```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
from sklearn.datasets import load_digits
digitsdataset = load_digits()
```

```
digitsdataset
```

```
          [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
           [ 0.,  0.,  0., ...,  9.,  0.,  0.],
           [ 0.,  0.,  3., ...,  6.,  0.,  0.],
           ...,
           [ 0.,  0.,  1., ...,  6.,  0.,  0.],
           [ 0.,  0.,  1., ...,  6.,  0.,  0.],
           [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

          [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
           [ 0.,  0.,  3., ..., 14.,  0.,  0.],
           [ 0.,  0.,  8., ..., 16.,  0.,  0.],
           ...,
           [ 0.,  9., 16., ...,  0.,  0.,  0.],
           [ 0.,  3., 13., ..., 11.,  5.,  0.],
           [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

          ...,

          [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
           [ 0.,  0., 13., ...,  2.,  1.,  0.],
           [ 0.,  0., 16., ..., 16.,  5.,  0.],
           ...,
           [ 0.,  0., 16., ..., 15.,  0.,  0.],
           [ 0.,  0., 15., ..., 16.,  0.,  0.],
           [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

          [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
           [ 0.,  0., 14., ..., 15.,  1.,  0.],
           [ 0.,  4., 16., ..., 16.,  7.,  0.],
           ...,
           [ 0.,  0.,  0., ..., 16.,  2.,  0.],
           [ 0.,  0.,  4., ..., 16.,  2.,  0.],
           [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

          [[ 0.,  0., 10., ...,  1.,  0.,  0.],
           [ 0.,  2., 16., ...,  1.,  0.,  0.],
           [ 0.,  0., 15., ..., 15.,  0.,  0.],
           ...,
           [ 0.,  4., 16., ..., 16.,  6.,  0.],
           [ 0.,  8., 16., ..., 16.,  8.,  0.],
           [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),
 'DESCR': ".. _digits_dataset:\n\nOptical recognition of handwritten digits dataset\n--------------------------------------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 1797\n    :Number of Attributes: 64\n    :Attribute Information: 8x8 image of integer pixels in the range 0..16.\n    :Missing Attribute Values: None\n    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n    :Date: July; 1998\n\nThis is a copy of the test set of the UCI ML hand-written digits datasets\nhttps://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits\n\nThe data set contains images of hand-written digits: 10 classes where\neach class refers to a digit.\n\nPreprocessing programs made available by NIST were used to extract\nnormalized bitmaps of handwritten digits from a preprinted form. From a\ntotal of 43 people, 30 contributed to the training set and different 13\nto the test set. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x4 and the number of on pixels are counted in each block. This generates\nan input matrix of 8x8 where each element is an integer in the range\n0..16. This reduces dimensionality and gives invariance to small\ndistortions.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,\n1994.\n\n.. topic:: References\n\n  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their\n    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of\n    Graduate Studies in Science and Engineering, Bogazici University.\n  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.\n    Linear dimensionalityreduction using relevance weighted LDA. School of\n    Electrical and Electronic Engineering Nanyang Technological University.\n    2005.\n  - Claudio Gentile. A New Approximate Maximal Margin Classification\n    Algorithm. NIPS. 2000.\n"}
```

```
digitsdataset.target_names
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
digitsdataset.feature_names
```

```
['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_1_1',
 'pixel_1_2',
 'pixel_1_3',
 'pixel_1_4',
```

```
 'pixel_1_5',
 'pixel_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',
 'pixel_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',
```

```
df = pd.DataFrame(digitsdataset.data)
```

```
df['target'] = digitsdataset.target
```

```
df
```

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	target
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	13.0	10.0	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	0.0	...	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0	1
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0	2
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0	3
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	2.0	14.0	15.0	9.0	0.0	0.0	9
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	16.0	14.0	6.0	0.0	0.0	0
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	2.0	9.0	13.0	6.0	0.0	0.0	8
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	5.0	12.0	16.0	12.0	0.0	0.0	9
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	0.0	0.0	1.0	8.0	12.0	14.0	12.0	1.0	0.0	8

1797 rows × 65 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1797 entries, 0 to 1796
Data columns (total 65 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       1797 non-null   float64
 1   1       1797 non-null   float64
 2   2       1797 non-null   float64
 3   3       1797 non-null   float64
 4   4       1797 non-null   float64
 5   5       1797 non-null   float64
```

```
 6   6       1797 non-null   float64
 7   7       1797 non-null   float64
 8   8       1797 non-null   float64
 9   9       1797 non-null   float64
 10  10      1797 non-null   float64
 11  11      1797 non-null   float64
 12  12      1797 non-null   float64
 13  13      1797 non-null   float64
 14  14      1797 non-null   float64
 15  15      1797 non-null   float64
 16  16      1797 non-null   float64
 17  17      1797 non-null   float64
 18  18      1797 non-null   float64
 19  19      1797 non-null   float64
 20  20      1797 non-null   float64
 21  21      1797 non-null   float64
 22  22      1797 non-null   float64
 23  23      1797 non-null   float64
 24  24      1797 non-null   float64
 25  25      1797 non-null   float64
 26  26      1797 non-null   float64
 27  27      1797 non-null   float64
 28  28      1797 non-null   float64
 29  29      1797 non-null   float64
 30  30      1797 non-null   float64
 31  31      1797 non-null   float64
 32  32      1797 non-null   float64
 33  33      1797 non-null   float64
 34  34      1797 non-null   float64
 35  35      1797 non-null   float64
 36  36      1797 non-null   float64
 37  37      1797 non-null   float64
 38  38      1797 non-null   float64
 39  39      1797 non-null   float64
 40  40      1797 non-null   float64
 41  41      1797 non-null   float64
 42  42      1797 non-null   float64
 43  43      1797 non-null   float64
 44  44      1797 non-null   float64
 45  45      1797 non-null   float64
 46  46      1797 non-null   float64
 47  47      1797 non-null   float64
 48  48      1797 non-null   float64
 49  49      1797 non-null   float64
 50  50      1797 non-null   float64
 51  51      1797 non-null   float64
 52  52      1797 non-null   float64
```

```
df.describe()
```

	0	1	2	3	4	5	6	7	8	9	...	55
count	1797.0	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	...	1797.000000
mean	0.0	0.303840	5.204786	11.835838	11.848080	5.781859	1.362270	0.129661	0.005565	1.993879	...	0.206455
std	0.0	0.907192	4.754826	4.248842	4.287388	5.666418	3.325775	1.037383	0.094222	3.196160	...	0.984401
min	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
25%	0.0	0.000000	1.000000	10.000000	10.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
50%	0.0	0.000000	4.000000	13.000000	13.000000	4.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
75%	0.0	0.000000	9.000000	15.000000	15.000000	11.000000	0.000000	0.000000	0.000000	3.000000	...	0.000000
max	0.0	8.000000	16.000000	16.000000	16.000000	16.000000	16.000000	15.000000	2.000000	16.000000	...	13.000000

8 rows × 65 columns

```
x = df.drop(['target'], axis='columns')
y = df.target
```

```
x
```

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.0	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	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0

```
y
```

```
0       0
1       1
2       2
3       3
4       4
       ..
1792    9
1793    0
1794    8
1795    9
1796    8
Name: target, Length: 1797, dtype: int64
```

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x,y,test_size=0.3)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

```
def get_score(model,xtrain,xtest,ytrain,ytest):
    model.fit(xtrain,ytrain)
    return model.score(xtest,ytest)
```

```
get_score(LogisticRegression(),xtrain,xtest,ytrain,ytest)
```

```
0.9574074074074074
```

```
get_score(SVC(),xtrain,xtest,ytrain,ytest)
```

```
0.9851851851851852
```

```
get_score(RandomForestClassifier(),xtrain,xtest,ytrain,ytest)
```

```
0.9666666666666667
```

```
from sklearn.model_selection import StratifiedKFold
folds = StratifiedKFold(n_splits=5)
```

```
from sklearn.model_selection import cross_val_score
```

```
s1=cross_val_score(LogisticRegression(),x,y)
s1
```

```
array([0.92222222, 0.86944444, 0.94150418, 0.93871866, 0.89693593])
```

```
s2=cross_val_score(SVC(),x,y)
s2
```

```
array([0.96111111, 0.94444444, 0.98328691, 0.98885794, 0.93871866])
```

```
s3=cross_val_score(RandomForestClassifier(),x,y)
s3
```

```
array([0.93611111, 0.91666667, 0.95264624, 0.96935933, 0.91643454])
```

```
#LogisticRegression
np.average(s1)
```

```
0.9137650882079852
```

+ Code + Text

```
#SVC
np.average(s2)
```

```
0.9632838130609718
```

```
#RandomForestClassifier
np.average(s3)
```

```
0.938243577839678
```

dation we can see SVC is showing the best accuracy amoung other algorithms