Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Read Dataset

In [2]:
```
df = pd.read_csv(r'C:\Users\DELL\Downloads\adult.csv', encoding='latin-1')
```

In [3]:
```
df
```

Out[3]:

| | age | workclass | fnlwgt | education | education.num | marital.status | occupation |
|---|---|---|---|---|---|---|---|
| **0** | 90 | ? | 77053 | HS-grad | 9 | Widowed | ? |
| **1** | 82 | Private | 132870 | HS-grad | 9 | Widowed | Exec-managerial |
| **2** | 66 | ? | 186061 | Some-college | 10 | Widowed | ? |
| **3** | 54 | Private | 140359 | 7th-8th | 4 | Divorced | Machine-op-inspct |
| **4** | 41 | Private | 264663 | Some-college | 10 | Separated | Prof-specialty |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **32556** | 22 | Private | 310152 | Some-college | 10 | Never-married | Protective-serv |
| **32557** | 27 | Private | 257302 | Assoc-acdm | 12 | Married-civ-spouse | Tech-support |
| **32558** | 40 | Private | 154374 | HS-grad | 9 | Married-civ-spouse | Machine-op-inspct |
| **32559** | 58 | Private | 151910 | HS-grad | 9 | Widowed | Adm-clerical |
| **32560** | 22 | Private | 201490 | HS-grad | 9 | Never-married | Adm-clerical |

32561 rows × 15 columns

# Exploratory Data Analysis

Check shape of dataset

In [4]:
```
df.shape
```

Out[4]: (32561, 15)

Preview dataset

In [5]:
```
df.head()
```

Out[5]:

| | age | workclass | fnlwgt | education | education.num | marital.status | occupation | relati |
|---|---|---|---|---|---|---|---|---|
| **0** | 90 | ? | 77053 | HS-grad | 9 | Widowed | ? | |
| **1** | 82 | Private | 132870 | HS-grad | 9 | Widowed | Exec-managerial | |
| **2** | 66 | ? | 186061 | Some-college | 10 | Widowed | ? | Unı |
| **3** | 54 | Private | 140359 | 7th-8th | 4 | Divorced | Machine-op-inspct | Unı |
| **4** | 41 | Private | 264663 | Some-college | 10 | Separated | Prof-specialty | Ow |

View summary of dataframe

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64 
 5   marital.status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64 
 11  capital.loss    32561 non-null  int64 
 12  hours.per.week  32561 non-null  int64 
 13  native.country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [8]:
```
df[df == '?'] = np.nan
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       30725 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64 
 5   marital.status  32561 non-null  object
 6   occupation      30718 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64 
 11  capital.loss    32561 non-null  int64 
 12  hours.per.week  32561 non-null  int64 
 13  native.country  31978 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Impute missing values with mode

In [10]:
```python
for col in ['workclass', 'occupation', 'native.country']:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

Check again for missing values

In [12]:
```python
df.isnull().sum()
```

Out[12]:
```
age               0
workclass         0
fnlwgt            0
education         0
education.num     0
marital.status    0
occupation        0
relationship      0
race              0
sex               0
capital.gain      0
capital.loss      0
hours.per.week    0
native.country    0
income            0
dtype: int64
```

Setting feature vector and target variable

In [14]:
```python
X = df.drop(['income'], axis=1)

y = df['income']
```

In [15]:
```python
X.head()
```

Out[15]:

| | age | workclass | fnlwgt | education | education.num | marital.status | occupation | relati |
|---|---|---|---|---|---|---|---|---|
| 0 | 90 | Private | 77053 | HS-grad | 9 | Widowed | Prof-specialty | |
| 1 | 82 | Private | 132870 | HS-grad | 9 | Widowed | Exec-managerial | |
| 2 | 66 | Private | 186061 | Some-college | 10 | Widowed | Prof-specialty | Unr |
| 3 | 54 | Private | 140359 | 7th-8th | 4 | Divorced | Machine-op-inspct | Unr |
| 4 | 41 | Private | 264663 | Some-college | 10 | Separated | Prof-specialty | Ow |

Split data into separate training and test set

In [16]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, randc
```

# Feature Engineering

Encode categorical variables

In [17]:

```
from sklearn import preprocessing

categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relati
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])
```

Feature Scaling

In [18]:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns = X.columns)
```

In [19]:

```
X_train.head()
```

Out[19]:

| | age | workclass | fnlwgt | education | education.num | marital.status | occupatio |
|---|---|---|---|---|---|---|---|
| **0** | 0.101484 | 2.600478 | -1.494279 | -0.332263 | 1.133894 | -0.402341 | -0.78223 |
| **1** | 0.028248 | -1.884720 | 0.438778 | 0.184396 | -0.423425 | -0.402341 | -0.02669 |
| **2** | 0.247956 | -0.090641 | 0.045292 | 1.217715 | -0.034095 | 0.926666 | -0.78223 |
| **3** | -0.850587 | -1.884720 | 0.793152 | 0.184396 | -0.423425 | 0.926666 | -0.53038 |
| **4** | -0.044989 | -2.781760 | -0.853275 | 0.442726 | 1.523223 | -0.402341 | -0.78223 |

Logistic Regression model with all features

In [20]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

print('Logistic Regression accuracy score with all the features: {0:0.4f}'. form
```

```
Logistic Regression accuracy score with all the features: 0.8218
```

Logistic Regression with PCA

In [21]:
```
from sklearn.decomposition import PCA
pca = PCA()
X_train = pca.fit_transform(X_train)
pca.explained_variance_ratio_
```

Out[21]:
```
array([0.14757168, 0.10182915, 0.08147199, 0.07880174, 0.07463545,
       0.07274281, 0.07009602, 0.06750902, 0.0647268 , 0.06131155,
       0.06084207, 0.04839584, 0.04265038, 0.02741548])
```

Logistic Regression with first 13 features

In [22]:
```
X = df.drop(['income','native.country'], axis=1)
y = df['income']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, randc


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relati
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])


X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns = X.columns)

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
y_pred = logreg.predict(X_test)

print('Logistic Regression accuracy score with the first 13 features: {0:0.4f}'.
```

```
Logistic Regression accuracy score with the first 13 features: 0.8213
```

Logistic Regression with first 12 features

In [23]:
```
X = df.drop(['income','native.country', 'hours.per.week'], axis=1)
y = df['income']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, rando


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relati
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])


X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns = X.columns)

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

print('Logistic Regression accuracy score with the first 12 features: {0:0.4f}'.
```

```
Logistic Regression accuracy score with the first 12 features: 0.8227
```

Logistic Regression with first 11 features

In [24]:
```
X = df.drop(['income','native.country', 'hours.per.week', 'capital.loss'], axis=
y = df['income']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, rando


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relati
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])


X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns = X.columns)

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

print('Logistic Regression accuracy score with the first 11 features: {0:0.4f}'.
```

```
Logistic Regression accuracy score with the first 11 features: 0.8186
```

Select right number of dimensions

In [25]:
```
X = df.drop(['income'], axis=1)
y = df['income']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, randc


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relati
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])


X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)


pca= PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
dim = np.argmax(cumsum >= 0.90) + 1
print('The number of dimensions required to preserve 90% of variance is',dim)
```

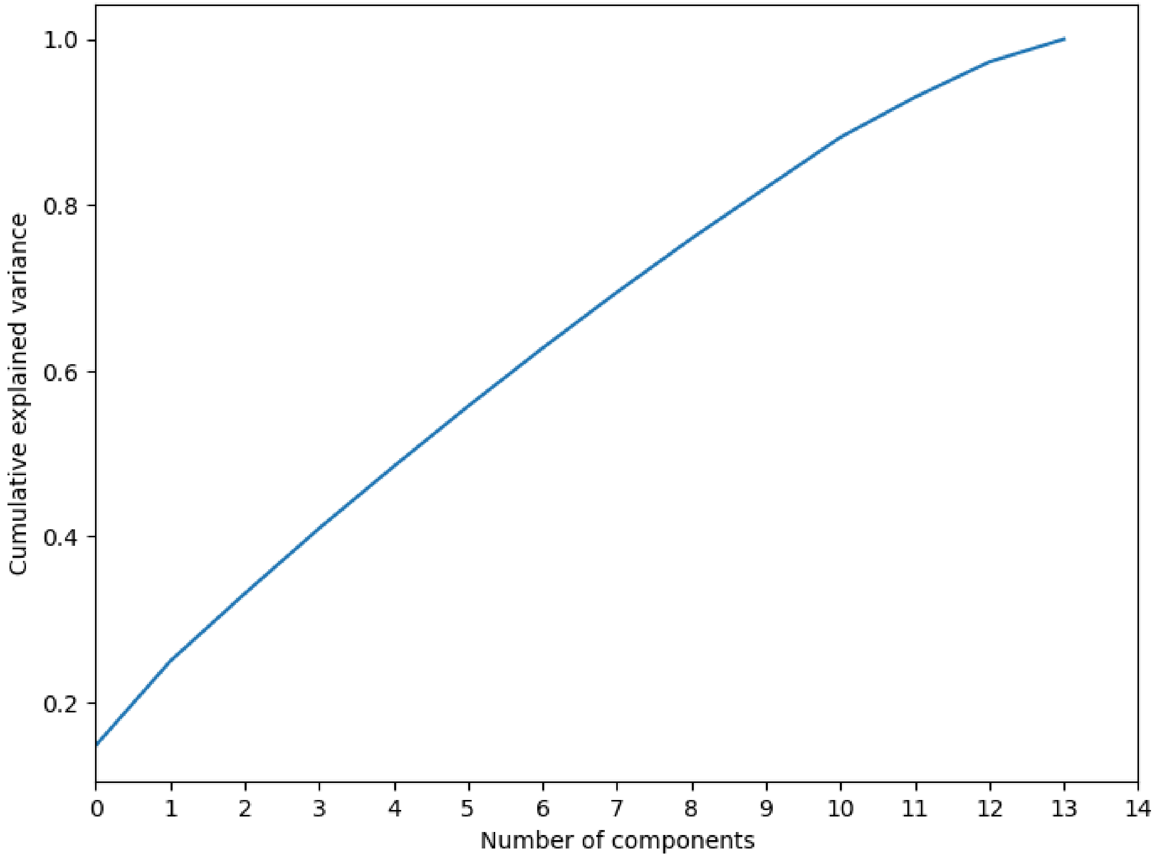
```
The number of dimensions required to preserve 90% of variance is 12
```

Plot explained variance ratio with number of dimensions

In [28]:
```
plt.figure(figsize=(8,6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,14)
plt.xticks(range(0, 15, 1))
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

Out[28]:
```
Text(0, 0.5, 'Cumulative explained variance')
```

In [29]:
```
plt.show()
```


1.0
0.8
0.6
0.4
0.2
Cumulative explained variance
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Number of components

```
In [ ]:
```