In [406…

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

In [408…

```
from sklearn.datasets import load_digits
```

In [410…

```
digits = load_digits()
```

In [412…

```
digits.keys()
```

Out[412…

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

In [414…

```
print(digits['DESCR'])
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

:Number of Instances: 1797
:Number of Attributes: 64
:Attribute Information: 8x8 image of integer pixels in the range 0..16.
:Missing Attribute Values: None
:Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
:Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

|details-start|
**References**
|details-split|

- C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
  Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
  Graduate Studies in Science and Engineering, Bogazici University.
- E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
- Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
  Linear dimensionalityreduction using relevance weighted LDA. School of
  Electrical and Electronic Engineering Nanyang Technological University.
  2005.
- Claudio Gentile. A New Approximate Maximal Margin Classification
  Algorithm. NIPS. 2000.

|details-end|
```

In [416…

```
# Get the target values
print(digits['target_names'])
```

```
[0 1 2 3 4 5 6 7 8 9]
```

In [418…

```
print(digits['feature_names'])
```

```
['pixel_0_0', 'pixel_0_1', 'pixel_0_2', 'pixel_0_3', 'pixel_0_4', 'pixel_0_5', 'pixel_0_6', 'p
ixel_0_7', 'pixel_1_0', 'pixel_1_1', 'pixel_1_2', 'pixel_1_3', 'pixel_1_4', 'pixel_1_5', 'pixe
l_1_6', 'pixel_1_7', 'pixel_2_0', 'pixel_2_1', 'pixel_2_2', 'pixel_2_3', 'pixel_2_4', 'pixel_2
_5', 'pixel_2_6', 'pixel_2_7', 'pixel_3_0', 'pixel_3_1', 'pixel_3_2', 'pixel_3_3', 'pixel_3_
4', 'pixel_3_5', 'pixel_3_6', 'pixel_3_7', 'pixel_4_0', 'pixel_4_1', 'pixel_4_2', 'pixel_4_3',
'pixel_4_4', 'pixel_4_5', 'pixel_4_6', 'pixel_4_7', 'pixel_5_0', 'pixel_5_1', 'pixel_5_2', 'pi
xel_5_3', 'pixel_5_4', 'pixel_5_5', 'pixel_5_6', 'pixel_5_7', 'pixel_6_0', 'pixel_6_1', 'pixel
_6_2', 'pixel_6_3', 'pixel_6_4', 'pixel_6_5', 'pixel_6_6', 'pixel_6_7', 'pixel_7_0', 'pixel_7_
1', 'pixel_7_2', 'pixel_7_3', 'pixel_7_4', 'pixel_7_5', 'pixel_7_6', 'pixel_7_7']
```

In [420…

```
# store feature and target in variables
feature = digits.data
target = digits.target
```

In [422…

```
from sklearn.model_selection import train_test_split
```

In [424…

```
df=pd.DataFrame(digits['data'],columns=digits['feature_names'])
```

In [426…

```
df.head()
```

Out[426…

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	

5 rows × 64 columns

In [428…

```
df.tail()
```

Out[428…

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	p
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	

5 rows × 64 columns

In [430…

```
X_train, X_test, y_train, y_test = train_test_split(df, target, train_size = 0.7, random_stat
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1257, 64)
(540, 64)
(1257,)
(540,)
```

Apply Algorithm

In [433…
```
from sklearn.linear_model import LogisticRegression
```

In [435…
```
my_model = LogisticRegression()
```

In [437…
```
my_model.fit(X_train,y_train)
```

Out[437…
```
▾ LogisticRegression

LogisticRegression()
```

In [439…
```
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [441…
```
preds = my_model.predict(X_test)
print(accuracy_score(y_test, preds))
```

```
0.9537037037037037
```

Apply PCA

In [444…
```
from sklearn.decomposition import PCA
```

In [446…
```
pca=PCA(n_components=25)
x_pca=pca.fit_transform(df)
```

In [447…
```
df.shape
```

Out[447…
```
(1797, 64)
```

In [450…
```
x_pca.shape
```

Out[450…
```
(1797, 25)
```

In [452…
```
x_pca
```

Out[452…
```
array([[ -1.25946711,  21.27488394,  -9.46305472, ...,   2.04085638,
          1.29293541,   1.2345628 ],
       [  7.95761077, -20.76869848,   4.43950564, ...,   0.85412705,
         -2.98820696,   5.32413317],
       [  6.99192421,  -9.9559875 ,   2.95855912, ...,   5.94517127,
          0.16213048,  -3.16464772],
       ...,
       [ 10.8012828 ,  -6.96025174,   5.5995542 , ...,   0.20073236,
          3.73814914,  -5.42350733],
       [ -4.87209984,  12.42395347, -10.17086634, ...,   3.1077846 ,
         -3.73072434,   5.0947119 ],
       [ -0.34438881,   6.36554845,  10.77370935, ...,   2.05668318,
         -0.13813437,  -4.19277817]])
```

In [454…

```
explained_variance = np.var(x_pca, axis=0)
print(explained_variance)
```

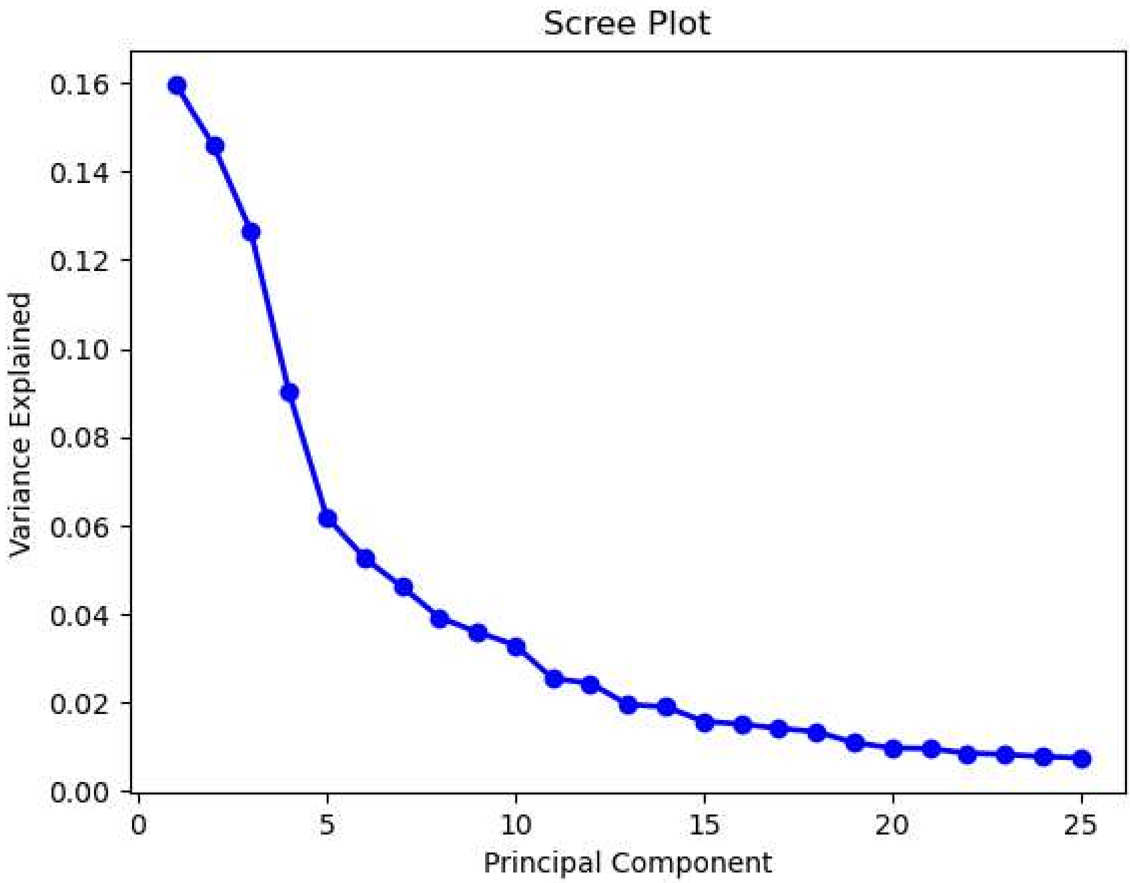
```
[178.90731578 163.62664073 141.70953623 101.04411456  69.47448269
  59.07563199  51.85566624  43.990613    40.2885629   36.99120196
  28.50317072  27.30596589  21.8892994   21.31248688  17.6268809
  16.9374175   15.84247909  14.99607398  12.22732907  10.88050117
  10.68748534   9.57644217   9.21971809   8.68488423   8.35826445]
```

In [456…

```
explained_variance_ratio = explained_variance / np.sum(explained_variance)
```

In [458…

```
PC_values = np.arange(pca.n_components) + 1
plt.plot(PC_values, explained_variance_ratio, 'o-', linewidth=2, color='blue')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.show()
```



In [460…

```
X_train, X_test, y_train, y_test = train_test_split(x_pca, target, train_size = 0.7, random_s
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1257, 25)
(540, 25)
(1257,)
(540,)
```

In [462…

```
my_model.fit(X_train,y_train)
```

Out[462…

```
▾ LogisticRegression
LogisticRegression()
```

In [464…

```
my_model_preds = my_model.predict(X_test)
print(accuracy_score(y_test, preds))
```

```
0.9537037037037037
```

In [466…

```
# Lists to store the results
accuracy_results = []
n_components_list = list(range(1, 60))
```

In [468…

```
# Lists to store the results
for n in n_components_list:
    pca = PCA(n_components=n)
    x_pca = pca.fit_transform(df)

    X_train, X_test, y_train, y_test = train_test_split(x_pca, target, train_size=0.7, random_

    my_model.fit(X_train, y_train)
    my_model_preds = my_model.predict(X_test)

    accuracy_results.append(accuracy_score(y_test, my_model_preds))
```

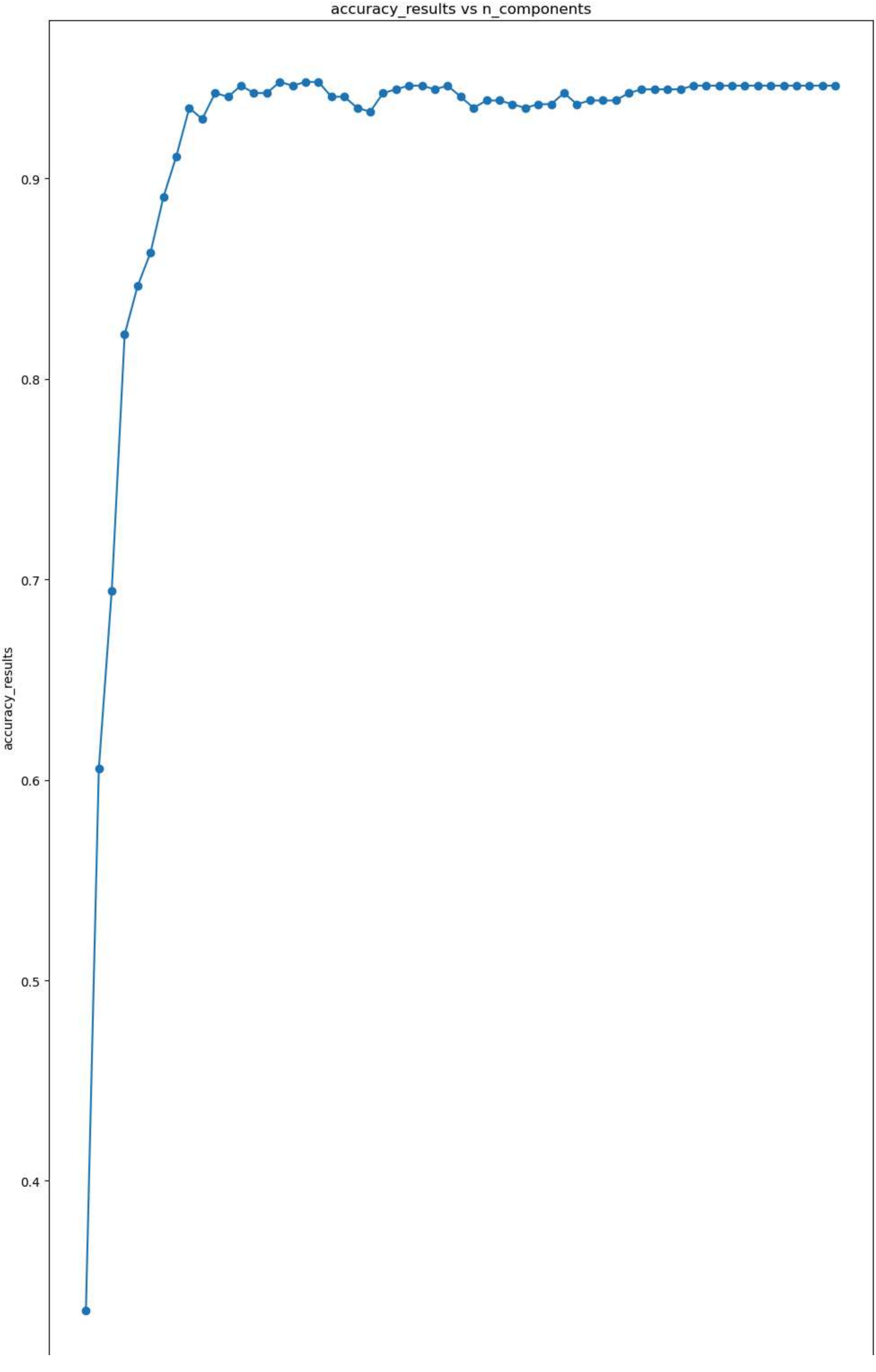
In [470…

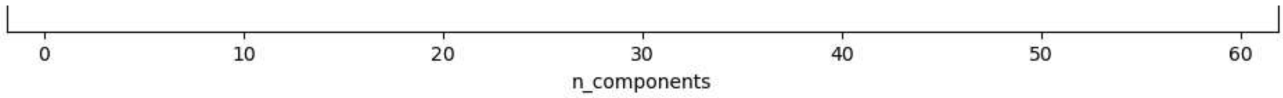
```
import numpy as np
import matplotlib.pyplot as plt

# Plotting the results in a 1-column, 4-row layout
plt.figure(figsize=(10, 16))  # Adjust the figure size for better display

# MAE plot
plt.plot(n_components_list, accuracy_results, marker='o')
plt.title('accuracy_results vs n_components')
plt.xlabel('n_components')
plt.ylabel('accuracy_results')

# Adjust layout to avoid overlap
plt.tight_layout()
plt.show()
```


accuracy_results vs n_components
accuracy_results
0.9
0.8
0.7
0.6
0.5
0.4



In [472…

```
best_n_accuracy = n_components_list[np.argmax(accuracy_results)]
best_accuracy = max(accuracy_results)

# Printing the result
print(f"Best n_components for Accuracy: {best_n_accuracy} with Accuracy = {best_accuracy}")
```

```
Best n_components for Accuracy: 16 with Accuracy = 0.9481481481481482
```

In [ ]: