Principal component analysis

```
PCA:PCA finds the hyperplane that best represents the directions of maximum variance in the data, and this hyperplane is defined by the principal components. The hyperplane serves as a lower-dimensional representation of the original feature space, enabling dimensionality reduction and data visualization.
Step 1: Standardization
We start by standardizing the variables to have zero mean and unit variance. This step ensures that variables with different scales are on the same scale.
Step 2: Covariance Matrix
Next, we calculate the covariance matrix based on the standardized data.
```

In [3]:

```python
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:

```python
# ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

In [7]:

```python
# Working with os module - os is a module in Python 3.
# Its main purpose is to interact with the operating system.
# It provides functionalities to manipulate files and folders.

current_directory =os.getcwd()
current_directory
```

Out[7]:

```
'/Users/myyntiimac'
```

In [8]:

```python
df = pd.read_csv("/Users/myyntiimac/Desktop/adult.csv")
df.head()
```

Out[8]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.p
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356	
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356	
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	

In [9]:

```python
df.shape
```

Out[9]:

```
(32561, 15)
```

```
insight:we have 3261 rows and 15 column
```

In [10]:

```python
df.columns
```

Out[10]:

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education.num',
       'marital.status', 'occupation', 'relationship', 'race', 'sex',
       'capital.gain', 'capital.loss', 'hours.per.week', 'native.country',
       'income'],
      dtype='object')
```

```
Insight:income is dependent and others is dependent
```

In [11]:

```
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 [12]:

```
df.isnull().any()
```

Out[12]:

```
age               False
workclass         False
fnlwgt            False
education         False
education.num     False
marital.status    False
occupation        False
relationship      False
race              False
sex               False
capital.gain      False
capital.loss      False
hours.per.week    False
native.country    False
income            False
dtype: bool
```

In [13]:

```
#our data have no nullvalue , lets check any extra charecter contain in our dataset
df[df == '?']
```

Out[13]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.p
0	NaN	?	NaN	NaN	NaN	NaN	?	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	?	NaN	NaN	NaN	NaN	?	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
32557	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
32558	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
32559	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
32560	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

32561 rows × 15 columns

In [14]:

```
#Lets tranfer this ? mark with  nan value
df[df == '?']=np.NaN
```

In [15]:

```
#check
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
```

In [16]:

```
df.isnull().any()
```

Out[16]:

```
age                  False
workclass             True
fnlwgt               False
education            False
education.num        False
marital.status       False
occupation            True
relationship         False
race                 False
sex                  False
capital.gain         False
capital.loss         False
hours.per.week       False
native.country        True
income               False
dtype: bool
```

```
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.
```

In [18]:

```
#The attribute containg null value are catagorical so mode strategy is good fit
# Define the columns you want to fill missing values for
columns_to_fill = ['workclass', 'occupation', 'native.country']

# Replace missing values in each column with the mode
df[columns_to_fill] = df[columns_to_fill].fillna(df[columns_to_fill].mode().iloc[0])
```

In [19]:

```
df.isnull().any()
```

Out[19]:

```
age               False
workclass         False
fnlwgt            False
education         False
education.num     False
marital.status    False
occupation        False
relationship      False
race              False
sex               False
capital.gain      False
capital.loss      False
hours.per.week    False
native.country    False
income            False
dtype: bool
```

In [28]:

```
# define independent and dependent variable
X = df.drop(['income'], axis=1)
X
```

Out[28]:

ge	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.wee
90	Private	77053	HS-grad	9	Widowed	Prof-specialty	Not-in-family	White	Female	0	4356	4
82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	1
66	Private	186061	Some-college	10	Widowed	Prof-specialty	Unmarried	Black	Female	0	4356	4
54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	4
41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	4
...	...	...	...	...	...	...	...	...	...	...	...	.
22	Private	310152	Some-college	10	Never-married	Protective-serv	Not-in-family	White	Male	0	0	4
27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	3
40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	4
58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	4
22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	2

ws × 14 columns

In [30]:

```
Y = df['income']
Y
```

Out[30]:

```
0        <=50K
1        <=50K
2        <=50K
3        <=50K
4        <=50K
         ...
32556    <=50K
32557    <=50K
32558     >50K
32559    <=50K
32560    <=50K
Name: income, Length: 32561, dtype: object
```

In [32]:

```
#split the data for training
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25, random_state = 0)
```

In [33]:

```
catagorical feature need to be numeric, so we can do it lebelencoder or one hot encodein
om sklearn import preprocessing

tegorical = ['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'native.country'
r feature in categorical:
  le = preprocessing.LabelEncoder()
  X_train[feature] = le.fit_transform(X_train[feature])
  X_test[feature] = le.transform(X_test[feature])
```

In [35]:

```
X_train
```

Out[35]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.p
26464	55	3	208019	4	3	2	6	0	4	1	0	0	
16134	42	3	175526	11	9	2	6	0	4	1	0	0	
4747	42	3	137390	11	9	2	9	0	4	1	0	0	
8369	25	3	228608	15	10	4	2	2	1	0	0	0	
5741	60	3	106850	0	6	0	6	1	4	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
13123	90	5	282095	15	10	2	4	0	4	1	0	0	
19648	36	3	279721	11	9	2	13	0	4	1	0	0	
9845	26	3	51961	2	8	4	11	2	2	1	0	0	
10799	44	3	115323	12	14	2	3	0	4	1	0	0	
2732	39	3	224531	11	9	2	2	0	4	1	7298	0	

24420 rows × 14 columns

```
Insght: we see our catagorical variable is converted to numerical, but the attribute values in different range ,
need toscalize the df
```

In [36]:

```
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 [37]:

```
X_test
```

Out[37]:

class	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.weel
37019	0.798919	-1.109480	-1.983058	2.257844	1.231225	1.593354	0.393899	-1.432255	-0.145098	-0.218566	-0.03923
37019	0.448727	0.183026	-0.426081	0.926124	-0.280453	0.970330	0.393899	-1.432255	-0.145098	-0.218566	-1.25599
37019	-0.610401	1.217030	-0.036836	0.926124	0.223440	-0.275717	-3.125678	-1.432255	-0.145098	-0.218566	-0.44481
73497	-1.336061	-0.333977	1.130897	-0.405597	0.727333	-0.898741	0.393899	0.698199	-0.145098	-0.218566	-0.03923
37019	2.205997	0.441527	1.520141	-0.405597	1.231225	-0.898741	0.393899	0.698199	-0.145098	-0.218566	0.77194
...	...	...	...	...	...	...	...	...	...	...	..
37019	-0.826060	0.183026	-0.426081	2.257844	0.727333	-0.275717	0.393899	0.698199	0.000331	-0.218566	-0.44481
37019	0.233371	-0.333977	1.130897	0.926124	-0.280453	0.347306	0.393899	0.698199	-0.145098	-0.218566	-1.25599
37019	1.083225	1.217030	-0.036836	-1.737317	-1.036292	-0.275717	0.393899	0.698199	-0.145098	-0.218566	1.17753
03966	-0.462132	0.183026	-0.426081	-0.405597	-0.532399	-0.898741	0.393899	0.698199	-0.145098	-0.218566	0.77194
37019	-0.296240	1.217030	-0.036836	0.926124	-1.540185	-0.275717	-1.952486	-1.432255	-0.145098	5.529140	0.36635

ns

```
Insight: we saw our value is scalized, now data is ready for build the model
```

In [38]:

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

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

```

Out[38]:

```
▾ LogisticRegression
LogisticRegression()
```

In [40]:

```
y_pred = logreg.predict(X_test)
y_pred


```

Out[40]:

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

In [41]:

```
# now calculate the accuracy score
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
ac
```

Out[41]:

```
0.8227490480284977
```

```
Insight:we see accuracy score of model is 82%
```

```
#Lets see , if we can increase the model with PCA,by this, we can reduce dimennsion of dataset, so it reduce overfillting, so it predict well, model will be good
So the tecnique which we determined important feature is called PCA(we can know corelation and pattern by visulization),
#we also do  the PCA with covarience matrix creation between allpairs of variable, inaddition corelation
if target variable is catagorical apply chi-square test , or cross tabulation table
```

In [46]:

```
#In here we will do PCA and find eigen value and explained varience ratio
from sklearn.decomposition import PCA
pca = PCA()
X_train = pca.fit_transform(X_train)
eigen_values = pca.explained_variance_
eigen_values
```

Out[46]:

```
array([2.07029437, 1.42313018, 1.13968246, 1.10345498, 1.04416595,
       1.02058858, 0.98132252, 0.94599909, 0.90588147, 0.85898119,
       0.85064514, 0.67665715, 0.59699058, 0.38277966])
```

In [48]:

```
explained_variance = pca.explained_variance_ratio_
explained_variance
```

Out[48]:

```
array([0.14787211, 0.10164799, 0.08140256, 0.07881498, 0.07458023,
       0.0728962 , 0.07009159, 0.0675686 , 0.06470317, 0.06135329,
       0.06075788, 0.04833067, 0.04264044, 0.02734028])
```

```
Insight:by evaluting eigen values and explained variance we  can conclude that 13 attribute contain more information , so we can drop the least information contain atribute and train model and predict again find accuray
```

In [49]:

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

```

In [50]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

```

In [51]:

```
categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex']
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 [52]:

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

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

In [53]:

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

Out[53]:

```
▾ LogisticRegression
LogisticRegression()
```

In [54]:

```
y_pred = logreg.predict(X_test)
y_pred
```

Out[54]:

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

In [55]:

```
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
ac
```

Out[55]:

```
0.8225033779633951
```

```
Insight:we see accuracy score litle decrease .8227 to .8225
```

In [57]:

```
#Lets check with 12 varible by removing last 2 variable which contain less info
X = df.drop(['income','native.country', 'hours.per.week', 'capital.loss'], axis=1)
y = df['income']
```

In [59]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [60]:

```
categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex']
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 [61]:

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

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

In [62]:

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

Out[62]:

```
▾ LogisticRegression
LogisticRegression()
```

In [63]:

```
y_pred = logreg.predict(X_test)
```

In [64]:

```
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
ac
```

Out[64]:

```
0.8205380174425746
```

```
Insight: we found the accuracy is derase with attribute deleting, lets chech which feature is most important
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 [65]:

```
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_state = 0)


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'native.co
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
```

```
Insight: this result shows if we take first 12 record that hold 90%of variance and also reduce dimension
```

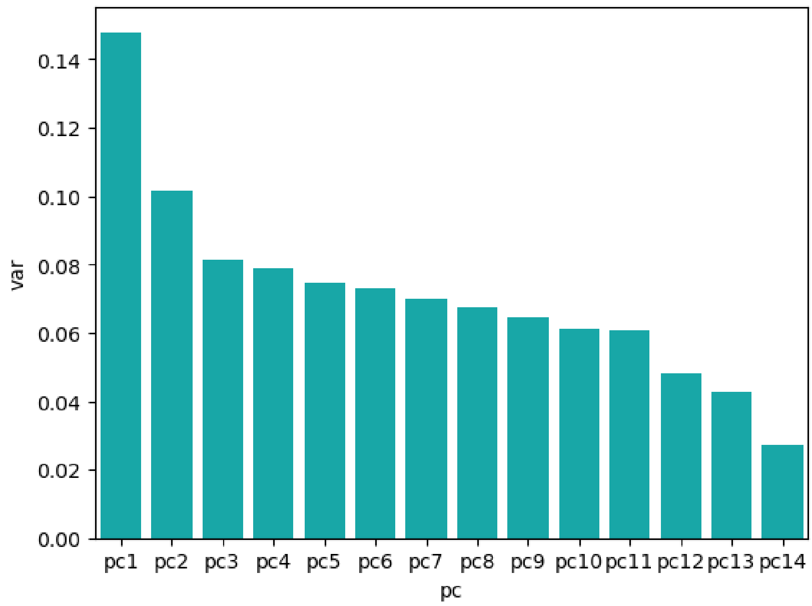
In [68]:

```
# we can aso plot this findiding
df2 = pd.DataFrame({'var': explained_variance, 'pc': ["pc1", "pc2", "pc3", "pc4", "pc5", "pc6", "pc7", "pc8", "pc9

sns.barplot(x="pc",y="var",data=df2,color="c")
```

Out[68]:

```
<Axes: xlabel='pc', ylabel='var'>
```



In [ ]:

```
Insight:we see that 90% of the variance is explained by first 12 attibute
```