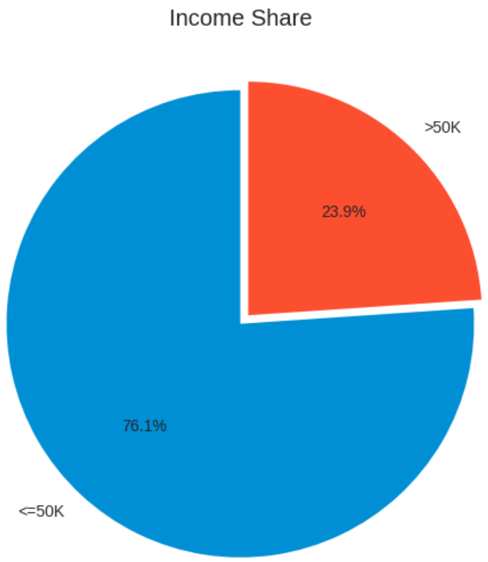

```
# Creating a barplot for 'Income'
income = data1['income'].value_counts()

plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(25, 8))
sns.barplot(income.index, income.values, palette='bright')
plt.title('Distribution of Income', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Income', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=10)
```

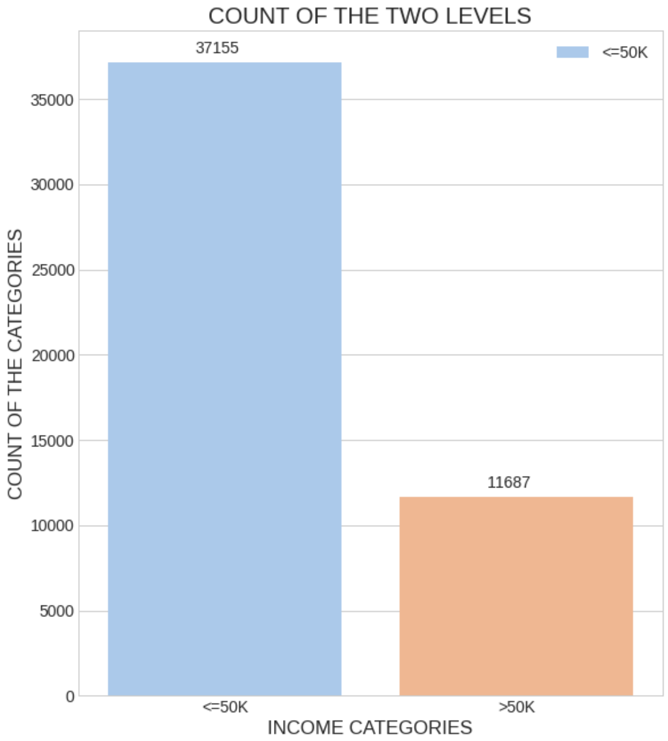
```
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```



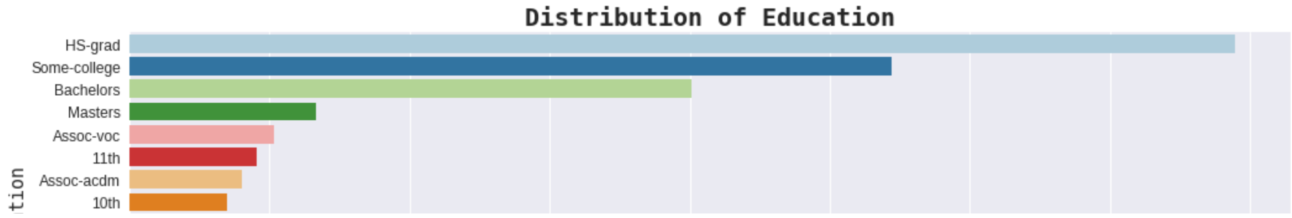

```
f,ax=plt.subplots(1,2,figsize=(18,10))
#plt.figure(figsize=(7,10))
income1=data1['income'].value_counts()
ax[0].pie(income1,explode=(0,0.05),autopct='%1.1f%%',startangle=90,labels=['<=50K','>50K']
ax[0].set_title('Income Share')
ax[1]=sns.countplot(x='income',data=data1,palette='pastel')
ax[1].legend(labels=['<=50K','>50K'])
ax[1].set(xlabel="INCOME CATEGORIES")
ax[1].set(ylabel='COUNT OF THE CATEGORIES')
ax[1].set_title('COUNT OF THE TWO LEVELS')

for p in ax[1].patches:
    ax[1].annotate(p.get_height(),(p.get_x()+0.3,p.get_height()+500))
```






```
# Creating a barplot for 'Education'
edu = data1['education'].value_counts()

plt.style.use('seaborn')
plt.figure(figsize=(15, 5))
sns.barplot(edu.values, edu.index, palette='Paired')
plt.title('Distribution of Education', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Education', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=12)
plt.show()
```

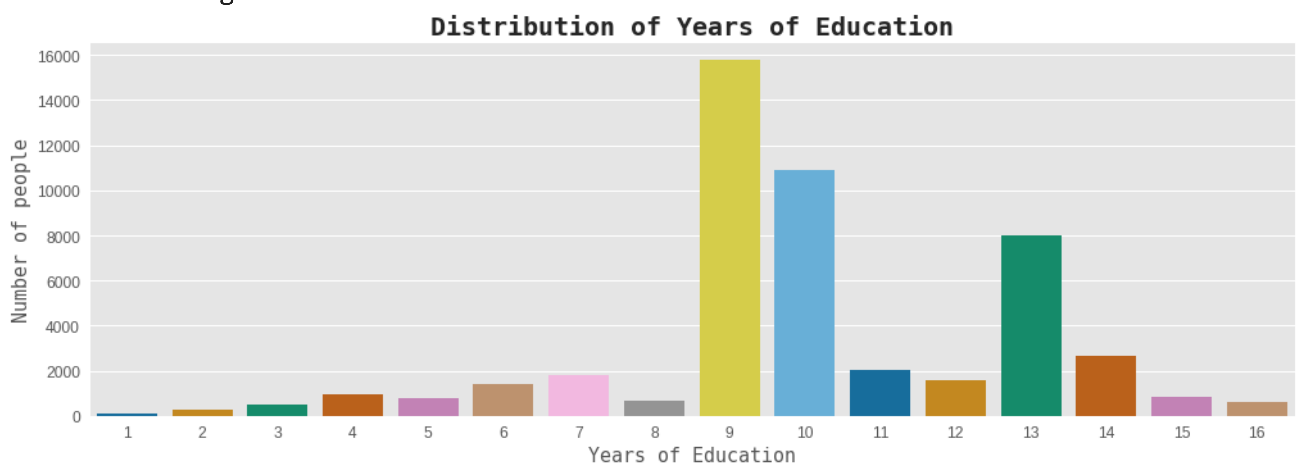
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```



Creating a barplot for 'Years of Education'

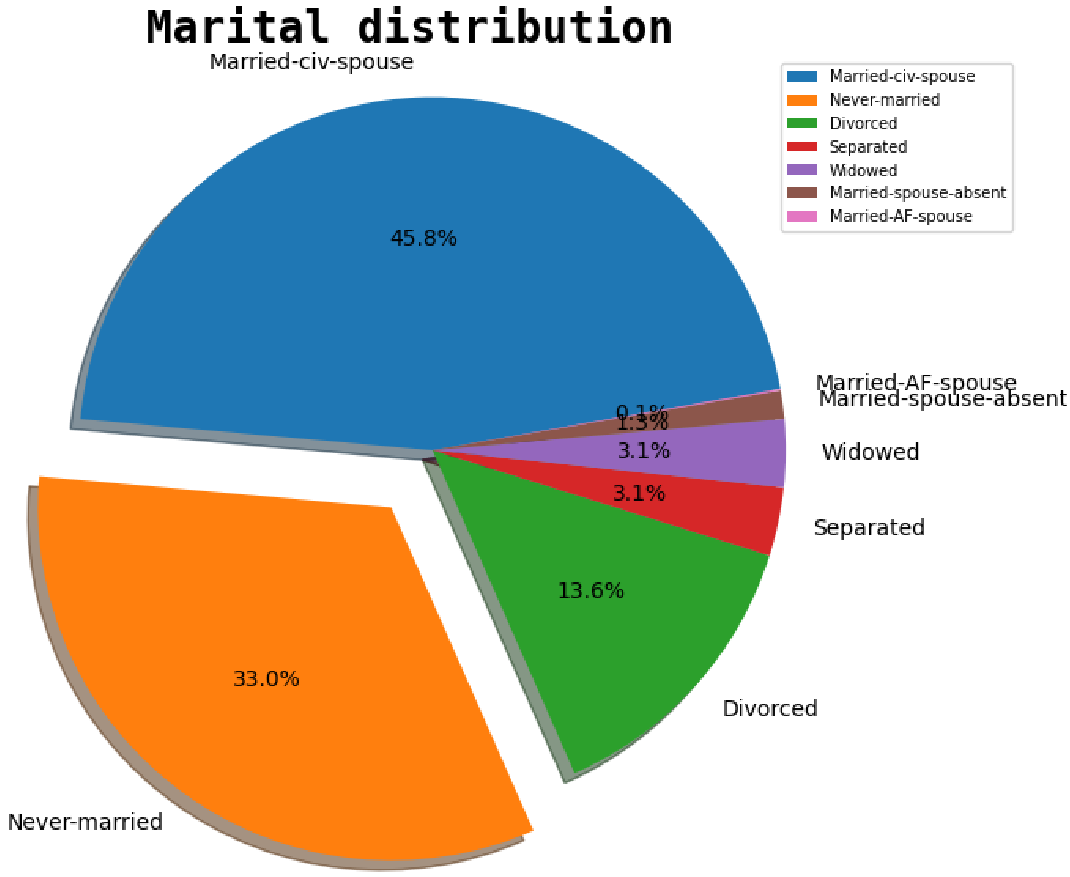
```
edu_num = data1['educational-num'].value_counts()

plt.style.use('ggplot')
plt.figure(figsize=(15, 5))
sns.barplot(edu_num.index, edu_num.values, palette='colorblind')
plt.title('Distribution of Years of Education', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Years of Education', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=12)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```



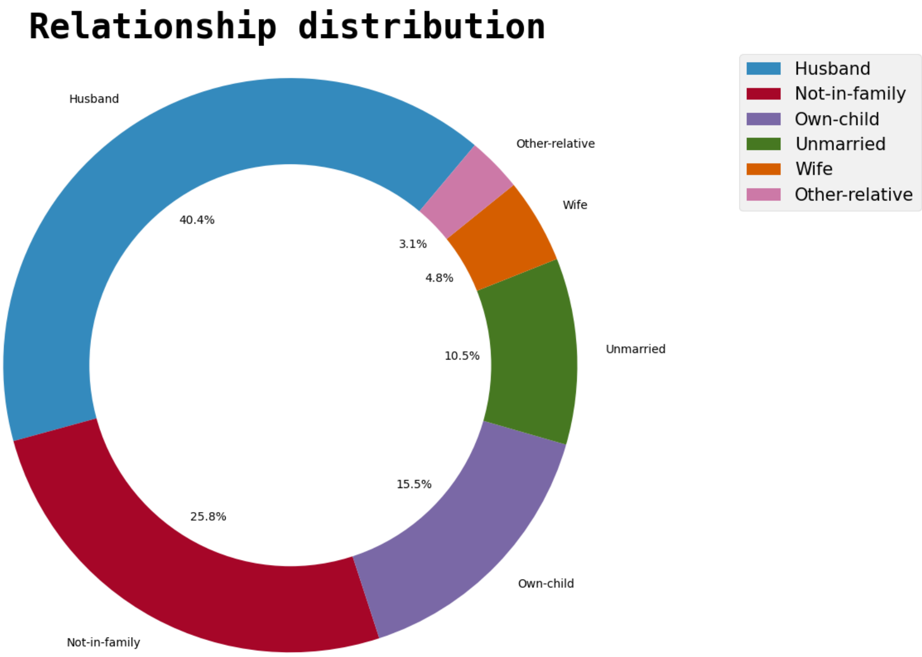
Creating a pie chart for 'Marital status'

```
marital = data1['marital-status'].value_counts()

plt.style.use('default')
plt.figure(figsize=(10, 7))
plt.pie(marital.values, labels=marital.index, startangle=10, explode=(
    0, 0.20, 0, 0, 0, 0, 0), shadow=True, autopct='%1.1f%%')
plt.title('Marital distribution', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.legend()
plt.legend(prop={'size': 7})
plt.axis('equal')
plt.show()
```



Creating a donut chart for 'Age'

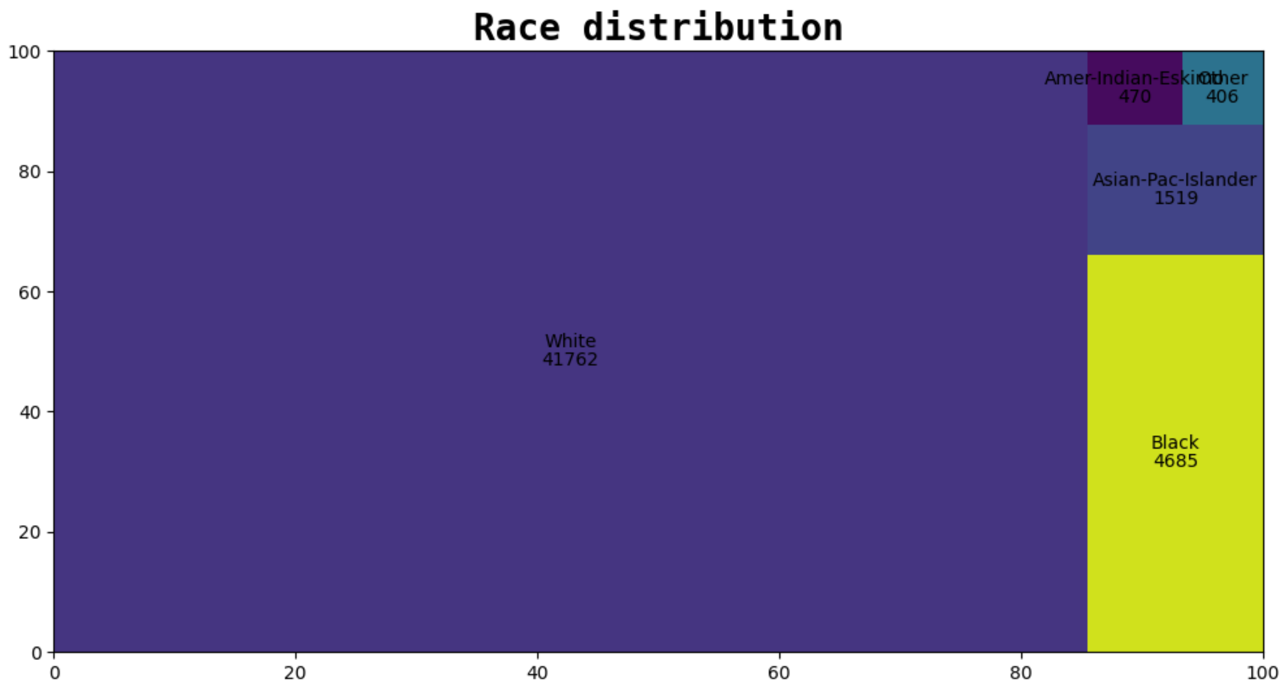
```
relation = data1['relationship'].value_counts()

plt.style.use('bmh')
plt.figure(figsize=(20, 10))
plt.pie(relation.values, labels=relation.index,
        startangle=50, autopct='%1.1f%%')
centre_circle = plt.Circle((0, 0), 0.7, fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.title('Relationship distribution', fontdict={
          'fontname': 'Monospace', 'fontsize': 30, 'fontweight': 'bold'})
plt.axis('equal')
plt.legend(prop={'size': 15})
plt.show()
```



```
pip install squarify
```

```
Collecting squarify
  Downloading https://files.pythonhosted.org/packages/0b/2b/2e77c35326efec19819cd1d72
Installing collected packages: squarify
Successfully installed squarify-0.4.3
```

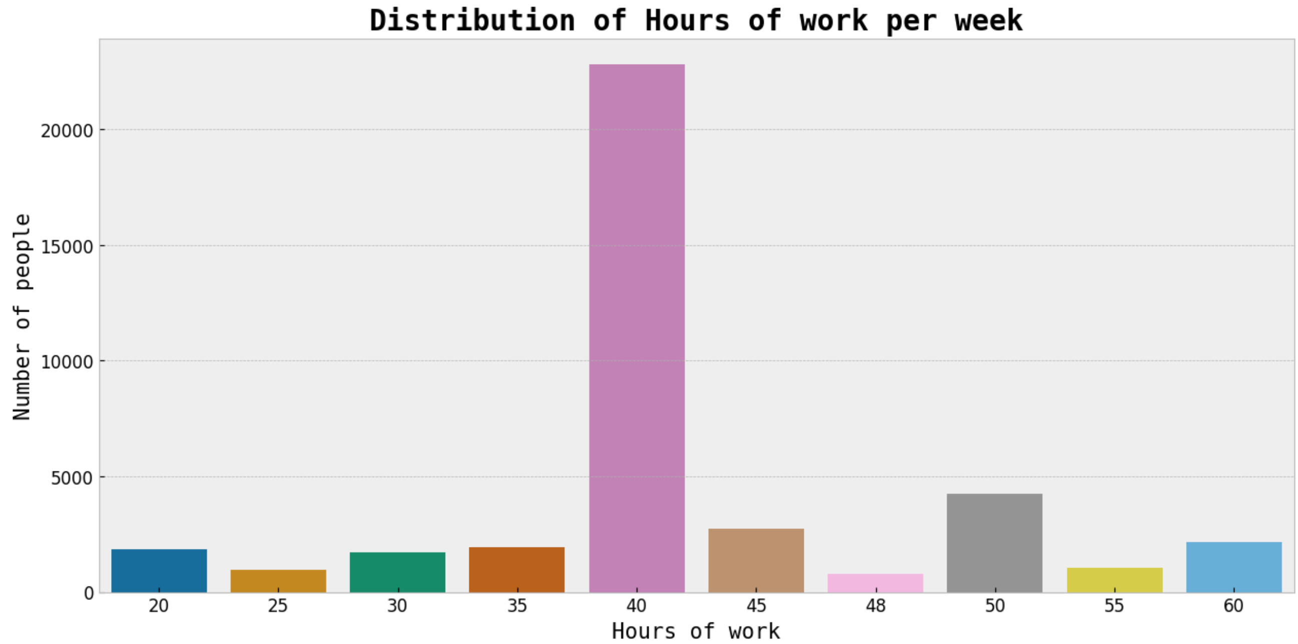
```
# Creating a Treemap for 'Race'
import squarify
race = data1['race'].value_counts()

plt.style.use('default')
plt.figure(figsize=(12, 6))
squarify.plot(sizes=race.values, label=race.index, value=race.values)
plt.title('Race distribution', fontdict={
        'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.show()
```




```
# Creating a barplot for 'Hours per week'
hours = data1['hours-per-week'].value_counts().head(10)

plt.style.use('bmh')
plt.figure(figsize=(15, 7))
sns.barplot(hours.index, hours.values, palette='colorblind')
```

```
sns.barplot(hours.index, hours.values, palette='colorblind')
plt.title('Distribution of Hours of work per week', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Hours of work', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=12)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```

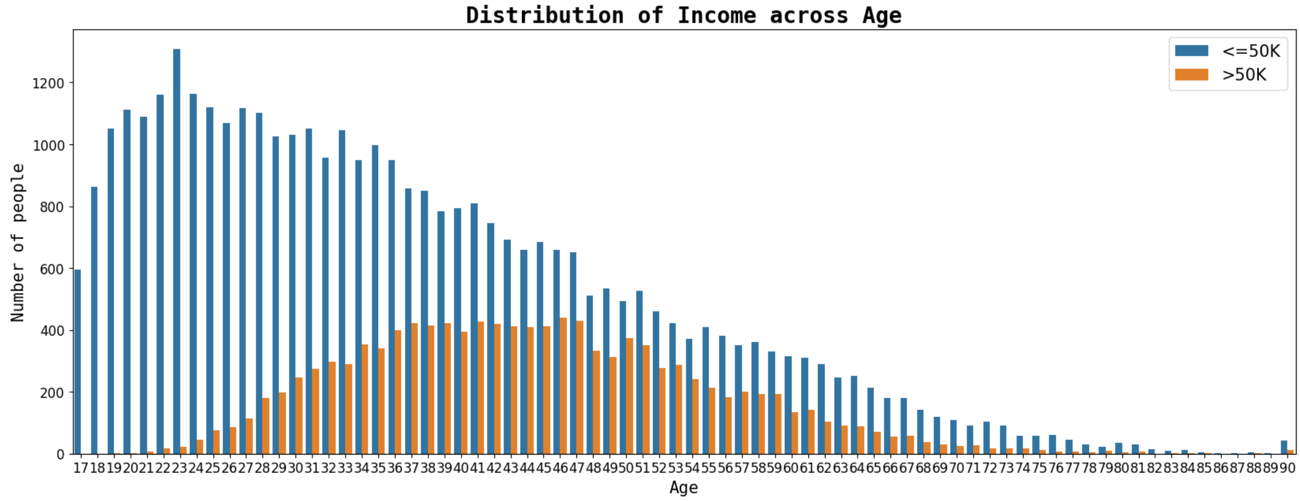


Bivariate Analysis

```
# Creating a countplot of income across age
plt.style.use('default')
plt.figure(figsize=(20, 7))
sns.countplot(data1['age'], hue=data1['income'])
```

```
plt.title('Distribution of Income across Age', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Age', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 15})
plt.show()
```

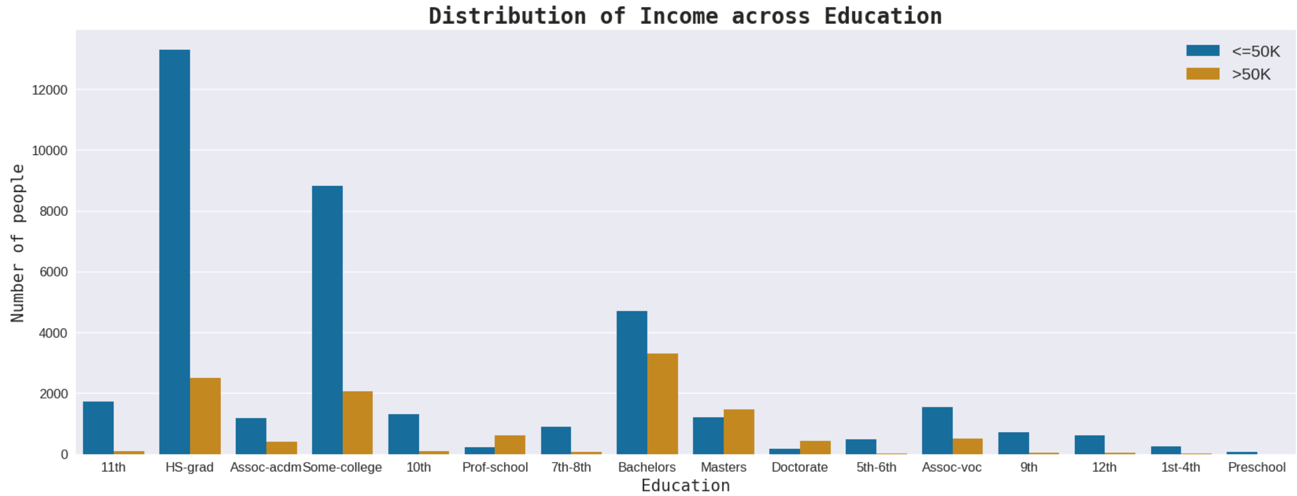
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```




```
# Creating a countplot of income across education
plt.style.use('seaborn')
plt.figure(figsize=(20, 7))
sns.countplot(data1['education'],
              hue=data1['income'], palette='colorblind')
plt.title('Distribution of Income across Education', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Education', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
```

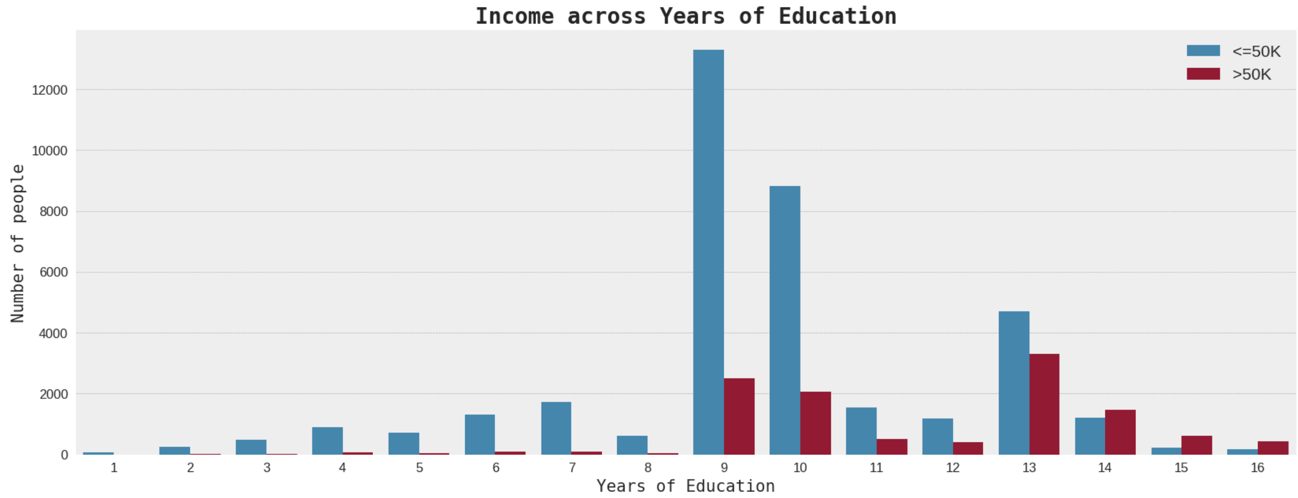
```
plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 15})
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```




```
# Creating a countplot of income across years of education
plt.style.use('bmh')
plt.figure(figsize=(20, 7))
sns.countplot(data1['educational-num'],
              hue=data1['income'])
plt.title('Income across Years of Education', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Years of Education', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 15})
plt.savefig('bi2.png')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```

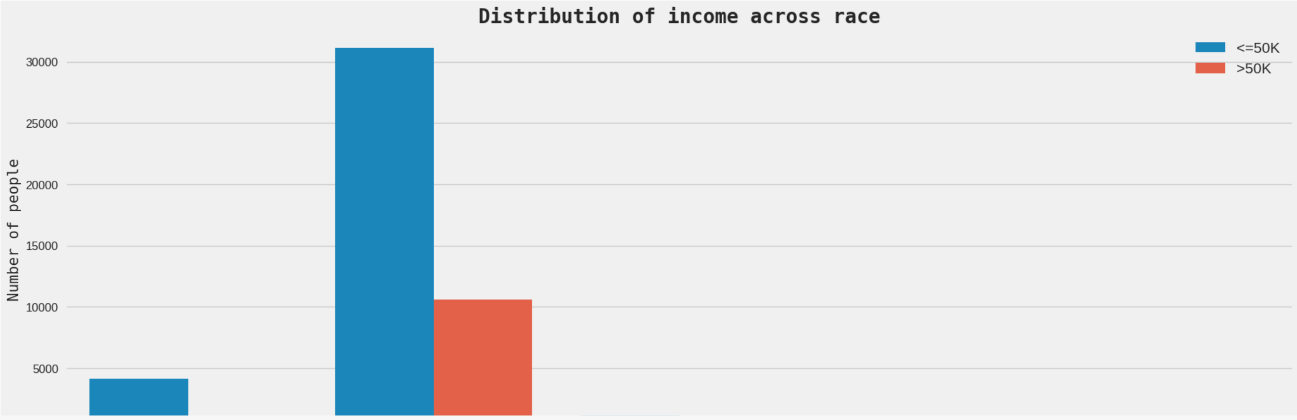



```
# Creating a countplot of income across Marital Status
plt.style.use('seaborn')
plt.figure(figsize=(20, 7))
sns.countplot(data1['marital-status'], hue=data1['income'])
plt.title('Income across Marital Status', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Marital Status', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 15})
plt.savefig('marital_status.png')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```

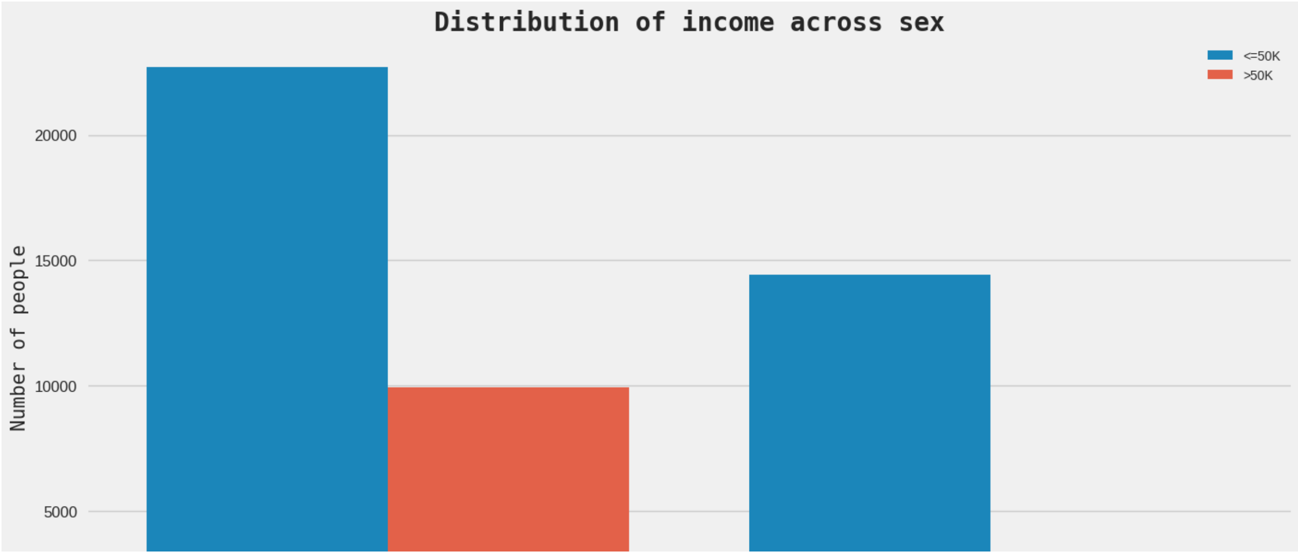

```
# Creating a countplot of income across race
plt.style.use('fivethirtyeight')
plt.figure(figsize=(20, 7))
sns.countplot(data1['race'], hue=data1['income'])
plt.title('Distribution of income across race', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Race', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 15})
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```




```
# Creating a countplot of income across sex
plt.style.use('fivethirtyeight')
plt.figure(figsize=(15, 8))
sns.countplot(data1['gender'], hue=data1['income'])
plt.title('Distribution of income across sex', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Sex', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 10})
plt.savefig('bi3.png')
plt.show()
```

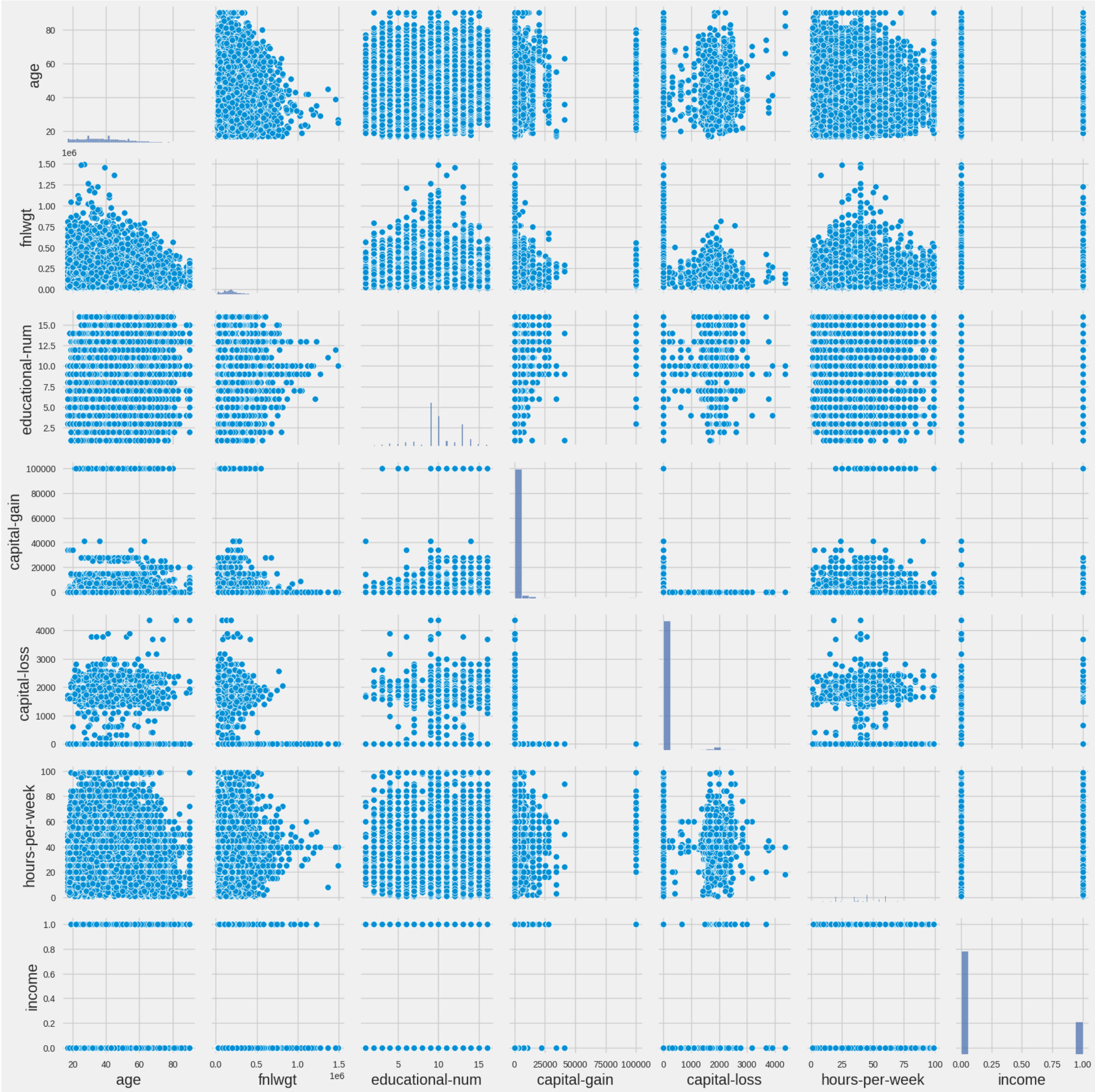
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```



Multivariate Analysis

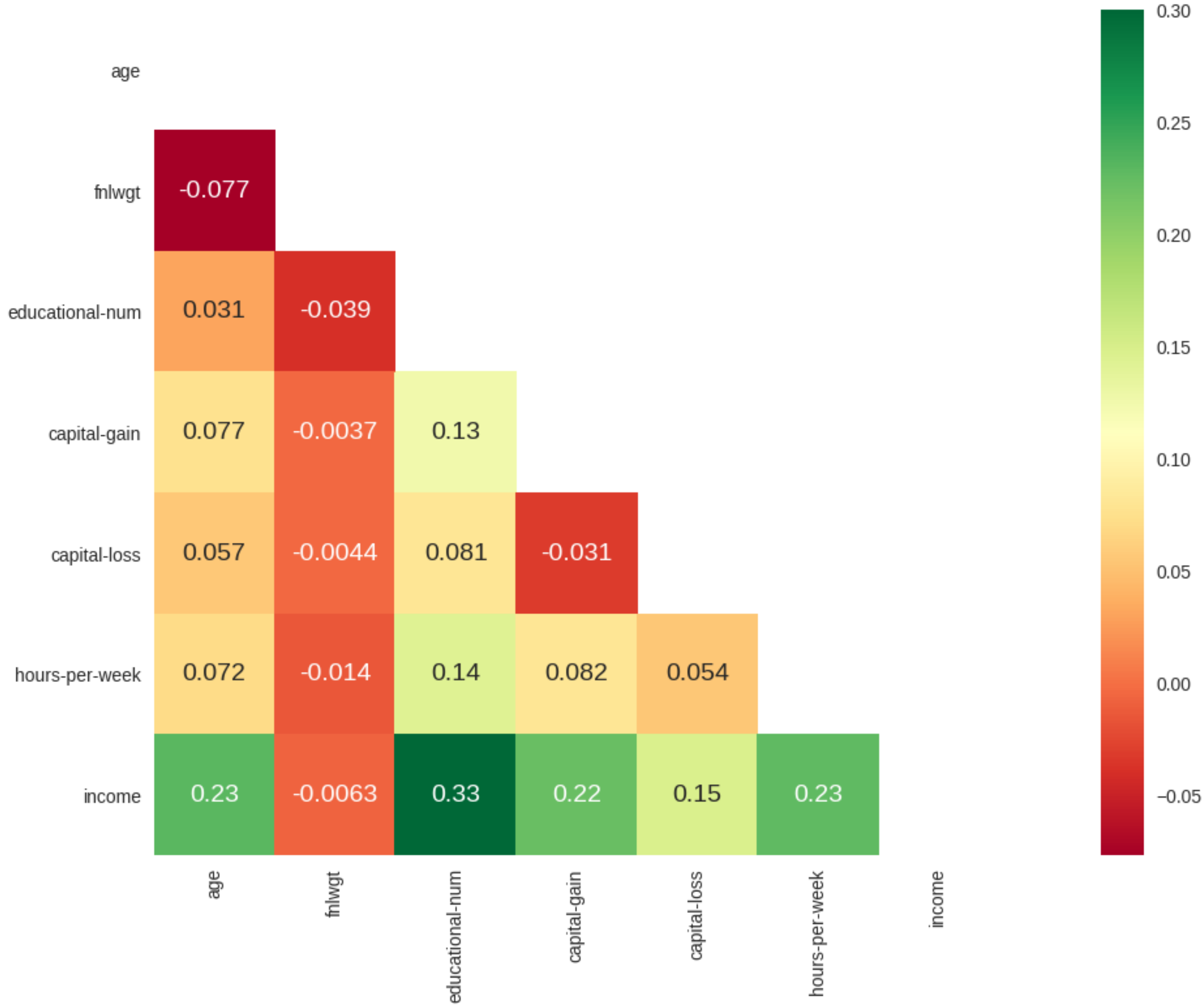
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data1['income'] = le.fit_transform(data1['income'])
```

```
# Creating a pairplot of dataset
sns.pairplot(data1)
plt.savefig('sns1.png')
plt.show()
```




```
corr = data1.corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
    f, ax = plt.subplots(figsize=(18, 8))
```

```
    ax = sns.heatmap(corr, mask=mask, vmax=.3, square=True,
                     annot=True, cmap='RdYlGn')
plt.savefig('corr1.png')
plt.show()
```



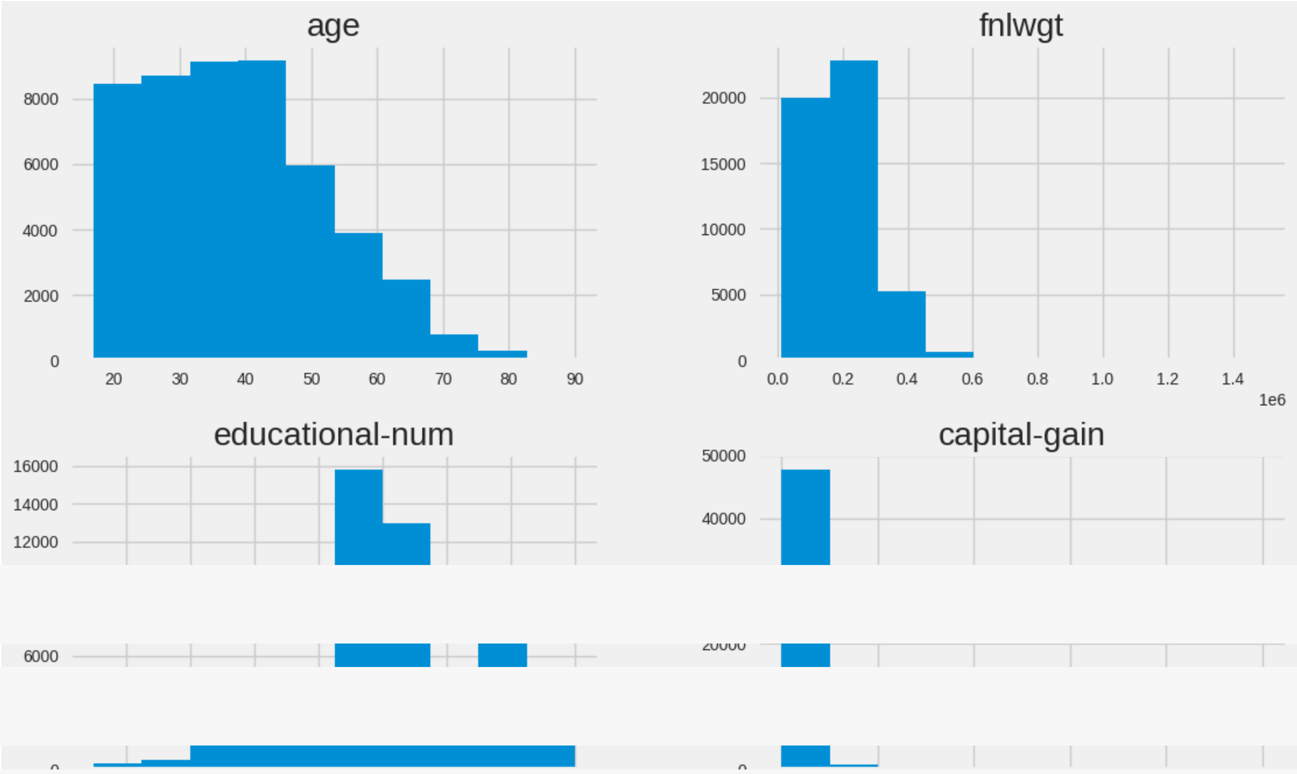
Observations:

> **In this dataset, the most number of people are young, white, male, high school graduates with 9 to 10 years of education and work 40 hours per week.**

> **From the correlation heatmap, we can see that the dependent feature 'income' is highly correlated with age, numbers of years of education, capital gain and number of hours per week.**

```
#Visualizing the numerical features of the dataset using histograms to analyze the distrib
from matplotlib import rcParams
rcParams['figure.figsize'] = 12, 12
data[['age', 'fnlwgt', 'educational-num', 'capital-gain', 'capital-loss', 'hours-per-week'

#Can visualise that data such as capital gain, capitaln loss, fnlwgt is right skewed an ot
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fa0ab18dc50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa0ab126350>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fa0ab14f9d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa0ab0fbb90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fa0ab0c9490>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa0ab07ea10>]],
      dtype=object)
```




```
data['workclass'].value_counts()
```

```
Private             33906
Self-emp-not-inc     3862
Local-gov            3136
?                    2799
State-gov            1981
Self-emp-inc         1695
Federal-gov          1432
Without-pay            21
Never-worked           10
Name: workclass, dtype: int64
```

```
data['income'].value_counts()
```

```
<=50K    37155
>50K     11687
Name: income, dtype: int64
```

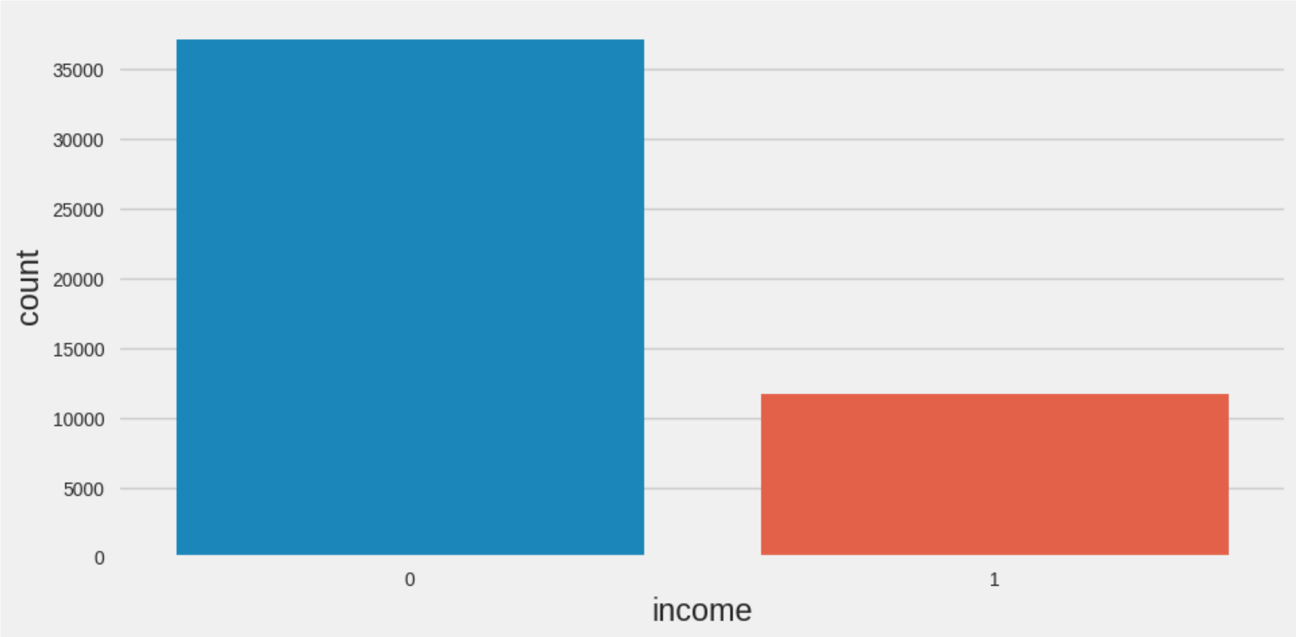
Encode the target variable to binary

```
data['income'] = data['income'].apply(lambda inc: 0 if inc == "<=50K" else 1) # Binary enc
```

Exploratory analysis

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10,5))
sns.countplot(data['income'])
```

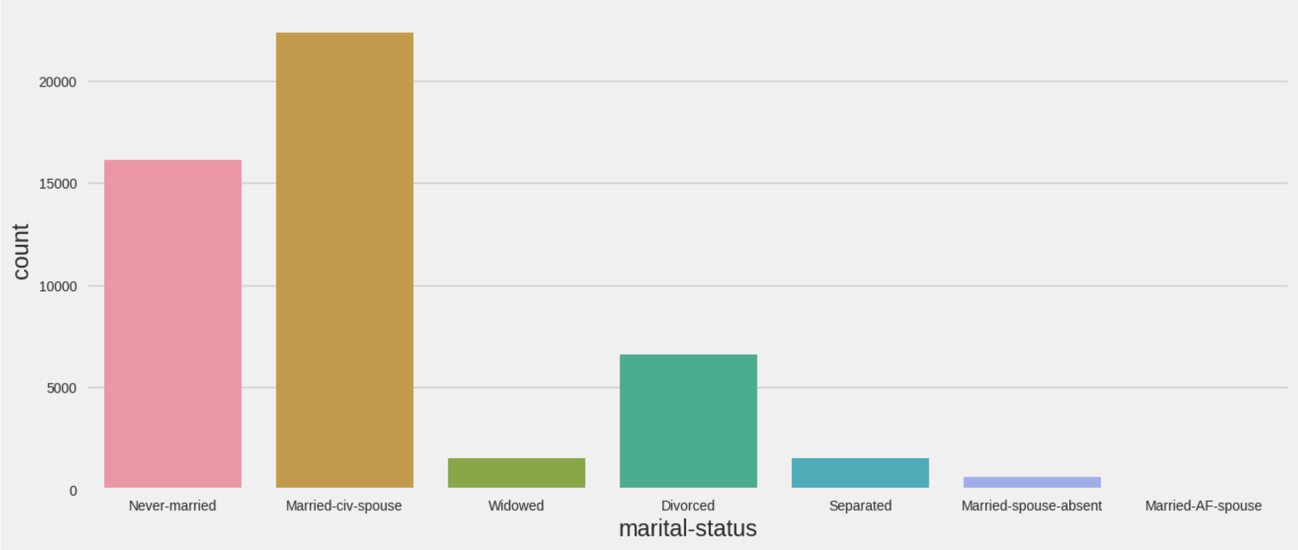
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0aaeed650>
```



As one can see, there is considerable class imbalance in the target variable, i.e. income. This is also intuitively obvious as one expects fewer 'rich' people (earning>50k/annum) than 'not-so-rich' people (earning <50k/annum). Therefore we might need to consider over-sampling techniques in our ML model to improve our accuracy.

```
plt.figure(figsize=(14,6))
sns.countplot(data['marital-status'])
```

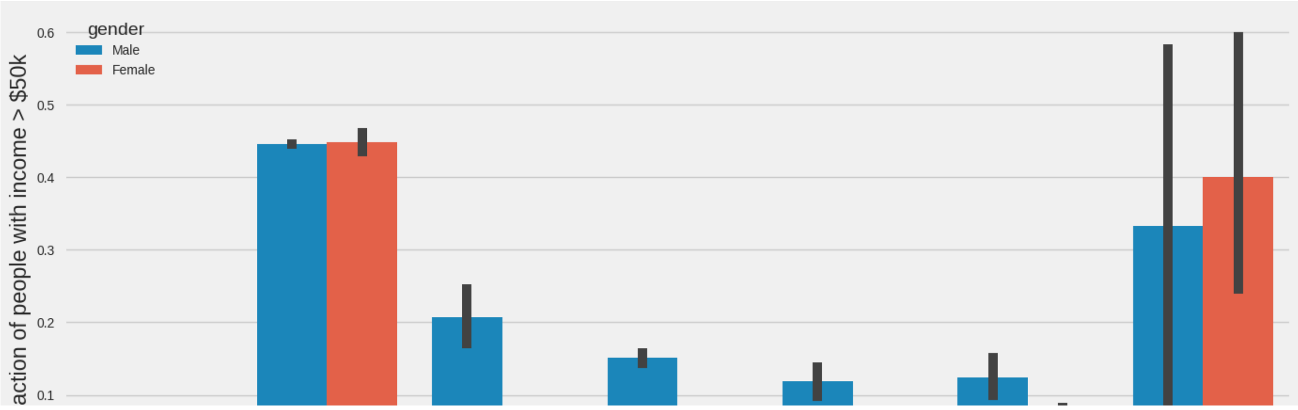
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0aae29ed0>
```



Those with Never-married and Married-civ-spouse labels dominate the dataset

```
plt.figure(figsize=(15,6))
ax=sns.barplot(x='marital-status',y='income',data=data,hue='gender')
ax.set(ylabel='Fraction of people with income > $50k')
data['marital-status'].value_counts()
```

```
Married-civ-spouse       22379
Never-married            16117
Divorced                  6633
Separated                 1530
Widowed                   1518
Married-spouse-absent      628
Married-AF-spouse           37
Name: marital-status, dtype: int64
```



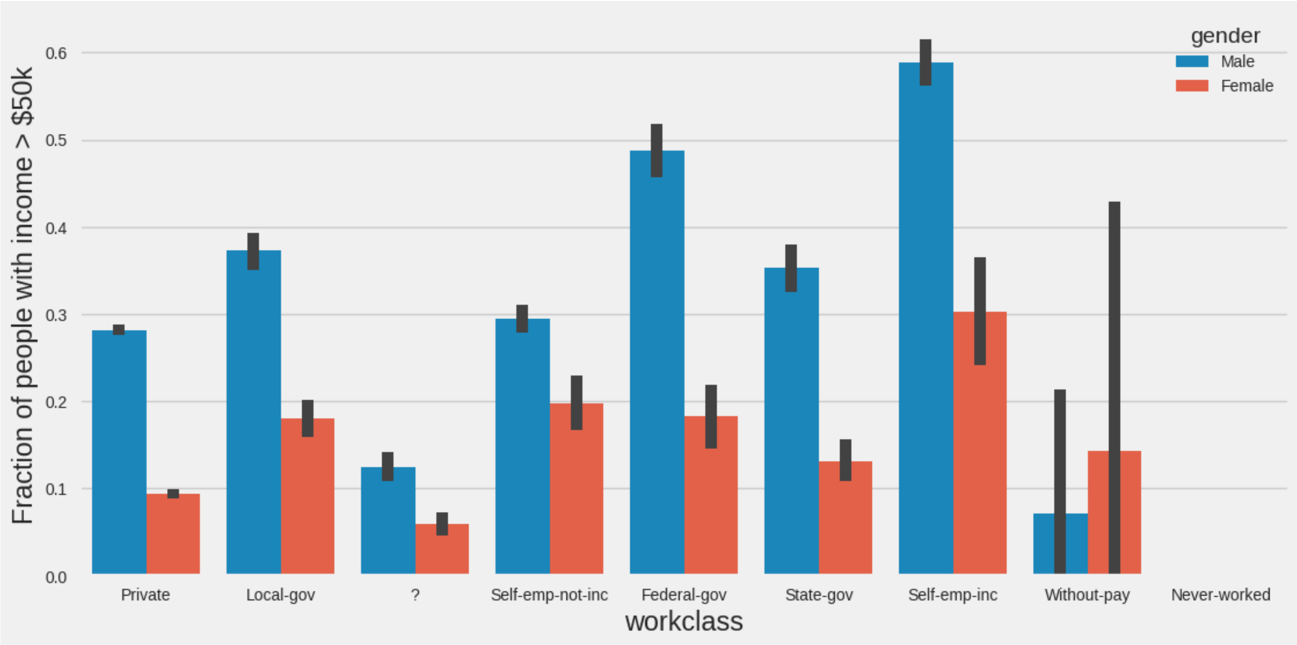
The above plot shows the the fraction of people earning more than $50k per annum, grouped by their marital status and gender. The data shows that married people have a higher %age of high-earners, compared to those who either never married or are widowed/divorced/separated. The black lines indicate 2 standard deviations (or 95% confidence interval) in the data set. The married spouses of armed forces personnel have a much higher variation in their income compared to civil spouses because of low-number statistics.

```
plt.figure(figsize=(12,6))
sns.countplot(data['workclass'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0aadc6c90>
```

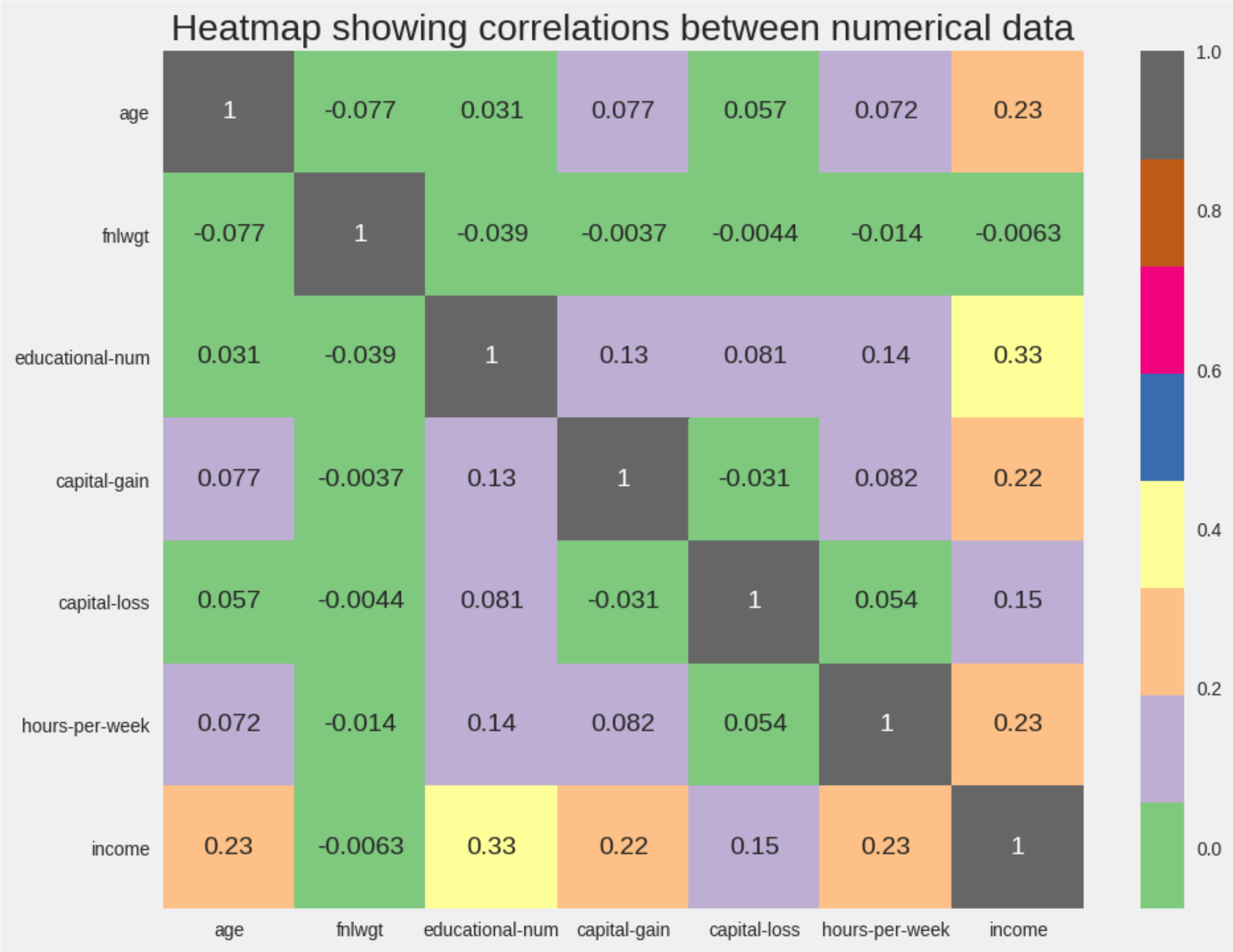

```
plt.figure(figsize=(12,6))
ax=sns.barplot('workclass', y='income', data=data, hue='gender')
ax.set(ylabel='Fraction of people with income > $50k')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
[Text(0, 0.5, 'Fraction of people with income > $50k')]
```



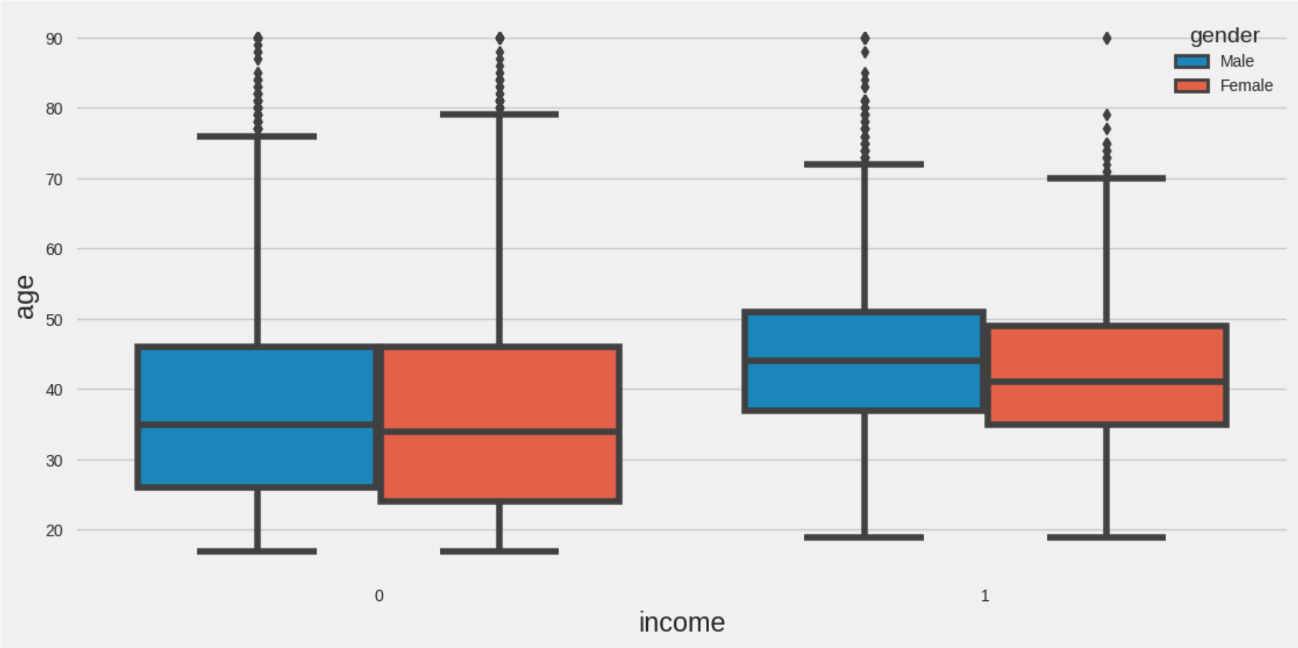

```
plt.figure(figsize=(10,8))
sns.heatmap(data.corr(),cmap='Accent',annot=True)
#data.corr()
plt.title('Heatmap showing correlations between numerical data')
```

```
Text(0.5, 1.0, 'Heatmap showing correlations between numerical data')
```




```
plt.figure(figsize=(12,6))
sns.boxplot(x="income", y="age", data=data, hue='gender')
#data[data['income']==0]['age'].mean()
```

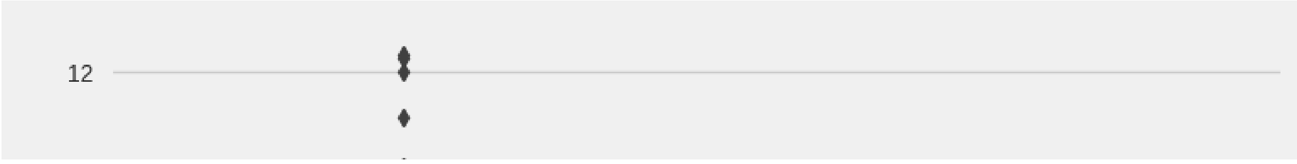
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0aac86550>
```



The mean age of people earning more than 50k per annum is around 44 whereas the mean age of of those earning less than 50k per annum is 36.

```
norm_fnl = (data["fnlwgt"] - data['fnlwgt'].mean())/data['fnlwgt'].std()
plt.figure(figsize=(8,6))
sns.boxplot(x="income", y=norm_fnl, data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0aa9e95d0>
```



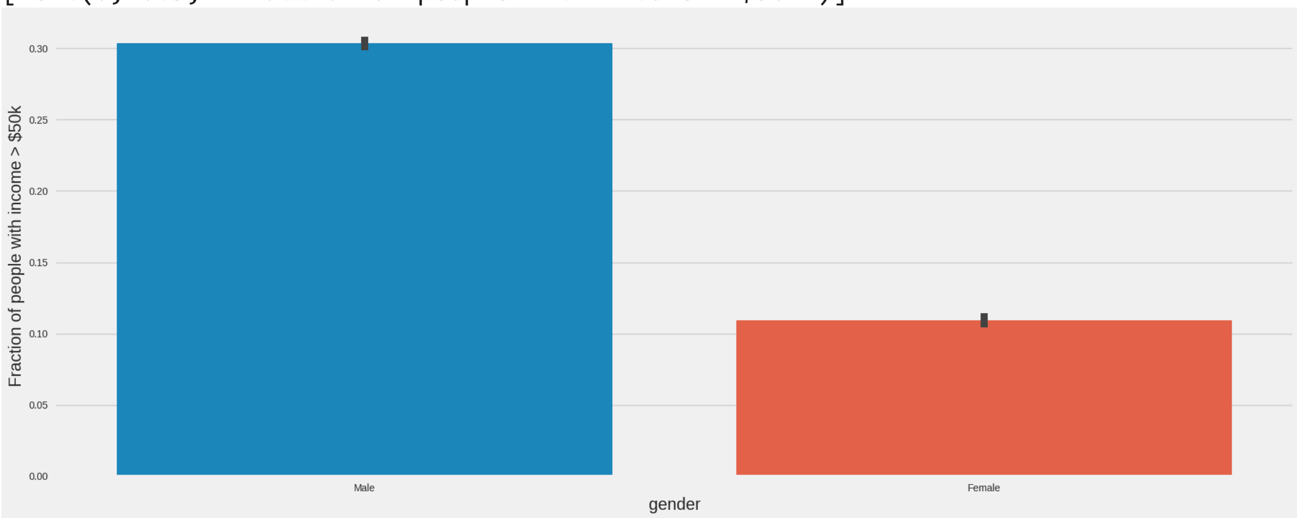
As evident from the plot above, there are many outliers in the fnlwgt column and this feature is uncorrelated with income, our target variable. The correlation coefficient (which one can read from the heatmap) is -0.0095. The number of outliers, i.e. the number of records which are more than 2 s.d's away from the mean, is 1866.

```
data[norm_fnl>2].shape
```

```
(1866, 15)
```

```
plt.figure(figsize=(20,8))
ax = sns.barplot(x='gender',y='income',data=data)
ax.set(ylabel='Fraction of people with income > $50k')
```

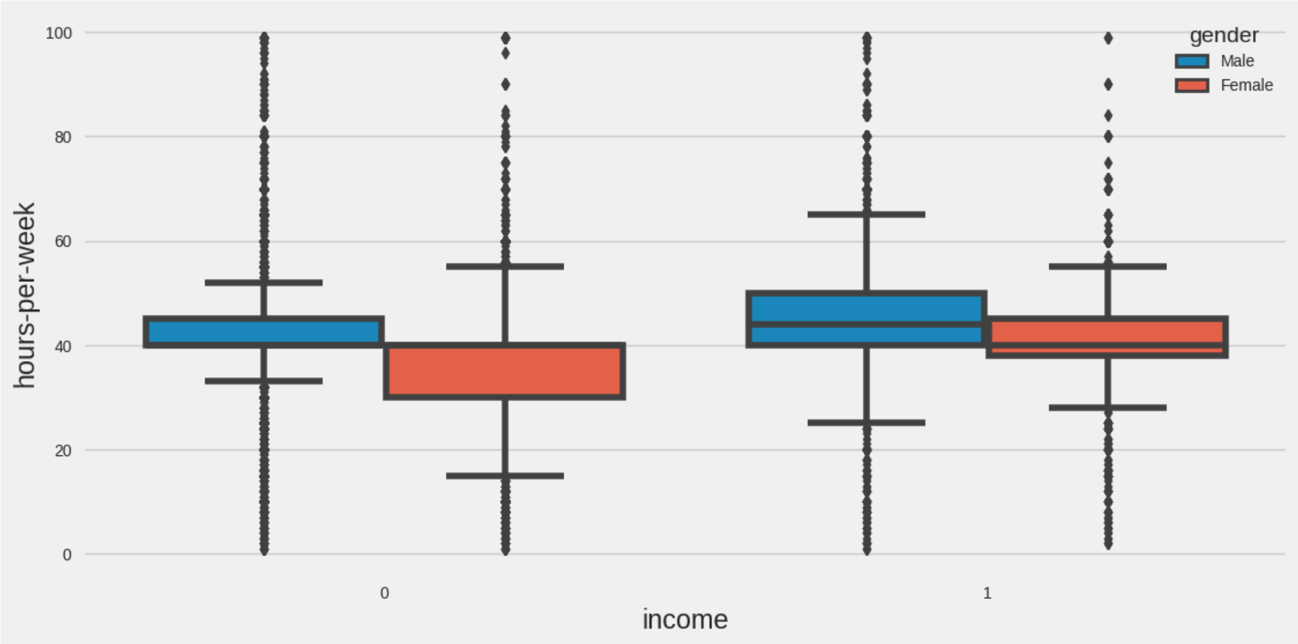
```
[Text(0, 0.5, 'Fraction of people with income > $50k')]
```



The fraction of rich among men is significantly higher than that among women.

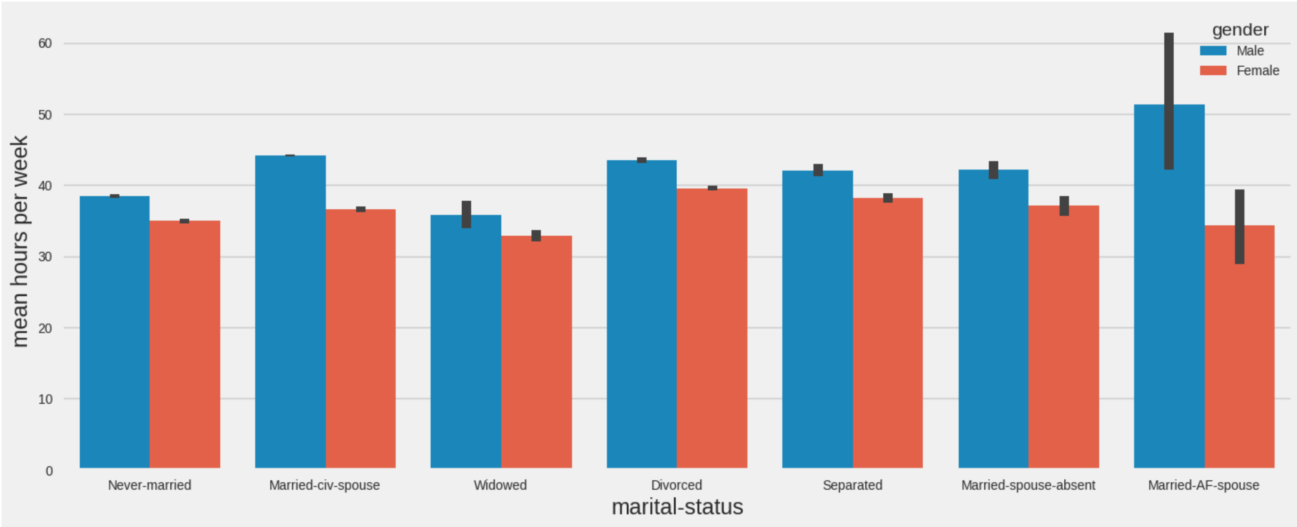
```
plt.figure(figsize=(12,6))
sns.boxplot(x='income',y ='hours-per-week', hue='gender',data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0abd94f90>
```



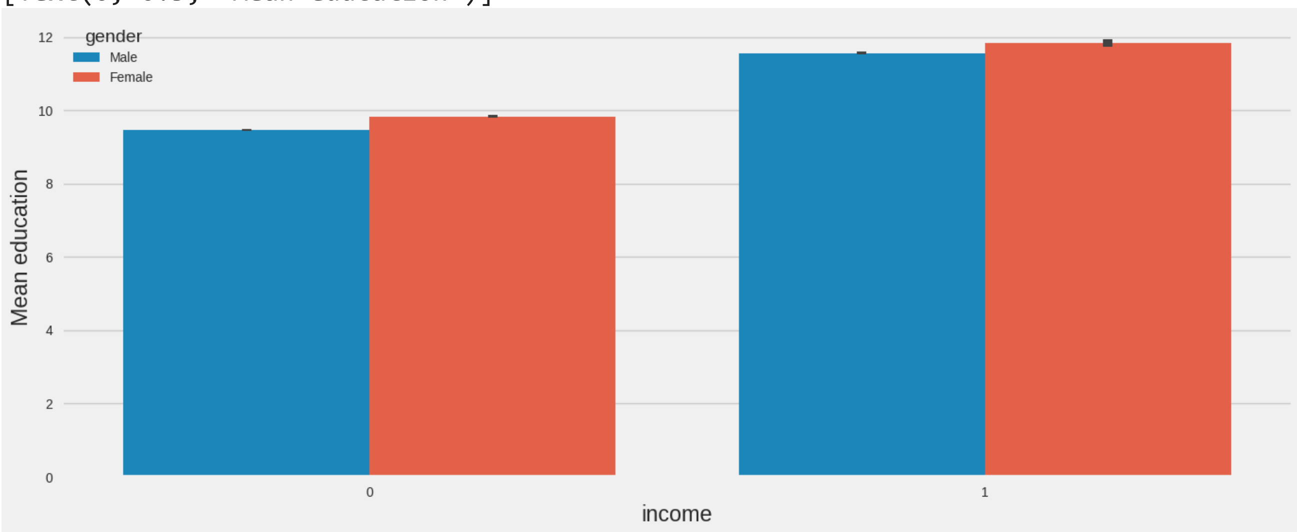

```
plt.figure(figsize=(15,6))
ax = sns.barplot(x='marital-status',y='hours-per-week',data=data,hue='gender')
ax.set(ylabel='mean hours per week')
```

[Text(0, 0.5, 'mean hours per week')]




```
plt.figure(figsize=(15,6))
ax = sns.barplot(x='income', y='educational-num',hue='gender', data=data)
ax.set(ylabel='Mean education')
```

[Text(0, 0.5, 'Mean education')]

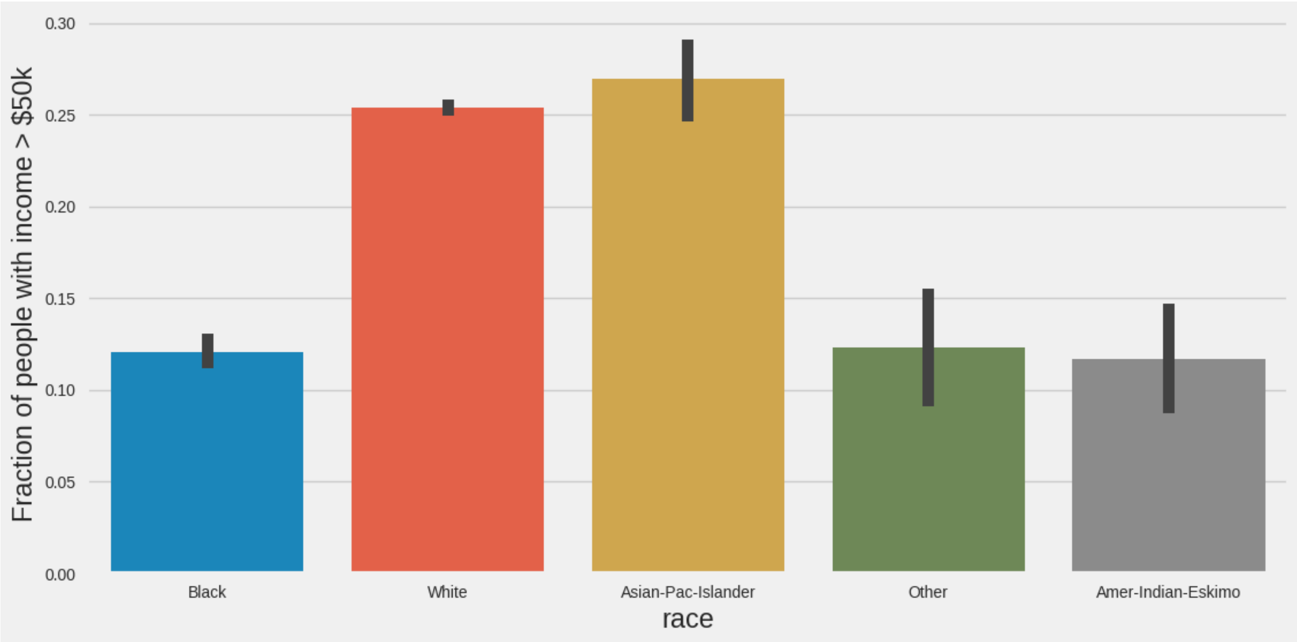


The education.num is label encoded such that a higher number corresponds to a higher level of education. As on would naïvely expect, people who earn more (>50k per annum) are also

highly educated. The mean education level for income=1 class is between 11 and 12 whereas that for the income=0 class is between 9 (HS-grad) and 10 (Some-college).

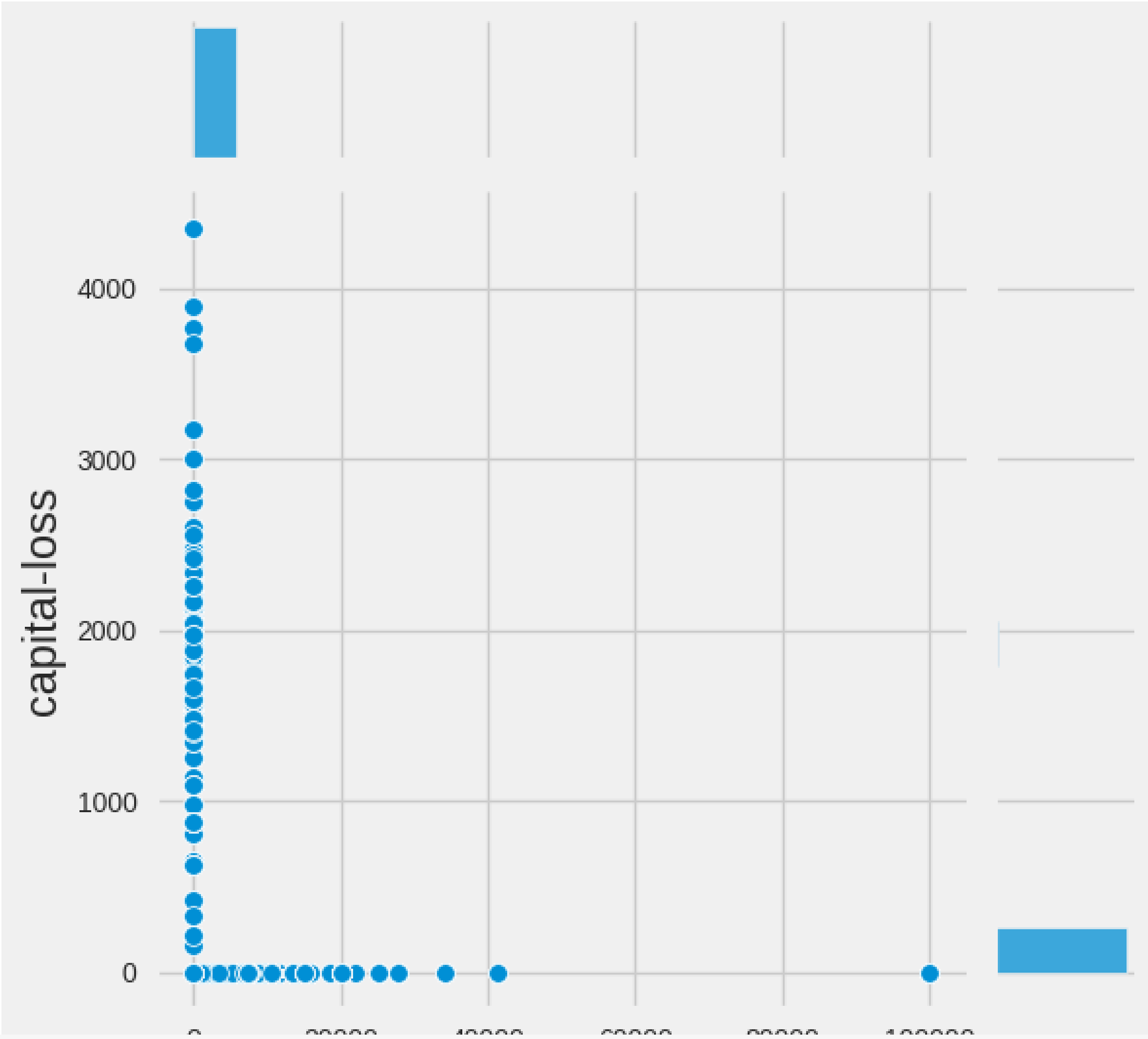
```
print(data['race'].value_counts())
plt.figure(figsize=(12,6))
ax=sns.barplot(x='race',y='income',data=data)
ax.set(ylabel='Fraction of people with income > $50k')
```

```
White                 41762
Black                  4685
Asian-Pac-Islander     1519
Amer-Indian-Eskimo      470
Other                   406
Name: race, dtype: int64
[Text(0, 0.5, 'Fraction of people with income > $50k')]
```



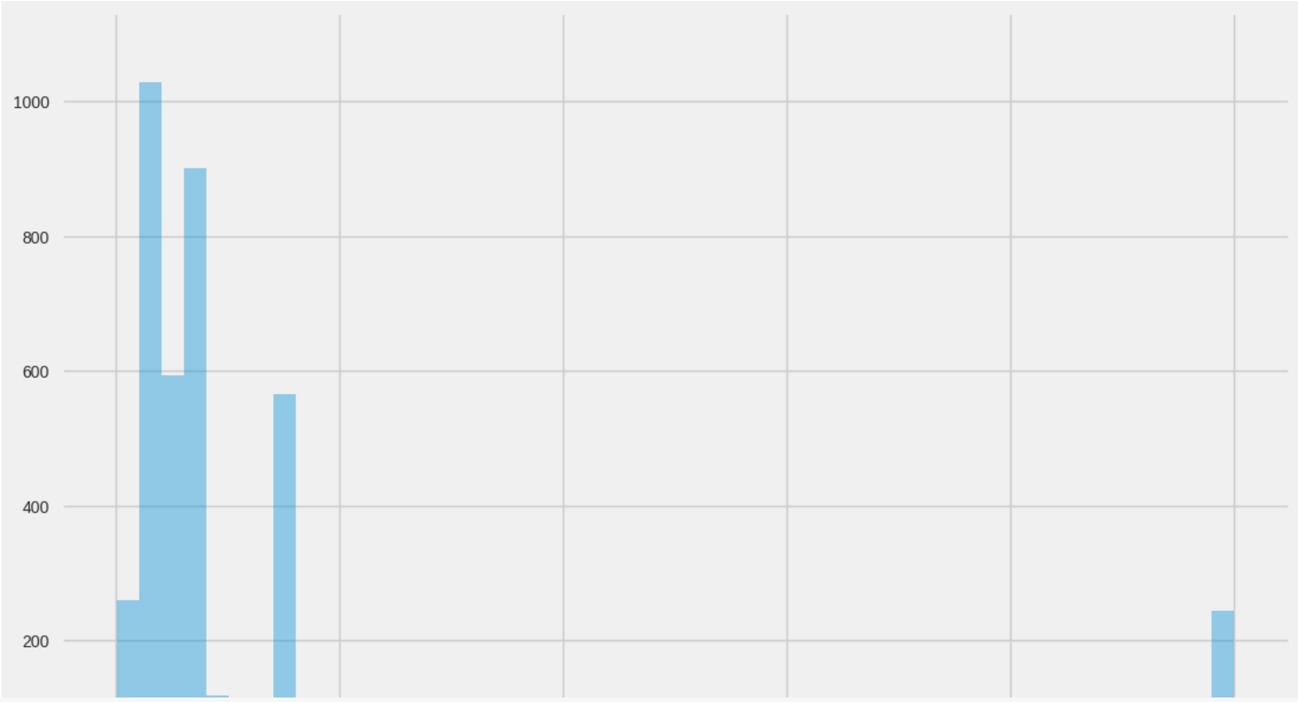

```
plt.figure(figsize=(12,6))
sns.jointplot(x=data['capital-gain'], y=data['capital-loss'])
```

```
<seaborn.axisgrid.JointGrid at 0x7fa0aee032d0>
<Figure size 1200x600 with 0 Axes>
```



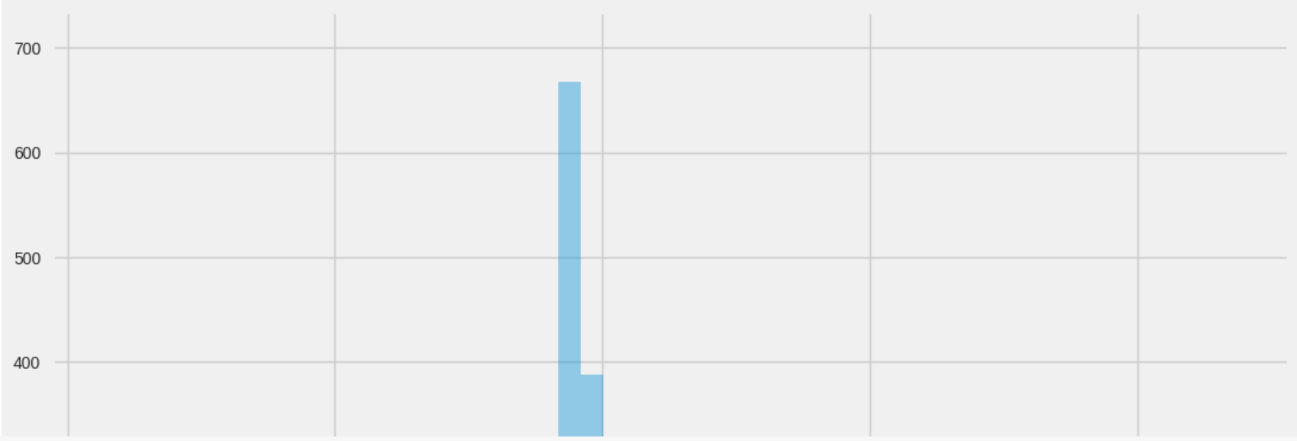

```
plt.figure(figsize=(12,8))
sns.distplot(data[(data['capital-gain']!=0)]['capital-gain'],kde=False, rug=True)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2056: FutureWarning:
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0ab53e490>
```




```
plt.figure(figsize=(12,8))
sns.distplot(data[(data['capital-loss']!=0)]['capital-loss'], kde=False,rug=True)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2056: FutureWarning:
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0ab7a0610>
```




```
plt.figure(figsize=(20,6))
ax=sns.barplot(x='occupation', y='income', data=data)
ax.set(ylabel='Fraction of people with income > $50k')
```

```
[Text(0, 0.5, 'Fraction of people with income > $50k')]
```


```
print(data['native-country'].value_counts())
not_from_US = np.sum(data['native-country']!='United-States')
print(not_from_US, 'people are not from the United States')
```

```
United-States              43832
Mexico                       951
?                            857
Philippines                  295
Germany                      206
```

```
Puerto-Rico                     184
Canada                          182
El-Salvador                     155
India                           151
Cuba                            138
England                         127
China                           122
South                           115
Jamaica                         106
Italy                           105
Dominican-Republic              103
Japan                            92
Guatemala                        88
Poland                           87
Vietnam                          86
Columbia                         85
Haiti                            75
Portugal                         67
Taiwan                           65
Iran                             59
Greece                           49
Nicaragua                        49
Peru                             46
Ecuador                          45
France                           38
Ireland                          37
Thailand                         30
Hong                             30
Cambodia                         28
Trinadad&Tobago                  27
Outlying-US(Guam-USVI-etc)       23
Yugoslavia                       23
Laos                             23
Scotland                         21
Honduras                         20
Hungary                          19
Holand-Netherlands                1
Name: native-country, dtype: int64
5010 people are not from the United States
```

Convert the native-country feature to binary since there is a huge imbalance in this feature

```
data['native-country'] = (data['native-country']=='United-States')*1
```

```
data['native-country'].value_counts()
```

```
1    43832
0     5010
Name: native-country, dtype: int64
```

Now time to work clean data set

```
data.select_dtypes(exclude=[np.number]).head()
```

	workclass	education	marital-status	occupation	relationship	race	gender
0	Private	11th	Never-married	Machine-op-inspct	Own-child	Black	Male
1	Private	HS-grad	Married-civ-spouse	Farming-fishing	Husband	White	Male
		Assoc-	Married-civ-				

```
#Replace all '?'s with NaNs.
data = data.applymap(lambda x: np.nan if x=='?' else x)
```

```
data.isnull().sum(axis=0) # How many issing values are there in the dataset?
```

```
age                   0
workclass          2799
fnlwgt                0
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        0
income                0
dtype: int64
```

```
data.shape[0] - data.dropna(axis=0).shape[0]   # how many rows will be removed if I remove
```

```
2809
```

```
data = data.dropna(axis=0) ## Drop all the NaNs
```

```
data.education.value_counts()  # I will label-encode the education column since it is an o
```

```
HS-grad         14972
Some-college    10036
Bachelors        7772
Masters          2590
Assoc-voc        1978
11th             1631
Assoc-acdm       1529
10th             1239
7th-8th           844
Prof-school       810
9th               687
12th              599
Doctorate         576
5th-6th           468
1st-4th           229
```

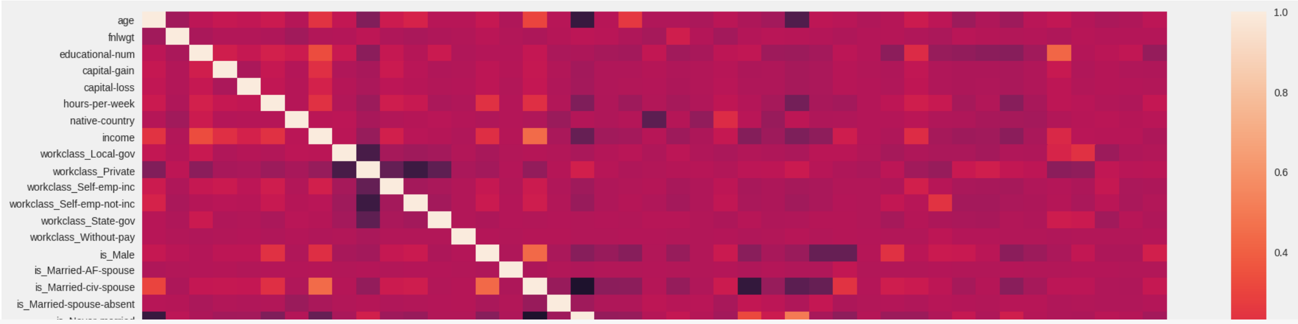
```
Preschool          73
Name: education, dtype: int64
```

One-hot encoding of the categorical columns

```
data = pd.get_dummies(data,columns=['workclass','gender', 'marital-status',
                                    'race','relationship','occupation'],
               prefix=['workclass', 'is', 'is', 'race_is', 'relation', 'is'], drop_first=T
### native country is ignored because that feature will be dropped later
```

```
plt.figure(figsize=(20,12))
sns.heatmap(data.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0aed34910>
```




```
data.select_dtypes(exclude=[np.number]).shape
```

```
(46033, 1)
```


```
data.groupby('income').mean()
```

	age	fnlwgt	educational-num	capital-gain	capital-loss	hours-per-week	na co
income							
0	36.756320	190220.927451	9.639479	147.992604	53.942128	39.383549	0.8
1	44.011819	188545.149536	11.612064	4042.540974	194.141744	45.690247	0.9

```
data.shape
```

```
(46033, 44)
```

```
y = data.income
X = data.drop(['income', 'education', 'native-country', 'fnlwgt'],axis=1)
```

> **income is dropped from X because it is the target variable.**

> **Education is dropped because it is already label-encoded in education.num. One can notice the high correlation between education and education.num in the heatmap.**

> **native country is dropped because it showed very little feature importance in random forest classifer.**

> **fnlwgt is dropped because it has no correlation with income.**

Modelling

This section explores different classification algorithms to maximise the accuracy for predicting income of a person (> 50k/yr or < 50k/yr).

```
from sklearn.model_selection import train_test_split
```

```
# Split the dataset into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from xgboost import XGBClassifier as xgb
from sklearn import metrics
```

Baseline model

> **In the baseline model, we predict the minority class for all our train and test (or validation) examples. The resulting accuracy will serve as a benchmark for the ML models. In other words, the sophisticated ML models should have an accuracy which should at least better the baseline one.**

```
baseline_train = np.zeros(y_train.shape[0])
baseline_test = np.zeros(y_test.shape[0])
print('Accuracy on train data: %f%%' % (metrics.accuracy_score(y_train, baseline_train)))
print('Accuracy on test data: %f%%' %  (metrics.accuracy_score(y_test, baseline_test)))
```

```
Accuracy on train data: 0.752506%
Accuracy on test data: 0.750398%
```

Random Forest classifier

```
rfmodel = RandomForestClassifier(n_estimators=300,oob_score=True,min_samples_split=5,max_d
rfmodel.fit(X_train,y_train)
print(rfmodel)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=300,
                       n_jobs=None, oob_score=True, random_state=10, verbose=0,
                       warm_start=False)
```

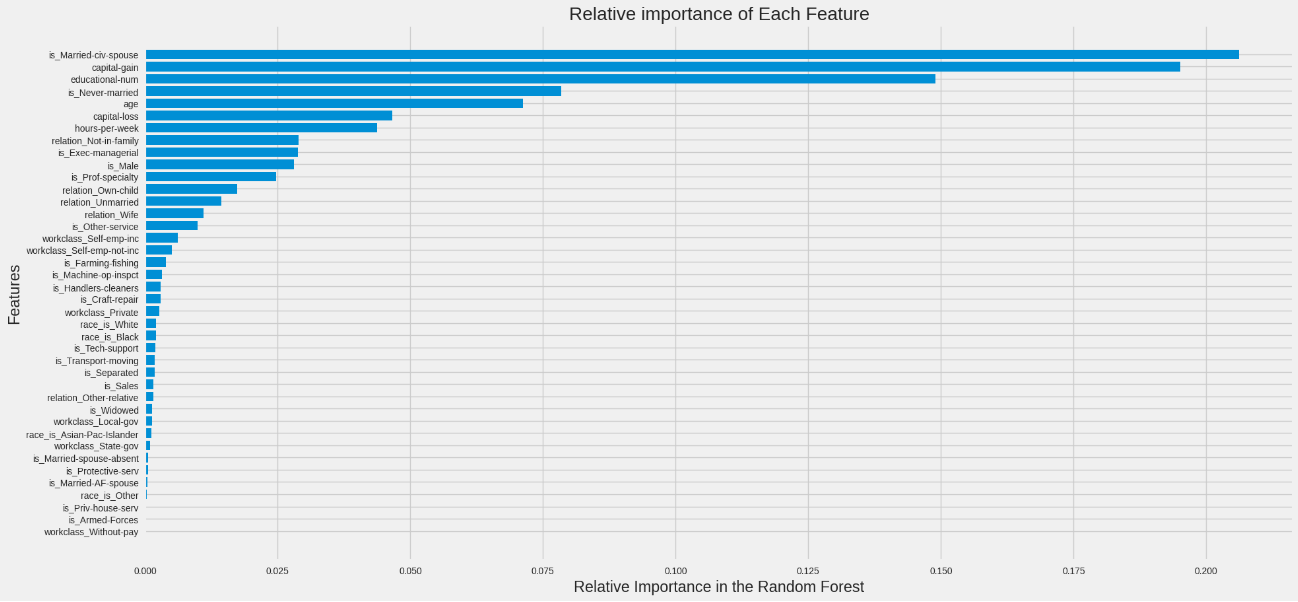
```
def show_classifier_metrics(clf, y_train=y_train,y_test=y_test, print_classification_repor
    print(clf)
    if print_confusion_matrix:
```

```
    if print_confusion_matrix:
        print('confusion matrix of training data')
        print(metrics.confusion_matrix(y_train, clf.predict(X_train)))
        print('confusion matrix of test data')
        print(metrics.confusion_matrix(y_test, clf.predict(X_test)))
    if print_classification_report:
        print('classification report of test data')
        print(metrics.classification_report(y_test, clf.predict(X_test)))
    print('Accuracy on test data: %f%%' % (metrics.accuracy_score(y_test, clf.predict(X_te
    print('Accuracy on training data: %f%%' % (metrics.accuracy_score(y_train, clf.predict
    print('Area under the ROC curve : %f' % (metrics.roc_auc_score(y_test, clf.predict(X_t
```

```
show_classifier_metrics(rfmodel,y_train)
print('RandomForestClassifier score = %f'% rfmodel.oob_score_)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=300,
                       n_jobs=None, oob_score=True, random_state=10, verbose=0,
                       warm_start=False)
confusion matrix of training data
[[23202  1046]
 [ 3371  4604]]
confusion matrix of test data
[[9885  478]
 [1489 1958]]
classification report of test data
              precision    recall  f1-score   support

           0       0.87      0.95      0.91     10363
           1       0.80      0.57      0.67      3447

    accuracy                           0.86     13810
   macro avg       0.84      0.76      0.79     13810
weighted avg       0.85      0.86      0.85     13810

Accuracy on test data: 85.756698%
Accuracy on training data: 86.292400%
Area under the ROC curve : 0.760952
RandomForestClassifier score = 0.856500
```

```
importance_list = rfmodel.feature_importances_
name_list = X_train.columns
importance_list, name_list = zip(*sorted(zip(importance_list, name_list)))
plt.figure(figsize=(20,10))
plt.barh(range(len(name_list)),importance_list,align='center')
plt.yticks(range(len(name_list)),name_list)
plt.xlabel('Relative Importance in the Random Forest')
plt.ylabel('Features')
plt.title('Relative importance of Each Feature')
plt.show()
```



Random forest: Grid Search and cross-validation

```
from sklearn.model_selection import cross_val_score, GridSearchCV

def grid_search(clf, parameters, X, y, n_jobs= -1, n_folds=4, score_func=None):
    if score_func:
        gs = GridSearchCV(clf, param_grid=parameters, cv=n_folds, n_jobs=n_jobs, scoring=s
    else:
        print('Doing grid search')
        gs = GridSearchCV(clf, param_grid=parameters, n_jobs=n_jobs, cv=n_folds, verbose =
    gs.fit(X, y)
    print("mean test score (weighted by split size) of CV rounds: ",gs.cv_results_['mean_t
    print ("Best parameter set", gs.best_params_, "Corresponding mean CV score",gs.best_sc
    best = gs.best_estimator_
    return best
```

```
rfmodel2 = RandomForestClassifier(min_samples_split=5,oob_score=True, n_jobs=-1,random_sta
parameters = {'n_estimators': [100,200,300], 'max_depth': [10,13,15,20]}
rfmodelCV = grid_search(rfmodel2, parameters,X_train,y_train)
```

```
Doing grid search
Fitting 4 folds for each of 12 candidates, totalling 48 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed:  2.7min finished
mean test score (weighted by split size) of CV rounds:  [0.85628277 0.85622071 0.856
 0.85988268 0.86003782 0.86006887 0.85916886 0.85907578 0.85895168]
Best parameter set {'max_depth': 15, 'n_estimators': 300} Corresponding mean CV score
```

```
rfmodelCV.fit(X_train,y_train)
show_classifier_metrics(rfmodelCV,y_train)
print('RandomForestClassifier score = %f'% rfmodelCV.oob_score_)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=15, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=300,
                       n_jobs=-1, oob_score=True, random_state=10, verbose=0,
                       warm_start=False)
confusion matrix of training data
[[23236  1012]
 [ 2839  5136]]
confusion matrix of test data
[[9819  544]
 [1389 2058]]
classification report of test data
              precision    recall  f1-score   support

           0       0.88      0.95      0.91     10363
           1       0.79      0.60      0.68      3447

    accuracy                           0.86     13810
   macro avg       0.83      0.77      0.80     13810
weighted avg       0.85      0.86      0.85     13810

Accuracy on test data: 86.002896%
Accuracy on training data: 88.048909%
Area under the ROC curve : 0.772273
RandomForestClassifier score = 0.860379
```

XGBoost

```
from xgboost.sklearn import XGBClassifier

param = {}
param['learning_rate'] = 0.1
param['verbosity'] = 1
param['colsample_bylevel'] = 0.9
param['colsample_bytree'] = 0.9
```

```
param['subsample'] = 0.9
param['reg_lambda']= 1.5
param['max_depth'] = 5
param['n_estimators'] = 400
param['seed']=10
xgb= XGBClassifier(**param)
xgb.fit(X_train, y_train, eval_metric=['error'], eval_set=[(X_train, y_train),(X_test, y_t
```

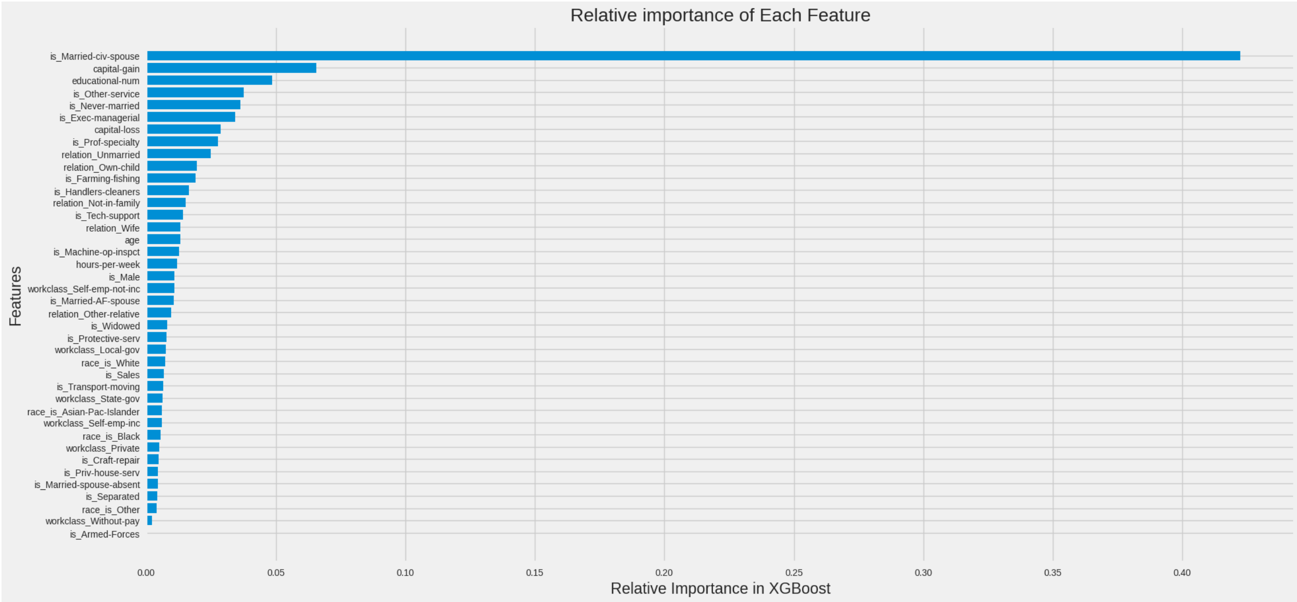
```
[220]   validation_0-error:0.119356     validation_1-error:0.129471
[221]   validation_0-error:0.119325     validation_1-error:0.129471
[222]   validation_0-error:0.119076     validation_1-error:0.129471
[223]   validation_0-error:0.119232     validation_1-error:0.129471
[224]   validation_0-error:0.118921     validation_1-error:0.129616
[225]   validation_0-error:0.119045     validation_1-error:0.129544
[226]   validation_0-error:0.119076     validation_1-error:0.129327
[227]   validation_0-error:0.119076     validation_1-error:0.129399
[228]   validation_0-error:0.118859     validation_1-error:0.129327
[229]   validation_0-error:0.118859     validation_1-error:0.129254
[230]   validation_0-error:0.11889      validation_1-error:0.129471
[231]   validation_0-error:0.118921     validation_1-error:0.129471
[232]   validation_0-error:0.118952     validation_1-error:0.129327
[233]   validation_0-error:0.118921     validation_1-error:0.129254
[234]   validation_0-error:0.118828     validation_1-error:0.129254
[235]   validation_0-error:0.11858      validation_1-error:0.12882
[236]   validation_0-error:0.118735     validation_1-error:0.128747
[237]   validation_0-error:0.11858      validation_1-error:0.12882
[238]   validation_0-error:0.118611     validation_1-error:0.128747
[239]   validation_0-error:0.118642     validation_1-error:0.12882
[240]   validation_0-error:0.11858      validation_1-error:0.12882
[241]   validation_0-error:0.118487     validation_1-error:0.128965
[242]   validation_0-error:0.118487     validation_1-error:0.128965
[243]   validation_0-error:0.118394     validation_1-error:0.128675
[244]   validation_0-error:0.118301     validation_1-error:0.128892
[245]   validation_0-error:0.118083     validation_1-error:0.128892
[246]   validation_0-error:0.11799      validation_1-error:0.128965
[247]   validation_0-error:0.117897     validation_1-error:0.128965
[248]   validation_0-error:0.117959     validation_1-error:0.129037
[249]   validation_0-error:0.117897     validation_1-error:0.12882
[250]   validation_0-error:0.118083     validation_1-error:0.128602
[251]   validation_0-error:0.118021     validation_1-error:0.128892
[252]   validation_0-error:0.118239     validation_1-error:0.128965
[253]   validation_0-error:0.118332     validation_1-error:0.128675
[254]   validation_0-error:0.118207     validation_1-error:0.128602
[255]   validation_0-error:0.118301     validation_1-error:0.128747
[256]   validation_0-error:0.11827      validation_1-error:0.128675
[257]   validation_0-error:0.118301     validation_1-error:0.128675
[258]   validation_0-error:0.118207     validation_1-error:0.12853
[259]   validation_0-error:0.11799      validation_1-error:0.128458
[260]   validation_0-error:0.117897     validation_1-error:0.128458
[261]   validation_0-error:0.118052     validation_1-error:0.128747
[262]   validation_0-error:0.117928     validation_1-error:0.12824
[263]   validation_0-error:0.117959     validation_1-error:0.128602
[264]   validation_0-error:0.118021     validation_1-error:0.128747
[265]   validation_0-error:0.118207     validation_1-error:0.12882
[266]   validation_0-error:0.118021     validation_1-error:0.128965
[267]   validation_0-error:0.117804     validation_1-error:0.129182
[268]   validation_0-error:0.117742     validation_1-error:0.129327
[269]   validation_0-error:0.117618     validation_1-error:0.129182
[270]   validation_0-error:0.117587     validation_1-error:0.129182
[271]   validation_0-error:0.11768      validation_1-error:0.129254
[272]   validation_0-error:0.117587     validation_1-error:0.129761
```

```
[273]	validation_0-error:0.117525	validation_1-error:0.129689
[274]	validation_0-error:0.117556	validation_1-error:0.129761
[275]	validation_0-error:0.117525	validation_1-error:0.129471
[276]	validation_0-error:0.11737	validation_1-error:0.129399
[277]	validation_0-error:0.117401	validation_1-error:0.129399
[278]	validation_0-error:0.117432	validation_1-error:0.129327
```

```
show_classifier_metrics(xgb,y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.9,
              colsample_bynode=1, colsample_bytree=0.9, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=400, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1.5, scale_pos_weight=1, seed=10,
              silent=None, subsample=0.9, verbosity=1)
confusion matrix of training data
[[22922  1326]
 [ 2474  5501]]
confusion matrix of test data
[[9735  628]
 [1143 2304]]
classification report of test data
              precision    recall  f1-score   support

           0       0.89      0.94      0.92     10363
           1       0.79      0.67      0.72      3447

    accuracy                           0.87     13810
   macro avg       0.84      0.80      0.82     13810
weighted avg       0.87      0.87      0.87     13810

Accuracy on test data: 87.175959%
Accuracy on training data: 88.207181%
Area under the ROC curve : 0.803904
```

```
importance_list = xgb.feature_importances_
name_list = X_train.columns
importance_list, name_list = zip(*sorted(zip(importance_list, name_list)))
plt.figure(figsize=(20,10))
plt.barh(range(len(name_list)),importance_list,align='center')
plt.yticks(range(len(name_list)),name_list)
plt.xlabel('Relative Importance in XGBoost')
plt.ylabel('Features')
plt.title('Relative importance of Each Feature')
plt.show()
```



Grid search with cross validation: XGBoost model

```
xgbmodel2 = XGBClassifier(seed=42)
param = {
'learning_rate': [0.1],#[0.1,0.2],
#'verbosity': [1],
'colsample_bylevel': [0.9],
'colsample_bytree': [0.9],
'subsample' : [0.9],
'n_estimators': [300],
'reg_lambda': [1.5,2,2.5],
'max_depth': [3,5,7],
 'seed': [10]
}
xgbCV = grid_search(xgbmodel2, param,X_train,y_train)
```

```
Doing grid search
Fitting 4 folds for each of 9 candidates, totalling 36 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:  5.0min finished
mean test score (weighted by split size) of CV rounds:  [0.86832378 0.86795134 0.868(
 0.8650032  0.86512736 0.8662445 ]
Best parameter set {'colsample_bylevel': 0.9, 'colsample_bytree': 0.9, 'learning_rate
```

```
xgbCV.fit(X_train, y_train, eval_metric=['error'], eval_set=[(X_train, y_train),(X_test, y
```

```
[189]   validation_0-error:0.130342     validation_1-error:0.133526
```

```
[190]	validation_0-error:0.130124	validation_1-error:0.132947
[191]	validation_0-error:0.130155	validation_1-error:0.132947
[192]	validation_0-error:0.130218	validation_1-error:0.133454
[193]	validation_0-error:0.130249	validation_1-error:0.133309
[194]	validation_0-error:0.130155	validation_1-error:0.13302
[195]	validation_0-error:0.130187	validation_1-error:0.13302
[196]	validation_0-error:0.130155	validation_1-error:0.13302
[197]	validation_0-error:0.129969	validation_1-error:0.132875
[198]	validation_0-error:0.130031	validation_1-error:0.13273
[199]	validation_0-error:0.129752	validation_1-error:0.132657
[200]	validation_0-error:0.12969	validation_1-error:0.132368
[201]	validation_0-error:0.13	validation_1-error:0.132223
[202]	validation_0-error:0.129969	validation_1-error:0.132223
[203]	validation_0-error:0.129845	validation_1-error:0.132078
[204]	validation_0-error:0.129721	validation_1-error:0.132223
[205]	validation_0-error:0.12969	validation_1-error:0.132223
[206]	validation_0-error:0.129659	validation_1-error:0.132151
[207]	validation_0-error:0.129566	validation_1-error:0.132078
[208]	validation_0-error:0.129504	validation_1-error:0.132006
[209]	validation_0-error:0.129628	validation_1-error:0.131933
[210]	validation_0-error:0.129473	validation_1-error:0.131861
[211]	validation_0-error:0.129473	validation_1-error:0.131861
[212]	validation_0-error:0.129504	validation_1-error:0.131861
[213]	validation_0-error:0.129411	validation_1-error:0.132223
[214]	validation_0-error:0.129473	validation_1-error:0.132295
[215]	validation_0-error:0.129349	validation_1-error:0.132006
[216]	validation_0-error:0.129193	validation_1-error:0.132078
[217]	validation_0-error:0.129193	validation_1-error:0.131861
[218]	validation_0-error:0.129038	validation_1-error:0.131644
[219]	validation_0-error:0.128914	validation_1-error:0.131861
[220]	validation_0-error:0.128852	validation_1-error:0.131644
[221]	validation_0-error:0.128883	validation_1-error:0.131789
[222]	validation_0-error:0.128821	validation_1-error:0.131789
[223]	validation_0-error:0.12879	validation_1-error:0.131571
[224]	validation_0-error:0.128759	validation_1-error:0.131354
[225]	validation_0-error:0.128759	validation_1-error:0.131426
[226]	validation_0-error:0.128759	validation_1-error:0.131571
[227]	validation_0-error:0.128666	validation_1-error:0.131209
[228]	validation_0-error:0.128666	validation_1-error:0.131426
[229]	validation_0-error:0.12848	validation_1-error:0.131499
[230]	validation_0-error:0.128542	validation_1-error:0.131499
[231]	validation_0-error:0.128542	validation_1-error:0.131716
[232]	validation_0-error:0.128511	validation_1-error:0.131571
[233]	validation_0-error:0.128511	validation_1-error:0.131426
[234]	validation_0-error:0.128511	validation_1-error:0.131426
[235]	validation_0-error:0.128418	validation_1-error:0.131644
[236]	validation_0-error:0.128356	validation_1-error:0.131716
[237]	validation_0-error:0.128231	validation_1-error:0.131789
[238]	validation_0-error:0.128045	validation_1-error:0.131789
[239]	validation_0-error:0.127983	validation_1-error:0.131716
[240]	validation_0-error:0.128138	validation_1-error:0.131571
[241]	validation_0-error:0.128293	validation_1-error:0.131716
[242]	validation_0-error:0.128324	validation_1-error:0.131716
[243]	validation_0-error:0.128356	validation_1-error:0.131789
[244]	validation_0-error:0.128449	validation_1-error:0.131571
[245]	validation_0-error:0.128387	validation_1-error:0.131789
[246]	validation_0-error:0.128387	validation_1-error:0.132006
[247]	validation_0-error:0.128356	validation_1-error:0.131933
```

```
show_classifier_metrics(xgbCV,y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.9,
              colsample_bynode=1, colsample_bytree=0.9, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=300, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1.5, scale_pos_weight=1, seed=10,
              silent=None, subsample=0.9, verbosity=1)
confusion matrix of training data
[[22891  1357]
 [ 2789  5186]]
confusion matrix of test data
[[9774  589]
 [1223 2224]]
classification report of test data
              precision    recall  f1-score   support

           0       0.89      0.94      0.92     10363
           1       0.79      0.65      0.71      3447

    accuracy                           0.87     13810
   macro avg       0.84      0.79      0.81     13810
weighted avg       0.86      0.87      0.86     13810

Accuracy on test data: 86.879073%
Accuracy on training data: 87.133414%
Area under the ROC curve : 0.794181
```

Logistic regression

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

```
param = {
'C': [3,5,10],
'verbose': [1],
    'max_iter': [100,200,500,700]
}
logreg = LogisticRegression(random_state=10)
logreg_grid = grid_search(logreg, param, X_train,y_train, n_folds=3)
```

```
Doing grid search
Fitting 3 folds for each of 12 candidates, totalling 36 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:   42.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
mean test score (weighted by split size) of CV rounds:  [0.82022158 0.83766254 0.8445
 0.84588648 0.84557614 0.81953884 0.84160382 0.84390032 0.84628992]
Best parameter set {'C': 3, 'max_iter': 700, 'verbose': 1} Corresponding mean CV scor
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

```
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    5.2s finished
```

```
logreg_grid.fit(X_train, y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    5.2s finished
LogisticRegression(C=3, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=700,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=10, solver='lbfgs', tol=0.0001, verbose=1,
                   warm_start=False)
```

```
show_classifier_metrics(logreg_grid)
```

```
LogisticRegression(C=3, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=700,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=10, solver='lbfgs', tol=0.0001, verbose=1,
                   warm_start=False)
confusion matrix of training data
[[22433  1815]
 [ 3082  4893]]
confusion matrix of test data
[[9603  760]
 [1352 2095]]
classification report of test data
              precision    recall  f1-score   support

           0       0.88      0.93      0.90     10363
           1       0.73      0.61      0.66      3447

    accuracy                           0.85     13810
   macro avg       0.81      0.77      0.78     13810
weighted avg       0.84      0.85      0.84     13810

Accuracy on test data: 84.706734%
Accuracy on training data: 84.802781%
Area under the ROC curve : 0.767219
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
NBmodel = GaussianNB()
NBmodel.fit(X_train, y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
NBmodel.predict(X_test)
```

```
array([1, 0, 1, ..., 0, 0, 1])
```

```
show_classifier_metrics(NBmodel,y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
confusion matrix of training data
[[20777  3471]
 [ 2198  5777]]
confusion matrix of test data
[[8827 1536]
 [ 949 2498]]
classification report of test data
              precision    recall  f1-score   support

           0       0.90      0.85      0.88     10363
           1       0.62      0.72      0.67      3447

    accuracy                           0.82     13810
   macro avg       0.76      0.79      0.77     13810
weighted avg       0.83      0.82      0.82     13810

Accuracy on test data: 82.005793%
Accuracy on training data: 82.406976%
Area under the ROC curve : 0.788234
```

Stacked model

```
def create_stacked_dataset(clfs,modelnames, X_train=X_train,X_test=X_test):
    X_train_stack, X_test_stack = X_train, X_test
    for clf,modelname in zip(clfs,modelnames):
        temptrain = pd.DataFrame(clf.predict(X_train),index = X_train.index,columns=[model
        temptest  = pd.DataFrame(clf.predict(X_test),index = X_test.index,columns=[modelna
        X_train_stack = pd.concat([X_train_stack, temptrain], axis=1)
        X_test_stack = pd.concat([X_test_stack, temptest], axis=1)
    return (X_train_stack,X_test_stack)
```

```
X_train_stack,X_test_stack = create_stacked_dataset([rfmodelCV,logreg_grid,xgbCV],modelnam
```

```
X_train_stack.head(5)
```

	age	educational-num	capital-gain	capital-loss	hours-per-week	workclass_Local-gov	workclass_Pr
27836	29	13	0	0	50	0	
39794	23	11	0	0	36	0	

```
param = {}
param['learning_rate'] = 0.1
param['verbosity'] = 1
param['colsample_bylevel'] = 0.9
param['colsample_bytree'] = 0.9
param['subsample'] = 0.9
param['reg_lambda']= 1.5
param['max_depth'] = 5#10
param['n_estimators'] = 400
param['seed']=10
xgbstack= XGBClassifier(**param)
xgbstack.fit(X_train_stack, y_train, eval_metric=['error'], eval_set=[(X_train_stack, y_tr
```

```
[0]	validation_0-error:0.118207	validation_1-error:0.136133
Multiple eval metrics have been passed: 'validation_1-error' will be used for early s

Will train until validation_1-error hasn't improved in 30 rounds.
[1]	validation_0-error:0.118239	validation_1-error:0.136785
[2]	validation_0-error:0.118114	validation_1-error:0.135988
[3]	validation_0-error:0.118797	validation_1-error:0.133599
[4]	validation_0-error:0.118332	validation_1-error:0.134902
[5]	validation_0-error:0.118301	validation_1-error:0.135988
[6]	validation_0-error:0.117773	validation_1-error:0.136857
[7]	validation_0-error:0.117773	validation_1-error:0.13693
[8]	validation_0-error:0.117897	validation_1-error:0.137364
[9]	validation_0-error:0.117866	validation_1-error:0.137437
[10]	validation_0-error:0.118083	validation_1-error:0.138088
[11]	validation_0-error:0.118207	validation_1-error:0.138378
[12]	validation_0-error:0.118456	validation_1-error:0.13874
[13]	validation_0-error:0.117711	validation_1-error:0.137292
[14]	validation_0-error:0.117742	validation_1-error:0.137799
[15]	validation_0-error:0.117928	validation_1-error:0.138306
[16]	validation_0-error:0.117959	validation_1-error:0.138378
[17]	validation_0-error:0.118021	validation_1-error:0.138378
[18]	validation_0-error:0.118176	validation_1-error:0.138306
[19]	validation_0-error:0.118207	validation_1-error:0.138378
[20]	validation_0-error:0.118176	validation_1-error:0.138378
[21]	validation_0-error:0.117959	validation_1-error:0.138016
[22]	validation_0-error:0.117804	validation_1-error:0.137726
[23]	validation_0-error:0.117804	validation_1-error:0.137726
[24]	validation_0-error:0.117711	validation_1-error:0.137654
[25]	validation_0-error:0.117711	validation_1-error:0.137726
[26]	validation_0-error:0.117804	validation_1-error:0.137726
[27]	validation_0-error:0.117773	validation_1-error:0.137654
[28]	validation_0-error:0.117711	validation_1-error:0.137654
[29]	validation_0-error:0.117742	validation_1-error:0.137654
[30]	validation_0-error:0.117525	validation_1-error:0.137654
[31]	validation_0-error:0.117463	validation_1-error:0.137002
[32]	validation_0-error:0.117463	validation_1-error:0.137002
[33]	validation_0-error:0.117432	validation_1-error:0.137002
Stopping. Best iteration:
[3]	validation_0-error:0.118797	validation_1-error:0.133599
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.9,
              colsample_bynode=1, colsample_bytree=0.9, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=400, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1.5, scale_pos_weight=1, seed=10,
              silent=None, subsample=0.9, verbosity=1)
```

```
print(metrics.classification_report(y_test, xgbstack.predict(X_test_stack)))
print('Accuracy on test data: %f%%' % (metrics.accuracy_score(y_test, xgbstack.predict(X_t
print('Accuracy on training data: %f%%' % (metrics.accuracy_score(y_train, xgbstack.predic
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91     10363
           1       0.78      0.65      0.71      3447

    accuracy                           0.87     13810
   macro avg       0.83      0.79      0.81     13810
weighted avg       0.86      0.87      0.86     13810

Accuracy on test data: 86.640116%
Accuracy on training data: 88.120287%
```

Stacked model Grid Search

```
xgbstackCV = XGBClassifier(seed=10)
param_grid = {}
param_grid['learning_rate'] = [0.1]
param_grid['colsample_bylevel'] = [0.9]
param_grid['colsample_bytree'] = [0.9]
param_grid['subsample'] = [0.9]
param_grid['n_estimators'] = [300]
param_grid['reg_lambda']= [1.5]
param_grid['seed'] =[10]
param_grid['max_depth'] = [3,5,8,10]
xgbstackCV_grid = grid_search(xgbstackCV, param_grid,X_train_stack,y_train)
```

```
Doing grid search
Fitting 4 folds for each of 4 candidates, totalling 16 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 out of  16 | elapsed:  3.1min finished
mean test score (weighted by split size) of CV rounds:  [0.88306472 0.88123373 0.875
Best parameter set {'colsample_bylevel': 0.9, 'colsample_bytree': 0.9, 'learning_rate
```

```
xgbstackCV_grid.fit(X_train_stack, y_train, eval_metric=['error'], eval_set=[(X_train_stac
```

```
[0]     validation_0-error:0.119511     validation_1-error:0.139971
Multiple eval metrics have been passed: 'validation_1-error' will be used for early s

Will train until validation_1-error hasn't improved in 30 rounds.
```

```
[1]	validation_0-error:0.119511	validation_1-error:0.139971
[2]	validation_0-error:0.118083	validation_1-error:0.137002
[3]	validation_0-error:0.121063	validation_1-error:0.134902
[4]	validation_0-error:0.118083	validation_1-error:0.137002
[5]	validation_0-error:0.118083	validation_1-error:0.137075
[6]	validation_0-error:0.118363	validation_1-error:0.137799
[7]	validation_0-error:0.119511	validation_1-error:0.139971
[8]	validation_0-error:0.119511	validation_1-error:0.139971
[9]	validation_0-error:0.119511	validation_1-error:0.139971
[10]	validation_0-error:0.119511	validation_1-error:0.139971
[11]	validation_0-error:0.119511	validation_1-error:0.139971
[12]	validation_0-error:0.119511	validation_1-error:0.139971
[13]	validation_0-error:0.119511	validation_1-error:0.139971
[14]	validation_0-error:0.119511	validation_1-error:0.139971
[15]	validation_0-error:0.119511	validation_1-error:0.139971
[16]	validation_0-error:0.119511	validation_1-error:0.139971
[17]	validation_0-error:0.119511	validation_1-error:0.139971
[18]	validation_0-error:0.119511	validation_1-error:0.139971
[19]	validation_0-error:0.119511	validation_1-error:0.139971
[20]	validation_0-error:0.119511	validation_1-error:0.139971
[21]	validation_0-error:0.119511	validation_1-error:0.139971
[22]	validation_0-error:0.119511	validation_1-error:0.139971
[23]	validation_0-error:0.11917	validation_1-error:0.139609
[24]	validation_0-error:0.11917	validation_1-error:0.139609
[25]	validation_0-error:0.11917	validation_1-error:0.139609
[26]	validation_0-error:0.11917	validation_1-error:0.139609
[27]	validation_0-error:0.11917	validation_1-error:0.139609
[28]	validation_0-error:0.119232	validation_1-error:0.139754
[29]	validation_0-error:0.119045	validation_1-error:0.139464
[30]	validation_0-error:0.119045	validation_1-error:0.139464
[31]	validation_0-error:0.119139	validation_1-error:0.139319
[32]	validation_0-error:0.119139	validation_1-error:0.139319
[33]	validation_0-error:0.119076	validation_1-error:0.139102
Stopping. Best iteration:
[3]	validation_0-error:0.121063	validation_1-error:0.134902

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.9,
              colsample_bynode=1, colsample_bytree=0.9, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=300, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1.5, scale_pos_weight=1, seed=10,
              silent=None, subsample=0.9, verbosity=1)
```

```
print(metrics.classification_report(y_test, xgbstack.predict(X_test_stack)))
print('Accuracy on test data: %f%%' % (metrics.accuracy_score(y_test, xgbstack.predict(X_t
print('Accuracy on training data: %f%%' % (metrics.accuracy_score(y_train, xgbstack.predic
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91     10363
           1       0.78      0.65      0.71      3447

    accuracy                           0.87     13810
   macro avg       0.83      0.79      0.81     13810
weighted avg       0.86      0.87      0.86     13810

Accuracy on test data: 86.640116%
Accuracy on training data: 88.120287%
```

```
pip install catboost
```

```
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/47/80/8e9c57ec32dfed6ba2922bc5c
     |████████████████████████████████| 67.3MB 56kB/s
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from ca
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (fror
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Installing collected packages: catboost
Successfully installed catboost-0.25.1
```

```
from catboost import CatBoostClassifier

catb = CatBoostClassifier(learning_rate=0.3,iterations=400,verbose=0,random_seed=10,eval_m
catb.fit(X_train,y_train,eval_set=[(X_train,y_train), (X_test,y_test)],early_stopping_roun
```

```
<catboost.core.CatBoostClassifier at 0x7fa0ac81ba10>
```

```
show_classifier_metrics(catb)
```

```
<catboost.core.CatBoostClassifier object at 0x7fa0ac81ba10>
confusion matrix of training data
[[22950  1298]
 [ 2598  5377]]
confusion matrix of test data
[[9763  600]
 [1186 2261]]
classification report of test data
              precision    recall  f1-score   support

           0       0.89      0.94      0.92     10363
           1       0.79      0.66      0.72      3447

    accuracy                           0.87     13810
   macro avg       0.84      0.80      0.82     13810
weighted avg       0.87      0.87      0.87     13810

Accuracy on test data: 87.067343%
Accuracy on training data: 87.909257%
Area under the ROC curve : 0.799017
```

Catboost grid search

```
catbCV = CatBoostClassifier(verbose=0,random_seed=10,eval_metric='Accuracy')
param_grid = {}
param_grid['learning_rate'] = [0.1]#, 0.3]
param_grid['rsm'] = [0.9]
#param_grid['subsample'] = [0.9]
param_grid['iterations'] = [200,300]
param_grid['reg_lambda']= [3] #2
param_grid['depth'] = [8,10]#5
catbCV_grid = grid_search(catbCV, param_grid,X_train,y_train)
```

```
Doing grid search
Fitting 4 folds for each of 4 candidates, totalling 16 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 out of  16 | elapsed:  2.5min finished
mean test score (weighted by split size) of CV rounds:  [0.8677032  0.86717554 0.864
Best parameter set {'depth': 8, 'iterations': 200, 'learning_rate': 0.1, 'reg_lambda
```

```
catbCV_grid.fit(X_train,y_train,eval_set=[(X_train,y_train), (X_test,y_test)],early_stoppi
```

```
<catboost.core.CatBoostClassifier at 0x7fa0aba2f0d0>
```

```
show_classifier_metrics(catbCV_grid)
```

```
<catboost.core.CatBoostClassifier object at 0x7fa0aba2f0d0>
confusion matrix of training data
[[23047  1201]
 [ 2505  5470]]
confusion matrix of test data
[[9764  599]
 [1194 2253]]
classification report of test data
              precision    recall  f1-score   support

           0       0.89      0.94      0.92     10363
           1       0.79      0.65      0.72      3447

    accuracy                           0.87     13810
   macro avg       0.84      0.80      0.82     13810
weighted avg       0.87      0.87      0.87     13810

Accuracy on test data: 87.016655%
Accuracy on training data: 88.498898%
Area under the ROC curve : 0.797905
```

```
from imblearn.over_sampling import RandomOverSampler
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/externals/six.py:31: FutureWarning: Th
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarnin
  warnings.warn(message, FutureWarning)
```

```
np.sum(y_train)/y_train.shape[0]
```

```
0.24749402600626882
```

```
ros = RandomOverSampler(random_state=1,sampling_strategy=0.8)
```

```
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)
catb_ros = CatBoostClassifier(learning_rate=0.1,iterations=400,reg_lambda=2,verbose=0,rand
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
```

```
catb_ros.fit(X_resampled,y_resampled,eval_set=[(X_resampled,y_resampled), (X_test,y_test)]
```

```
<catboost.core.CatBoostClassifier at 0x7fa0ac195ed0>
```

```
print('Accuracy on test data: %f%%' % (metrics.accuracy_score(y_test, catb_ros.predict(X_t
print('Accuracy on training data: %f%%' % (metrics.accuracy_score(y_resampled, catb_ros.pr
print('Area under the ROC curve : %f' % (metrics.roc_auc_score(y_test, catb_ros.predict(X_
```

```
Accuracy on test data: 84.786387%
Accuracy on training data: 86.612748%
Area under the ROC curve : 0.838804
```

SMOTE

```
from imblearn.over_sampling import SMOTE
```

```
smt = SMOTE(random_state=10,sampling_strategy=0.7)
X_train_smt, y_train_smt = smt.fit_sample(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
```

```
y_train.value_counts()
```

```
0    24248
1     7975
Name: income, dtype: int64
```

```
np.bincount(y_train_smt)
```

```
array([24248, 16973])
```

```
catb_smote = CatBoostClassifier(learning_rate=0.1,iterations=400,reg_lambda=2,verbose=0,ra
catb_smote.fit(X_train_smt,y_train_smt,eval_set=[(X_train_smt,y_train_smt), (X_test,y_test
```

```
<catboost.core.CatBoostClassifier at 0x7fa0ac10b1d0>
```

```
print(metrics.classification_report(y_test, catb_smote.predict(X_test)))
print('Accuracy on test data: %f%%' % (metrics.accuracy_score(y_test, catb_smote.predict(X_
print('Accuracy on training data: %f%%' % (metrics.accuracy_score(y_train_smt, catb_smote.
print('Area under the ROC curve : %f' % (metrics.roc_auc_score(y_test, catb_ros.predict(X_
```

```
              precision    recall  f1-score   support

           0       0.91      0.89      0.90     10363
           1       0.70      0.75      0.72      3447

    accuracy                           0.86     13810
   macro avg       0.81      0.82      0.81     13810
weighted avg       0.86      0.86      0.86     13810

Accuracy on test data: 85.684287%
Accuracy on training data: 88.632008%
Area under the ROC curve : 0.838804
```

SMOTE with XGBoost

```
param = {}
param['learning_rate'] = 0.1
param['verbosity'] = 1
param['colsample_bylevel'] = 0.9
param['colsample_bytree'] = 0.9
param['subsample'] = 0.9
param['reg_lambda']= 1.5
param['max_depth'] = 5
param['n_estimators'] = 400
param['seed']=10
xgb_smote= XGBClassifier(**param)
xgb_smote.fit(X_train_smt, y_train_smt, eval_metric=['error'], eval_set=[(X_train_smt, y_t
```

```
[147]	validation_0-error:0.120133	validation_1-error:0.144678
[148]	validation_0-error:0.12023	validation_1-error:0.143809
[149]	validation_0-error:0.119721	validation_1-error:0.144533
[150]	validation_0-error:0.120181	validation_1-error:0.143736
[151]	validation_0-error:0.119696	validation_1-error:0.144605
[152]	validation_0-error:0.119575	validation_1-error:0.144461
[153]	validation_0-error:0.119357	validation_1-error:0.144243
[154]	validation_0-error:0.119357	validation_1-error:0.143519
[155]	validation_0-error:0.119308	validation_1-error:0.143519
[156]	validation_0-error:0.119308	validation_1-error:0.143592
[157]	validation_0-error:0.119502	validation_1-error:0.143447
[158]	validation_0-error:0.118823	validation_1-error:0.143519
[159]	validation_0-error:0.118871	validation_1-error:0.143664
[160]	validation_0-error:0.118702	validation_1-error:0.143664
[161]	validation_0-error:0.118702	validation_1-error:0.143664
[162]	validation_0-error:0.118411	validation_1-error:0.143302
[163]	validation_0-error:0.118362	validation_1-error:0.143302
[164]	validation_0-error:0.118241	validation_1-error:0.14323
[165]	validation_0-error:0.118216	validation_1-error:0.143447
[166]	validation_0-error:0.118483	validation_1-error:0.142578
```

```
[167]	validation_0-error:0.118241	validation_1-error:0.142361
[168]	validation_0-error:0.118411	validation_1-error:0.142433
[169]	validation_0-error:0.117877	validation_1-error:0.143302
[170]	validation_0-error:0.117756	validation_1-error:0.14323
[171]	validation_0-error:0.118119	validation_1-error:0.142216
[172]	validation_0-error:0.117707	validation_1-error:0.143085
[173]	validation_0-error:0.117561	validation_1-error:0.142867
[174]	validation_0-error:0.117586	validation_1-error:0.142867
[175]	validation_0-error:0.117222	validation_1-error:0.142867
[176]	validation_0-error:0.117731	validation_1-error:0.142216
[177]	validation_0-error:0.117513	validation_1-error:0.142071
[178]	validation_0-error:0.117561	validation_1-error:0.142071
[179]	validation_0-error:0.116834	validation_1-error:0.14294
[180]	validation_0-error:0.116785	validation_1-error:0.142795
[181]	validation_0-error:0.116882	validation_1-error:0.142723
[182]	validation_0-error:0.117125	validation_1-error:0.142216
[183]	validation_0-error:0.117003	validation_1-error:0.142071
[184]	validation_0-error:0.116858	validation_1-error:0.141999
[185]	validation_0-error:0.116154	validation_1-error:0.142795
[186]	validation_0-error:0.116106	validation_1-error:0.14265
[187]	validation_0-error:0.116543	validation_1-error:0.141854
[188]	validation_0-error:0.116421	validation_1-error:0.141781
[189]	validation_0-error:0.116397	validation_1-error:0.141854
[190]	validation_0-error:0.116324	validation_1-error:0.141781
[191]	validation_0-error:0.116397	validation_1-error:0.141999
[192]	validation_0-error:0.116106	validation_1-error:0.142143
[193]	validation_0-error:0.115888	validation_1-error:0.142288
[194]	validation_0-error:0.115863	validation_1-error:0.142288
[195]	validation_0-error:0.115985	validation_1-error:0.142216
[196]	validation_0-error:0.116009	validation_1-error:0.142505
[197]	validation_0-error:0.115742	validation_1-error:0.142361
[198]	validation_0-error:0.115839	validation_1-error:0.142361
[199]	validation_0-error:0.115815	validation_1-error:0.142216
[200]	validation_0-error:0.115742	validation_1-error:0.142071
[201]	validation_0-error:0.115693	validation_1-error:0.141999
[202]	validation_0-error:0.115451	validation_1-error:0.142288
[203]	validation_0-error:0.115475	validation_1-error:0.142216
[204]	validation_0-error:0.115402	validation_1-error:0.142071
```

```
print(metrics.classification_report(y_test, xgb_smote.predict(X_test.values)))
print('Accuracy on test data: %f%%' % (metrics.accuracy_score(y_test, xgb_smote.predict(X_
print('Accuracy on training data: %f%%' % (metrics.accuracy_score(y_train_smt, xgb_smote.p
print('Area under the ROC curve : %f' % (metrics.roc_auc_score(y_test, xgb_smote.predict(X
```

```
              precision    recall  f1-score   support

           0       0.92      0.89      0.90     10363
           1       0.70      0.75      0.73      3447

    accuracy                           0.86     13810
   macro avg       0.81      0.82      0.82     13810
weighted avg       0.86      0.86      0.86     13810

Accuracy on test data: 85.821868%
Accuracy on training data: 88.357876%
Area under the ROC curve : 0.823439
```

Save model

```
import pickle
pickle.dump(xgb_smote, open('model.pkl','wb'))

model = pickle.load(open('model.pkl','rb'))

#prediction = model.predict(np.array([[20, 10, 3, 5, 40,20, 10, 3, 5, 40,20, 10, 3, 5, 40,
```

Conclusion:

> **In this project, we build various models like logistic regression, knn classifier, support vector classifier, decision tree classifier, random forest classifier and xgboost classifier.**

> **A hyperparameter tuned random forest classifier gives the highest accuracy score of 92.77 and f1 score of 93.08.**

Thank-you

0s completed at 6:21 PM