Import Python 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')
```

In [5]:
```
import os
print(os.getcwd())  # To check the current working directory
```

```
C:\Users\lenovo
```

In [7]:
```
data = pd.read_csv(r"C:\Users\lenovo\Desktop\spyder\adult.csv")
```

In [8]:
```
data
```

Out[8]:

	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

In [10]:
```
data.shape
```

Out[10]: (32561, 15)

In [11]:
```
data.head()
```

Out[11]:

	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	?	Unr
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unr
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Ow

In [12]:
```
data.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
```

Encode `?` as `NaNs`

In [13]:
```
data[data == '?'] = np.nan
```

In [15]:
```
data.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 [17]:
```
for col in['workclass', 'occupation', 'native.country']:
    data[col].fillna(data[col].mode()[0], inplace = True)
```

In [18]:
```
# 'workclass', 'occupation', 'native.country', These three are categorical data.
```

In [19]:
```
# We used mode bcze, only mode strategy is used for categorical data.
```

Check again for missing values

In [21]:
```
data.isnull().sum()
```

Out[21]:
```
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
```

In [22]:
```
# Now there are no missing values...
```

Setting feature vector and target variable

In [23]:
```
x = data.drop(['income'], axis = 1)
y = data['income']
```

In [24]:
```
x.head()
```

Out[24]:

	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

In [25]:
```
y.head()
```

Out[25]:
```
0    <=50K
1    <=50K
2    <=50K
3    <=50K
4    <=50K
Name: income, dtype: object
```

In [26]:
```
# We droped 'income variable' from 'x', and added only 'income variable' in y.
```

Split data into seperate training and test set

In [28]:
```
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 [30]:
```
from sklearn import preprocessing

categorical = ['workclass',     'education', 'marital.status', 'occupation', 're
```

```
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 [32]:
```
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 [33]:
```
x_train.head()
```

Out[33]:

	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 [35]:
```
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 [36]:
```
# importing PCA from scikit-learn(sklearn) library
```

In [38]:
```
from sklearn.decomposition import PCA
pca = PCA()
x_train = pca.fit_transform(x_train)
pca.explained_variance_ratio_
```

Out[38]:
```
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])
```

Comment

- We can see that approximately 97.25% of variance is explained by the first 13 variables.
- Only 2.75% of variance is explained by the last variable. So, we can assume that it carries little information.
- So, I will drop it, train the model again and calculate the accuracy.

In [39]:
```
# 14%, The vaue is very less. So let's re-train the model again.
```

Logistic Regression With First 12 Features.

In [41]:
```
x = data.drop(['income','native.country', 'hours.per.week'], axis=1)
y = data['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 12 features: {0:0.4f}'.
```

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

Comment

- Now, it can be seen that the accuracy has been increased to 0.8227, if the model is trained with 12 features.
- Lastly, I will take the last three features combined. Approximately 11.83% of variance is explained by them.

- I will repeat the process, drop these features, train the model again and calculate the accuracy.

Logistic Regression With First 11 Features

In [47]:
```
x = data.drop(['income','native.country', 'hours.per.week', 'capital.loss'], axi
y = data['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 11 features: {0:0.4f}'.
```

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

Comment

- We can see that accuracy has significantly decreased to 0.8186 if I drop the last three features.
- Our aim is to maximize the accuracy. We get maximum accuracy with the first 12 features and the accuracy is 0.8227.

Select right number of dimensions

- The above process works well if the number of dimensions are small.
- But, it is quite cumbersome if we have large number of dimensions.
- In that case, a better approach is to compute the number of dimensions that can explain significantly large portion of the variance.
- The following code computes PCA without reducing dimensionality, then computes the minimum number of dimensions required to preserve 90% of the training set variance.

In [48]:
```
x = data.drop(['income'], axis=1)
y = data['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
```

Comment

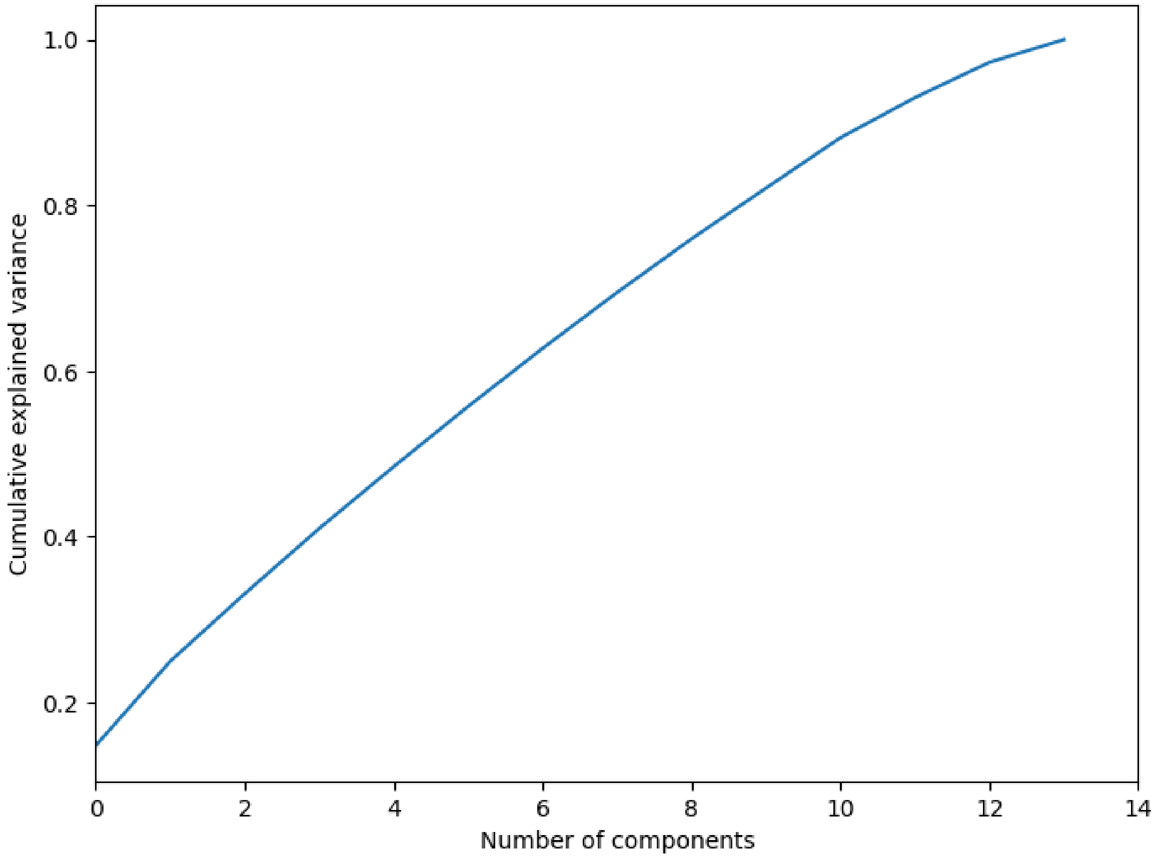
- With the required number of dimensions found, we can then set number of dimensions to `dim` and run PCA again.
- With the number of dimensions set to `dim`, we can then calculate the required accuracy.

Plot explained variance ratio with number of dimensions

- An alternative option is to plot the explained variance as a function of the number of dimensions.
- In the plot, we should look for an elbow where the explained variance stops growing fast.
- This can be thought of as the intrinsic dimensionality of the dataset.
- Now, I will plot cumulative explained variance ratio with number of components to show how variance ratio varies with number of components.

In [63]:
```
plt.figure(figsize=(8, 6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0, 14)
plt.xticks(np.arange(0, 16, 2))  # Adjusting x-ticks
```

```
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
plt.show()
```



In [66]:
```
# plt.xlim(0, 14): This sets the limits of the x-axis to range from 0 to 14.
# It defines the portion of the x-axis that will be visible in the plot.

# plt.xticks(np.arange(0, 16, 2)): This specifies the locations of the ticks on
# np.arange(0, 16, 2) generates an array of values from 0 to 14 (inclusive) with
# 2, ensuring that ticks appear at every integer point within this range.
```

Comment

-

The above plot shows that almost 90% of variance is explained by the first 12 components.

Conclusion

- In this kernelWe'have discussed Principal Component Analysis – the most popular dimensionality reduction technique.

We' have demonstrated PCA implementation with Logistic Regression on the adult dataset.We f I found the maximum accuracy with the first 12 features and it is found to be 0.8227.

- As expected, the number of dimensions required to preserve 90 % of variance is found to be 12.
- Finally, I plot the explained variance ratio with number of dimensions. The graph confirms that approximately 90% of variance is explained by the first 12 components.