Exploratory Data Analysis and Predictive Modeling using Logistic Regression with PCA

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
df = pd.read_csv(r"C:\Users\JANHAVI\OneDrive\FSDS\MACHINE LEARNING\1st, 2nd Logisti
```

In [9]:
```
import os

path = r"C:\Users\JANHAVI\OneDrive\FSDS\MACHINE LEARNING\1st, 2nd Logistic Regrassi

print(os.listdir(path))
```

```
['adult.csv', 'eda-logistic-regression-pca.ipynb']
```

In [13]:
```
import os

print('# File sizes')
path = r"C:\Users\JANHAVI\OneDrive\FSDS\MACHINE LEARNING\1st, 2nd Logistic Regrassi

for f in os.listdir(path):
    size = round(os.path.getsize(os.path.join(path, f)) / 1000000, 2)
    print(f.ljust(30) + str(size) + ' MB')
```

```
# File sizes
adult.csv                     0.0 MB
eda-logistic-regression-pca.ipynb0.06 MB
```

In [14]:
```
%%time

file = (r"C:\Users\JANHAVI\OneDrive\FSDS\MACHINE LEARNING\1st, 2nd Logistic Regrass
df = pd.read_csv(file, encoding='latin-1')
```

```
CPU times: total: 31.2 ms
Wall time: 51.1 ms
```

Exploratory Data Analysis

In [15]:
```
df.shape
```

Out[15]:
```
(32561, 15)
```

In [16]:
```
df.head()
```

Out[16]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	ra
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	Wh
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	Wh
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Bl
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	Wh
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	Wh

In [17]:
```
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 [18]:
```
df[df == '?'] = np.nan
```

In [19]:
```
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 Value

In [20]:
```
for col in ['workclass', 'occupation', 'native.country']:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

In [21]:
```
df.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]:
```
X = df.drop(['income'], axis=1)
y = df['income']
```

In [23]:
```
X.head()
```

Out[23]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	ra
0	90	Private	77053	HS-grad	9	Widowed	Prof-specialty	Not-in-family	Wh
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	Wh
2	66	Private	186061	Some-college	10	Widowed	Prof-specialty	Unmarried	Bl
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	Wh
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	Wh

Split data into separate training and test set

In [24]:

```
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, random_s
```

Feature Engineering

Encode Categorical Variables

In [25]:

```
from sklearn import preprocessing

categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relations
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 [26]:

```
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 [27]:

```
X_train.head()
```

Out[27]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relations
0	0.101484	2.600478	-1.494279	-0.332263	1.133894	-0.402341	-0.782234	2.214
1	0.028248	-1.884720	0.438778	0.184396	-0.423425	-0.402341	-0.026696	-0.899
2	0.247956	-0.090641	0.045292	1.217715	-0.034095	0.926666	-0.782234	-0.276
3	-0.850587	-1.884720	0.793152	0.184396	-0.423425	0.926666	-0.530388	0.968
4	-0.044989	-2.781760	-0.853275	0.442726	1.523223	-0.402341	-0.782234	-0.899

Logistic Regression model with all features

In [28]:

```
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}'. format(
```

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

Logistic Regression with PCA

In [30]:

```
from sklearn.decomposition import PCA
pca = PCA()
X_train = pca.fit_transform(X_train)
pca.explained_variance_ratio_
```

Out[30]:

```
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 [31]:

```
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, random_s


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relations
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}'. fo
```

Logistic Regression accuracy score with the first 13 features: 0.8213

Logistic Regression with first 12 features

In [32]:

```
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, random_s


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relations
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}'. fo
```

Logistic Regression accuracy score with the first 12 features: 0.8227

Logistic Regression with first 11 features

In [33]:

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


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


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relations
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}'. fo
```

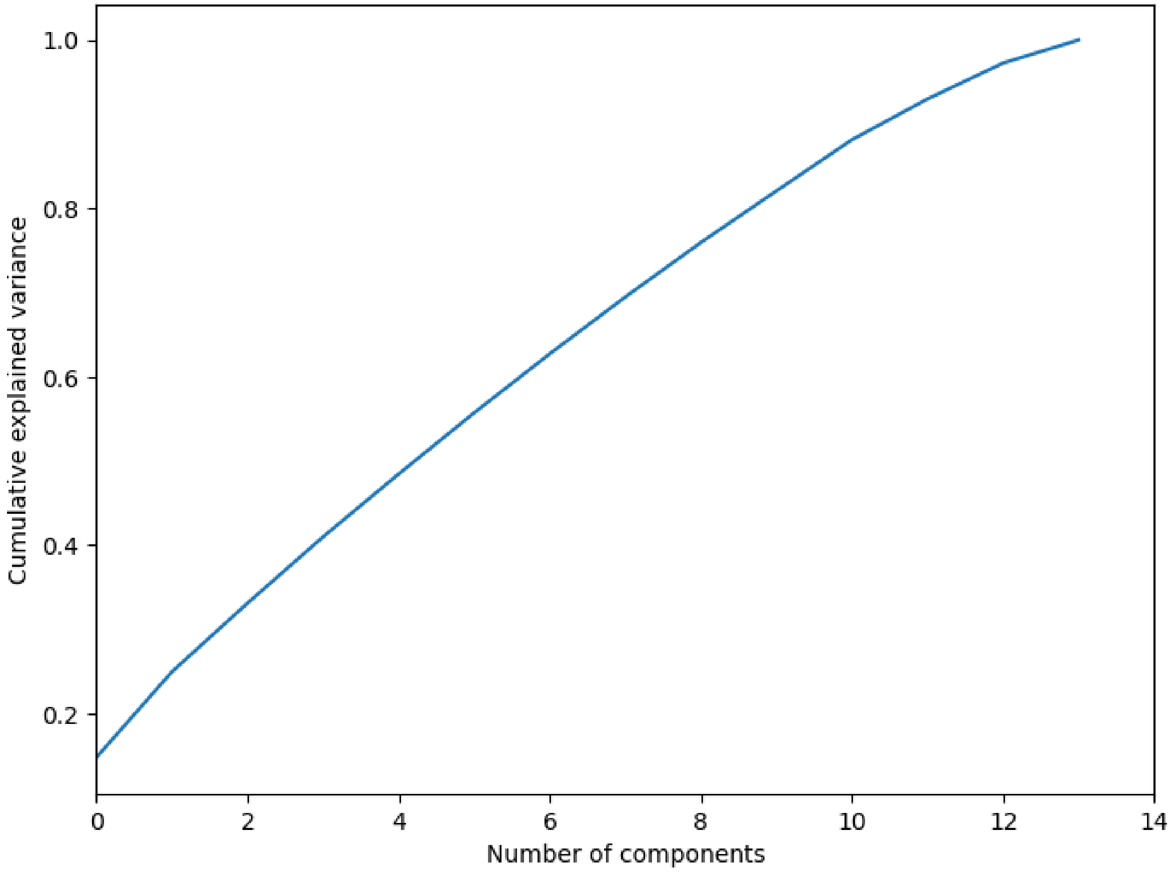
Logistic Regression accuracy score with the first 11 features: 0.8186

In [34]:
```
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, random_s


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relations
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 Variance ratio with number of dimensions

In [35]:
```
plt.figure(figsize=(8,6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,14,1)
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
plt.show()
```


1.0
0.8
0.6
0.4
0.2
Cumulative explained variance
0
2
4
6
8
10
12
14
Number of components