LAB MANUAL

Learn Machine Learning with Python

Experiment 1: Working with NumPy in Python

AIM: To understand the fundamentals and application of NumPy library In Machine Learning

Operations:

1. Importing & Checking version
2. Array Creation in NumPy
3. Array Operations in NumPy

Algorithm:

1. Import the library
2. Check the version of the library
3. Create the variable with object and input data as input arguments
4. Print the output

Program:

1. Importing & Checking version

```
import numpy as np
np. version.version
```

Result: '1.16.5' [Based on the version in the system]

2. Array Creation in NumPy

2.1. Creating 1D array

```
first_array = np.array([1,2,3])
print(first_array)
```

Result: [1 2 3]

2.2. Creating 2D array

```
second_array = np.array([(4,5,6),(7,8,9)])
print(second_array)
```

Result: [[4 5 6] [7 8 9]]

2.3. Creating 3D array

```
third_array = np.array([[(10,11,12),(13,14,15)],[(16,17,18),(13,14,15)]])
print(third_array)
```

Result: [[[10 11 12] [13 14 15]] [[16 17 18] [13 14 15]]

2.4. Array of Zeros

```
zero_array = np.zeros((2,2))
print(zero_array)
```

Result: [[0. 0.] [0. 0.]]

2.5. Array of Ones

```
ones_array = np.ones((3,4))
print(ones_array)
```

Result: [[1. 1. 1. 1.] [1. 1. 1. 1.] [1. 1. 1. 1.]]

2.6. Matrix using NumPy

```
a = np.matrix('1 2; 3 4')
print(a)
```

Result: `matrix([[1, 2], [3, 4]])`

3. Array Operations in NumPy

3.1. Create a Matrix

```
my_matrix = np.array([(11,17),(23,25)])
print(my_matrix)
```

Result: `[[11 17] [23 25]]`

3.2. Transpose Operation

```
matrix_transpose = np.transpose(my_matrix)
print(matrix_transpose)
```

Result: `[[11 23] [17 25]]`

3.3. Determinant Operation

```
det = np.linalg.det(my_matrix)
print(det)
```

Result: `-115.99999999999999`

3.4. Inverse Operation

```
inverse=np.linalg.inv(my_matrix)
inverse
```

Result:
```
array([[-0.21551724,  0.14655172],
[ 0.19827586, -0.09482759]])
```

3.5. Resize an Array

Note: Please use the array with ones which was created above

```
arr_ones.resize((4,1))
arr_ones
```

Result:
```
array([[1.],
       [1.],
       [1.],
       [1.]
```

Experiment 2: Working with Pandas in Python

AIM: To understand the fundamentals and application of Pandas library In Machine Learning

Operations:

1. Importing Pandas Library
2. Creating a series in Pandas
3. Creating Data frame in Pandas
4. Data Frame Operations
5. Data Manipulation

Algorithm:

1. Import the library
2. Create a series using Pandas library
3. Create a data frame using Pandas library
4. Print the output

Program:

1. **Importing & Checking version**

```
import pandas as pd
```

2. **Creating a series in Pandas**

```
alphabet = pd.Series([1,2,3,4],index=['A','B','C','D'])
print(alphabet)
```

Result :

```
A    1
B    2
C    3
D    4
dtype: int64
```

3. **Creating a dataframe in Pandas**

```
data = {'Games': ['GTA V','NFS Rivals','Cricket 19'],
        'Rating':[9,7,9]}
dataframe = pd.DataFrame(data,columns=['Games','Rating'])

dataframe
```

Result :

	Games	Rating
0	GTA V	9
1	NFS Rivals	7
2	Cricket 19	9

4. **Data Frame Operations**

4.1. Creating a Data frame with Random Numbers

```
random = pd.DataFrame(np.random.randint(0,300,size=(20,4)),columns=list('ABCD'))
random
```

Result :

	A	B	C	D
0	3	205	68	196
1	116	155	36	216
2	285	282	234	248
3	250	40	70	273
4	121	205	180	160

4.2. Saving a Data frame

```
random.to_csv('C:/Users/Sandeap/Documents/Py Testing/IRP - Machine Learning Using Python/Notebooks/Pandas/random.csv')
```

Note : Please give the location where you want to save the document along with document name and the extension. Upon saving, please go the given location and fetch the file

5. Data Manipulation

5.1. Importing external data

```
data = pd.read_csv('C:/Users/Sandeap/Videos/OBS Studio/IRP ML/Pandas/random.csv')
data
```

Result:

	Unnamed: 0	A	B	C	D
0	0	205	220	10	183
1	1	293	59	4	267
2	2	269	183	172	211
3	3	138	276	79	54
4	4	162	275	227	143

5.2. Dropping a Data frame

```
data.drop('Unnamed: 0',axis=1)
```

Result :

	A	B	C	D
0	205	220	10	183
1	293	59	4	267
2	269	183	172	211
3	138	276	79	54
4	162	275	227	143

5.3. Shape of a Data frame

data.shape

Result : `(20, 5)`

5.4. Get information about the Data frame

data.info()

Result :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  20 non-null     int64
 1   A           20 non-null     int64
 2   B           20 non-null     int64
 3   C           20 non-null     int64
 4   D           20 non-null     int64
dtypes: int64(5)
memory usage: 928.0 bytes
```

5.5. Shuffling the data frame

```
from sklearn.utils import shuffle
shuffle_data = shuffle(data).reset_index()
shuffle_data
```

Result :

	index	Unnamed: 0	A	B	C	D
0	9	9	286	245	255	176
1	1	1	293	59	4	267
2	17	17	290	38	245	194
3	19	19	79	291	83	149
4	14	14	202	171	214	276

Experiment 3: Data Visualization using Matplotlib and Seaborn

AIM: To understand the fundamentals of Data visualization and extracting insights from data using data visualization

Operations:

1. Importing Matplotlib library
2. Creating Data for visualization
3. Data Visualization using Matplotlib
4. Importing Searborn library
5. Advanced Data Visualization using Seaborn

Algorithm:

1. Import the library
2. Create data
3. Perform data visualization
4. Print the graph

Program:

1. **Import library**

```
import matplotlib.pyplot as plt
%matplotlib inline
```

2. **Creating data**

```
Days = ['Mon','Tue','Wed','Thur','Fri','Sat','Sun']
Temperature = [33,34,37,32,36,39,31]
Days
Temperature
```

Result:

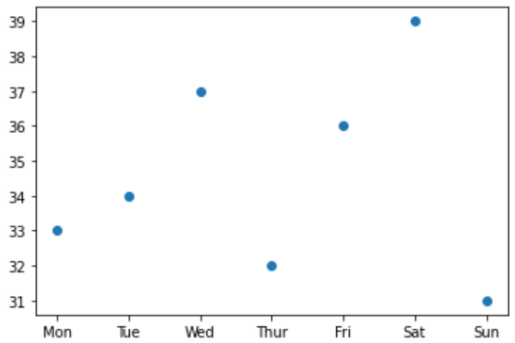
```
['Mon','Tue','Wed','Thur','Fri','Sat','Sun']
[33,34,37,32,36,39,31]
```

3. **Data Visualization using Matplotlib**

3.1. Scatter Plot

```
plt.scatter (Days, Temperature)
plt.show()
```

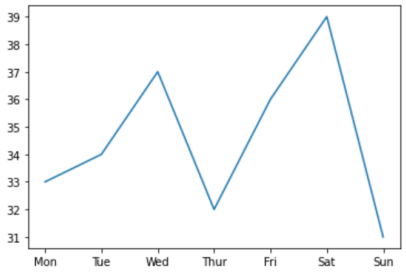
Result :



3.2. Line plot

```
plt.plot(Days, Temperature, linestyle='solid')
plt.show()
```

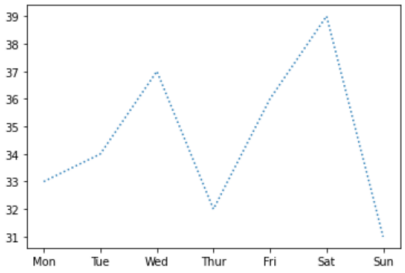
Result :



3.3. Line plot with dotted line

```
plt.plot(Days, Temperature, linestyle='dotted')
plt.show()
```

Result :



3.4. Pie Chart

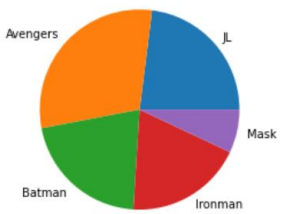
3.4.1. Creating Data

```
Movies = ['JL','Avengers','Batman','Ironman','Mask']
Percentage = [23,30,21,19,7]
```

3.4.2. Plot Pie chart

```
plt.pie(Percentage, labels=Movies)
```

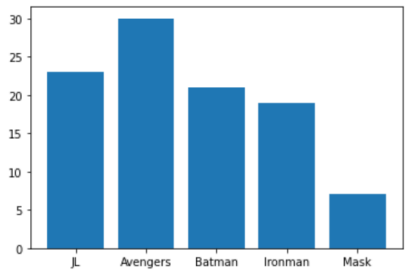
Result:



3.5. Bar Plot

plt.bar (Movies, Percentage)

Result:



4. Importing Seaborn Library

4.1. Importing Seaborn Library and other required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

Note: Creating data, plotting using seaborn needs other dependent libraries

4.2. Check for existing default datasets in seaborn

sns.get_dataset_names()

Result :

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'geyser',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'taxis',
 'tips',
 'titanic']
```

4.3. Loading tips dataset

tips = sns.load_dataset('tips')

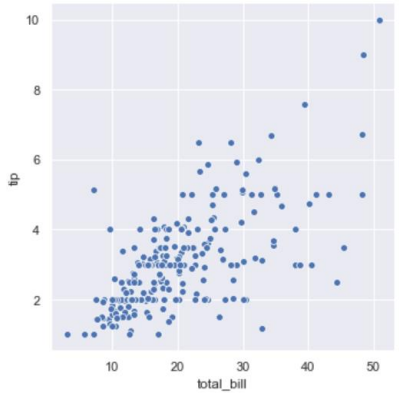
tips.head()

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

4.4. Relational Plot

sns.relplot(x='total_bill',y='tip',data=tips)

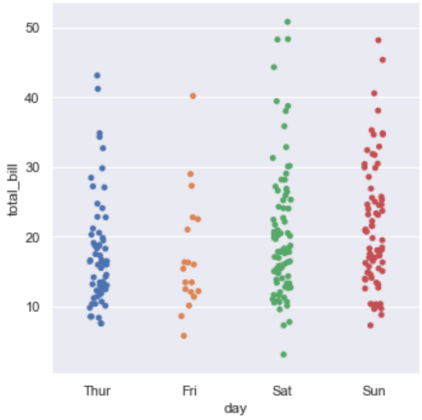
Result :



4.5. Categorical Plot

sns.catplot(x='day',y='total_bill',data=tips)

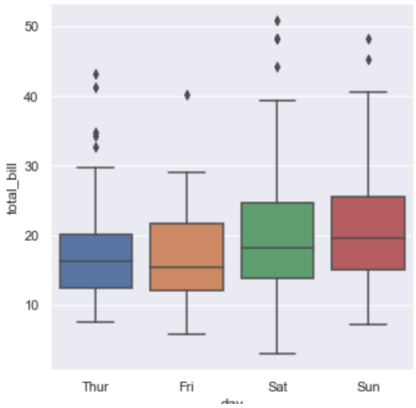
Result :



4.6. Box Plot

sns.catplot(x='day',y='total_bill',kind='box',data=tips)

Result :



Experiment 5: Data Preprocessing & Feature Scaling in Python

AIM: To clean data and perform feature scaling

Algorithms:

1. Import required libraries & Data
2. Remove Missing values
3. Handle Categorical Data
4. Feature Scaling

5.1. Importing libraries

```
import numpy as np
import pandas as pd
dataset = pd.read_csv('Data.csv')
dataset
```

Result :

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes

5.2. Handling Missing data

5.2.1. Reshaping dataset to dataframe

```
x = dataset.iloc[:,:-1].values
y = dataset.iloc[:,-1].values
x
y
```

Result :

```
array([['France', 44.0, 72000.0],
['Spain', 27.0, 48000.0],
['Germany', 30.0, 54000.0],
['Spain', 38.0, 61000.0],
['Germany', 40.0, nan],
['France', 35.0, 58000.0],
['Spain', nan, 52000.0],
['France', 48.0, 79000.0],
['Germany', 50.0, 83000.0],
['France', 37.0, 67000.0]], dtype=object)

array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

5.2.2. Finding Null Elements

dataset.isnull().sum()

Result:

```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

5.2.3. Importing Imputer Function

from sklearn.impute import SimpleImputer

5.2.4. Applying Simple Imputer

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

5.2.5. Fitting and transform values in imputer

imputer.fit(x[:,1:3])

x[:,1:3] = imputer.transform(x[:,1:3])

5.2.6. Printing filled values

print(x)

Result :

```
[['France' 44.0 72000.0]
['Spain' 27.0 48000.0]
['Germany' 30.0 54000.0]
['Spain' 38.0 61000.0]
['Germany' 40.0 63777.77777777778]
['France' 35.0 58000.0]
['Spain' 38.77777777777778 52000.0]
['France' 48.0 79000.0]
['Germany' 50.0 83000.0]
['France' 37.0 67000.0]]
```

5.3. Handling Categorical Data

5.3.1. Importing libraries

import pandas as pd

from sklearn.preprocessing import LabelEncoder

5.3.2. Creating data

```
salary_class = pd.DataFrame({'Salary':[5000,84000,22000,8000,75000],
        'Class':['Low','High','Medium','Low','High']})
salary_class
```

Result :

	Salary	Class
0	5000	Low
1	84000	High
2	22000	Medium
3	8000	Low
4	75000	High

5.3.3. Applying label encoder

lab_enc ode = LabelEncoder()

5.3.4. Fitting the values and transforming

salary_class['Class'] = lab_encode.fit_transform(salary_class['Class'])

5.3.5. Label encoded output

salary_class

Result:

	Salary	Class
0	5000	1
1	84000	0
2	22000	2
3	8000	1
4	75000	0

So the above encoded values are,

0 - High
1 - Low
2 - Medium

5.4. Feature Scaling

5.4.1. Importing library

```
import pandas as pd
import numpy as np
```

5.4.2. Creating data frame for minmax scaler

```
mms = pd.DataFrame({'x1':np.random.randint(0,10,size=100),
                    'x2':np.random.randint(-10,10,size=100)*40,
                    'x3':np.random.randint(50,100,size=100)})
mms
```

Result :

	x1	x2	x3
0	6	200	76
1	2	-120	78
2	4	-200	80

5.4.3. Importing Minmaxscaler

from sklearn.preprocessing import MinMaxScaler

5.4.4. Create an object with min max scaler

minmax = MinMaxScaler()

5.4.5. Fitting & viewing data frame in the object

mms = minmax.fit_transform(mms)

mms

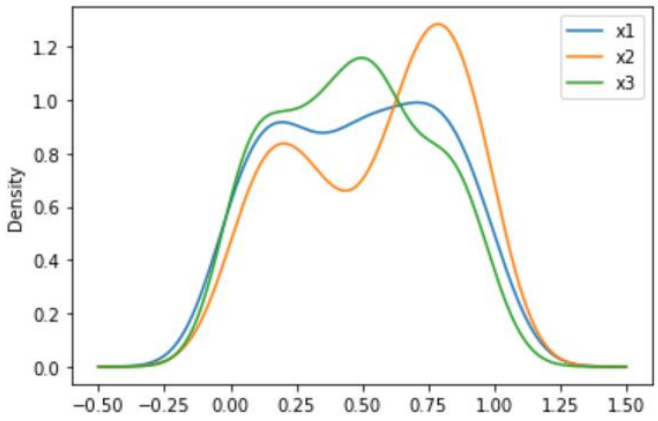
Result :

```
array([[0.66666667, 0.78947368, 0.53061224],
[0.22222222, 0.36842105, 0.57142857],
[0.44444444, 0.26315789, 0.6122449 ],
[0.44444444, 0.89473684, 0.40816327],
[0.77777778, 1.        , 0.67346939],
[1.        , 0.05263158, 0.04081633]
```

5.4.6. Plot Density plot

mms.plot.kde()

Result :



Experiment 6: Working with Descriptive Statistics using SciPy

AIM: To perform statistical analysis on data using SciPy library

Algorithm:

1. Importing the necessary library for descriptive statistics
2. Load the dataset we want to calculate descriptive statistics
3. Calculate the descriptive statistics parameters using scipy

Program :

Import Libraries

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

Import & View the data

```
mtcars = pd.read_csv("mtcars.csv")
mtcars
mtcars =  mtcars.rename(columns={'Unnamed: 0': 'model'})
mtcars
```

Remove unnecessary data

```
del mtcars["model"]
mtcars.head()
```

Measure of Central Tendency

Mean

```
mtcars.mean()
mtcars.mean(axis=1)
```

Median

```
mtcars.median()
mtcars.median(axis=1)
```

Mode

```
mtcars.mode()
```

Measure of Spread

Range

```
max(mtcars["mpg"]) - min(mtcars["mpg"])
```

```
23.5
```

Variance

```
mtcars["mpg"].var()
```

```
36.32410282258064
```

Standard Deviation

```
mtcars["mpg"].std()
```

```
6.026948052089104
```

Measure of Shape

Skewness

```
mtcars["mpg"].skew()
  0.6723771376290805
```

Kurtosis

```
mtcars["mpg"].kurt()
```

```
-0.0220062914240855
```

Result:

Mean:

```
mpg      20.090625
cyl       6.187500
disp    230.721875
hp      146.687500
drat      3.596563
wt        3.217250
qsec     17.848750
vs        0.437500
am        0.406250
gear      3.687500
carb      2.812500
dtype: float64
```

Median:

```
mpg      19.200
cyl       6.000
disp    196.300
hp      123.000
drat      3.695
wt        3.325
qsec     17.710
vs        0.000
am        0.000
gear      4.000
carb      2.000
dtype: float64
```

Experiment 7: Building a Simple Linear Regression Model using Scikit Learn

AIM: To build a simple linear regression model using Scikit learn library

Algorithm:

1. Import all the required python libraries
2. Import Dataset
3. View the dataset
4. Remove unnecessary columns
5. Reshape the dataset
6. Divide dataset into training set and testing set
7. Import linear regression class
8. Create an object of the linear regression class
9. Fitting the data
10. Predicting the output

Program :

```
import warnings
warnings.simplefilter("ignore")
import numpy as np
import pandas as pd
dataset = pd.read_csv("Admission_Predict_Ver1.1.csv")
dataset
dataset = dataset.drop(['Serial No.','TOEFL Score','University
Rating','SOP','LOR','CGPA','Research'],axis=1)
dataset
x = dataset.iloc[:,0].values.reshape(-1,1)
y = dataset.iloc[:,-1].values.reshape(-1,1)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.2,random_state=0)
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train,y_train)
y_pred = lm.predict(x_test)
```

Result:

```
array([[0.73841192],[0.76942347], [0.76942347],[0.84178376],
       [0.56267979],
       [0.69706318],
       [0.53166823],
       [0.57301697]])
```

Note : This is the sample output. The output we displayed is the predicted probability of getting admissio n. Students are expected to compare the actual test set output with the predicted output to appreciate predic tion model

Experiment 8: Building a Multiple Linear Regression Model using Scikit Learn

AIM: To build a Multiple linear regression model using Scikit learn library

Algorithm:

1. Import all the required python libraries
2. Import Dataset
3. View the dataset
4. Remove unnecessary columns
5. Reshape the dataset
6. Divide dataset into training set and testing set
7. Import linear regression class
8. Create an object of the linear regression class
9. Fitting the data
10. Predicting the output

Program :

```
import warnings
warnings.simplefilter("ignore")
import numpy as np
import pandas as pd
dataset = pd.read_csv("Admission_Predict_Ver1.1.csv")
dataset
dataset = dataset.drop(['Serial No.', axis=1)
dataset
x = dataset.iloc[:,0].values.reshape(-1,1)
y = dataset.iloc[:,-1].values.reshape(-1,1)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.2,random_state=0)
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train,y_train)
y_pred = lm.predict(x_test)
```

Result:

```
array([[0.73841192],
       [0.76942347],
       [0.76942347],
       [0.84178376],
       [0.56267979],
       [0.69706318],
       [0.53166823],
       [0.57301697]])
```

Note : This is the sample output. The output we displayed is the predicted probability of getting admissio n. Students are expected to compare the actual test set output with the predicted output to appreciate predic tion mode

Experiment 10: Building an Image recognition model using SVM and PCA

AIM: To build an Image recognition model using SVM and PCA

Algorithm:

1. Import required libraries
2. Assign directories for dataset
3. Read Images
4. View the Output images
5. Convert Images to gray scale image
6. Resize the images
7. Flatten the images
8. Stack the images
9. Convert the dataset into Data frame
10. Add label to the flatten images
11. Perform the same for other set of images
12. Merge all the three sets
13. Save the file
14. Identify the dependent and independent data
15. Divide the dataset into training set and testing set
16. Import PCA model
17. Fit the PCA model with independent data
18. Extract Eigen components
19. Fit data into support vector machines model
20. Predict on new images
21. Visualize the images

Program :

#Import required libraries

```
import warnings
warnings.simplefilter('ignore')
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from skimage.io import imread, imshow
from skimage.transform import resize
from skimage import exposure
from skimage.color import rgb2gray
%matplotlib inline
```

 # Print the working directory

```
print(os.getcwd())
```

Set the working directory

```
rajini=os.listdir("C:/Users/naveen/Documents/Image
recognition/rajinikanth")
vijay=os.listdir("C:/Users/naveen/Documents/Image recognition/Vijay")
dhanush=os.listdir("C:/Users/naveen/Documents/Image recognition/dhanush")
```

 # Read Images

```
limit=100
vijay_images=[None]*limit
j=0
for i in vijay:
```

```
    if(j<limit):
        vijay_images[j]=imread("C:/Users/naveen/Documents/Image
recognition/Vijay/"+i)
        j+=1
    else:
        break

dhanush_images=[None]*limit
j=0
for i in dhanush:
    if(j<limit):
        dhanush_images[j]=imread("C:/Users/naveen/Documents/Image
recognition/dhanush/"+i)
        j+=1
    else:
        break

rajini_images=[None]*limit
j=0
for i in rajini:
    if(j<limit):
        rajini_images[j]=imread("C:/Users/naveen/Documents/Image
recognition/rajinikanth/"+i)
        j+=1
    else:
        break
```

 # View the images

```
imshow(rajini_images[1])
imshow(dhanush_images[1])
imshow(vijay_images[10])
```

 # Convert image to gray scale

```
dhanush_gray=[None]*limit
j=0
for i in dhanush:
    if(j<limit):
        dhanush_gray[j]=rgb2gray(dhanush_images[j])
        j+=1
    else:
        break

rajinikanth_gray=[None]*limit
j=0
for i in rajini:
    if(j<limit):
        rajinikanth_gray[j]=rgb2gray(rajini_images[j])
        j+=1
    else:
        break



vijay_gray=[None]*limit
j=0
for i in vijay:
    if(j<limit):
        vijay_gray[j]=rgb2gray(vijay_images[j])
        j+=1
    else:
```

```
        break
```

 # View grayscale image

```
imshow(vijay_gray[1])
```

 # Resize the image

```
for j in range (100):
    f=rajinikanth_gray[j]
    rajinikanth_gray[j]=resize(f,(512,512))

for j in range (100):
    g=dhanush_gray[j]
    dhanush_gray[j]=resize(g,(512,512))

for j in range (100):
    k=vijay_gray[j]
    vijay_gray[j]=resize(k,(512,512))
```

 # View resized images

```
imshow(vijay_gray[1])
vijay_gray[1].shape
```

 # Flatten the image

```
len_of_images=len(rajinikanth_gray)
len_of_images
image_size=(512,512)
flatten_size=image_size[0]*image_size[1]
flatten_size
for i in range(len_of_images):

rajinikanth_gray[i]=np.ndarray.flatten(rajinikanth_gray[i]).reshape(flatten
_size)
rajinikanth_gray=np.dstack(rajinikanth_gray)
rajinikanth_gray
rajinikanth_gray.shape
rajinikanth_gray=np.rollaxis(rajinikanth_gray,axis=2,start=0)
rajinikanth_gray=rajinikanth_gray.reshape(len_of_images,flatten_size)
rajinikanth_gray.shape
```

 # Convert the dataset into Data frame

```
rajini_data=pd.DataFrame(rajinikanth_gray)
rajini_data
```

 # Add label to the dataset

```
rajini_data["label"]="rajinikanth"
```

Note: Perform the same for the other two sets of images (Vijay and Dhanush). The next assumes that you have created a data frame of all the three sets of images

 # Merging Data frames

```
actor_1=pd.concat([rajini_data,dhanush_data])
actor=pd.concat([actor_1,vijay_data])
actor
actor.shape
```

#Shuffle the dataset

```
from sklearn.utils import shuffle
```

```
kollywood_indexed=shuffle(actor).reset_index()
kollywood_indexed
```

 # Remove Index

```
kollywood_actors=kollywood_indexed.drop(['index'],axis=1)

kollywood_actors
```

 # Identify dependent and independent data

```
x=kollywood_actors.values[:,:-1]
y=kollywood_actors.values[:,-1]
```

 # Divide the dataset into Training set and testing set

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_sta
te=0)
```

 # Import PCA model

```
from sklearn import decomposition
```

 # Fit the PCA model with independent data

```
pca = decomposition.PCA(n_components =70, whiten=True)
pca.fit(x_train)
```

 # Extract Eigen components

```
X_train_pca = pca.transform(x_train)
X_test_pca = pca.transform(x_test)
print(X_train_pca.shape)
```

 # Fit data into support vector machines model

```
from sklearn import svm
clf = svm.SVC(C=2., gamma=0.006)
clf.fit(X_train_pca, y_train)
```

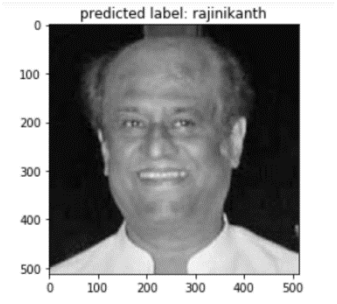
 # Perform prediction using Test set

```
y_pred = clf.predict(X_test_pca)
```

 # Visualize the predicted images

```
for i in (np.random.randint(0,60,35)):
 two_d = (np.reshape(x_test[i], (512,512)).astype(np.float64))
 plt.title('predicted label: {0}'. format(y_pred[i]))
 plt.imshow(two_d, interpolation='nearest', cmap='gray')
 plt.show()
```

Result:



Experiment 15: Implementation of Q learning in Python

AIM: To build a q learning program to solve the cartpole problem using python

Algorithm:

1. Initialize Environment:
2. Create the OpenAI Gym environment (CartPole-v1 in this case).
3. Initialize Q-Learning Agent:
4. Define a QLearning Agent class with methods for choosing actions and updating the Q-table based on rewards.
5. The Q-learning agent has parameters such as learning rate (alpha), discount factor (gamma), and exploration rate (epsilon).
6. Initialize the Q-table with zeros.
7. Training Loop:
8. For a specified number of episodes:
9. Reset the environment to the initial state.
10. Initialize the total reward for the episode to zero.
11. While the episode is not done:
12. Choose an action using epsilon-greedy policy (with exploration rate epsilon).
13. Take the chosen action and observe the next state and reward.
14. Update the Q-table using the Q-learning update equation.
15. Update the total reward for the episode.
16. Transition to the next state.
17. Print Progress:
18. Optionally, print the total reward obtained in each episode to track the agent's progress.
19. Close Environment:
20. Close the environment after training is completed.

Program :

```
import gym
import numpy as np

class QLearningAgent:
    def __init__(self, num_states, num_actions, alpha=0.1, gamma=0.99,
epsilon=0.1):
        self.num_states = num_states
        self.num_actions = num_actions
        self.alpha = alpha  # learning rate
        self.gamma = gamma  # discount factor
        self.epsilon = epsilon  # exploration rate
        self.q_table = np.zeros((num_states, num_actions))

    def choose_action(self, state):
        if np.random.uniform(0, 1) < self.epsilon:
            return np.random.randint(0, self.num_actions)  # Explore action space
        else:
            return np.argmax(self.q_table[state])  # Exploit learned values

    def update_q_table(self, state, action, reward, next_state):
        best_next_action = np.argmax(self.q_table[next_state])
        td_target = reward + self.gamma * self.q_table[next_state,
best_next_action]
        td_error = td_target - self.q_table[state, action]
        self.q_table[state, action] += self.alpha * td_error

# Create the CartPole environment
env = gym.make('CartPole-v1')
```

```
# Initialize the Q-learning agent
num_states = env.observation_space.shape[0]
num_actions = env.action_space.n
q_learning_agent = QLearningAgent(num_states, num_actions)

# Training the Q-learning agent
num_episodes = 1000
for episode in range(num_episodes):
    state = env.reset()
    done = False
    total_reward = 0

    while not done:
        action = q_learning_agent.choose_action(state)
        next_state, reward, done, _ = env.step(action)
        q_learning_agent.update_q_table(state, action, reward, next_state)
        total_reward += reward
        state = next_state

    if (episode + 1) % 50 == 0:
        print(f"Episode {episode + 1}/{num_episodes}, Total Reward:
{total_reward}")

env.close()
```

Result:
Prints the total reward obtained in each episode every 50 episodes.