EDA + Logistic Regression + PCA

This kernel is all about **Principal Component Analysis** - a **Dimensionality Reduction** technique.

I have used the **adult** data set for this kernel. This dataset is very small for PCA purpose. My main purpose is to demonstrate PCA implementation with this dataset.

Table of Contents

The contents of this kernel is divided into various topics which are as follows:-

- The Curse of Dimensionality
- Strategies to Mitigate the Curse of Dimensionality
- Introduction to Principal Component Analysis
- Import Python libraries
- Import dataset
- Exploratory data analysis
- Split data into training and test set
- Feature engineering
- Feature scaling
- Logistic regression model with all features
- Logistic Regression with PCA
- Select right number of dimensions
- Plot explained variance ratio with number of dimensions
- Conclusion
- References

The Curse of Dimensionality

Generally, real world datasets contain thousands or millions of features to train for. This is very time consuming task as this makes training extremely slow. In such cases, it is very difficult to find a good solution. This problem is often referred to as the curse of dimensionality.

The curse of dimensionality refers to various phenomena that arise when we analyze and organize data in high dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings. The problem is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance.

In real-world problems, it is often possible to reduce the number of dimensions considerably. This process is called **dimensionality reduction**. It refers to the process of reducing the number of dimensions under consideration by obtaining a set of principal variables. It helps to speed up training and is also extremely useful for data visualization.

The most popular dimensionality reduction technique is Principal Component Analysis (PCA), which is discussed below.

Strategies to Mitigate the Curse of Dimensionality:

Dimensionality Reduction: Techniques like Principal Component Analysis (PCA) and t-SNE reduce the number of features while preserving important information.

Feature Selection: Identifying and selecting the most relevant features can improve model performance.

Regularization: Adding penalties to complex models (e.g., Lasso or Ridge regression) helps prevent overfitting in high dimensions.

Introduction to Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique that can be used to reduce a larger set of feature variables into a smaller set that still contains most of the variance in the larger set.

Preserve the variance

PCA, first identifies the hyperplane that lies closest to the data and then it projects the data onto it. Before, we can project the training set onto a lower-dimensional hyperplane, we need to select the right hyperplane. The projection can be done in such a way so as to preserve the maximum variance. This is the idea behind PCA.

Principal Components

PCA identifies the axes that accounts for the maximum amount of cumulative sum of variance in the training set. These are called Principal Components. PCA assumes that the dataset is centered around the origin. Scikit-Learn’s PCA classes take care of centering the data automatically.

Projecting down to d Dimensions

Once, we have identified all the principal components, we can reduce the dimensionality of the dataset down to d dimensions by projecting it onto the hyperplane defined by the first d principal components. This ensures that the projection will preserve as much variance as possible.

Import Python libraries

In [15]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

import os
print(os.listdir('C:/Users/user/Documents'))
```

```
['!qhlogs.doc', '3D Objects - Shortcut.lnk', 'adult.csv', 'car-mpg.csv', 'Custom
Office Templates', 'Data.csv', 'desktop.ini', 'emp_sal.csv', 'FIFA.csv', 'final1.
csv', 'Future prediction1.csv', 'heart.csv', 'House_data.csv', 'Inc_Exp_Data.cs
v', 'Investment.csv', 'iris flower.png', 'Iris.csv', 'logit classification.csv',
'Movie-Rating.csv', 'movie.csv', 'My Music', 'My Pictures', 'My Videos', 'Python
Scripts', 'randomnew_records.csv', 'rating.csv', 'Rawdata.xlsx', 'Salary_Data.cs
v', 'Sample - Superstore_Orders.csv', 'sample1-json.json', 'sample1.xml', 'sample
pdf.pdf', 'ShareX', 'Social_Network_Ads.csv', 'statistics.jpg', 'statistics.PNG',
'table.html', 'tag.csv', 'TASK -- convert raw data - clean data.xlsx', 'Tasks.tx
t', 'titanic dataset.csv', 'toy_dataset.csv']
```

Check file size

In [24]:
```
# File sizes for only .csv files
print('# CSV File sizes')
for f in os.listdir('C:/Users/user/Documents'):
    if f.endswith('.csv'):  # Check if the file has a .csv extension
        file_path = os.path.join('C:/Users/user/Documents', f)  # Join directory
        file_size = round(os.path.getsize(file_path) / 1000000, 2)  # Get file s
        print(f.ljust(30) + str(file_size) + 'MB')
```

```
# CSV File sizes
adult.csv                     4.1MB
car-mpg.csv                   0.02MB
Data.csv                      0.0MB
emp_sal.csv                   0.0MB
FIFA.csv                      9.14MB
final1.csv                    0.0MB
Future prediction1.csv        0.0MB
heart.csv                     0.01MB
House_data.csv                2.36MB
Inc_Exp_Data.csv              0.0MB
Investment.csv                0.0MB
Iris.csv                      0.01MB
logit classification.csv      0.01MB
Movie-Rating.csv              0.02MB
movie.csv                     1.49MB
randomnew_records.csv         0.0MB
rating.csv                    690.35MB
Salary_Data.csv               0.0MB
Sample - Superstore_Orders.csv2.12MB
Social_Network_Ads.csv        0.01MB
tag.csv                       21.73MB
titanic dataset.csv           0.06MB
toy_dataset.csv               5.74MB
```

Import dataset

In [31]:
```
%%time

file = (r'C:\Users\user\Documents\adult.csv')
df = pd.read_csv(file, encoding='latin-1')
```

```
CPU times: total: 156 ms
Wall time: 139 ms
```

Exploratory Data Analysis

Check shape of dataset

In [37]:
```
df.shape
```

Out[37]: (32561, 15)

We can see that there are 32561 instances and 15 attributes in the data set.

Preview dataset

In [41]:
```
df.head()
```

Out[41]:

	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

View summary of dataframe

In [46]:
```
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
```

Summary of the dataset shows that there are no missing values. But the preview shows that the dataset contains values coded as `?`. So, lets encode `?` as NaN values.

Encode `?` as `NaNs`

In [52]:
```
df[df == '?'] = np.nan
```

Again check the summary of dataframe

In [56]:
```
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
```

Now, the summary shows that the variables - `workclass`, `occupation` and `native.country` contain missing values. All of these variables are categorical data type. So, I will impute the missing values with the most frequent value- the mode.

Impute missing values with mode

In [60]:
```
for col in ['workclass', 'occupation', 'native.country']:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

Check again for missing values

In [63]:
```
df.isnull().sum()
```

Out[63]:
```
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
```

Now we can see that there are no missing values in the dataset.

Setting feature vector and target variable

In [69]:
```
x = df.drop(['income'], axis=1)

y = df['income']
```

In [71]:
```
x.head()
```

Out[71]:

	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 [73]:

```
y.head()
```

Out[73]:

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

Split data into separate training and test set

In [96]:

```
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 [100…

```
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])
```

In [102…

```
x_train.head()
```

Out[102...

	age	workclass	fnlwgt	education	education.num	marital.status	occupation
32098	40	6	31627	9	13	2	3
25206	39	1	236391	11	9	2	6
23491	42	3	194710	15	10	4	3
12367	27	1	273929	11	9	4	4
7054	38	0	99527	12	14	2	3

Feature Scaling

In [105...

```python
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 [107...

```python
x_train.head()
```

Out[107...

	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 [113...

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

logireg = LogisticRegression()
logireg.fit(x_train, y_train)
y_pred = logireg.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

Scikit-Learn's PCA class implements PCA algorithm using the code below. Before diving deep, lets see another important concept called explained variance ratio.

Explained Variance Ratio

A very useful piece of information is the **explained variance ratio** of each principal component. It is available via the `explained_variance_ratio_` variable. It indicates the proportion of the dataset's variance that lies along the axis of each principal component.

Now, let's get to the PCA implementation.

In [123…

```
from sklearn.decomposition import PCA
pca = PCA()
x_train = pca.fit_transform(x_train)
v=pca.explained_variance_ratio_
print(v)
print()
print("approximate percentage:",np.sum(v[:13]))
```

```
[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]

approximate percentage: 0.9725845155276274
```

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.

Logistic Regression with first 13 features

In [127…

```
# getting new dependent and independent variable
X = df.drop(['income','native.country'], axis=1)
y = df['income']

# traing the model with new X,Y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, randc

# encoding categorical variables(Feature engineering)
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
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
```

Comment

- We can see that accuracy has been decreased from 0.8218 to 0.8213 after dropping the last feature.
- Now, if I take the last two features combined, then we can see that approximately 7% of variance is explained by them.
- I will drop them, train the model again and calculate the accuracy.

Logistic Regression with first 12 features

In [131…

```
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, 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 [135…

```
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, 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 0.8187 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 [163…

```
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, 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)


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

dim = np.argmax(cumsum >= 0.90) + 1
# This line is determining how many principal components (dimensions) are needed
# cumsum >= 0.90 creates a boolean array where each element is True if the cumul
# np.argmax(cumsum >= 0.90) returns the index of the first True value, i.e., the
# + 1 is added because indices are zero-based, but the number of components is t
```

```
[0.14757168 0.24940083 0.33087282 0.40967457 0.48431002 0.55705283
 0.62714886 0.69465787 0.75938468 0.82069623 0.8815383  0.92993414
 0.97258452 1.        ]
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.

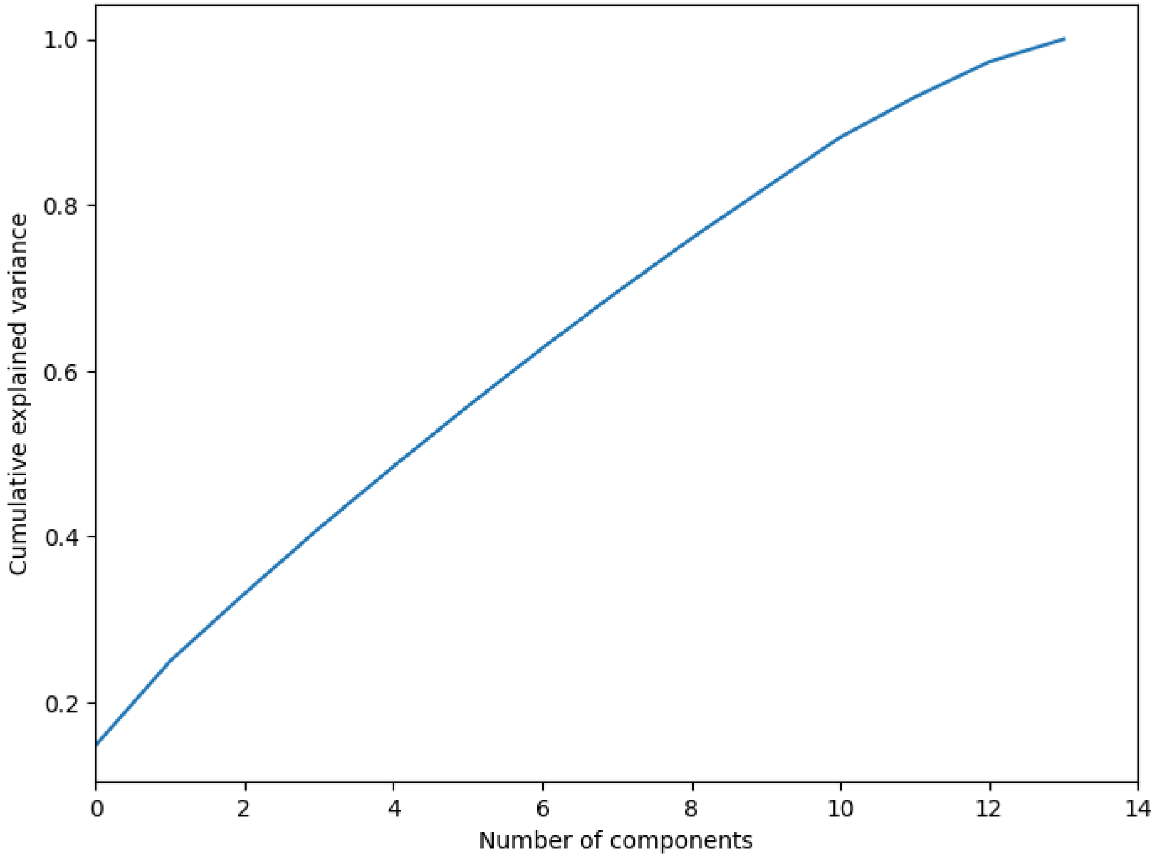
Plot explained variance ratio with number of dimensions

- 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 [151…

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



In [177…

```
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, 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}'.
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[177], line 1
----> 1 X = df.drop(['income','native.country', 'hours.per.week', 'capital.los
s'], axis=1)
      2 y = df['income']
      5 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
3, random_state = 0)

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:5581, in DataFrame.drop(s
elf, labels, axis, index, columns, level, inplace, errors)
   5433 def drop(
   5434     self,
   5435     labels: IndexLabel | None = None,
   (...)
   5442     errors: IgnoreRaise = "raise",
   5443 ) -> DataFrame | None:
   5444     """
   5445     Drop specified labels from rows or columns.
   5446 
   (...)
   5579             weight  1.0     0.8
   5580     """
-> 5581     return super().drop(
   5582         labels=labels,
   5583         axis=axis,
   5584         index=index,
   5585         columns=columns,
   5586         level=level,
   5587         inplace=inplace,
   5588         errors=errors,
   5589     )

File ~\anaconda3\Lib\site-packages\pandas\core\generic.py:4788, in NDFrame.drop(s
elf, labels, axis, index, columns, level, inplace, errors)
   4786 for axis, labels in axes.items():
   4787     if labels is not None:
-> 4788         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4790 if inplace:
   4791     self._update_inplace(obj)

File ~\anaconda3\Lib\site-packages\pandas\core\generic.py:4830, in NDFrame._drop_
axis(self, labels, axis, level, errors, only_slice)
   4828         new_axis = axis.drop(labels, level=level, errors=errors)
   4829     else:
-> 4830         new_axis = axis.drop(labels, errors=errors)
   4831     indexer = axis.get_indexer(new_axis)
   4833 # Case for non-unique axis
   4834 else:

File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:7070, in Index.dro
p(self, labels, errors)
   7068 if mask.any():
   7069     if errors != "ignore":
-> 7070         raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7071     indexer = indexer[~mask]
   7072 return self.delete(indexer)

KeyError: "['income'] not found in axis"
```

Comment

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

Conclusion

- In this kernel, I have discussed Principal Component Analysis – the most popular dimensionality reduction technique.
- I have demonstrated PCA implementation with Logistic Regression on the adult dataset.
- 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.

References

The ideas and concepts in this kernel are taken from the following book.

- Hands on Machine Learning with Scikit-Learn and Tensorflow by Aurelien Geron.

```
In [ ]:
```