Experiment No. 7
Apply Dimensionality Reduction on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:
Date of Submission:

Aim: Apply Dimensionality Reduction on Adult Census Income Dataset and analyze the performance of the model.

Objective: Able to perform various feature engineering tasks, perform dimetionality reduction on the given dataset and maximize the accuracy, Precision, Recall, F1 score.

Theory:

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Dataset:

Predict whether income exceeds $50K/yr based on census data. Also known as "Adult" dataset.

Attribute Information:

Listing of attributes:

>50K, <=50K.

age: continuous.

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.

education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.

capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad & Tobago, Peru, Hong, Holand-Netherlands.

Conclusion:

- The logistic regression model achieves an accuracy of around 83% post-dimensionality reduction.
- The precision for the >50K class is 0.72, recall is 0.43, and F1-score is 0.54.
- The precision for the >50K class is 0.72, recall is 0.43, and F1-score is 0.54.

- Dimensionality reduction enhanced model performance by reducing overfitting and noise, simplifying the data, and highlighting key patterns, resulting in improved accuracy, precision, and recall.
- Dimensionality reduction improved model efficiency by reducing computational complexity and memory usage. By retaining the most informative features, it streamlined training and inference, resulting in faster execution without significant loss in predictive accuracy.

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

```
dataset=pd.read_csv("/content/adult_data.csv")
```

```
# 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)
```

```
adult_df=dataset
```

```
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	cap
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	

```
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	cap
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	

```
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_pe
0	50	Self-emp-not-inc	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	
1	38	Private	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	
2	53	Private	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	

```
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())
```

```
[50 38 53 28 37 49 52 31 42 30 23 32 40 34 25 43 54 35 59 56 19 39 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]
[' Self-emp-not-inc' ' Private' ' State-gov' ' 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]
[' Married-civ-spouse' ' Divorced' ' Married-spouse-absent'
 ' Never-married' ' Separated' ' Married-AF-spouse' ' Widowed']
[' Exec-managerial' ' Handlers-cleaners' ' Prof-specialty'
 ' Other-service' ' Adm-clerical' ' Sales' ' Craft-repair'
 ' Transport-moving' ' Farming-fishing' ' Machine-op-inspct'
 ' Tech-support' ' ?' ' Protective-serv' ' Armed-Forces'
 ' Priv-house-serv']
[' Husband' ' Not-in-family' ' Wife' ' Own-child' ' Unmarried'
 ' Other-relative']
[' White' ' Black' ' Asian-Pac-Islander' ' Amer-Indian-Eskimo' ' Other']
[' Male' ' Female']
[    0 14084  5178  5013  2407 14344 15024  7688 34095  4064  4386  7298
  1409  3674  1055  3464  2050  2176  2174   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]
[13 40 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         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 ["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
```

```
adult_df_rev.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32560 entries, 0 to 32559
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32560 non-null  int64 
 1   workclass       32560 non-null  object
 2   education_num   32560 non-null  int64 
 3   marital_status  32560 non-null  object
 4   occupation      32560 non-null  object
 5   relationship    32560 non-null  object
 6   race            32560 non-null  object
 7   sex             32560 non-null  object
 8   capital_gain    32560 non-null  int64 
 9   capital_loss    32560 non-null  int64 
 10  hours_per_week  32560 non-null  int64 
 11  native_country  32560 non-null  object
 12  income          32560 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	32560.000000	32560.000000	32560.000000	32560.000000	32560.000000
mean	38.581634	10.080590	1077.615172	87.306511	40.437469
std	13.640642	2.572709	7385.402999	402.966116	12.347618
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

```
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 : [(' ?', 0), (' Federal-gov', 1), (' Local-gov', 2), (' Never-worked', 3), (' Private', 4), (' Self-emp-inc', 5), (' Self-emp-r
Feature : marital_status
Mapping : [(' Divorced', 0), (' Married-AF-spouse', 1), (' Married-civ-spouse', 2), (' Married-spouse-absent', 3), (' Never-married', 4)
Feature : occupation
Mapping : [(' ?', 0), (' Adm-clerical', 1), (' Armed-Forces', 2), (' Craft-repair', 3), (' Exec-managerial', 4), (' Farming-fishing', 5)
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 : [(' ?', 0), (' Cambodia', 1), (' Canada', 2), (' China', 3), (' Columbia', 4), (' Cuba', 5), (' Dominican-Republic', 6), (' Ec
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_week	native
0	50	6	13	2	4	0	4	1	0	0	13	
1	38	4	9	0	6	1	4	1	0	0	40	
2	53	4	7	2	6	0	2	1	0	0	40	
3	28	4	13	2	10	5	2	0	0	0	40	
4	37	4	14	2	4	5	4	0	0	0	40	

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

```
print(X)
```

```
[[50  6 13 ...  0 13 39]
 [38  4  9 ...  0 40 39]
 [53  4  7 ...  0 40 39]
 ...
 [58  4  9 ...  0 40 39]
 [22  4  9 ...  0 20 39]
 [52  5  9 ...  0 40 39]]
```

```
print(Y)
```

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

```
# 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)
```

```
print(X_train)
```

```
[[38  2 13 ...  0 40 39]
 [25  4  8 ...  0 60 26]
 [21  4 10 ...  0 16 39]
 ...
 [32  4  9 ...  0 50 39]
 [53  4  9 ...  0 40 39]
 [36  7  9 ...  0 90 39]]
```

```
print(X_test)
```

```
[[41  4 13 ...  0 50 39]
 [38  4 10 ...  0 60 39]
 [24  4  9 ...  0 40 39]
 ...
 [46  2 10 ...  0 40 39]
 [46  4 10 ...  0 40 39]
 [34  5 14 ...  0 40 39]]
```

```
print(Y_train)
```

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

```
print(Y_test)
```

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

```
from sklearn.preprocessing import StandardScaler

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

```
## pca

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)
```

```
[0.17613611 0.10628616 0.09490602 0.09364629 0.08570574 0.08310758
 0.07397927 0.07072631 0.0676082  0.05945211 0.0566263  0.03181992]
```

```
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)
```

```
[0.17613611 0.10628616 0.09490602 0.09364629 0.08570574 0.08310758
 0.07397927 0.07072631]
```

```
pca.n_components_
```

```
8
```

```
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)))
```

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

```
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  447]
 [1247 1061]]

Accuracy Score =  0.8265765765765766

Classification Report =
              precision    recall  f1-score   support

           0       0.85      0.94      0.89      7460
           1       0.70      0.46      0.56      2308

    accuracy                           0.83      9768
   macro avg       0.78      0.70      0.72      9768
weighted avg       0.81      0.83      0.81      9768
```

```
# splitting the data into training and testing data set.
# IF WE PUT N=0.85
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)
```

```
[0.17613611 0.10628616 0.09490602 0.09364629 0.08570574 0.08310758
 0.07397927 0.07072631 0.0676082 ]
```

```
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 = 
[[7042  418]
 [1276 1032]]

Accuracy Score =  0.8265765765765766

Classification Report = 
              precision    recall  f1-score   support

           0       0.85      0.94      0.89      7460
           1       0.71      0.45      0.55      2308

    accuracy                           0.83      9768
   macro avg       0.78      0.70      0.72      9768
weighted avg       0.81      0.83      0.81      9768
```

```
## N = 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)
```

```
[0.17613611 0.10628616 0.09490602 0.09364629 0.08570574 0.08310758
 0.07397927 0.07072631 0.0676082  0.05945211 0.0566263 ]
```

```
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 =
[[7049  411]
 [1266 1042]]

Accuracy Score =  0.8283169533169533

Classification Report =
              precision    recall  f1-score   support

           0       0.85      0.94      0.89      7460
           1       0.72      0.45      0.55      2308

    accuracy                           0.83      9768
   macro avg       0.78      0.70      0.72      9768
weighted avg       0.82      0.83      0.81      9768
```

```
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)                    # Note: for Data Visualization we use n_components.
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

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

```
[0.17613611 0.10628616]
```

```
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 =
[[7102  358]
 [1683  625]]

Accuracy Score =  0.791052416052416

Classification Report =
              precision    recall  f1-score   support

           0       0.81      0.95      0.87      7460
           1       0.64      0.27      0.38      2308

    accuracy                           0.79      9768
   macro avg       0.72      0.61      0.63      9768
```

```
weighted avg       0.77      0.79      0.76      9768
```

```
# 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(('red', '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(('red', 'black'))(i), label = j)
plt.title('LR (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```