Census Income Project

Problem Statement:

This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics). A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) && (AGI>100) && (AFNLWGT>1) && (HRSWK>0)). The prediction task is to determine whether a person makes over $50K a year.

Description of fnlwgt (final weight) The weights on the Current Population Survey (CPS) files are controlled to independent estimates of the civilian non-institutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau. We use 3 sets of controls. These are:

- A single cell estimate of the population 16+ for each state.
- Controls for Hispanic Origin by age and sex.
- Controls by Race, age and sex.

We use all three sets of controls in our weighting program and "rake" through them 6 times so that by the end we come back to all the controls we used. The term estimate refers to population totals derived from CPS by creating "weighted tallies" of any specified socio-economic characteristics of the population. People with similar demographic characteristics should have similar weights. There is one important caveat to remember about this statement. That is that since the CPS sample is actually a collection of 51 state samples, each with its own probability of selection, the statement only applies within state.

Step 0: Load libraries and dataset

In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

Step 1: Descriptive analysis

In [2]:
```
dataset=pd.read_csv('Census Income Project.csv')
dataset.head()
```

Out[2]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	Ra
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	Wh
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	Wh
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Bla
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Bla
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	Wh

In [3]:
```
# Shape of dataset
print('Rows: {} Columns: {}'.format(dataset.shape[0], dataset.shape[1]))
```

```
Rows: 32560 Columns: 15
```

In [4]:
```
# Features data-type
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32560 entries, 0 to 32559
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Age             32560 non-null  int64 
 1   Workclass       32560 non-null  object
 2   Fnlwgt          32560 non-null  int64 
 3   Education       32560 non-null  object
 4   Education_num   32560 non-null  int64 
 5   Marital_status  32560 non-null  object
 6   Occupation      32560 non-null  object
 7   Relationship    32560 non-null  object
 8   Race            32560 non-null  object
 9   Sex             32560 non-null  object
 10  Capital_gain    32560 non-null  int64 
 11  Capital_loss    32560 non-null  int64 
 12  Hours_per_week  32560 non-null  int64 
 13  Native_country  32560 non-null  object
 14  Income          32560 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [5]:
```
# Statistical summary
dataset.describe()
```

Out[5]:

	Age	Fnlwgt	Education_num	Capital_gain	Capital_loss	Hours_per_week
count	32560.000000	3.256000e+04	32560.000000	32560.000000	32560.000000	32560.000000
mean	38.581634	1.897818e+05	10.080590	1077.615172	87.306511	40.437469
std	13.640642	1.055498e+05	2.572709	7385.402999	402.966116	12.347618
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178315e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783630e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370545e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

In [6]:
```
# Check for null values
round((dataset.isnull().sum() / dataset.shape[0]) * 100, 2).astype(str) + ' %'
```

Out[6]:
```
Age               0.0 %
Workclass         0.0 %
Fnlwgt            0.0 %
Education         0.0 %
Education_num     0.0 %
Marital_status    0.0 %
Occupation        0.0 %
Relationship      0.0 %
Race              0.0 %
Sex               0.0 %
Capital_gain      0.0 %
Capital_loss      0.0 %
Hours_per_week    0.0 %
Native_country    0.0 %
Income            0.0 %
dtype: object
```

In [7]:
```
# Check for '?' in dataset
round((dataset.isin(['?']).sum() / dataset.shape[0])
      * 100, 2).astype(str) + ' %'
```

Out[7]:
```
Age               0.0 %
Workclass         0.0 %
Fnlwgt            0.0 %
Education         0.0 %
Education_num     0.0 %
Marital_status    0.0 %
Occupation        0.0 %
Relationship      0.0 %
Race              0.0 %
Sex               0.0 %
Capital_gain      0.0 %
Capital_loss      0.0 %
Hours_per_week    0.0 %
Native_country    0.0 %
Income            0.0 %
dtype: object
```

In [8]:
```
# Checking the counts of label categories
Income = dataset['Income'].value_counts(normalize=True)
```

```
round(Income * 100, 2).astype('str') + ' %'
```

Out[8]:
```
 <=50K    75.92 %
 >50K     24.08 %
Name: Income, dtype: object
```

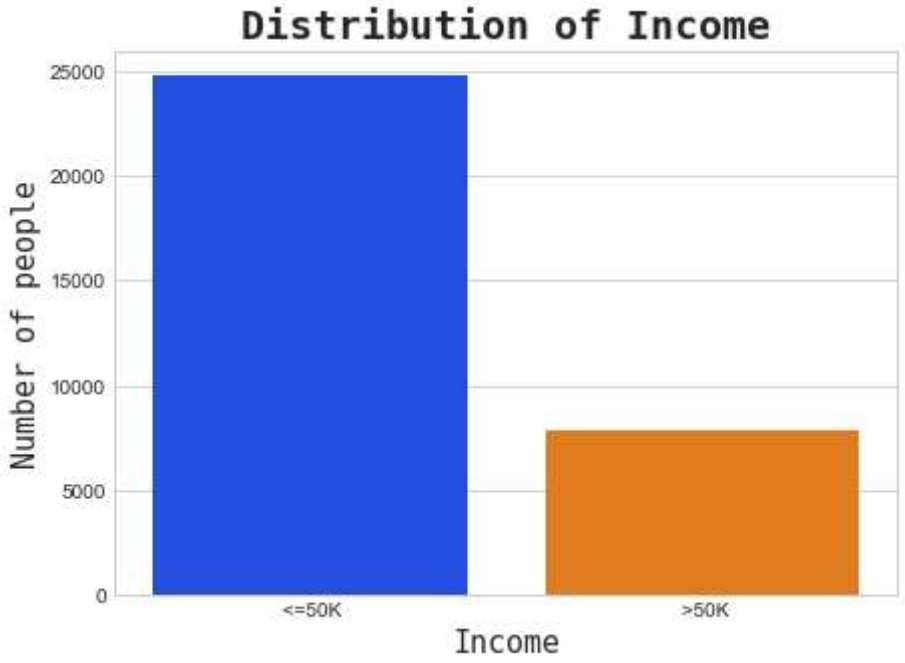
Observations:

- 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 75.92% values have income less than 50k and 24.08% values have income more than 50k.

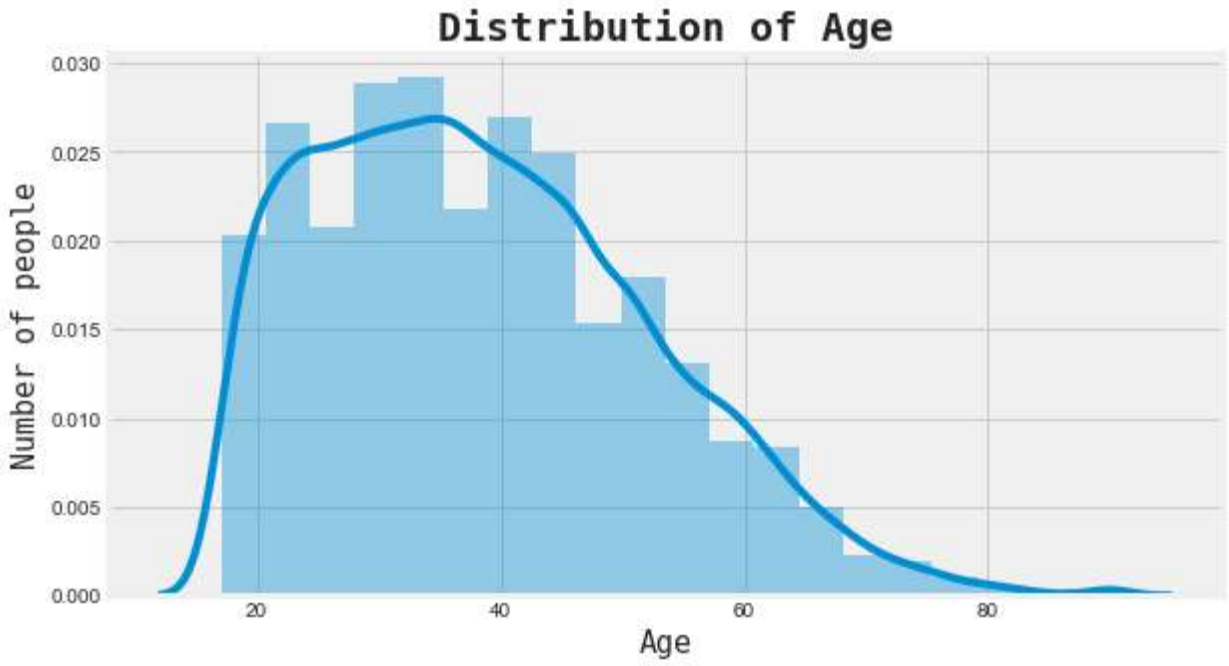
Step 2: Exploratory Data Analysis

Univariate Analysis

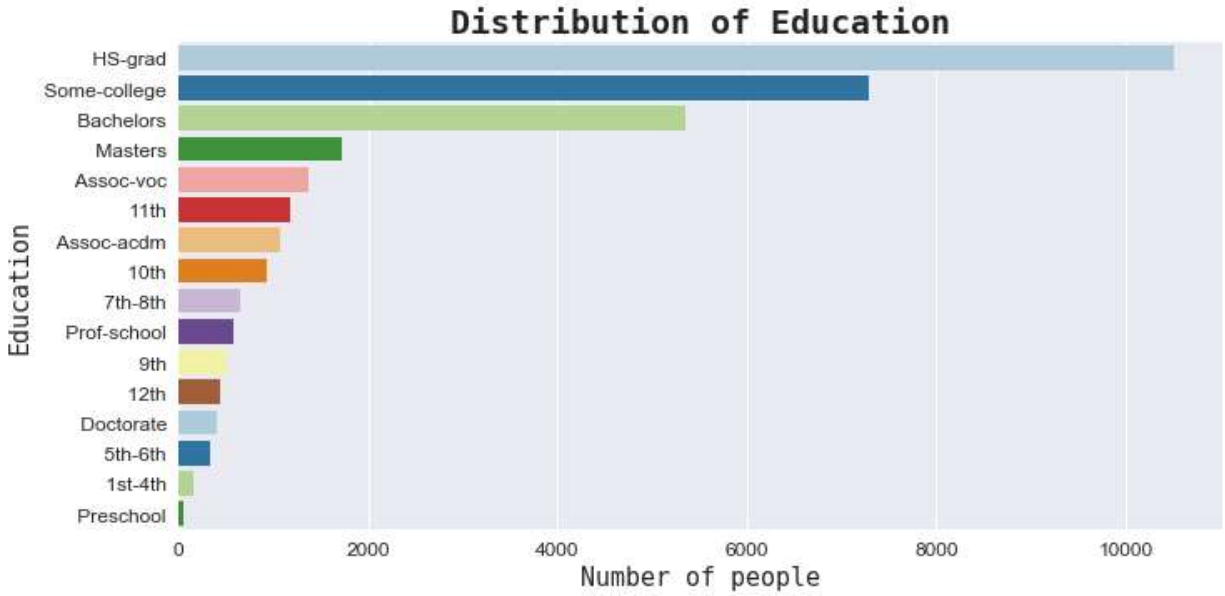
In [9]:
```
# Creating a barplot for 'Income'
Income = dataset['Income'].value_counts()

plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(7, 5))
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()
```



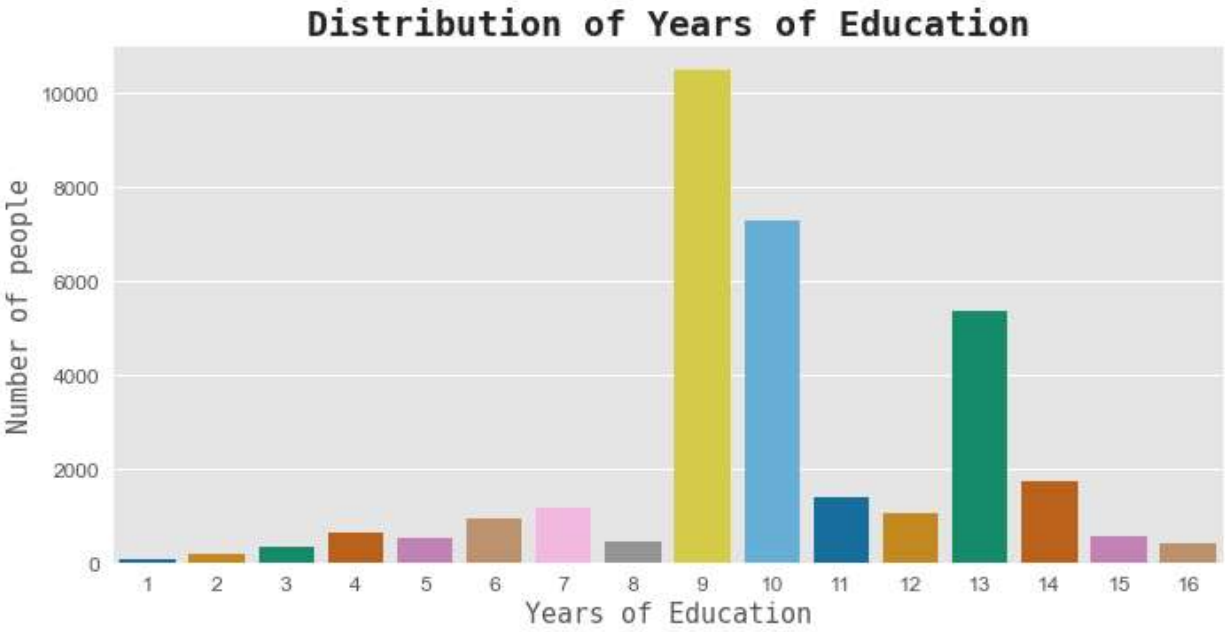
In [10]:
```
# Creating a distribution plot for 'Age'
Age = dataset['Age'].value_counts()

plt.figure(figsize=(10, 5))
plt.style.use('fivethirtyeight')
sns.distplot(dataset['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()
```



In [11]:
```
# Creating a barplot for 'Education'
edu = dataset['Education'].value_counts()

plt.style.use('seaborn')
plt.figure(figsize=(10, 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()
```

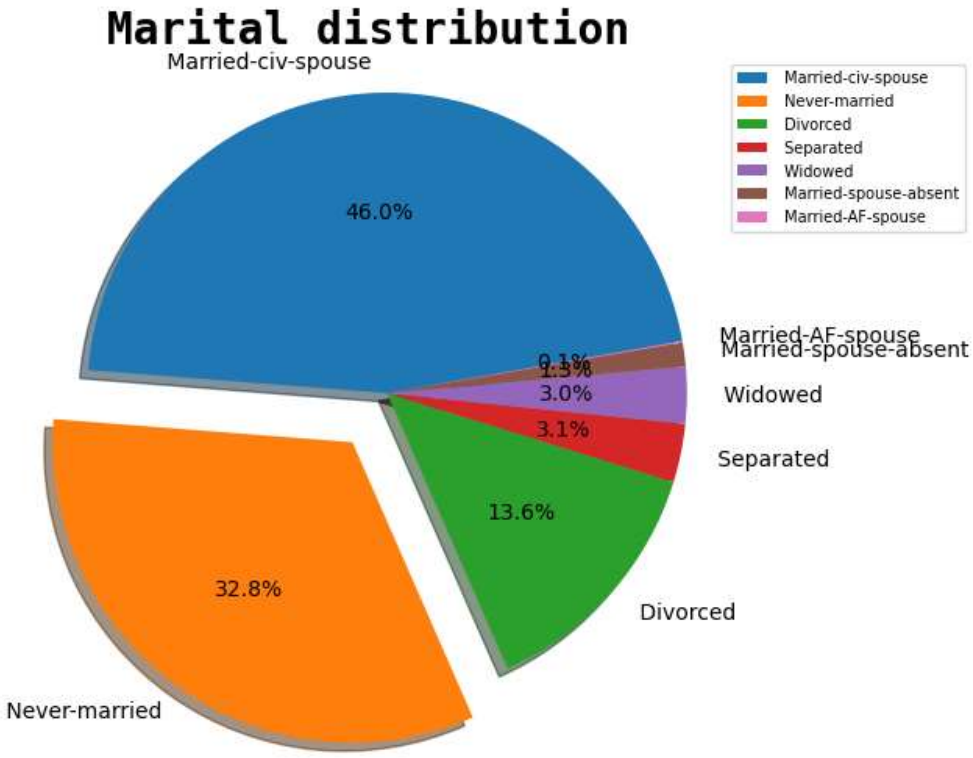


In [12]:
```
# Creating a barplot for 'Years of Education'
edu_num = dataset['Education_num'].value_counts()

plt.style.use('ggplot')
plt.figure(figsize=(10, 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()
```



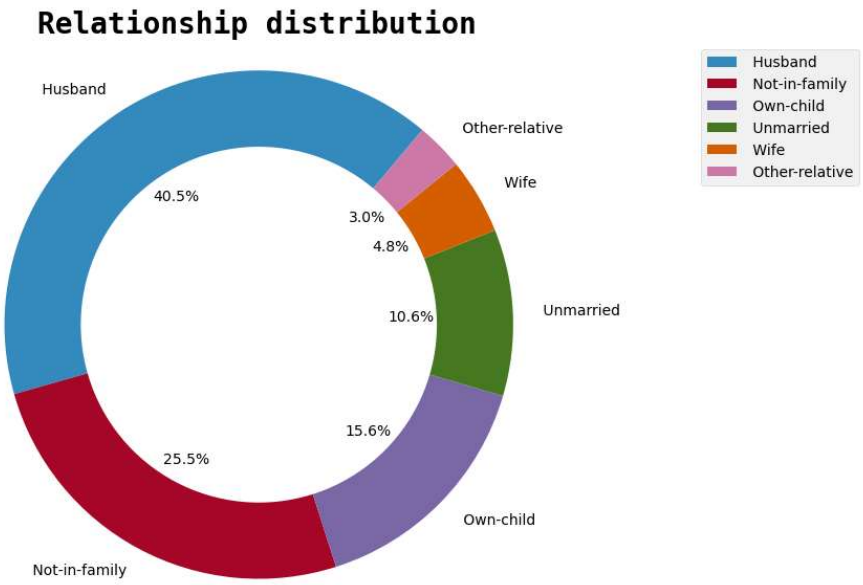
In [13]:
```
# Creating a pie chart for 'Marital status'
marital = dataset['Marital_status'].value_counts()

plt.style.use('default')
```

```
plt.figure(figsize=(10, 6))
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()
```

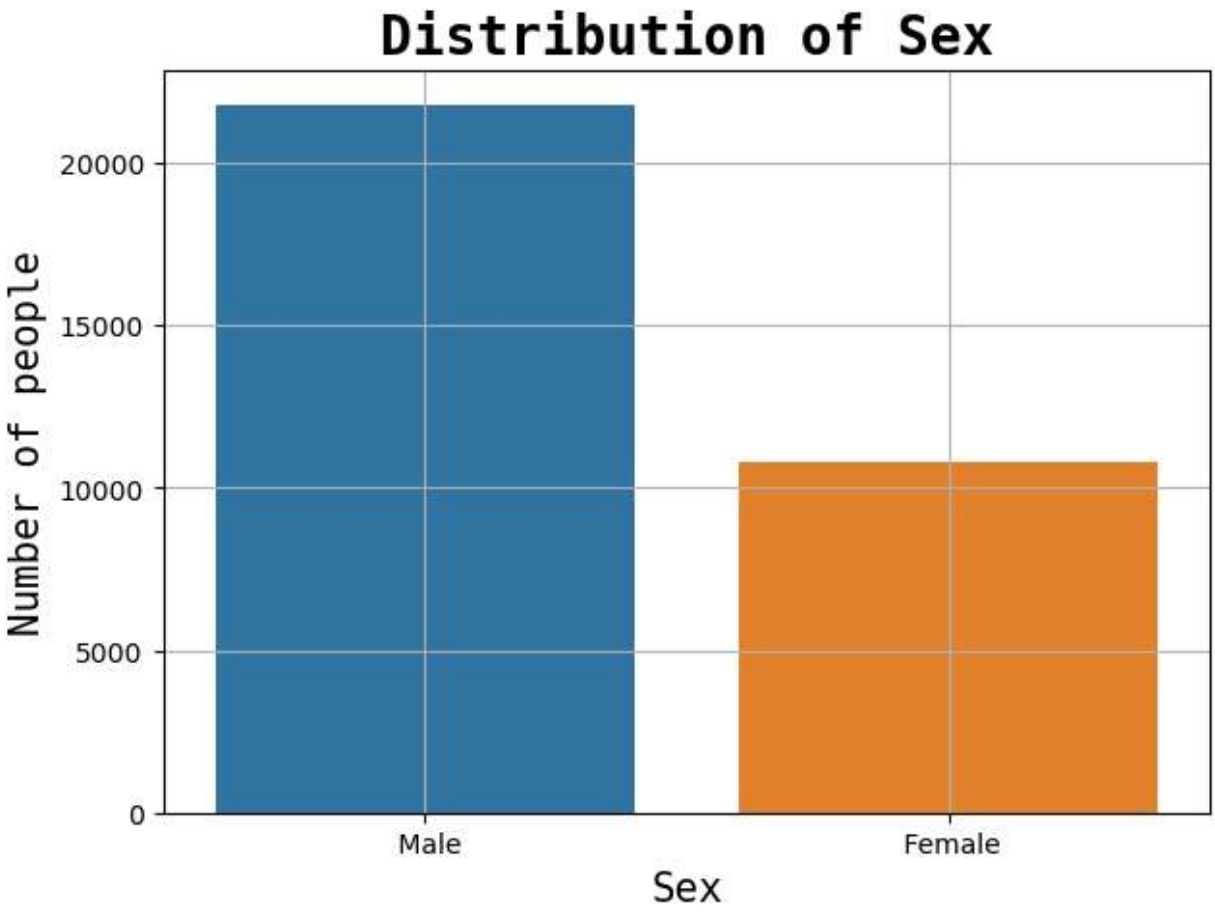


In [14]:

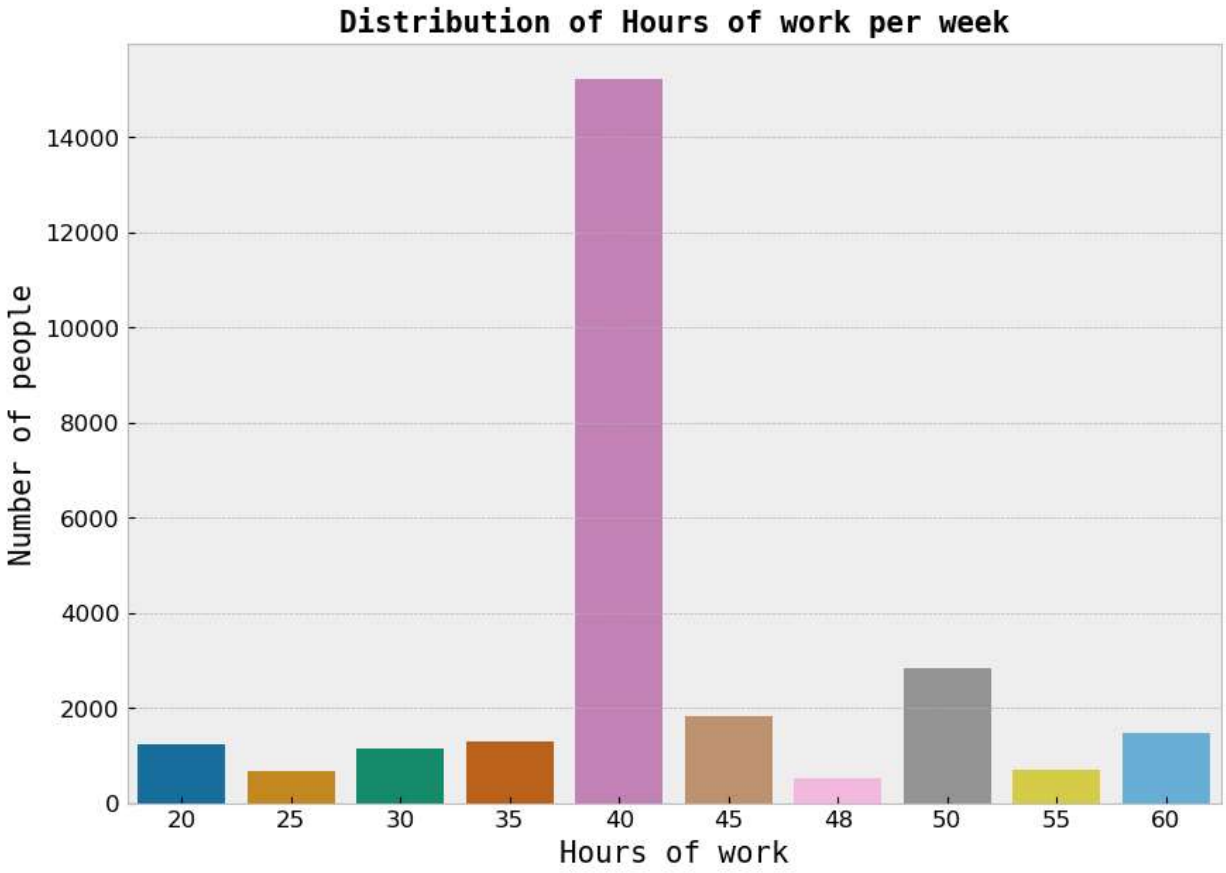
```
# Creating a donut chart for 'Age'
relation = dataset['Relationship'].value_counts()

plt.style.use('bmh')
plt.figure(figsize=(15, 7))
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': 20, 'fontweight': 'bold'})

plt.axis('equal')
plt.legend(prop={'size': 10})
plt.show()
```



In [15]:
```
# Creating a barplot for 'Sex'
sex = dataset['Sex'].value_counts()

plt.style.use('default')
plt.figure(figsize=(7, 5))
sns.barplot(sex.index, sex.values)
plt.title('Distribution of 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=10)
plt.grid()
plt.show()
```

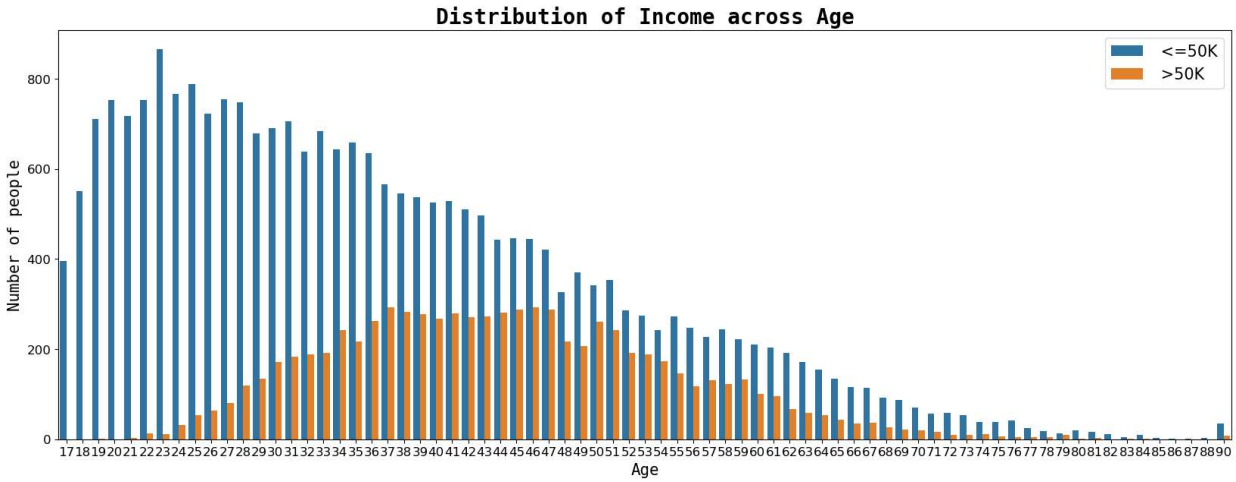


In [16]:
```
# Creating a barplot for 'Hours per week'
hours = dataset['Hours_per_week'].value_counts().head(10)

plt.style.use('bmh')
plt.figure(figsize=(10, 7))
sns.barplot(hours.index, hours.values, palette='colorblind')

plt.title('Distribution of Hours of work per week', fontdict={
          'fontname': 'Monospace', 'fontsize': 15, '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()
```

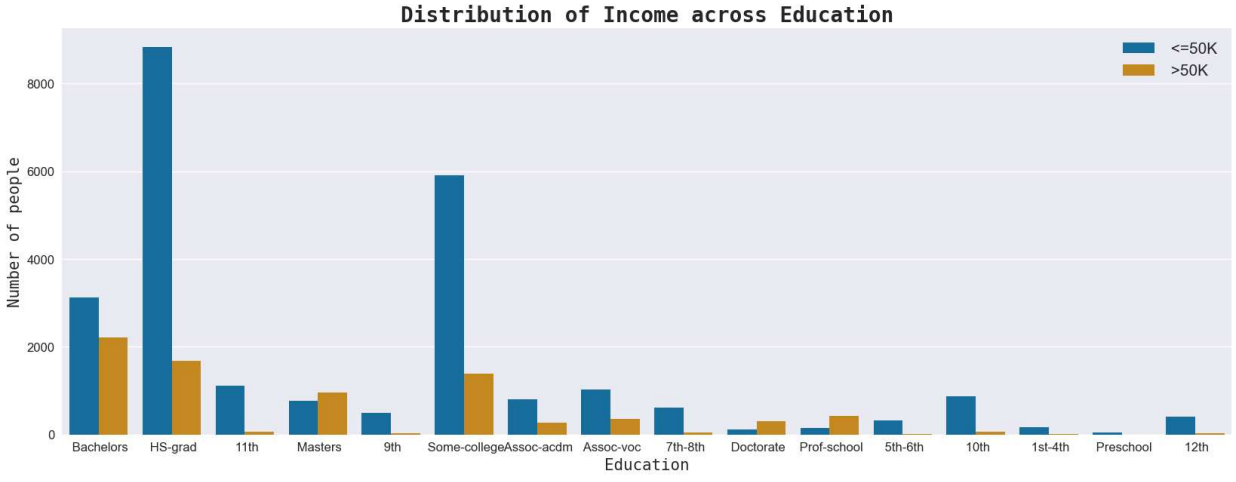


Bivariate Analysis

In [17]:
```
# Creating a countplot of income across age
plt.style.use('default')
plt.figure(figsize=(20, 7))
sns.countplot(dataset['Age'], hue=dataset['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()
```

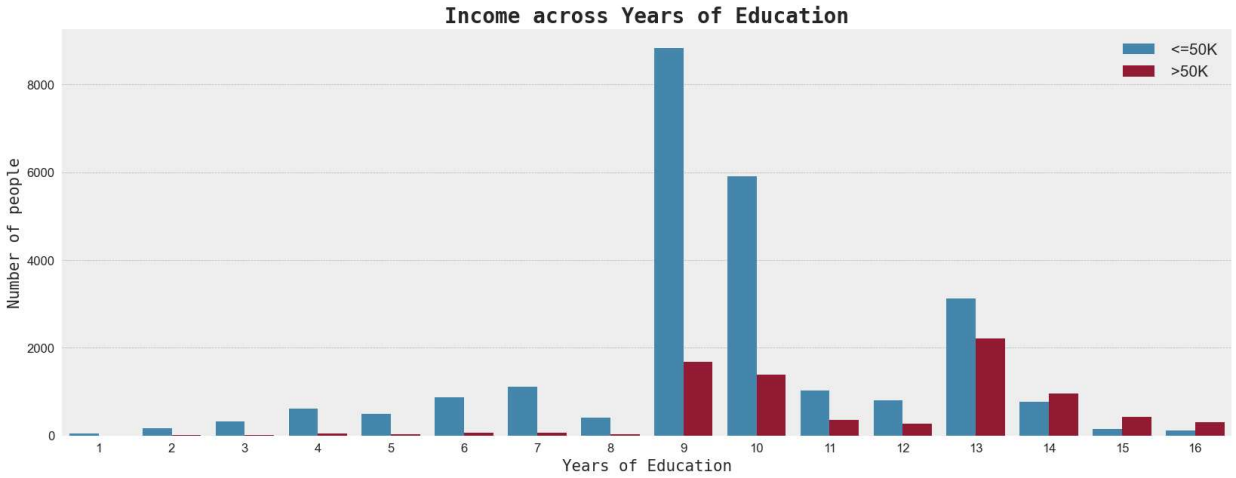


In [18]:
```
# Creating a countplot of income across education
plt.style.use('seaborn')
plt.figure(figsize=(20, 7))
sns.countplot(dataset['Education'],
              hue=dataset['Income'], palette='colorblind')

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()
```

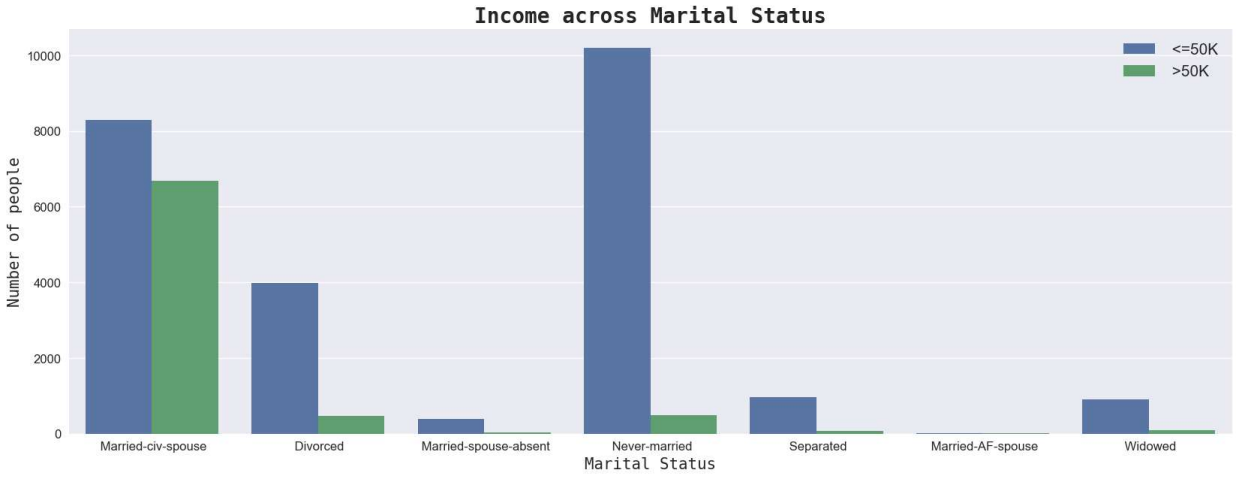


In [19]:
```
# Creating a countplot of income across years of education
plt.style.use('bmh')
plt.figure(figsize=(20, 7))
sns.countplot(dataset['Education_num'],
              hue=dataset['Income'])

plt.title('Income across Years of Education', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})

plt.xlabel('Years of Education', fontdict={
```

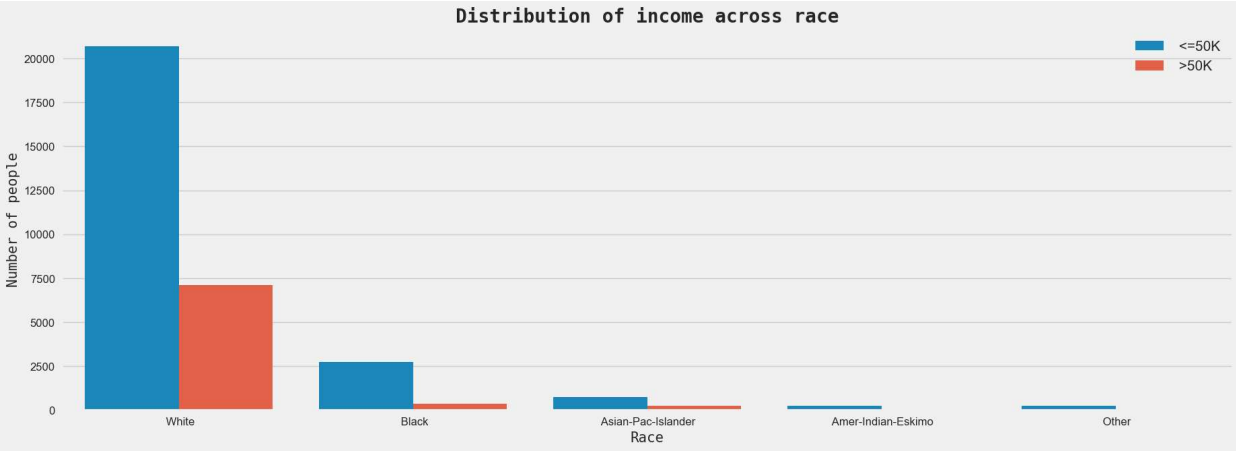
```
            'fontname': 'Monospace', 'fontsize': 15})

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()
```



In [20]:
```
# Creating a countplot of income across Marital Status
plt.style.use('seaborn')
plt.figure(figsize=(20, 7))
sns.countplot(dataset['Marital_status'], hue=dataset['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.show()
```

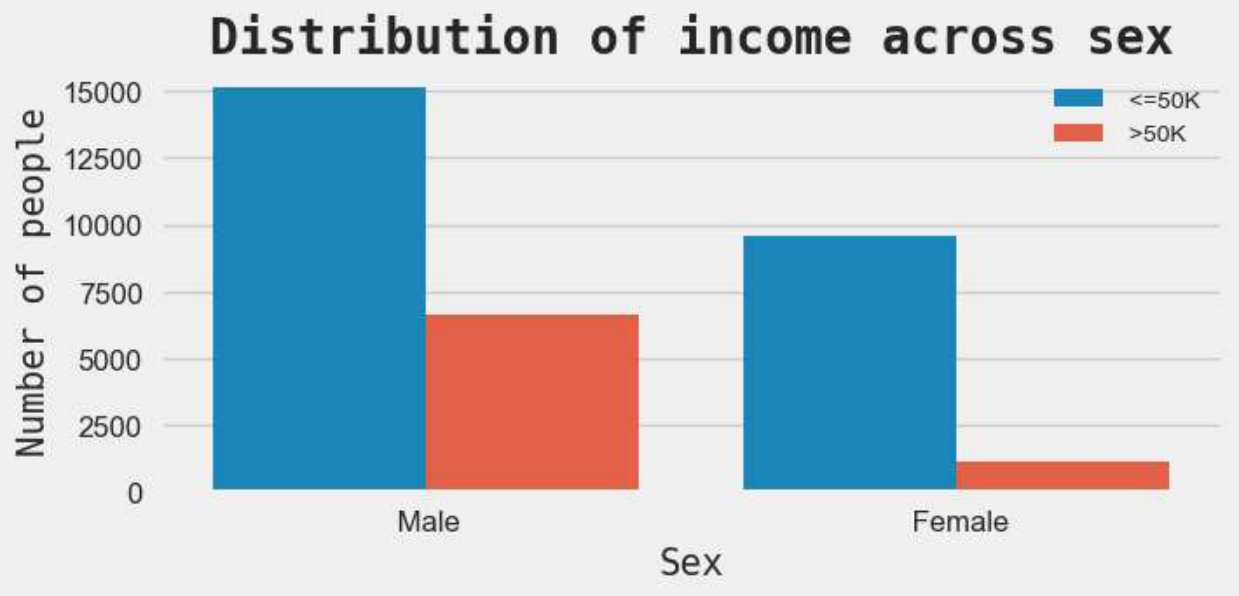


In [21]:
```
# Creating a countplot of income across race
plt.style.use('fivethirtyeight')
plt.figure(figsize=(20, 7))

sns.countplot(dataset['Race'], hue=dataset['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()
```



In [22]:
```
# Creating a countplot of income across sex
plt.style.use('fivethirtyeight')
plt.figure(figsize=(7, 3))
sns.countplot(dataset['Sex'], hue=dataset['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()
```

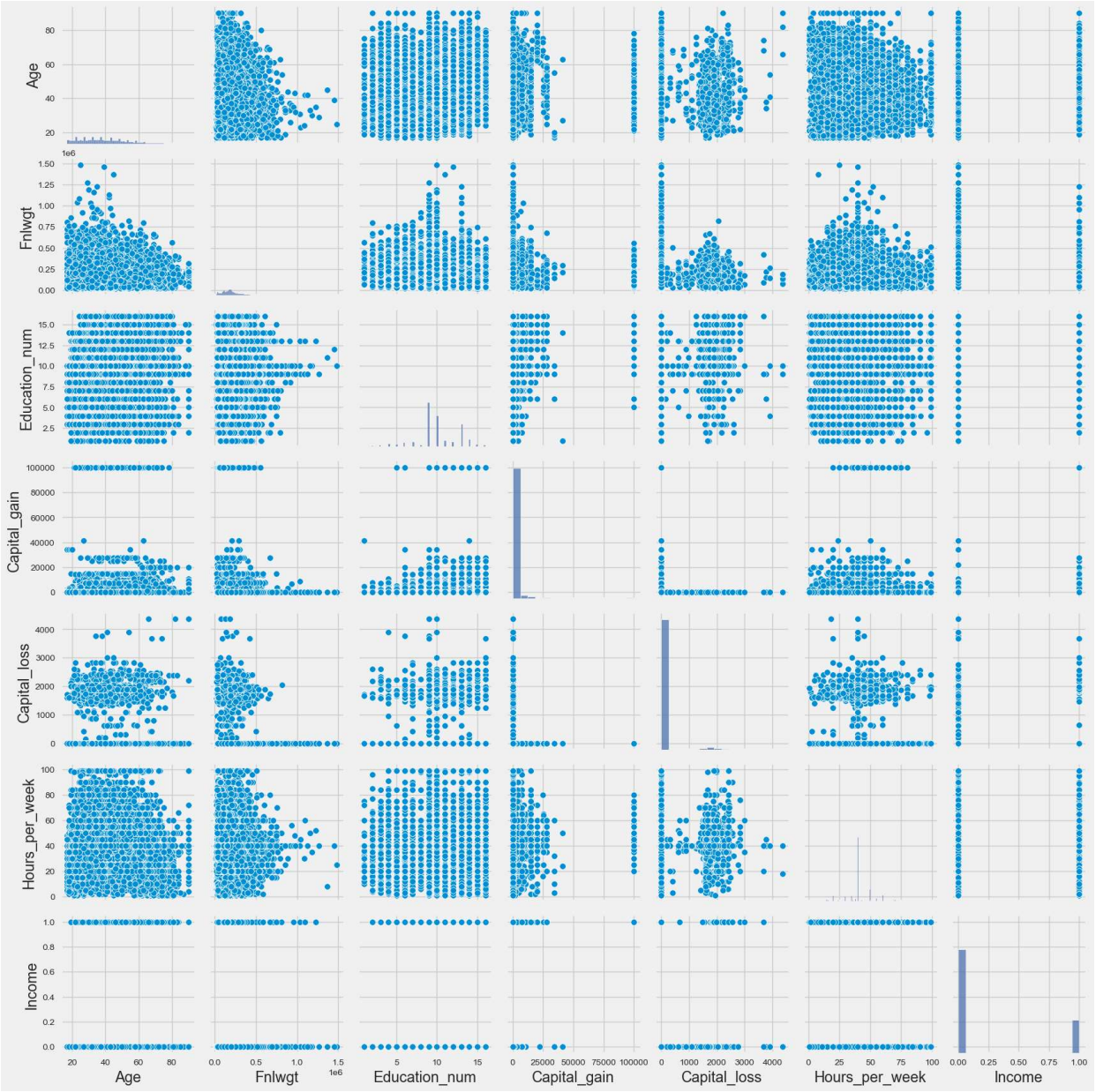


Multivariate Analysis

In [23]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

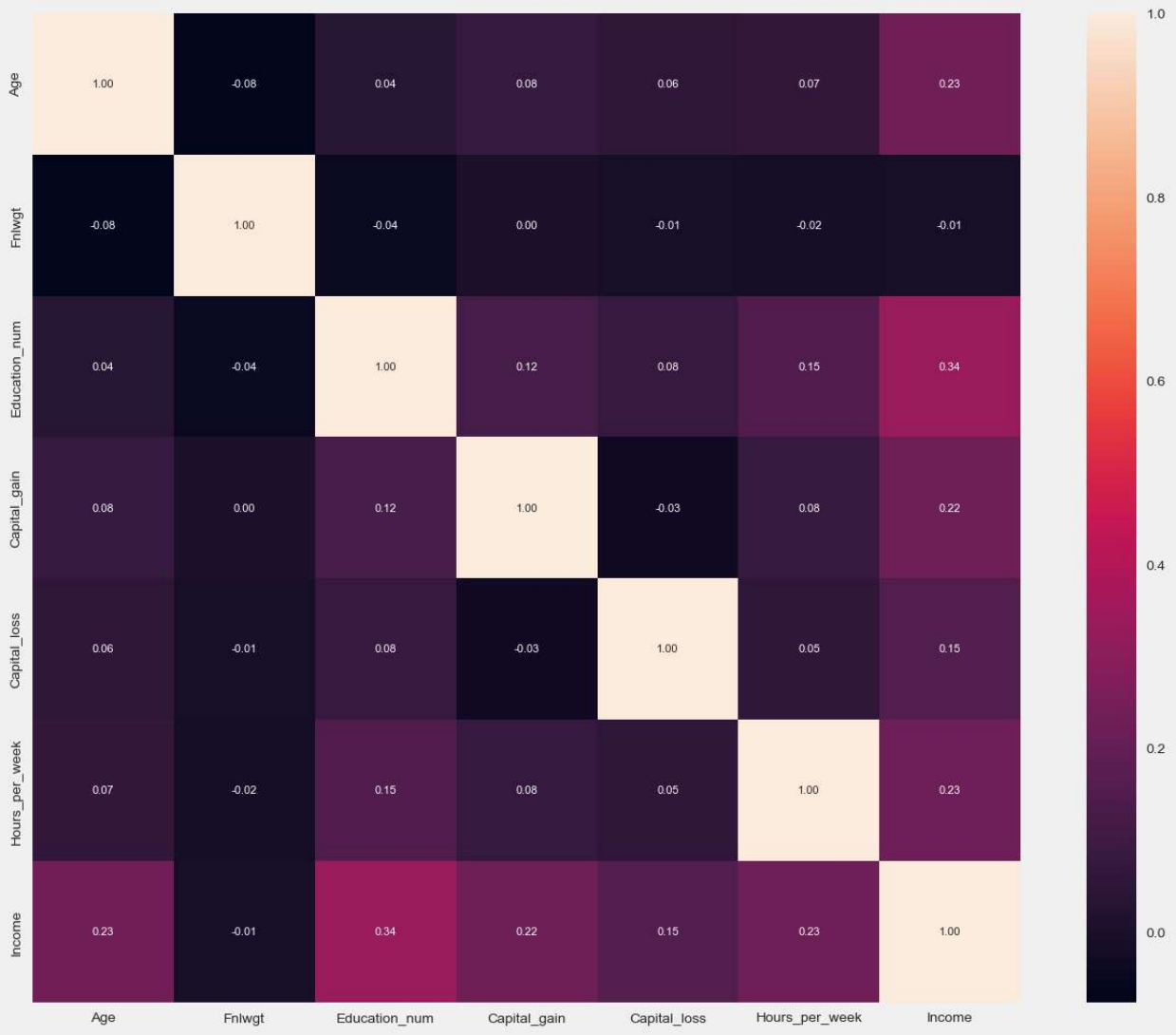
In [24]:
```
dataset['Income'] = le.fit_transform(dataset['Income'])
```

In [25]:
```
# Creating a pairplot of dataset
sns.pairplot(dataset)
plt.savefig('multi1.png')
plt.show()
```



In [26]:
```
plt.figure(figsize=(15,12))
sns.heatmap(dataset.corr(),cbar=True,square=True,fmt='.2f',annot=True,annot_kws={'size
```

Out[26]:
```
<AxesSubplot:>
```



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.

Step 3: Data Preprocessing

In [27]:
```
dataset = dataset.replace('?', np.nan)
```

In [28]:
```
# Checking null values
round((dataset.isnull().sum() / dataset.shape[0]) * 100, 2).astype(str) + ' %'
```

Out[28]:
```
Age               0.0 %
Workclass         0.0 %
Fnlwgt            0.0 %
Education         0.0 %
Education_num     0.0 %
Marital_status    0.0 %
Occupation        0.0 %
Relationship      0.0 %
Race              0.0 %
Sex               0.0 %
Capital_gain      0.0 %
Capital_loss      0.0 %
Hours_per_week    0.0 %
Native_country    0.0 %
Income            0.0 %
dtype: object
```

In [29]:
```
#columns_with_nan = ['Workclass', 'Occupation', 'Native_country']
```

In [30]:
```
#for col in columns_with_nan:
#    dataset[col].fillna(dataset[col].mode()[0], inplace=True)
```

Label Encoding

In [31]:
```
from sklearn.preprocessing import LabelEncoder
```

In [32]:
```
for col in dataset.columns:
    if dataset[col].dtypes == 'object':
        encoder = LabelEncoder()
        dataset[col] = encoder.fit_transform(dataset[col])
```

In [33]:
```
dataset.head()
```

Out[33]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	Rac
0	50	6	83311	9	13	2	4	0	
1	38	4	215646	11	9	0	6	1	
2	53	4	234721	1	7	2	6	0	
3	28	4	338409	9	13	2	10	5	
4	37	4	284582	12	14	2	4	5	

Feature Selection

In [34]:
```
X = dataset.drop('Income', axis=1)
Y = dataset['Income']
```

In [35]:
```
from sklearn.ensemble import ExtraTreesClassifier
selector = ExtraTreesClassifier(random_state=42)
```

In [36]:
```
selector.fit(X, Y)
```

Out[36]:
```
ExtraTreesClassifier
ExtraTreesClassifier(random_state=42)
```

In [37]:
```
feature_imp = selector.feature_importances_
```

In [38]:
```
for index, val in enumerate(feature_imp):
    print(index, round((val * 100), 2))
```

```
0 15.36
1 4.45
2 16.52
3 3.56
4 8.98
5 7.33
6 7.48
7 9.46
8 1.45
9 2.7
10 8.84
11 2.77
12 9.36
13 1.74
```

In [39]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32560 entries, 0 to 32559
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             32560 non-null  int64
 1   Workclass       32560 non-null  int32
 2   Fnlwgt          32560 non-null  int64
 3   Education       32560 non-null  int32
 4   Education_num   32560 non-null  int64
 5   Marital_status  32560 non-null  int32
 6   Occupation      32560 non-null  int32
 7   Relationship    32560 non-null  int32
 8   Race            32560 non-null  int32
 9   Sex             32560 non-null  int32
 10  Capital_gain    32560 non-null  int64
 11  Capital_loss    32560 non-null  int64
 12  Hours_per_week  32560 non-null  int64
 13  Native_country  32560 non-null  int32
dtypes: int32(8), int64(6)
memory usage: 2.5 MB
```

In [40]:
```
X = X.drop(['Workclass', 'Education', 'Race', 'Sex',
            'Capital_loss', 'Native_country'], axis=1)
```

Feature Scaling

In [41]:
```
from sklearn.preprocessing import StandardScaler
```

In [42]:
```
for col in X.columns:
    scaler = StandardScaler()
```

```
    X[col] = scaler.fit_transform(X[col].values.reshape(-1, 1))
```

Fixing imbalanced dataset using Oversampling

In [43]:
```
round(Y.value_counts(normalize=True) * 100, 2).astype('str') + ' %'
```

Out[43]:
```
0    75.92 %
1    24.08 %
Name: Income, dtype: object
```

Creating a train test split

In [44]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state
```

In [45]:
```
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("Y_train shape:", Y_train.shape)
print("Y_test shape:", Y_test.shape)
```

```
X_train shape: (26048, 8)
X_test shape: (6512, 8)
Y_train shape: (26048,)
Y_test shape: (6512,)
```

Step 4: Data Modelling

Logistic Regression

In [46]:
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(random_state=42)
```

In [47]:
```
log_reg.fit(X_train,Y_train)
```

Out[47]:
```
LogisticRegression(random_state=42)
```

In [48]:
```
Y_pred_log_reg = log_reg.predict(X_test)
Y_pred_log_reg
```

Out[48]:
```
array([0, 0, 0, ..., 0, 0, 1])
```

KNN Classifier

In [49]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
```

In [50]:
```
knn.fit(X_train, Y_train)
```

Out[50]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [51]:
```
Y_pred_knn = knn.predict(X_test)
Y_pred_knn
```

Out[51]:
```
array([0, 0, 1, ..., 0, 0, 0])
```

Support Vector Classifier

In [52]:
```
from sklearn.svm import SVC
svc = SVC(random_state=42)
```

In [53]:
```
svc.fit(X_train, Y_train)
```

Out[53]:
```
SVC
SVC(random_state=42)
```

In [54]:
```
Y_pred_svc = svc.predict(X_test)
Y_pred_svc
```

Out[54]:
```
array([0, 0, 1, ..., 0, 0, 1])
```

Naive Bayes Classifier

In [55]:
```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
```

In [56]:
```
nb.fit(X_train, Y_train)
```

Out[56]:
```
GaussianNB
GaussianNB()
```

In [57]:
```
Y_pred_nb = nb.predict(X_test)
Y_pred_nb
```

Out[57]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

Decision Tree Classifier

In [58]:
```
from sklearn.tree import DecisionTreeClassifier
dec_tree = DecisionTreeClassifier(random_state=42)
```

In [59]:
```
dec_tree.fit(X_train, Y_train)
```

Out[59]:
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [60]:
```
Y_pred_dec_tree = dec_tree.predict(X_test)
Y_pred_dec_tree
```

Out[60]:
```
array([0, 1, 0, ..., 0, 0, 0])
```

Random Forest Classifier

In [61]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42)
```

In [62]:
```
rfc.fit(X_train, Y_train)
```

Out[62]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [63]:
```
Y_pred_ran_for = rfc.predict(X_test)
Y_pred_ran_for
```

Out[63]:
```
array([0, 0, 1, ..., 0, 0, 1])
```

Step 5: Model Evaluation

In [64]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

In [65]:
```
print('Logistic Regression:')
print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_log_reg) * 100, 2))
print('F1 score:', round(f1_score(Y_test, Y_pred_log_reg) * 100, 2))
```

```
Logistic Regression:
Accuracy score: 81.65
F1 score: 54.16
```

In [66]:
```
print('KNN Classifier:')
print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_knn) * 100, 2))
print('F1 score:', round(f1_score(Y_test, Y_pred_knn) * 100, 2))
```

```
KNN Classifier:
Accuracy score: 83.22
F1 score: 63.7
```

In [67]:
```
print('Support Vector Classifier:')
print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_svc) * 100, 2))
print('F1 score:', round(f1_score(Y_test, Y_pred_svc) * 100, 2))
```

```
Support Vector Classifier:
Accuracy score: 84.49
F1 score: 62.73
```

In [68]:
```
print('Naive Bayes Classifier:')
print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_nb) * 100, 2))
print('F1 score:', round(f1_score(Y_test, Y_pred_nb) * 100, 2))
```

```
Naive Bayes Classifier:
Accuracy score: 79.51
F1 score: 36.05
```

In [69]:
```
print('Decision Tree Classifier:')
print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_dec_tree) * 100, 2))
print('F1 score:', round(f1_score(Y_test, Y_pred_dec_tree) * 100, 2))
```

```
Decision Tree Classifier:
Accuracy score: 79.47
F1 score: 59.05
```

In [70]:
```
print('Random Forest Classifier:')
print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_ran_for) * 100, 2))
print('F1 score:', round(f1_score(Y_test, Y_pred_ran_for) * 100, 2))
```

```
Random Forest Classifier:
Accuracy score: 84.31
F1 score: 65.43
```

Step 6: Hyperparameter Tuning

In [71]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [72]:
```
n_estimators = [int(x) for x in np.linspace(start=40, stop=150, num=15)]
max_depth = [int(x) for x in np.linspace(40, 150, num=15)]
```

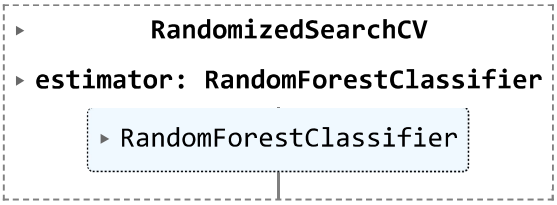
In [73]:
```
param_dist = {
    'n_estimators': n_estimators,
    'max_depth': max_depth,
}
```

In [74]:
```
rf_tuned = RandomForestClassifier(random_state=42)
```

In [75]:
```
rf_cv = RandomizedSearchCV(
    estimator=rf_tuned, param_distributions=param_dist, cv=5, random_state=42)
```

In [76]:
```
rf_cv.fit(X_train, Y_train)
```

Out[76]:



In [77]:
```
rf_cv.best_score_
```

Out[77]:
```
0.8470512390153024
```

In [78]:
```
rf_cv.best_params_
```

Out[78]:
```
{'n_estimators': 118, 'max_depth': 126}
```

In [79]:
```
rf_best = RandomForestClassifier(
    max_depth=102, n_estimators=40, random_state=42)
```

In [80]:
```
rf_best.fit(X_train, Y_train)
```

Out[80]:
```
RandomForestClassifier(max_depth=102, n_estimators=40, random_state=42)
```

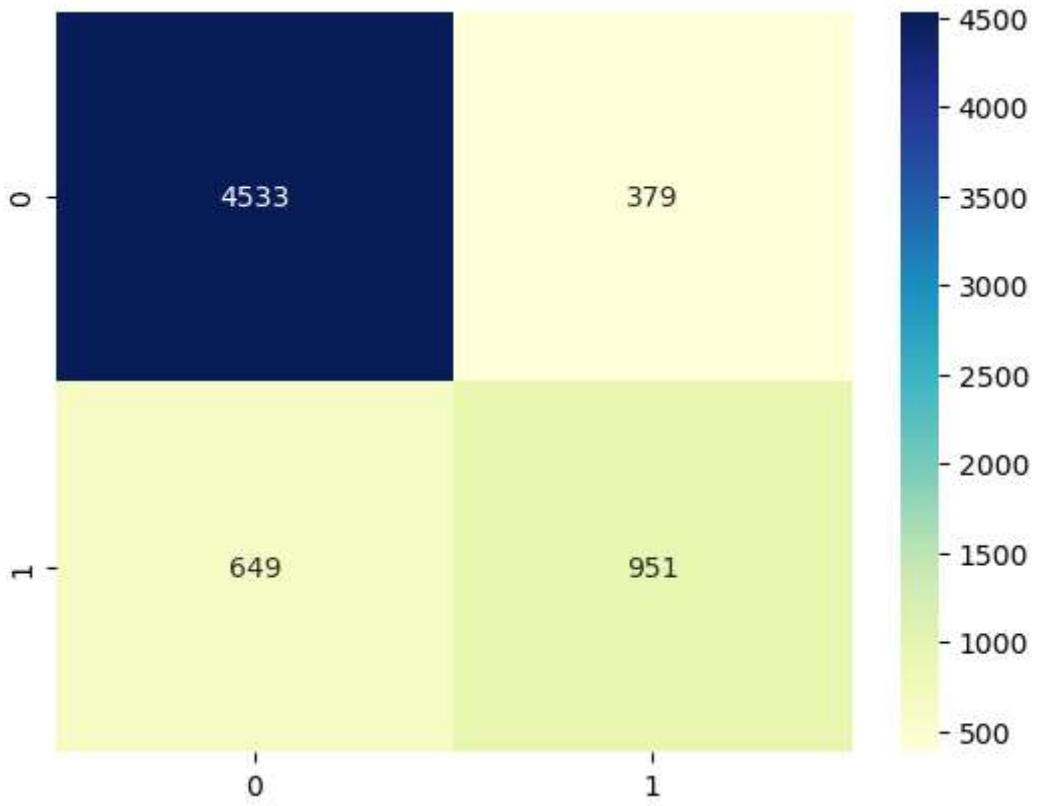
In [81]:
```
Y_pred_rf_best = rf_best.predict(X_test)
```

In [82]:
```
print('Random Forest Classifier:')
print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_rf_best) * 100, 2))
print('F1 score:', round(f1_score(Y_test, Y_pred_rf_best) * 100, 2))
```

```
Random Forest Classifier:
Accuracy score: 84.21
F1 score: 64.91
```

In [83]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred_rf_best)
```

In [84]:
```
plt.style.use('default')
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu')
plt.savefig('heatmap.png')
plt.show()
```



In [85]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_pred_rf_best))
```

```
              precision    recall  f1-score   support

           0       0.87      0.92      0.90      4912
           1       0.72      0.59      0.65      1600

    accuracy                           0.84      6512
   macro avg       0.79      0.76      0.77      6512
weighted avg       0.84      0.84      0.84      6512
```

Conclusion:

- In this project, we build various models like logistic regression, knn classifier, support vector classifier, decision tree classifier, random forest classifier.
- A hyperparameter tuned random forest classifier gives the highest accuracy score of 84.21 and f1 score of 64.91

In [ ]: