Principal Component Analysis (PCA)

Importing the Libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
```

```
# Code for filtering out the warning.
warnings.filterwarnings("ignore")
```

```
# code to show all the columns in the table
pd.set_option("display.max_columns", None)
```

Creating the data frame.

```
adult_df = pd.read_csv(r"/content/adult_data.csv",
                       header = None, index_col = None, delimiter=' *, *')
adult_df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

```
adult_df.shape
```

```
(32561, 15)
```

```
adult_df.columns = ['age','workclass','fnlwgt','education','education_num',
                    'marital_status','occupation','relationship','race','sex',
                    'capital_gain','capital_loss','hours_per_week',
                    'native_country','income']

adult_df.head()
```

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0

Pre-processing the data.

Making a copy of the dataset.

```
adult_df_rev = pd.DataFrame.copy(adult_df)
adult_df_rev.head()
```

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0

Feature Selection.

```
adult_df_rev = adult_df_rev.drop(["fnlwgt","education"], axis = 1)
adult_df_rev.head()
```

	age	workclass	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours
0	39	State-gov	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	
1	50	Self-emp-not-inc	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	
2	38	Private	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	
3	53	Private	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	
4	28	Private	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	

Handling the missing values.

```
adult_df_rev.isnull().sum()
```

```
age               0
workclass         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
```

```
for i in adult_df_rev.columns:
    print(adult_df_rev[i].unique())
```

```
[39 50 38 53 28 37 49 52 31 42 30 23 32 40 34 25 43 54 35 59 56 19 20 45
 22 48 21 24 57 44 41 29 18 47 46 36 79 27 67 33 76 17 55 61 70 64 71 68
 66 51 58 26 60 90 75 65 77 62 63 80 72 74 69 73 81 78 88 82 83 84 85 86
 87]
['State-gov' 'Self-emp-not-inc' 'Private' 'Federal-gov' 'Local-gov' '?'
 'Self-emp-inc' 'Without-pay' 'Never-worked']
[13  9  7 14  5 10 12 11  4 16 15  3  6  2  1  8]
```

```
['Never-married' 'Married-civ-spouse' 'Divorced' 'Married-spouse-absent'
 'Separated' 'Married-AF-spouse' 'Widowed']
['Adm-clerical' 'Exec-managerial' 'Handlers-cleaners' 'Prof-specialty'
 'Other-service' 'Sales' 'Craft-repair' 'Transport-moving'
 'Farming-fishing' 'Machine-op-inspct' 'Tech-support' '?'
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv']
['Not-in-family' 'Husband' 'Wife' 'Own-child' 'Unmarried' 'Other-relative']
['White' 'Black' 'Asian-Pac-Islander' 'Amer-Indian-Eskimo' 'Other']
['Male' 'Female']
[ 2174     0 14084  5178  5013  2407 14344 15024  7688 34095  4064  4386
  7298  1409  3674  1055  3464  2050  2176   594 20051  6849  4101  1111
  8614  3411  2597 25236  4650  9386  2463  3103 10605  2964  3325  2580
  3471  4865 99999  6514  1471  2329  2105  2885 25124 10520  2202  2961
 27828  6767  2228  1506 13550  2635  5556  4787  3781  3137  3818  3942
   914   401  2829  2977  4934  2062  2354  5455 15020  1424  3273 22040
  4416  3908 10566   991  4931  1086  7430  6497   114  7896  2346  3418
  3432  2907  1151  2414  2290 15831 41310  4508  2538  3456  6418  1848
  3887  5721  9562  1455  2036  1831 11678  2936  2993  7443  6360  1797
  1173  4687  6723  2009  6097  2653  1639 18481  7978  2387  5060]
[   0 2042 1408 1902 1573 1887 1719 1762 1564 2179 1816 1980 1977 1876
 1340 2206 1741 1485 2339 2415 1380 1721 2051 2377 1669 2352 1672  653
 2392 1504 2001 1590 1651 1628 1848 1740 2002 1579 2258 1602  419 2547
 2174 2205 1726 2444 1138 2238  625  213 1539  880 1668 1092 1594 3004
 2231 1844  810 2824 2559 2057 1974  974 2149 1825 1735 1258 2129 2603
 2282  323 4356 2246 1617 1648 2489 3770 1755 3683 2267 2080 2457  155
 3900 2201 1944 2467 2163 2754 2472 1411]
[40 13 16 45 50 80 30 35 60 20 52 44 15 25 38 43 55 48 58 32 70  2 22 56
 41 28 36 24 46 42 12 65  1 10 34 75 98 33 54  8  6 64 19 18 72  5  9 47
 37 21 26 14  4 59  7 99 53 39 62 57 78 90 66 11 49 84  3 17 68 27 85 31
 51 77 63 23 87 88 73 89 97 94 29 96 67 82 86 91 81 76 92 61 74 95]
['United-States' 'Cuba' 'Jamaica' 'India' '?' 'Mexico' 'South'
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand' 'Ecuador'
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-Salvador'
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'
 'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinadad&Tobago' 'Greece'
 'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary' 'Holand-Netherlands']
['<=50K' '>50K']
```

```
adult_df_rev = adult_df_rev.replace(["?"], np.nan)
adult_df_rev.isnull().sum()
```

```
age                  0
workclass         1836
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
```

```
for i in ["workclass","occupation","native_country"]:
    adult_df_rev[i].fillna(adult_df_rev[i].mode()[0], inplace = True)
```

```
adult_df_rev.isnull().sum()
```

```
age               0
workclass         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
```

Outlier Handling.

```
adult_df_rev.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   education_num   32561 non-null  int64
 3   marital_status  32561 non-null  object
 4   occupation      32561 non-null  object
 5   relationship    32561 non-null  object
 6   race            32561 non-null  object
 7   sex             32561 non-null  object
 8   capital_gain    32561 non-null  int64
 9   capital_loss    32561 non-null  int64
 10  hours_per_week  32561 non-null  int64
 11  native_country  32561 non-null  object
 12  income          32561 non-null  object
dtypes: int64(5), object(8)
memory usage: 3.2+ MB
```

```
adult_df_rev.describe()
```

	age	education_num	capital_gain	capital_loss	hours_per_week
count	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	10.080679	1077.648844	87.303830	40.437456
std	13.640433	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	48.000000	12.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	4356.000000	99.000000

Encoding of categorical variables into numerical.

```
colname = []

for i in adult_df_rev.columns:
    if(adult_df_rev[i].dtype == "object"):
        colname.append(i)

colname
```

```
['workclass',
 'marital_status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native_country',
 'income']
```

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for i in colname:
    adult_df_rev[i] = le.fit_transform(adult_df_rev[i])

    le_name_mapping = list(zip(le.classes_, le.transform(le.classes_)))
    print("Feature :", i)
    print("Mapping :", le_name_mapping)
```

```
Feature : workclass
Mapping : [('Federal-gov', 0), ('Local-gov', 1), ('Never-worked', 2), ('Private', 3), ('Self-emp-inc', 4), ('Self-emp-not-inc', 5), ('S
Feature : marital_status
Mapping : [('Divorced', 0), ('Married-AF-spouse', 1), ('Married-civ-spouse', 2), ('Married-spouse-absent', 3), ('Never-married', 4), ('
Feature : occupation
```

```
Mapping : [('Adm-clerical', 0), ('Armed-Forces', 1), ('Craft-repair', 2), ('Exec-managerial', 3), ('Farming-fishing', 4), ('Handlers-cl
Feature : relationship
Mapping : [('Husband', 0), ('Not-in-family', 1), ('Other-relative', 2), ('Own-child', 3), ('Unmarried', 4), ('Wife', 5)]
Feature : race
Mapping : [('Amer-Indian-Eskimo', 0), ('Asian-Pac-Islander', 1), ('Black', 2), ('Other', 3), ('White', 4)]
Feature : sex
Mapping : [('Female', 0), ('Male', 1)]
Feature : native_country
Mapping : [('Cambodia', 0), ('Canada', 1), ('China', 2), ('Columbia', 3), ('Cuba', 4), ('Dominican-Republic', 5), ('Ecuador', 6), ('El-
Feature : income
Mapping : [('<=50K', 0), ('>50K', 1)]
```

```
adult_df_rev.head()
```

	age	workclass	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per
0	39	6	13	4	0	1	4	1	2174	0	
1	50	5	13	2	3	0	4	1	0	0	
2	38	3	9	0	5	1	4	1	0	0	
3	53	3	7	2	5	0	2	1	0	0	
4	28	3	13	2	9	5	2	0	0	0	

Create X & Y

```
X = adult_df_rev.values[:,:-1]
Y = adult_df_rev.values[:,-1]
Y = Y.astype(int)
```

```
print(X)
```

```
[[39  6 13 ...  0 40 38]
 [50  5 13 ...  0 13 38]
 [38  3  9 ...  0 40 38]
 ...
 [58  3  9 ...  0 40 38]
 [22  3  9 ...  0 20 38]
 [52  4  9 ...  0 40 38]]
```

```
print(Y)
```

```
[0 0 0 ... 0 0 1]
```

Spliting the data.

```
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 = 10)
```

```
print(X_train)
```

```
[[20  3 10 ...  0 15 38]
 [32  3  9 ...  0 40 38]
 [37  3  9 ...  0 40 38]
 ...
 [19  3 10 ...  0 40 38]
 [34  3  7 ...  0 40 20]
 [19  3  9 ...  0 40 38]]
```

```
print(X_test)
```

```
[[58  6  9 ...  0 16 38]
 [23  1  9 ...  0 40 38]
 [41  3 11 ...  0 60 38]
 ...
 [64  1  7 ...  0 40 38]
```

```
[33  3 11 ...  0 45 38]
[90  3 13 ...  0 45 38]]
```

```
print(Y_train)
```

```
[0 1 1 ... 0 0 0]
```

```
print(Y_test)
```

```
[0 0 1 ... 1 0 0]
```

Scaling the data.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Applying the PCA. (For Feature Selection)

```
from sklearn.decomposition import PCA

pca = PCA(n_components = None)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[0.17179376 0.09693511 0.09261766 0.08972459 0.08663778 0.08176419
 0.08053327 0.0744331  0.07230738 0.06430992 0.05664846 0.03229477]
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 0.75)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[0.17179376 0.09693511 0.09261766 0.08972459 0.08663778 0.08176419
 0.08053327 0.0744331 ]
```

How to find PCA components?

```
pca.n_components_
```

```
8
```

Building the Logistic Regression Model.

```
from sklearn.linear_model import LogisticRegression

# Build the model.
model = LogisticRegression()

# Train the model.
model.fit(X_train, Y_train)

# Predict using model
Y_pred = model.predict(X_test)
```

```
print(list(zip(Y_test, Y_pred)))
```

```
[(0, 0), (0, 0), (1, 1), (0, 0), (1, 1), (0, 0), (0, 0), (1, 1), (0, 0), (0, 0), (0, 0), (0, 0), (0, 1), (0, 0), (0, 0), (0, 0), (0, 0)
```

Evaluating the Building model.

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

print("Confusion Matrix = ")
print(confusion_matrix(Y_test, Y_pred), "\n")
print("Accuracy Score = ", accuracy_score(Y_test, Y_pred), "\n")
print("Classification Report = ")
print(classification_report(Y_test, Y_pred))
```

```
Confusion Matrix =
[[7008  415]
 [1280 1066]]

Accuracy Score =  0.8264919643771113

Classification Report =
              precision    recall  f1-score   support

           0       0.85      0.94      0.89      7423
           1       0.72      0.45      0.56      2346

    accuracy                           0.83      9769
   macro avg       0.78      0.70      0.72      9769
weighted avg       0.82      0.83      0.81      9769
```

Now if we put n_components = 0.85

```
# splitting the data into training and testing data set.

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 = 10)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 0.85)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[0.17179376 0.09693511 0.09261766 0.08972459 0.08663778 0.08176419
 0.08053327 0.0744331  0.07230738 0.06430992]
```

```
from sklearn.linear_model import LogisticRegression

# Build the model.
model = LogisticRegression()

# Train the model.
model.fit(X_train, Y_train)

# Predict using model
Y_pred = model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

print("Confusion Matrix = ")
print(confusion_matrix(Y_test, Y_pred), "\n")
print("Accuracy Score = ", accuracy_score(Y_test, Y_pred), "\n")
print("Classification Report = ")
print(classification_report(Y_test, Y_pred))
```

```
Confusion Matrix =
[[7012  411]
 [1318 1028]]

Accuracy Score =  0.8230115672023749

Classification Report =
              precision    recall  f1-score   support

           0       0.84      0.94      0.89      7423
           1       0.71      0.44      0.54      2346

    accuracy                           0.82      9769
   macro avg       0.78      0.69      0.72      9769
weighted avg       0.81      0.82      0.81      9769
```

```
# Thus now for n_components = 0.85 we get accuracy = 82.30 % --> variables needs = 10
```

Now if we put n_components = 0.95

```
# splitting the data into training and testing data set.

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 = 10)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 0.95)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[0.17179376 0.09693511 0.09261766 0.08972459 0.08663778 0.08176419
 0.08053327 0.0744331  0.07230738 0.06430992 0.05664846]
```

```
from sklearn.linear_model import LogisticRegression

# Build the model.
model = LogisticRegression()

# Train the model.
model.fit(X_train, Y_train)

# Predict using model
Y_pred = model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

print("Confusion Matrix = ")
print(confusion_matrix(Y_test, Y_pred), "\n")
print("Accuracy Score = ", accuracy_score(Y_test, Y_pred), "\n")
print("Classification Report = ")
print(classification_report(Y_test, Y_pred))
```

```
Confusion Matrix = 
[[7027  396]
 [1310 1036]] 

Accuracy Score =  0.8253659535264612 

Classification Report = 
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      7423
           1       0.72      0.44      0.55      2346

    accuracy                           0.83      9769
   macro avg       0.78      0.69      0.72      9769
weighted avg       0.81      0.83      0.81      9769
```

Applying PCA (For Data Visualization.)

```
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 = 10)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Applying PCA
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[0.17179376 0.09693511]
```

```
from sklearn.linear_model import LogisticRegression

# Build the model.
model = LogisticRegression()

# Train the model.
model.fit(X_train, Y_train)

# Predict using model.
Y_pred = model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
print("Confusion Matrix = ")
print(confusion_matrix(Y_test, Y_pred), "\n")
print("Accuracy Score = ", accuracy_score(Y_test, Y_pred), "\n")
print("Classification Report = ")
print(classification_report(Y_test, Y_pred))
```

```
Confusion Matrix = 
[[7013  410]
 [1435  911]] 

Accuracy Score =  0.8111372709591566 

Classification Report = 
              precision    recall  f1-score   support

           0       0.83      0.94      0.88      7423
           1       0.69      0.39      0.50      2346

    accuracy                           0.81      9769
   macro avg       0.76      0.67      0.69      9769
```

```
weighted avg       0.80      0.81      0.79      9769
```

```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, Y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, model.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.5, cmap = ListedColormap(('yellow', 'black')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('yellow', 'black'))(i), label = j)
plt.title('LR (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```