Experiment: 1 Working with NumPy and Pandas 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
4. Importing Pandas Library
5. Creating n series in Pandas
6. Creating Data frame in Pandas
7. Data Frame Operations

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. Create a series using Pandas library
5. Create a data frame using Pandas library
6. Print the output

Program:

1.Importing & Checking version

import numpy as np

np. version. version

Output '16.5' [Based on the version in the system]

2. Array Creation in NumPy
2.1 Creating ID array

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

print(first_array)

Output [1 2 3]

2.2. Creating 2D array

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

print(second_array)

Output [[4 5 6] [7 8 9]]

2.3.Creating 3D array

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

print(third_array)

Output [[[10 11 12] [13 14 15]] [[16 17 18] [13 14 151]

2.4 Array of Zeros

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

print(zero_array)

Output [[0. 0] [0. 0.]

2.5. Array of Ones

one_array = np.ones((3,4))

print(one_array)

Output [[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)

Output 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)

Output [[11 17] [23 25]]

3.2. Transpose Operation

matrix _ transpose =np.transpose(my_matrix)

print(matrix_transpose)

Output [[11 23] [17 25]]

3.3. Determinant Operation

det = np.linalg.det(my_matrix)

print(det)

Output -115.99999999999999

3.4. Inverse Operation

inverse = np. linalg.inv(my_matrix)

inverse

Output 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, I))
art_ones
Outputarray ([[1.],
[1.],
[1.],
[1.]])

Pandas:

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)

Output

```
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

	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('ABCDE'))

random

Output:

	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

1.2. Saving a Data frame

random.to_csv('C:/Users/Admin/Documents/VIKKI 4TH/Pandas.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/Admin/Documents/VIKKI 4TH/Pandas.csv')
data

Output:

	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)

Output:

	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 Data frame

data.shape

Output:(20,5)

5.4.Get information about the Data frame

data.info()

Output:

```
<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

Output:

	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

Result:

The experiment aimed at understanding the fundamentals and application of the NumPy library in machine learning.

Experiment 2: Data Visualization using Matplotlib and Seaborn

Aim: To understand the fundamentals of Data Visualization and extracting insight using matplotlib and Seaborn

Operations:

1. Importing Matplotlib library
2. Creating Data for visualization
3. Data Visualization using Matplotlib
4. Importing Seaborn 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 pit

%matplotlib inline

2. Creating data

movies = ['Interstellar', 'Inception', 'Infinity War' ,'Dune', 'Harry Potter','Oppie','FordvsFerrari']

percentage = [9.5,8.5,9,8,9,9,7]

Output: ['Interstellar', 'Inception', 'Infinity War' ,'Dune', 'Harry Potter','Oppie','FordvsFerrari']

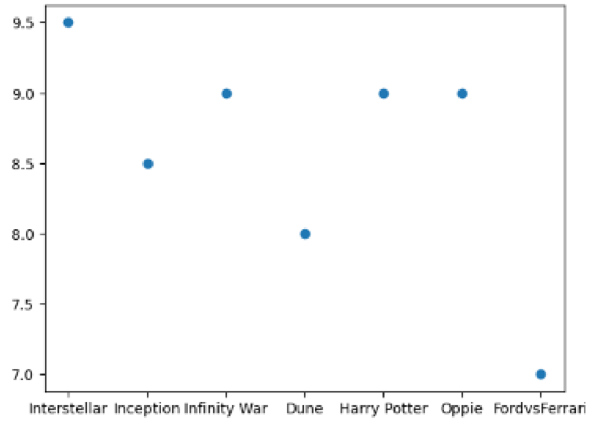
[9.5,8.5,9,8,9,9,7]

3. Data Visualization using Matplotlib

3.1. Scatter Plot

plt.scatter(movies,percentage)

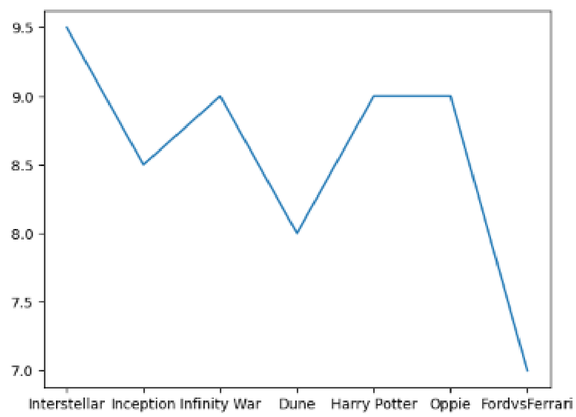
Output:



3.2 Scatter Plot

plt.plot(movies, percentage, linestyle= 'solid')

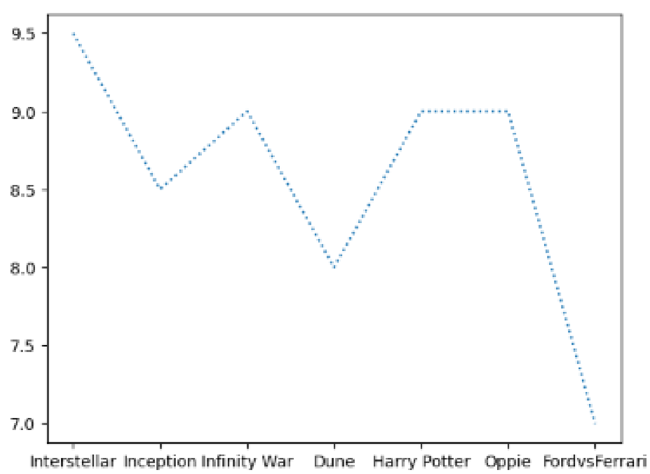
plt.show()



7.3. Line plot with dotted line

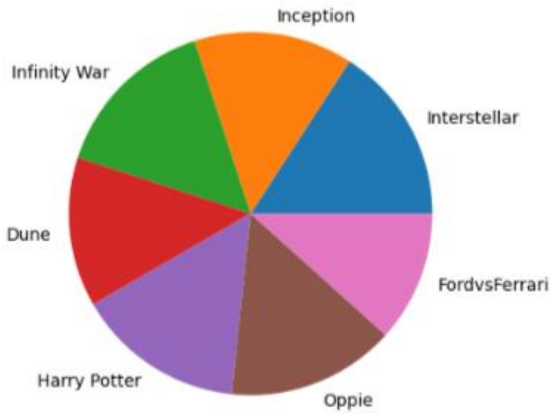
plt.plot(movies, percentage, linestyle= 'dotted')

plt.show()



3.4. Pie Chart

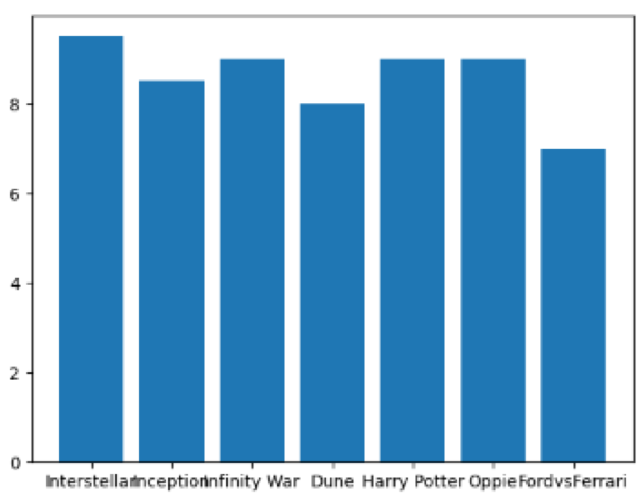
plt.pie(percentage, labels = movies)



3.4 Bar Plot

plt.bar(movies , percentage, linestyle= 'dotted')

Output:



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 pit

%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()

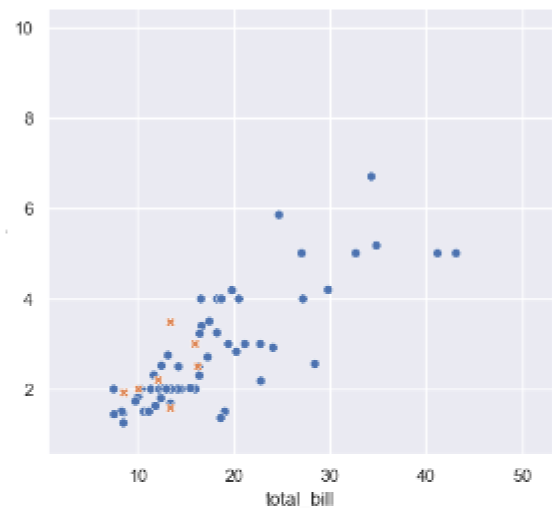
4.3. Loading tips dataset

tip=sns.load_dataset('tips')
tip
tip.tail()

	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

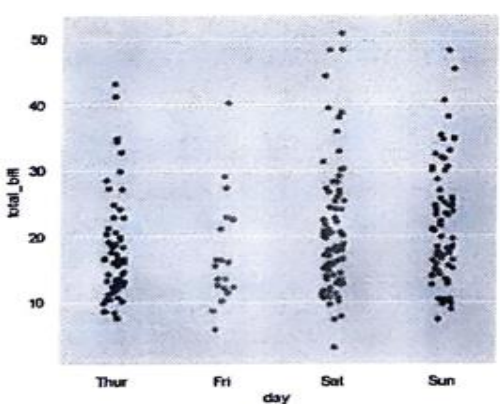
1.3. Relational Plot

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



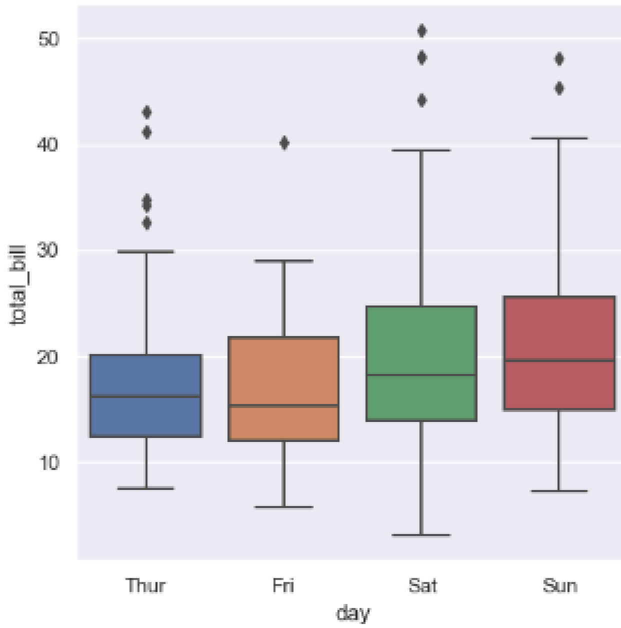
1.4. Categorical Plot

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



1.5. Box Plot

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



Result:

The experiment aimed to acquire a comprehensive understanding of data visualization fundamentals and extracting insights using Matplotlib and Seaborn.

Experiment 3: Building a Data dashboard using Google Looker studio

Aim: To build a data dashboard in Google Looker Studio

PROCEDURE/STEPS:

Step I : Open google chrome
Step 2: Sign in to the google account

Step 3: Go to looker studio.

Step 4: Click on " + Create" option

Step 5: In add data to report pop up window Click on File Upload.

Step 6: Click on "Authorize" button if prompted.

Step 7: Click on "Click to Upload File"

Step 8: Select a CSV or Excel worksheet which has the data.

Step 9: Once the file get uploaded, Click on Add button at the bottom.

Step 10: Click on "Add to report" if prompted in pop up window.

Step 11 : Click on Add chart option.

step 12: From the list, select Table to insert table.

Step 13: Check for the setup tab in the right side panel.

Step 14: In dimension and parameter select the required data.

Step 15: Click on Bar chart to insert Bar graph.

step 16: Similarly in the right side panel select the parameters in the Setup.

step 17: Click on Pie chart.

Step 18: Select the parameters for Pie chart.

Step 19: Click on tree map chart.

Step 20: Select the parameters for tree chart.

Step21: Click on scatter chart.

step 22: Select the parameters for scatter chart.

step 23: Click on Line chart.

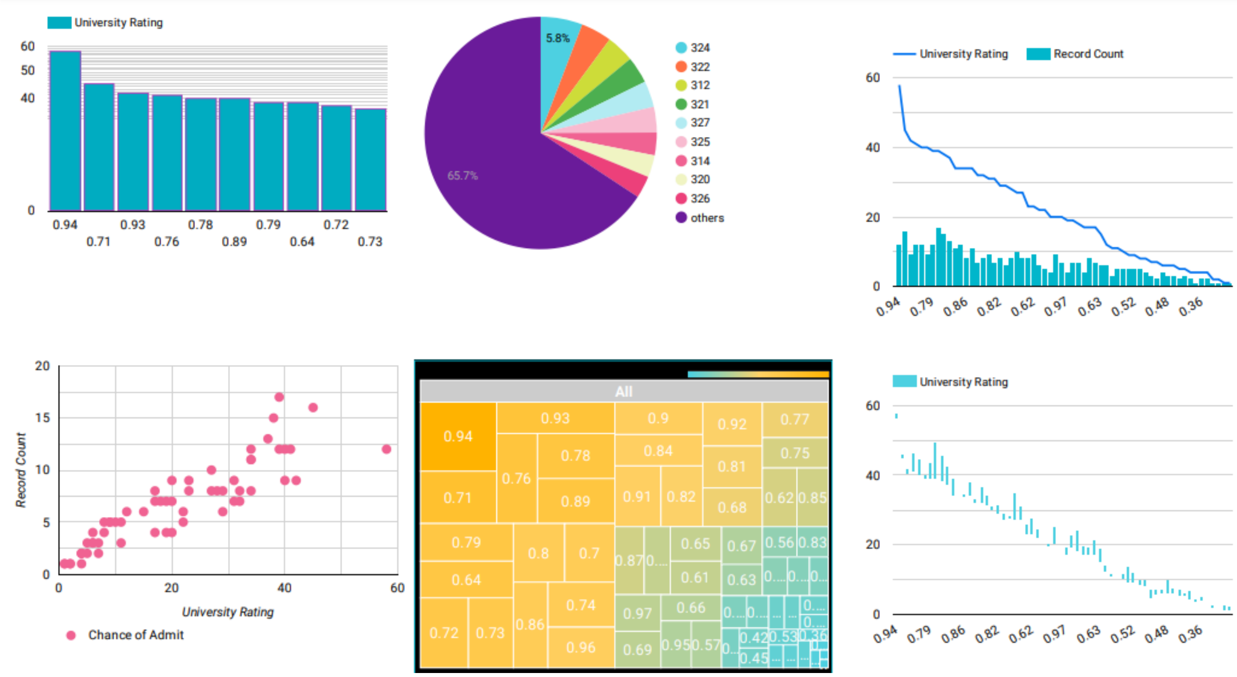
step 24: Select the parameters for Line chart.

Step 25: Click on File menu

Step 26: Click on "download as"

step 27: Select PDF format

OUTPUT:



RESULT:

The dashboard with different graphs are create using google looker studio.

Experiment 4:Data Preprocessing & Feature Scaling in Python

Aim: To Clean data and perform feature scaling

Algorithm:
1. Import required libraries & Data

2. Remove Missing values

3. Handle Categorical Data

4. Feature Scaling

Importing libraries

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

Importing data

```
dataset = pd.read_csv('Data.csv')
dataset
```

Output:

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

Handling Missing data

Reshaping dataset to dataframe

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

Finding Null Element:

```
dataset.isnull().sum()
Country      0
Age          3
Salary       3
Purchased    0
dtype: int64
```

Importing Imputer Function

```
from sklearn.impute import SimpleImputer
```

Applying simple imputer

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

Filtering imputer

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

Printing filled values

```
print(x)
[['France' 44.0 72000.0]
 ['Spain' 39.142857142857146 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 63714.28571428572]
 ['Germany' 40.0 63714.28571428572]
 ['France' 35.0 58000.0]
 ['Spain' 39.142857142857146 52000.0]
 ['France' 39.142857142857146 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 63714.28571428572]]
```

Handling categorical Data

Importing Libraries

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

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

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

Applying lab encoder

```
lab_encode = LabelEncoder()

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

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

Feature Scaling

```
import pandas as pd
import numpy as np


stand_scaler= pd.DataFrame({'[x1':np.random.normal(0,2,100),
                            'x2':np.random.normal(3,5,100),
                            'x3':np.random.normal(-2,2,100)})
```

 # Instantiate MinMaxScaler

```
scaler = MinMaxScaler()
```

 # Fit the scaler to the data

```
scaler.fit(stand_scaler)
```

 # Transform the data

```
scaled_data = scaler.transform(stand_scaler)
```

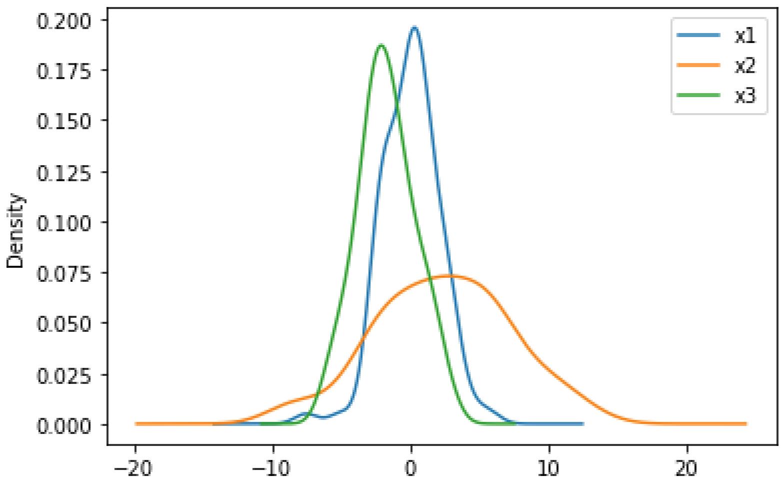
 # Convert the scaled data back to a DataFrame

```
scaled_df = pd.DataFrame(scaled_data,
columns=stand_scaler.columns)

print(scaled_df.head())
```

Plot Density Plot

stand_scaler.plot.kde()

OUTPUT:



Result:
Successfully executed data preprocessing tasks and applied featurSuccessfully executed data preprocessing tasks and applied feature scaling techniques to enhance data quality and prepare it for machine learning models.

Experiment 5: Working with Descriptive Statistics using SciPy

Aim: To perform statistical analysis on data 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 pit
%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
```

Output:

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
```

Result:

The experiment successfully executed statistical analysis on the dataset using SciPy, providing essential descriptive statistics parameters for further analysis.

Experiment: 6 Inferential Statistics and Hypothesis Test

Aim:

To Show the data Inferential Statistics and Hypothesis test using scipy

Algorithm:

1. Import Libraries
2. Point Interval Estimation:
3. Confidence Interval
4. Hypothesis Test
5. One-Tailed T-Test
6. Two-Tailed T-Test

Program:

Point Interval Estimation

```
importnumpyasnp
fromscipy.statsimportt

defmean_confidence_interval(data,confidence=0.80):
n=len(data)
mean=np.mean(data)
std_err=np.std(data,ddof=1)/np.sqrt(n)
margin_of_error=std_err*t.ppf((1+confidence)/2,n-1)
lower_bound=mean-margin_of_error
upper_bound=mean+margin_of_error
returnmean,lower_bound,upper_bound

data=[1,2,4,8,12,13,22,33,42,52]
confidence_level=0.80
mean,lower_bound,upper_bound=mean_confidence_interval(data,
confidence_level)
print(f"Mean: {mean}")
print(f"Confidence Interval ({int(confidence_level*100)}%):
[{lower_bound}, {upper_bound}]")
```

Output:

```
Mean: 18.9
Confidence Interval (80%): [11.094221522552104, 26.7057784774478
93]
```

Confidence Intervel

```
defcompare_means_and_confidence_interval(data1,data2,confidence=
0.80):

n1=len(data1)
n2=len(data2)
mean1=np.mean(data1)
mean2=np.mean(data2)
std1=np.std(data1,ddof=1)
std2=np.std(data2,ddof=1)
std_err=np.sqrt((std1**2/n1)+(std2**2/n2))
t_critical=t.ppf((1+confidence)/2,n1+n2-2)

# Hypothesis testing
t_statistic=(mean1-mean2)/std_err
reject_null=np.abs(t_statistic)>t_critical

# Confidence interval calculation
mean_diff=mean1-mean2
margin_of_error=t_critical*std_err
lower_bound=mean_diff-margin_of_error
upper_bound=mean_diff+margin_of_error

returnreject_null,(lower_bound,upper_bound)

# Example usage:
data1=[35,45,50,65,75]
data2=[25,32,44,55,66]
confidence_level=0.80
reject_null,confidence_interval=compare_means_and_confidence_int
erval(data1,data2,confidence_level)

ifreject_null:
print("Null hypothesis rejected: There is a significant
difference between the means.")
else:
print("Null hypothesis not rejected: There is no significant
difference between the means.")

print(f"Confidence Interval ({int(confidence_level*100)}%):
{confidence_interval}")
```

Output:

```
Null hypothesis not rejected: There is no significant difference
between the means.
```

```
Confidence Interval (80%): (-4.812264284107039, 24.0122642841070
4)
```

Hypothesis Test

```
def student_t_test(sample1, sample2, a=0.07):
   n1 = len(sample1)
   n2 = len(sample2)
   mean1 = np.mean(sample1)
   mean2 = np.mean(sample2)
   std1 = np.std(sample1, ddof=1)
   std2 = np.std(sample2, ddof=1)
pooled_std = np.sqrt((std1**2 / n1) + (std2**2 / n2))
t_statistic = (mean1 - mean2) / pooled_std
degrees_of_freedom = n1 + n2 - 2
p_value = 2 * (1 - t.cdf(abs(t_statistic), df=degrees_of_freedom))
reject_null = p_value< a
   return reject_null, t_statistic, p_value
sample1 = [22, 35, 20, 72, 77]
sample2 = [44, 27, 45, 10, 34]
a= 0.07
reject_null, t_statistic, p_value = student_t_test(sample1, sample2, a)
if reject_null:
   print("Reject the null hypothesis: There is a significant difference between the means.")
else:
   print("Fail to reject the null hypothesis: There is no significant difference between the
means.")
print(f"t-statistic: {t_statistic}")
print(f"p-value: {p_value}")
```

Output:

```
Fail to reject the null hypothesis: There is no significant diff
erence between the means.
t-statistic: 0.9535222907320946
p-value: 0.368244058919005
```

One Tailed T-Test

```
defone_tailed_t_test(sample,null_mean,alternative='Greater',a=0.07):

n=len(sample)
sample_mean=np.mean(sample)
sample_std=np.std(sample,ddof=1)
t_statistic=(sample_mean-null_mean)/(sample_std/np.sqrt(n))

ifalternative=='Greater':
p_value=1-t.cdf(t_statistic,df=n-1)
reject_null=p_value<a
elifalternative=='Less':
p_value=t.cdf(t_statistic,df=n-1)
reject_null=p_value<a
else:
raiseValueError("Invalid alternative hypothesis. Choose either 'greater' or 'less'.")
returnreject_null,t_statistic,p_value

sample=[10,11,18,21,24,17,47,33]
null_mean=12
alternative='Greater'
a=0.07

reject_null,t_statistic,p_value=one_tailed_t_test(sample,null_mean,alternative,a)
ifreject_null:
print("Reject the null hypothesis: The sample mean is significantly greater than the null
mean")
else:
print("Fail to reject the null hypothesis: The sample mean is not significantly greater than
the null mean")
print(f"t-statistic: {t_statistic}")
print(f"p-value: {p_value}")
```

Output:

```
Reject the null hypothesis: The sample mean is significantly gre
ater than the null mean
t-statistic: 2.449223405732373
p-value: 0.02207876149853427
```

Two Tailed T-Test

```
deftwo_tailed_t_test(sample,null_mean,a=0.07):

n=len(sample)
sample_mean=np.mean(sample)
sample_std=np.std(sample,ddof=1)
t_statistic=(sample_mean-null_mean)/(sample_std/np.sqrt(n))
degrees_of_freedom=n-1

p_value=2*(1-t.cdf(abs(t_statistic),df=degrees_of_freedom))
reject_null=p_value<a
returnreject_null,t_statistic,p_value
sample=[10,14,15,20,22,17,45,21]
null_mean=12
a=0.07

reject_null,t_statistic,p_value=two_tailed_t_test(sample,null_me
an,a)

ifreject_null:
print("Reject the null hypothesis: The sample mean is
significantly different from the null mean.")
else:
print("Fail to reject the null hypothesis: The sample mean is
not significantly different from the null mean.")

print(f"t-statistic: {t_statistic}")
print(f"p-value: {p_value}")
```

Output:

```
Reject the null hypothesis: The sample mean is significantly dif
ferent from the null mean.
t-statistic: 2.2517050070105746
p-value: 0.05904942200368035
```

Result:

The experiment successfully showcased the application of inferential statistics and hypothesis testing techniques using the SciPy library.

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_Verl . 1. csv")
dataset

dataset=dataset.drop(['Serial
No.','TOEFLScore','UniversityRating','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)
```

Output:

```
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 ofgetting admission. Students are expected to compare the actual test set output with the predicted output to appreciate prediction model

Result:

The experiment achieved its aim by successfully constructing a simple linear regression model using the Scikit-learn library.

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

Aim: To build a Multiple linear regression model using Scikit 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:

```
importwarnings
warnings.simplefilter('ignore')
import numpy as np
import pandas as pd
dataset=pd.read_csv("Admission_Predict.csv")
dataset
dataset = dataset.drop(['Serial No.'],axis=1)

x = dataset.iloc[:,:-2].values.reshape(-1,1)
y = dataset.iloc[:,-1].values

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_predict = lm.predict(x_test)
```

Output:

```
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 admission. Students are expected to compare the actual test set output with the predicted output to appreciate prediction model

Result:

Successfully implemented a multiple linear regression model using Scikit-learn, allowing for analysis of the relationship between multiple independent variables and a dependent variable

Experiment 9: Building a Logistic Regression Model in Scikit Learn

Aim: To build a Logistic regression model using Scikit learn Library

Algorithm:

1. Import libraries
2. Import Data
3. Perform Exploratory Data Analysis
4. Identify dependent and independent data
5. Divide Dataset into training and test set
6. Fit the model
7. Perform Prediction using Test set

Program:
#Import libraries

```
importnumpyasnp
importpandasaspd
importmatplotlib.pyplotasplt
from sklearn.preprocessing
import matplotlib .pyplot as plt
from sklearn.linear_model import LogisticRegression
importseabornassns
%matplotlib inline
```

#Import data
```
data—pd. read_csv ( diabetes.csv )
```

#Exploratory Data Analysis
```
data. shape
data. Columns
data info()
data [‘Outcome’].value_counts()
data.corr(method=’spearman’)
```

#Identifying dependent and independent data

```
feature_cols = [‘Pregnancies’, ‘Glucose’, ‘Blood pressure’,
‘SkinThickness’, ‘Insulin’, ‘BMI’, ‘DiabetesPedigreeFunction’,
‘Age’]
X data [feature_cols]
y data . Outcome
```

 # Dividing the dataset into training set and testing set

```
from skiearn. 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=30)
```

 # Fitting the model

```
from sklearn.linear_model import LogisticRegression
model LogisticRegression ()
model
model.fit(X_train,y_train)
```

 # Perform Prediction using test set

```
Y_predmodel.preduct(X_test)
Y_pred
```

Output :

```
array ( [0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0..])= int64
```

Result:

The experiment was successful in building a logistic regression model using the Scikit-learn library.

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

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

Algorithm:

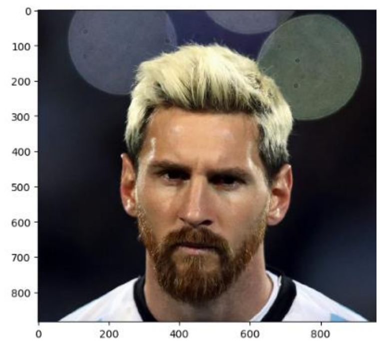
1. Import required libraries
2. Assign directories for dataset
3. Read Images
4. View the Output images
5. Convert Images to gray scale image
6. Resize the images
7. Flatten the images
8. Stack the images
9. Convert the dataset into Data frame
10. Add label to the flatten images
11. Perform the same 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 os
import warnings
warnings.simplefilter('ignore')

importnumpyasnp
importpandasaspd
importmatplotlib.pyplotasplt
%matplotlib inline
from skimage.io importimread, imshow
fromskimage.transformimport resize
fromskimage.colorimport rgb2gray


leo=os.listdir("C:\\Users\\Admin\\Documents\\VIKKI 4TH\\Cropped
DS\\leo")
```

```
maldini=os.listdir("C:\\Users\\Admin\\Documents\\VIKKI
4TH\\Cropped DS\\maldini")
david=os.listdir("C:\\Users\\Admin\\Documents\\VIKKI
4TH\\Cropped DS\\david")

limit=10
leo_img=[None]*limit
j=0
fori in leo:
    if(j<limit):
leo_img[j]=imread("C:\\Users\\Admin\\Documents\\VIKKI
4TH\\Cropped DS\\leo\\"+i)
        j+=1
else:
        break

limit=10
maldini_img=[None]*limit
j=0
foriinmaldini:
if(j<limit):
maldini_img[j]=imread("C:\\Users\\Admin\\Documents\\VIKKI
4TH\\Cropped DS\\maldini\\"+i)
        j+=1
else:
break

limit=10
david_img=[None]*limit
j=0
foriindavid:
if(j<limit):
david_img[j]=imread("C:\\Users\\Admin\\Documents\\VIKKI
4TH\\Cropped DS\\david\\"+i)
        j+=1
else:
break
imshow(leo_img[0])
```



B-3 GRAY RGB2

```
leo_gray=[None]*limit
j=0
foriinleo:
if(j<limit):
leo_gray[j]=rgb2gray(leo_img[j])
        j+=1
else:
break

david_gray=[None]*limit
j=0
foriindavid:
if(j<limit):
david_gray[j]=rgb2gray(david_img[j])
        j+=1
else:
break

maldini_gray=[None]*limit
j=0
foriinmaldini:
if(j<limit):
maldini_gray[j]=rgb2gray(maldini_img[j][:,:,:3])
        j+=1
else:
break

for j in range (10):
lm=leo_gray[j]
leo_gray[j]=resize(lm,(512,512))

for j in range (10):
    pm=maldini_gray[j]
maldini_gray[j]=resize(pm,(512,512))
```

```
for j in range (10):
db=david_gray[j]
david_gray[j]=resize(db,(512,512))
```

```
leo_gray[0].shape
 out[18]:(512, 512)
```

B-6 Find out the number of gray_scale img

For Leo

```
len_of_img_leo=len(leo_gray)
len_of_img_leo#output:10

img_size_leo=leo_gray[1].shape
img_size_leo #output:(512, 512)
```

Flatten Size

```
flatten_size_leo=img_size_leo[0]*img_size_leo[1]
flatten_size_leo#output:262144
```

```
foriin range(len_of_img_leo):

leo_gray[i]=np.ndarray.flatten(leo_gray[i]).reshape(flatten_size
_leo,1)


np.ndarray.flatten


leo_gray=np.dstack(leo_gray)
leo_gray

leo_gray.shape
(262144, 1, 10)

leo_gray=np.rollaxis(leo_gray,axis=2,start=0)
leo_gray.shape
(10, 262144, 1)

leo_data=pd.DataFrame(leo_gray)
leo_data

leo_data["Label"]="leo"
leo_data
```

For Maldini

```
len_of_img_maldini=len(maldini_gray)
len_of_img_maldini#Output: 10

img_size_maldini=maldini_gray[1].shape
img_size_maldini

flatten_size_maldini=img_size_maldini[0]*img_size_maldini[1]
flatten_size_maldini

foriinrange(len_of_img_maldini):
maldini_gray[i]=np.ndarray.flatten(maldini_gray[i]).reshape(flat
ten_size_maldini,1)

np.ndarray.flatten

maldini_gray=np.dstack(maldini_gray)
maldini_gray

maldini_gray.shape
(262144, 1, 10)

maldini_gray=np.rollaxis(maldini_gray,axis=2,start=0)
maldini_gray.shape
(10, 262144, 1)

maldini_gray=maldini_gray.reshape(len_of_img_maldini,flatten_siz
e_maldini)
maldini_gray.shape

maldini_data=pd.DataFrame(maldini_gray)
maldini_data

maldini_data["Label"]="maldini"
maldini_data
```

For David

```
len_of_img_david=len(david_gray)
len_of_img_david#Output: 10

img_size_david=david_gray[1].shape
img_size_david
```

```
flatten_size_david=img_size_david[0]*img_size_david[1]
flatten_size_david

foriinrange(len_of_img_david):
david_gray[i]=np.ndarray.flatten(david_gray[i]).reshape(flatten_
size_david,1)

np.ndarray.flatten

david_gray=np.dstack(david_gray)
david_gray

david_gray.shape
(262144, 1, 10)

david_gray=np.rollaxis(david_gray,axis=2,start=0)
david_gray.shape
(10, 262144, 1)

david_gray=david_gray.reshape(len_of_img_david,flatten_size_davi
d)
david_gray.shape

david_data=pd.DataFrame(david_gray)
david_data

david_data["Label"]="david"
david_data
```

Merge Images

```
man_1=pd.concat([leo_data,maldini_data])
man=pd.concat([man_1,david_data])
man
```

Shuffling

```
fromsklearn.utilsimportshuffle

fb_indexed=shuffle(man).reset_index()

fb_indexed

fb_man=fb_indexed.drop(['index'],axis=1)

fb_man.to_csv("Players.csv")
```

```
x =fb_man.values[:,:-1]

y =fb_man.values[:,-1]

fromsklearn.model_selectionimporttrain_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2
,random_state=0)

x_train.shape

(24, 262144)

x_test.shape

(6, 262144)
```

Decomposition

```
fromsklearnimport decomposition
pca=decomposition.PCA(n_components=20, whiten=True,
random_state=1)
```

Fitting Training Set

```
pca.fit(x_train)
```

Change Train set

```
x_train_pca=pca.transform(x_train)
x_test_pca=pca.transform(x_test)
x_train_pca.shape
(24, 20)

x_test_pca.shape
(6, 20)
```

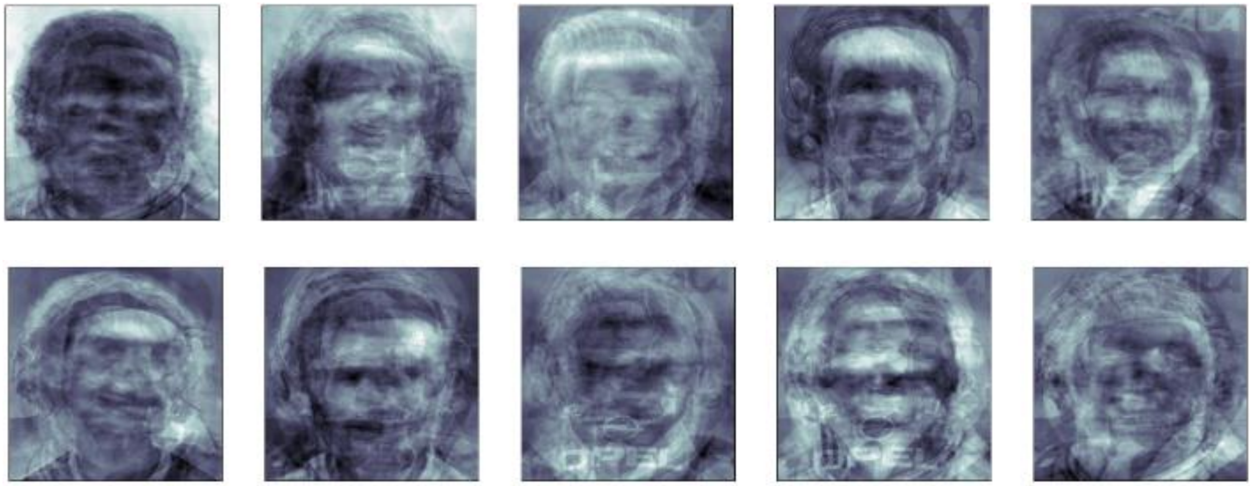
Viewing the Principle components or eigen faces

```
eigen=(np.reshape(x[10],(512,512)).astype(np.float64))
eigen
```

Plotting

```
fig=plt.figure(figsize=(30,30))
foriinrange(10):
ax=fig.add_subplot(2,5,i+1,xticks=[],yticks=[])
```

```
ax.imshow(pca.components_[i].reshape(eigen.shape),cmap=plt.cm.
bone)
```



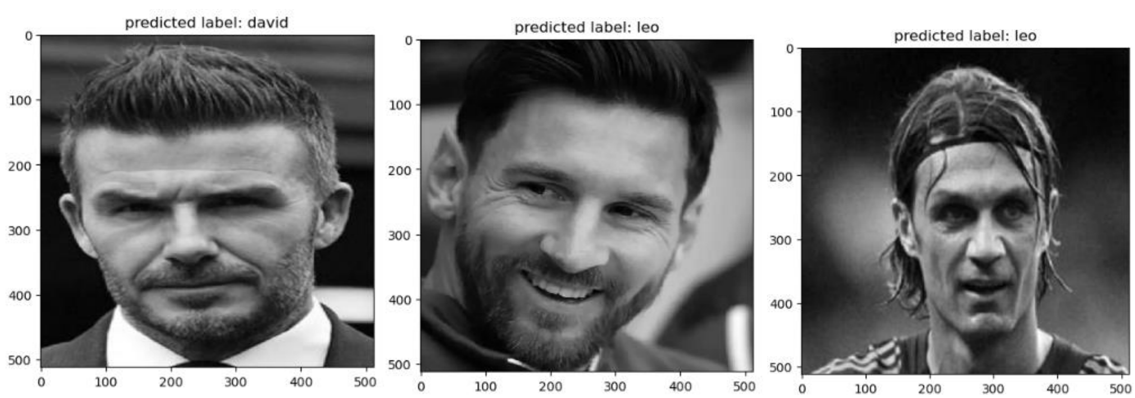
Support vector Machine Implementation

```
fromsklearnimportsvm
clf=svm.SVC(C=2,gamma=0.006,kernel='rbf')
clf.fit(x_train_pca,y_train)
```

Image Prediction

```
y_predict=clf.predict(x_test_pca)
y_predict
```

```
for in(np.random.randint(0,6,6)):
predicted_images=(np.reshape(x_test[i],(512,512)).astype
(np.float64))
plt.title('predicted label: {0}'.format(y_predict[i]))
plt.imshow(predicted_images,interpolation='nearest',cmap='gray')
plt.show()
```



```
fromsklearnimportmetrics
accuracy=metrics.accuracy_score(y_test,y_predict)
accuracy
```

Output:

```
Out[79]:  1.0
```

Result:

The experiment successfully resulted in an image recognition model using SVM and PCA, demonstrating accurate classification of images.

Experiment 11: Building an Emoji Classification model using SVM and PCA

Aim:
To build an Emojis Classification 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 os
import warnings
warnings.simplefilter('ignore')


importnumpyasnp
importpandasaspd
importmatplotlib.pyplotasplt
%matplotlib inline


from skimage.io importimread, imshow
fromskimage.transformimport resize
fromskimage.colorimport rgb2gray
```

```
smile=os.listdir("C:\\Users\\Vikneshraj\\Documents\\IV SEMESTER\\ML\\Data Emojis\\Smile")
anger=os.listdir("C:\\Users\\Vikneshraj\\Documents\\IV SEMESTER\\ML\\Data Emojis\\Anger")
sad=os.listdir("C:\\Users\\Vikneshraj\\Documents\\IV SEMESTER\\ML\\Data Emojis\\Sad")


limit=10
smile_img=[None]*limit
j=0
foriinsmile:
if(j<limit):
smile_img[j]=imread("C:\\Users\\Vikneshraj\\Documents\\IV SEMESTER\\ML\\Data Emojis\\Smile\\"+i)
j+=1
else:
break


limit=10
anger_img=[None]*limit
j=0
foriinanger:
if(j<limit):
anger_img[j]=imread("C:\\Users\\Vikneshraj\\Documents\\IV SEMESTER\\ML\\Data Emojis\\Anger\\"+i)
j+=1
else:
break


limit=10
sad_img=[None]*limit
j=0
foriinsad:
if(j<limit):
sad_img[j]=imread("C:\\Users\\Vikneshraj\\Documents\\IV SEMESTER\\ML\\Data Emojis\\Sad\\"+i)
j+=1
else:
break
imshow(smile_img[0])
```

B-3 Gray RGB2

```
smile_gray=[None]*limit
j=0
foriinsmile:
if(j<limit):
smile_gray[j]=rgb2gray(smile_img[j][:,:,:3])
j+=1
else:
break


anger_gray=[None]*limit
j=0
foriin anger:
if(j<limit):
anger_gray[j]=rgb2gray(anger_img[j][:,:,:3])
        j+=1
else:
break

sad_gray=[None]*limit
j=0
foriin sad:
if(j<limit):
sad_gray[j]=rgb2gray(sad_img[j][:,:,:3])
        j+=1
else:
break


for j in range (10):
    se=smile_gray[j]
smile_gray[j]=resize(se,(512,512))

for j in range (10):
    ae=anger_gray[j]
anger_gray[j]=resize(ae,(512,512))

for j in range (10):
    he=sad_gray[j]
sad_gray[j]=resize(he,(512,512))


smile_gray[0].shape
anger_gray[0].shape
```

```
sad_gray[0].shape
```

For Smil Emoji

```
len_of_img_smile=len(smile_gray)
len_of_img_smile
```

```
img_size_smile=smile_gray[1].shape
img_size_smile
```

Flatten

```
flatten_size_smile=img_size_smile[0]*img_size_smile[1]
flatten_size_smile
```

```
foriin range(len_of_img_smile):
smile_gray[i]=np.ndarray.flatten(smile_gray[i]).reshape(flatten_
size_smile,1)
np.ndarray.flatten
```

```
smile_gray=np.dstack(smile_gray)
smile_gray
```

```
smile_gray.shape
smile_gray=np.rollaxis(smile_gray,axis=2,start=0)
smile_gray.shape
```

```
smile_gray=smile_gray.reshape(len_of_img_smile,flatten_size_smil
e)
smile_gray.shape
```

```
smile_data=pd.DataFrame(smile_gray)
smile_data
smile_data["Label"]="smile"
smile_data
```

For Angry Emoji

```
len_of_img_anger=len(anger_gray)
len_of_img_anger
```

```
img_size_anger=anger_gray[1].shape
img_size_anger
```

```
flatten_size_anger=img_size_anger[0]*img_size_anger[1]
flatten_size_anger

foriinrange(len_of_img_anger):anger_gray[i]=np.ndarray.flatten(a
nger_gray[i]).reshape(flatten_size_anger,1)
np.ndarray.flatten

anger_gray=np.dstack(anger_gray)
anger_gray

anger_gray.shape

anger_gray=np.rollaxis(anger_gray,axis=2,start=0)
anger_gray.shape

anger_gray=anger_gray.reshape(len_of_img_anger,flatten_size_ange
r)
anger_gray.shape

anger_data=pd.DataFrame(anger_gray)
anger_data

anger_data["Label"]="anger"
anger_data
```

For Sad Emoji

```
len_of_img_sad=len(sad_gray)

len_of_img_sad

img_size_sad=sad_gray[1].shape
img_size_sad
flatten_size_sad=img_size_sad[0]*img_size_sad[1]
flatten_size_sad

foriinrange(len_of_img_sad):
sad_gray[i]=np.ndarray.flatten(sad_gray[i]).reshape(flatten_size
_sad,1)

np.ndarray.flatten

sad_gray=np.dstack(sad_gray)
```

```
sad_gray

sad_gray.shape

sad_gray=np.rollaxis(sad_gray,axis=2,start=0)
sad_gray.shape

sad_gray=sad_gray.reshape(len_of_img_sad,flatten_size_sad)
sad_gray.shape

sad_data=pd.DataFrame(sad_gray)
sad_data

sad_data["Label"]="sad"
sad_data
```

Merge Images

```
a_1=pd.concat([smile_data,sad_data])

a=pd.concat([a_1,anger_data])
a
```

Shuffling

```
fromsklearn.utilsimportshuffle
fb_indexed=shuffle(man).reset_index()
fb_indexed

fb_man=fb_indexed.drop(['index'],axis=1)

fb_man.to_csv("Emojis.csv")

x=fb_man.values[:,:-1]

y=fb_man.values[:,-1]

fromsklearn.model_selectionimporttrain_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2
,random_state=1)
x_train.shape
x_test.shape
```

Decomposition

```
fromsklearnimport decomposition
pca=decomposition.PCA(n_components=10, whiten=True,
random_state=1)
```

Fitting Training Set

```
pca.fit(x_train)
```

Change Training Set

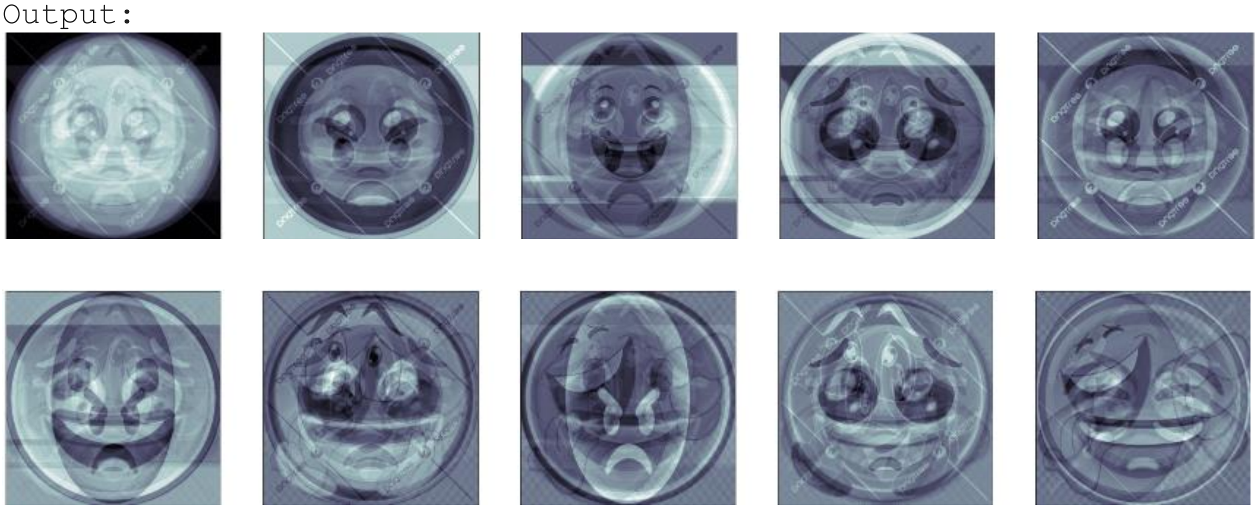
```
x_train_pca=pca.transform(x_train)
x_test_pca=pca.transform(x_test)
x_train_pca.shape
x_test_pca.shape
```

Viewing The Princeple Components or eigen

```
eigen = (np.reshape(x[10],(512,512)).astype(np.float64))
eigen
```

Plotting

```
fig =plt.figure(figsize=(30,30))
forin range(10):
    ax =fig.add_subplot(2, 5, i+1, xticks=[], yticks=[])
ax.imshow(pca.components_[i].reshape(eigen.shape),cmap=plt.cm.
bone)
Output:
```



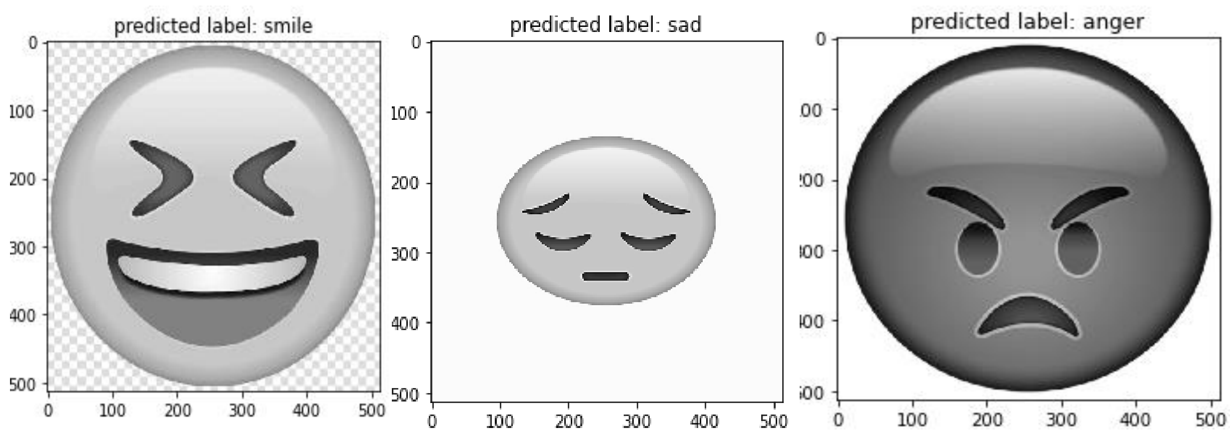
Support Vector Machine

```
From sklearn.ensembleimportRandomForestClassifier
```

```
clf=RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(x_train_pca, y_train)
```

Image Prediction

```
y_predict=clf.predict(x_test_pca)

for in (np.random.randint(0,6,6)):
predicted_images= (np.reshape(x_test[i],
(512,512)).astype(np.float64))
    plt.title('predicted label: {0}'. format(y_predict[i]))
plt.imshow(predicted_images,interpolation= 'nearest',cmap=
'gray')
    plt.show()
```




```
fromsklearnimportmetrics
accuracy=metrics.accuracy_score(y_test,y_predict)
accuracy
```

OUTPUT:

```
Out[79]:  1.0
```

Result:

The experiment successfully resulted in an image recognition model using SVM and PCA, demonstrating accurate classification of Emojis

Experiment 12: Spam Detection method using Naïve Bayes Method

Aim: To build a Classification model to detect Spam using Naïve Bayes method

Algorithm:

1. **Import required libraries**
2. **Import data**
3. **Exploratory Dato Analysis**
4. **Applying Count vectorizer**
5. **Identity dependent and independent data**
6. **Dividing cell for training and testing set**
7. **Import naive bayes classifier**
8. **Fit the data**
9. **Predict the output**
10. **Plot confusion matrix**

Program:

 # Import required libraries

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

#Import data

```
email=pd.read_csv('emails.csv')

email
```

 # Data Analysis

```
email.describe ( )
email.info ( )
spam0= email [email ['spam'] == 0]
spam1 = email[email['spam'] == 1]
sns.countplot (x = email( spam'), label
print('spam percentage =' ,(len(spam0) / len(email))*100, '%')

#spam percentage = 76.11731843575419 %
print('spam percentage =' ,(len(spam1) / len(email))*100, '%')

#spam percentage = 23.88268156424581 %
sns.countplot(x = email['spam'],label = 'spam vs spam0')
```

 # Apply Count Vectorizer

```
fromsklearn.feature_extraction.textimportCountVectorizer
vectorizer=CountVectorizer()
spam1spam0_countVectorizer=vectorizer.fit_transform(email['text'
])
print(vectorizer.get_feature_names_out())
```

```
spam1spam0_countVectorizer.shape
```

 # Identify Dependent and Independent Data

```
label = email['spam']
X = spam1spam0_CountVectorizer
Y = label
```

 # Dividing the data into training set and testing set

```
fromsklearn.model_selectionimporttrain_test_split
fromsklearn.naive_bayesimportMultinomialNB
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2
)
```

#Fit the data

```
NB_classifier=MultinomialNB()
NB_classifier.fit(x_train,y_train)
```

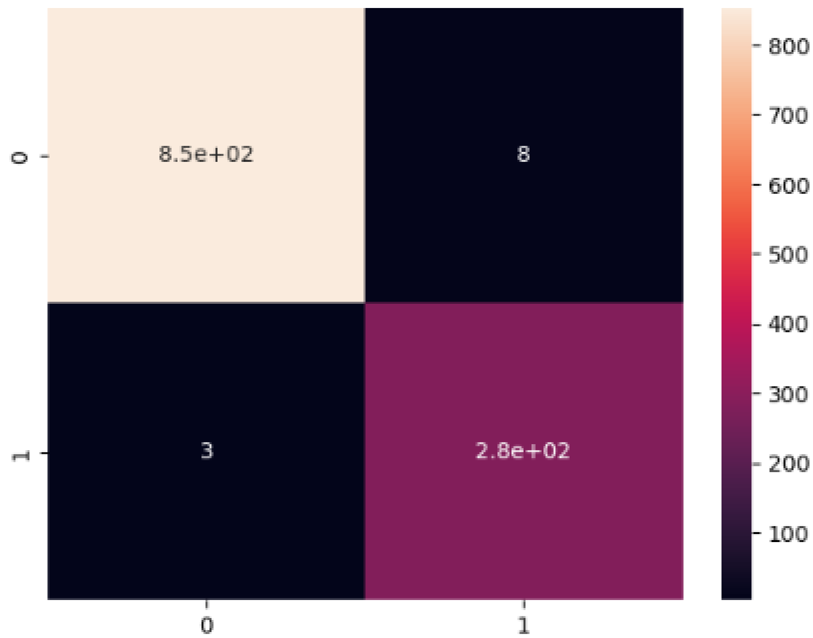
#Predict the output

```
y_predict_test=NB_classifier.predict(x_test)
y_predict_test
```

#Plot Confusion Matrix

```
cm=confusion_matrix(y_test,y_predict_test)
```

Output:
`array([0, 0, 0, ..., 1, 0, 1], dtype=int64)`



Result:

The experiment successfully built a classification model for spam detection using the Naïve Bayes method, achieving reliable performance metrics

Experiment 13: Building an Unsupervised Learning Model using Hierarchical Clustering

Aim: To build a Programthat would solveFibonacci series dynamic programming

Algorithm:

1. Import Libraries
2. DataCleaning
3. Fitting the Model
4. Visualizing Clusters

Program:

#lmport libraries

```
import pandas as pd
import numpy as np
import mat plotlib.pyplot as pit
import matplotlib.pyplot as pit
import scipy.cluster . hierarchy as sc
from s k learn import datasets
from sklearn.cluster import AgglomerativeC1ustering
```

Import dataset

```
iris=datasets.load_iris()
iris
```

#***Convert to Dataframe***

```
iris_data=pd.DataFrame(iris.data)
iris_data
```

#Removing Label fromdataset

```
iris_X=iris_data.iloc[:,[0,1,2,3]].values
iris_Y=iris_data.iloc[:,4].values
```

 # Fit the Model

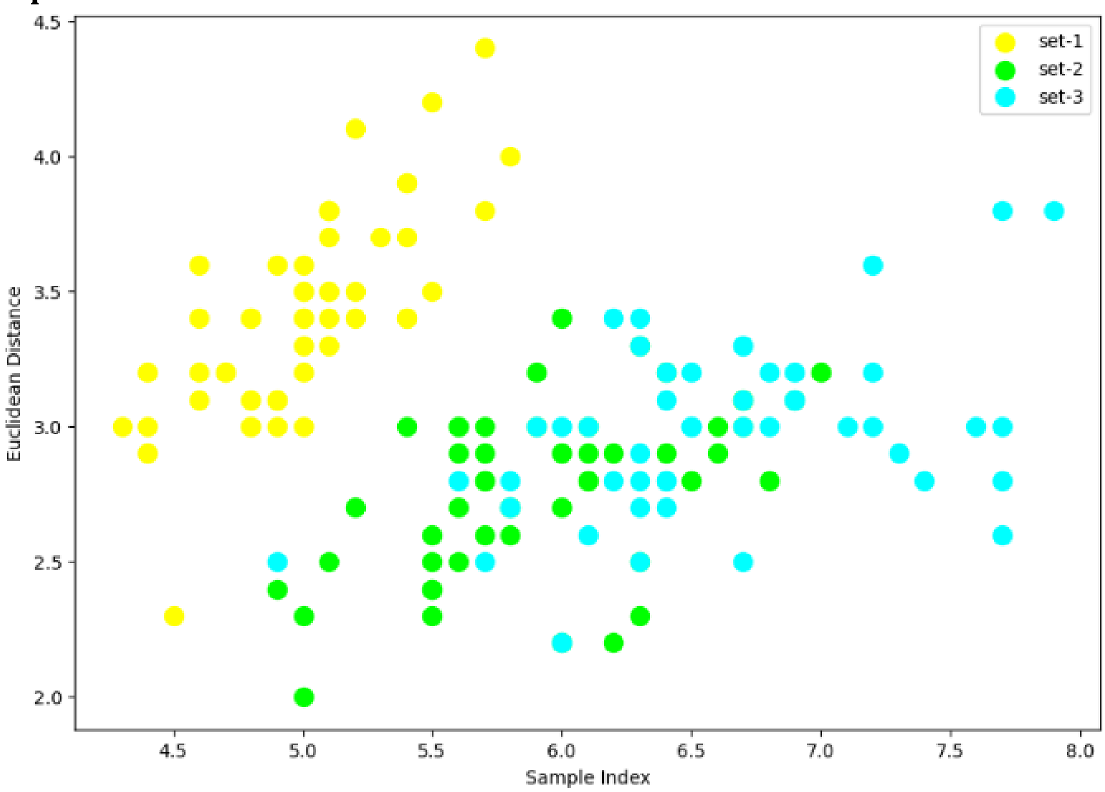
```
cluster AgglomerativeCiustering(n_clusters= 3, affinity =
'euclidean',
linkage 'ward' )
cluster. fit (iris _ X)
```

 # Print the labels
```
labels cluster. labels
labels
```

#Visuailze the classes

```
plt.figure(figsize=(10,7))
plt.scatter(iris_X[iris_Y==0,0],iris_X[iris_Y==0,1],s=100,c='yel
low',label='set-1')
plt.scatter(iris_X[iris_Y==1,0],iris_X[iris_Y==1,1],s=100,c='lim
e',label='set-2')
plt.scatter(iris_X[iris_Y==2,0],iris_X[iris_Y==2,1],s=100,c='aqu
a',label='set-3')
plt.legend()
plt.xlabel('Sample Index')
plt.ylabel('Euclidean Distance')
plt.show()
```

Output:



Result:

The program successfully implements dynamic programming to solve the Fibonacci series, demonstrating improved time complexity compared to naive recursive approaches.

Experiment 14: Building an Recommender Systems in Python

Aim: To build a Recommender System to suggest movies

Algorithm:

1. Import libraries
2. Import Data
3. Exploratory data Analysis
4. Framing Pivot Table
5. Displaying the sorted tables
6. Extracting desired movie ratings
7. Correlation
8. Viewing Recommendation

Import data

```
Import numpy  as np
Import Pandas as py
importmatplotlib.pyplotasplt
importseabornassns
sns.set_style('white')
%matplotlib inline

column_names=['user_id','item_id','rating','timestamp']
df=pd.read_csv('u.data',sep='\t',names=column_names)
df.head()
movie_titles=pd.read_csv("Movie_Id_Titles.csv")
movie_titles.head()
df=pd.merge(df,movie_titles,on='item_id')
df.head()
```

#Exploratory Data Analysis

```
df.groupby('title')['rating'].mean().sort_values
(ascending=False)
df.groupby('title')['rating'].count().sort_values
(ascending=False)
ratings =pd.DataFrame(df.groupby('title')['rating'].mean())
ratings
ratings['num of
ratings']=pd.DataFrame(df.groupby('title')['rating'].count())
ratings.head()
```

#Framing Pivot Table

```
moviemat=df.pivot_table(index='user_id',columns='title',values='
rating')
moviemat
```

#Display The sorted tables

```
ratings.sort_values('num of ratings',ascending=False).head(10)
ratings
```

#Extrating desired movie ratings

```
starwars_user_ratings=moviemat['Star Wars (1977)']
liarliar_user_ratings=moviemat['Liar Liar (1997)']
starwars_user_ratings.head(50)
```

#Correlation

```
similar_to_starwars=moviemat.corrwith(starwars_user_ratings)
similar_to_liarliar=moviemat.corrwith(liarliar_user_ratings)
```

```
#Viewing Recommendation
corr_starwars=pd.DataFrame(similar_to_starwars,columns=['correla
tion'])
corr_starwars.dropna(inplace=True)
corr_starwars.head()
corr_starwars.sort_values('correlation',ascending=False).head(50
)
corr_starwars=corr_starwars.join(ratings['num of ratings'])
corr_starwars.head()
corr_starwars[corr_starwars['num of
ratings']>100].sort_values('correlation',ascending=False).head()
```

Output:

title	correlation	num of ratings
Star Wars (1977)	1.000000	583
Empire Strikes Back, The (1980)	0.747981	367
Return of the Jedi (1983)	0.672556	507
Raiders of the Lost Ark (1981)	0.536117	420
Austin Powers: International Man of Mystery (1997)	0.377433	130

Result:

Successfully developed a recommender system in Python that effectively suggests movies based on user preferences, enhancing user experience and engagement with the platform.

Experiment 15: Implementation of Q- learning in Python

Aim: To build Q learning program to solve the cartpole problem using python

Algorithm:

1. Initialize Environment:
2. Create the OpenAI Gym environment (CartPole – v1 in cage).
3. Initialize Q-Learning Agent:
4. Define a Q-Learning Agent class With methods for choosing actions and updating the Q-table based rewards.
5. The Q-learning agent has parameters such ng 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:

```
def eps_greedy(Q, s, eps=0.1):

ifnp.random.uniform(0,1) < eps:

returnnp.random.randint(Q.shape[1])

else:

return greedy(Q, s)


defgreedy (Q, s):
return np.argmax(Q[s])

defrun_episodes(env, Q, num_episodes=100, to_print=False):
```

```
tot_rew = []
    state = env.reset()

for _ inrange(num_episodes):
        done = False
game_rew = 0

whilenot done:
next_state, rew, done, _ = env.step(greedy(Q, state))

            state = next_state
game_rew += rew
if done:
                state = env.reset()
tot_rew.append(game_rew)

ifto_print:
print('Mean score: %.3f of %i games!' % (np.mean(tot_rew),
num_episodes))
returnnp.mean(tot_rew)
defQ_learning(env, lr=0.01, num_episodes=10000, eps=0.3,
gamma=0.95, eps_decay=0.00005):
nA = env.action_space.n
nS = env.observation_space.n

    Q = np.zeros((nS, nA))
games_reward = []
test_rewards = []

for ep inrange(num_episodes):
        state = env.reset()
tot_rew = 0

if eps >0.01:
            eps -= eps_decay

        done = False
whilenot done:
            action = eps_greedy(Q, state, eps)
next_state, rew, done, _ = env.step(action)

            Q[state][action] = Q[state][action] + lr * (rew +
gamma * np.max(Q[next_state]) - Q[state][action])

            state = next_state
tot_rew += rew
if done:
```

```
games_reward.append(tot_rew)

if (ep % 300) == 0:
test_rew = run_episodes(env, Q, 1000)
print("Episode:{:5d} Eps:{:2.4f} Rew:{:2.4f}".format(ep, eps,
test_rew))
test_rewards.append(test_rew)

return Q


if __name__ == '__main__':
   env = gym.make('Taxi-v3')
print("Q-Learning")
Q_learning = Q_learning(env, lr=.1, num_episodes= 5000, eps= 0.4
, gamma = 0.95, eps_decay=0.001)
```

Output

```
Q-Learning
Episode:    0 Eps:0.3990 Rew:-241.3190
Episode:  300 Eps:0.0990 Rew:-212.2510
Episode:  600 Eps:0.0100 Rew:-227.5580
Episode:  900 Eps:0.0100 Rew:-190.4110
Episode: 1200 Eps:0.0100 Rew:-119.1710
Episode: 1500 Eps:0.0100 Rew:-73.5610
Episode: 1800 Eps:0.0100 Rew:-54.2760
Episode: 2100 Eps:0.0100 Rew:-21.2480
Episode: 2400 Eps:0.0100 Rew:-5.1300
Episode: 2700 Eps:0.0100 Rew:0.1800
Episode: 3000 Eps:0.0100 Rew:4.6000
Episode: 3300 Eps:0.0100 Rew:2.9400
Episode: 3600 Eps:0.0100 Rew:7.8860
Episode: 3900 Eps:0.0100 Rew:7.9900
Episode: 4200 Eps:0.0100 Rew:7.8780
Episode: 4500 Eps:0.0100 Rew:7.7900
Episode: 4800 Eps:0.0100 Rew:7.9870
```

Result:

The Q-learning program to solve the CartPole problem using Python has been successfully verified.