EDA + Logistic Regression + PCA

- This project is all about **Principal Component Analysis -** a **Dimensionality Reduction** technique.
- I have discussed **Principal Component Analysis (PCA)**. In particular, I have introduced PCA, explained variance ratio, Logistic Regression with PCA, find right number of dimensions and plotting explained variance ratio with number of dimensions.
- 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.

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.

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.

Importing necessary Libraries

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

%matplotlib inline

# To ignore pvm warnings
import warnings
warnings.filterwarnings('ignore')
```

Load the dataset

In [2]:
```
adult=pd.read_csv(r"D:\Project\adult.csv\adult.csv")
adult
```

Out[2]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	s
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Fem
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Fem
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Fem
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Fem
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Fem
...	...	...	...	...	...	...	...	...	...	
32556	22	Private	310152	Some-college	10	Never-married	Protective-serv	Not-in-family	White	M
32557	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Fem
32558	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	M
32559	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Fem
32560	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	M

32561 rows × 15 columns

Exploratory Data Analysis

Check the shape of dataset

In [3]:

```
adult.shape
```

Out[3]:

```
(32561, 15)
```

There are 32561 instances and 15 attributes in our dataset.

Preview dataset

In [4]:

```
adult.head()
```

Out[4]:

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

Summary of dataset

In [5]:
```
adult.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
```

Findings

Summary of the dataset shows that there are no missing values but the preview shows that the dataset contains values coded as '?' So, I will encode '?' as NAN values

Encode ? as NaN

In [6]:
```
# Replace '?' as nan value
adult[adult=='?']=np.nan
```

There is no null values in our dataset.

Again check the summary of dataset

In [7]:
```
adult.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
```

In [8]:
```
adult.isnull().sum()
```

Out[8]:
```
age                  0
workclass         1836
fnlwgt               0
education            0
education.num        0
marital.status       0
occupation        1843
relationship         0
race                 0
sex                  0
capital.gain         0
capital.loss         0
hours.per.week       0
native.country     583
income               0
dtype: int64
```

Interpretation

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 [9]:
```
for col in ['workclass','occupation','native.country']:
    adult[col].fillna(adult[col].mode()[0],inplace=True)
```

Check again missing values

In [10]:
```
adult.isnull().sum()
```

Out[10]:
```
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.

Preview dataset

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

Out[11]:

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

Split the data into dependent and independent

In [12]:

```
## Setting feature vector and target variable
X=adult.iloc[:,:-1]
y=adult.iloc[:,-1]
```

Split data into training and testing set

In [13]:

```
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_state=0)
```

Feature Engineering

Encode Categorical Variables

In [14]:

```
from sklearn import preprocessing
categorical=['workclass','education','marital.status','occupation','relationship','r
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 [15]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_train=pd.DataFrame(scaler.fit_transform(X_train),columns=X.columns)
X_test=pd.DataFrame(scaler.fit_transform(X_test),columns=X.columns)
```

In [16]:

```
X_train.head()
```

Out[16]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race
0	0.101484	2.600478	-1.494279	-0.332263	1.133894	-0.402341	-0.782234	2.214196	0.39298
1	0.028248	-1.884720	0.438778	0.184396	-0.423425	-0.402341	-0.026696	-0.899410	0.39298
2	0.247956	-0.090641	0.045292	1.217715	-0.034095	0.926666	-0.782234	-0.276689	0.39298
3	-0.850587	-1.884720	0.793152	0.184396	-0.423425	0.926666	-0.530388	0.968753	0.39298
4	-0.044989	-2.781760	-0.853275	0.442726	1.523223	-0.402341	-0.782234	-0.899410	0.39298

Logistic Model with all features

In [17]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
model=LogisticRegression()
model.fit(X_train,y_train)

y_pred=model.predict(X_test)

accuracy=accuracy_score(y_test,y_pred)
accuracy
```

Out[17]:
```
0.8217831917289384
```

Logistic Rgression accuracy score with all the features is 82.17%

Logistic Regression with PCA

Scikit-Learn's PCA class implements PCA algorithm using the code below. Before diving deep, I will explain 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 [18]:
```
from sklearn.decomposition import PCA
pca=PCA()
X_train=pca.fit_transform(X_train)
pca.explained_variance_ratio_  #It indicates the proportion of the dataset’s varianc
```

Out[18]:
```
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])
```

Interpretation

- 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.
- Remove low variance value
- So i will drop it,train the model again and calculate the acuracy.

Logistic Regression with the first 13 features

In [19]:

```
X=adult.iloc[:,:-2]
y=adult.iloc[:,-1]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)

categorical=['workclass','education','marital.status','occupation','relationship','r
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)

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

```

In [20]:

```
y_pred
```

Out[20]:

```
array(['<=50K', '<=50K', '<=50K', ..., '<=50K', '<=50K', '<=50K'],
      dtype=object)
```

In [21]:

```
# To find accuracy of model
accuracy=accuracy_score(y_test,y_pred)
print(f'Logistic Regression accuracy score with the first 13 features:{accuracy:.4f}
```

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

Interpretation

- Accuracy of model for 13 features is 82.13%
- Here we can see that accuracy is decreased,from 82.17% to 82.13%
- If i take the last two features combined ,then we can see that approximately 7% variance is explained by them.
- I will drop them,train the model again and calculate the accuracy.

Logistic Regression with first 12 features.

In [22]:
```
# Split data into dependent and independent variable take X is first 12 columns.
X=adult.iloc[:,:-3]
y=adult.iloc[:,-1]

# Train and test split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)

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

scaler=preprocessing.StandardScaler()

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

model=LogisticRegression()
model.fit(X_train,y_train)

y_pred=model.predict(X_test)

accuracy=accuracy_score(y_test,y_pred)
print(f'Logistic Regression accuracy score with the first 13 features:{accuracy:.4f}
```

```
Logistic Regression accuracy score with the first 13 features:0.8227%
```

Interpretation

- The accuracy of Logistic Regression model for 12 features is 82.27%.
- We can see that accuray is increased from 82.12% to 82.27%
- 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 [23]:

```
X=adult.iloc[:,:-4]
y=adult.iloc[:,-1]

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)

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

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)

model=LogisticRegression()
model.fit(X_train,y_train)

y_pred=model.predict(X_test)

accuracy=accuracy_score(y_test,y_pred)
print(f'Logistic Regression accuracy score with the first 13 features:{accuracy:.4f}
```

```
Logistic Regression accuracy score with the first 13 features:0.8186%
```

Interpretation

- The accuracy of logistic regression model for 11 features is 81.86%.
- we can see that model accuracy is decreased from 82.27% to 81.86%
- Our main aim is maximize accuracy .
- We get maximum accuracy with the first 12 features i.e 82.27%

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 [24]:

```python
# Split data into dependent and independent variables
X=adult.iloc[:,:-1]
y=adult.iloc[:,-1]

# Split data into train test set
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)

categorical=['workclass','education','marital.status','occupation','relationship','r

for feature in categorical:
    le=preprocessing.LabelEncoder()
    X_train[feature]=le.fit_transform(X_train[feature])
    X_test[feature]=le.transform(X_test[feature])

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

pca=PCA()
pca.fit(X_train)

# cumulative explained variance ratio from a PCA (Principal Component Analysis)
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
```

Interpretation

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

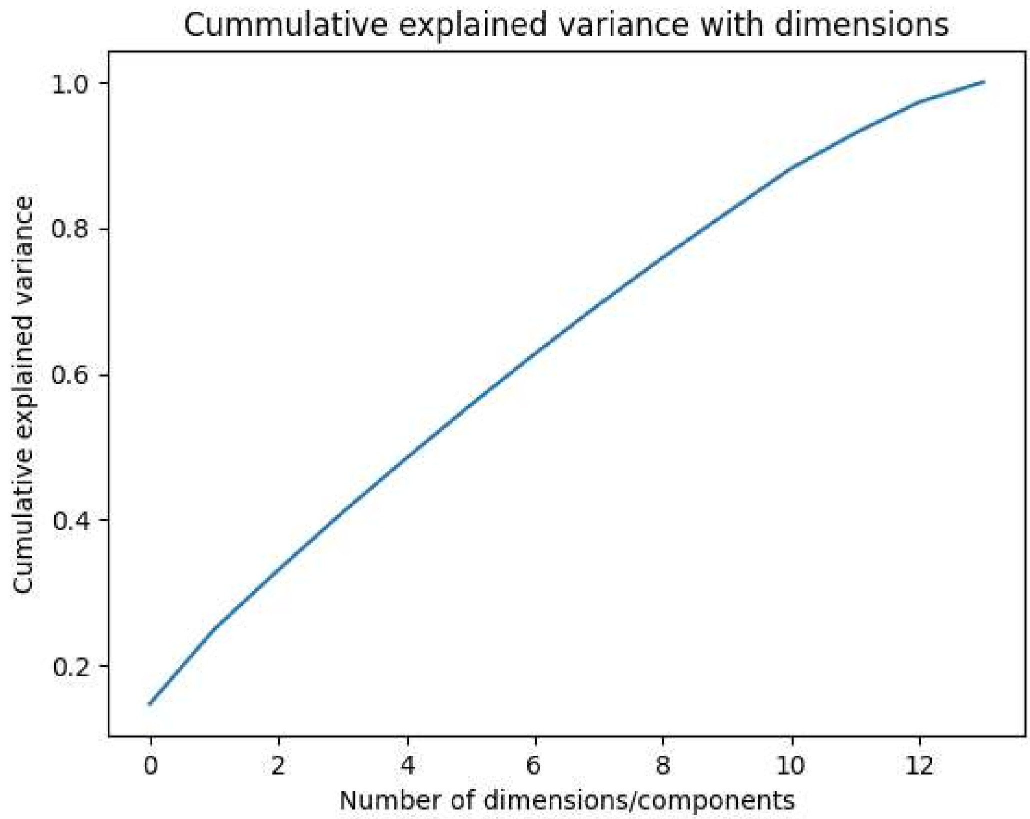
Plot explained variance ratio with number of dimensions

- 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 [25]:

```
plt.plot(np.cumsum(pca.explained_variance_ratio_));
plt.title('Cummulative explained variance with dimensions')
plt.xlabel('Number of dimensions/components')
plt.ylabel('Cumulative explained variance');

```



Interpretation

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

Conclusion

- PCA is the most powerful dimensionality reduction technique
- I have demonstrated PCA implementation with Logistic Regression on the adult dataset.
- I found maximum accuracy with the first 12 features it is 82.27%
- As expected the number of dimensions required to preserve 90% of variance is found to be 12.
- Finally i plot the explained variance ratio graph with the number of dimensions.The graph confirms that approximately 90% variance is explained by the first 12 components.

In [ ]:

```

```