```python
# import all necessary libraries
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split

from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

#import file and reading few lines
numbers = pd.read_csv('digit_svm.csv')
numbers.head(10)
```

```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0   
1      0       0       0       0       0       0       0       0       0   
2      1       0       0       0       0       0       0       0       0   
3      4       0       0       0       0       0       0       0       0   
4      0       0       0       0       0       0       0       0       0   
5      0       0       0       0       0       0       0       0       0   
6      7       0       0       0       0       0       0       0       0   
7      3       0       0       0       0       0       0       0       0   
8      5       0       0       0       0       0       0       0       0   
9      3       0       0       0       0       0       0       0       0   

   pixel8  ...  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  ...         0         0         0         0         0         0   
1       0  ...         0         0         0         0         0         0   
2       0  ...         0         0         0         0         0         0   
3       0  ...         0         0         0         0         0         0   
4       0  ...         0         0         0         0         0         0   
5       0  ...         0         0         0         0         0         0   
6       0  ...         0         0         0         0         0         0   
7       0  ...         0         0         0         0         0         0   
8       0  ...         0         0         0         0         0         0   
9       0  ...         0         0         0         0         0         0   
```

```
   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0
5         0         0         0         0
6         0         0         0         0
7         0         0         0         0
8         0         0         0         0
9         0         0         0         0

[10 rows x 785 columns]
```

```
numbers.shape
```

```
(42000, 785)
```

```
#checking datatype
numbers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

```
numbers.describe(percentiles = [0.05,0.10,0.25,0.50,0.75,0.90,0.99])
```

```
              label    pixel0    pixel1    pixel2    pixel3    pixel4    pixel5  \
count  42000.000000   42000.0   42000.0   42000.0   42000.0   42000.0   42000.0
mean       4.456643       0.0       0.0       0.0       0.0       0.0       0.0
std        2.887730       0.0       0.0       0.0       0.0       0.0       0.0
min        0.000000       0.0       0.0       0.0       0.0       0.0       0.0
5%         0.000000       0.0       0.0       0.0       0.0       0.0       0.0
10%        1.000000       0.0       0.0       0.0       0.0       0.0       0.0
25%        2.000000       0.0       0.0       0.0       0.0       0.0       0.0
50%        4.000000       0.0       0.0       0.0       0.0       0.0       0.0
75%        7.000000       0.0       0.0       0.0       0.0       0.0       0.0
90%        8.000000       0.0       0.0       0.0       0.0       0.0       0.0
99%        9.000000       0.0       0.0       0.0       0.0       0.0       0.0
max        9.000000       0.0       0.0       0.0       0.0       0.0       0.0

        pixel6    pixel7    pixel8  ...      pixel774      pixel775  \
count  42000.0   42000.0   42000.0  ...  42000.000000  42000.000000
mean       0.0       0.0       0.0  ...      0.219286      0.117095
std        0.0       0.0       0.0  ...      6.312890      4.633819
min        0.0       0.0       0.0  ...      0.000000      0.000000
5%         0.0       0.0       0.0  ...      0.000000      0.000000
10%        0.0       0.0       0.0  ...      0.000000      0.000000
25%        0.0       0.0       0.0  ...      0.000000      0.000000
```

```
50%        0.0      0.0      0.0  ...       0.000000      0.000000
75%        0.0      0.0      0.0  ...       0.000000      0.000000
90%        0.0      0.0      0.0  ...       0.000000      0.000000
99%        0.0      0.0      0.0  ...       0.000000      0.000000
max        0.0      0.0      0.0  ...     254.000000    254.000000

           pixel776     pixel777      pixel778      pixel779  pixel780  \
count  42000.000000  42000.00000  42000.000000  42000.000000   42000.0
mean       0.059024      0.02019      0.017238      0.002857       0.0
std        3.274488      1.75987      1.894498      0.414264       0.0
min        0.000000      0.00000      0.000000      0.000000       0.0
5%         0.000000      0.00000      0.000000      0.000000       0.0
10%        0.000000      0.00000      0.000000      0.000000       0.0
25%        0.000000      0.00000      0.000000      0.000000       0.0
50%        0.000000      0.00000      0.000000      0.000000       0.0
75%        0.000000      0.00000      0.000000      0.000000       0.0
90%        0.000000      0.00000      0.000000      0.000000       0.0
99%        0.000000      0.00000      0.000000      0.000000       0.0
max      253.000000    253.00000    254.000000     62.000000       0.0

       pixel781  pixel782  pixel783
count   42000.0   42000.0   42000.0
mean        0.0       0.0       0.0
std         0.0       0.0       0.0
min         0.0       0.0       0.0
5%          0.0       0.0       0.0
10%         0.0       0.0       0.0
25%         0.0       0.0       0.0
50%         0.0       0.0       0.0
75%         0.0       0.0       0.0
90%         0.0       0.0       0.0
99%         0.0       0.0       0.0
max         0.0       0.0       0.0

[12 rows x 785 columns]
```

```
#checking for null values
round(100*(numbers.isnull().sum()/(len(numbers.index))),2).sort_values(ascend
ing = False)
```

```
pixel783    0.0
pixel267    0.0
pixel265    0.0
pixel264    0.0
pixel263    0.0
pixel262    0.0
pixel261    0.0
pixel260    0.0
pixel259    0.0
pixel258    0.0
```

```
pixel257    0.0
pixel256    0.0
pixel255    0.0
pixel254    0.0
pixel253    0.0
pixel252    0.0
pixel251    0.0
pixel250    0.0
pixel249    0.0
pixel248    0.0
pixel247    0.0
pixel246    0.0
pixel245    0.0
pixel266    0.0
pixel268    0.0
pixel390    0.0
pixel269    0.0
pixel290    0.0
pixel289    0.0
pixel288    0.0
           ...
pixel495    0.0
pixel494    0.0
pixel493    0.0
pixel492    0.0
pixel491    0.0
pixel512    0.0
pixel513    0.0
pixel514    0.0
pixel526    0.0
pixel535    0.0
pixel534    0.0
pixel533    0.0
pixel532    0.0
pixel531    0.0
pixel530    0.0
pixel529    0.0
pixel528    0.0
pixel527    0.0
pixel525    0.0
pixel515    0.0
pixel524    0.0
pixel523    0.0
pixel522    0.0
pixel521    0.0
pixel520    0.0
pixel519    0.0
pixel518    0.0
pixel517    0.0
pixel516    0.0
```

```
label        0.0
Length: 785, dtype: float64
```

```
# let us check unique entries of label column
np.unique(numbers['label'])
```

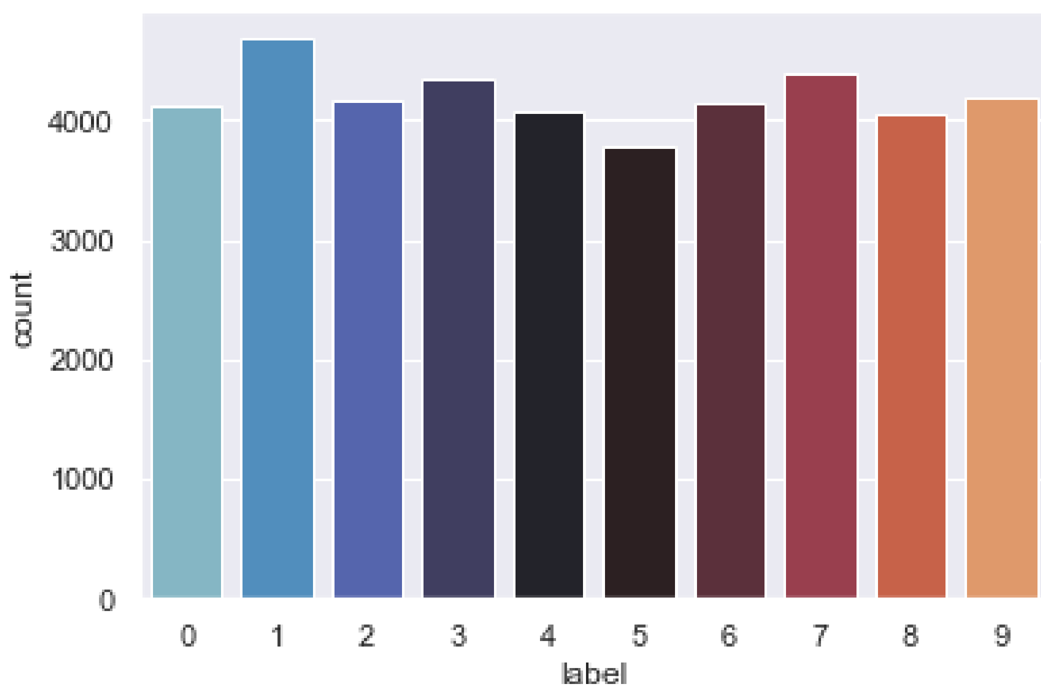
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int64)
```

```
numbers['label'].value_counts()
```

```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

```
#visualising the column - label
sns.countplot(numbers['label'],palette = 'icefire')
```

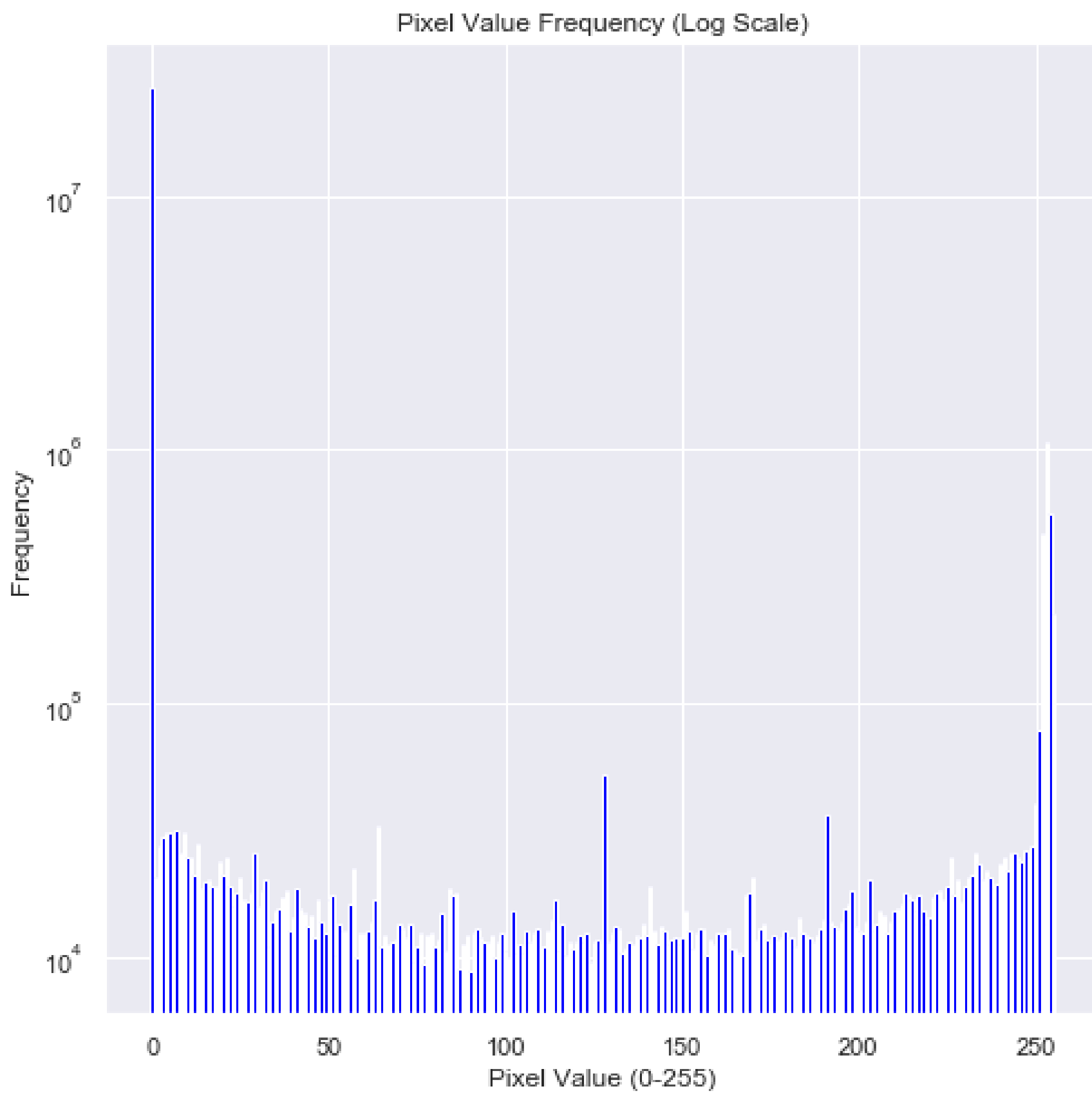
```
<matplotlib.axes._subplots.AxesSubplot at 0x2cb8f69b6a0>
```



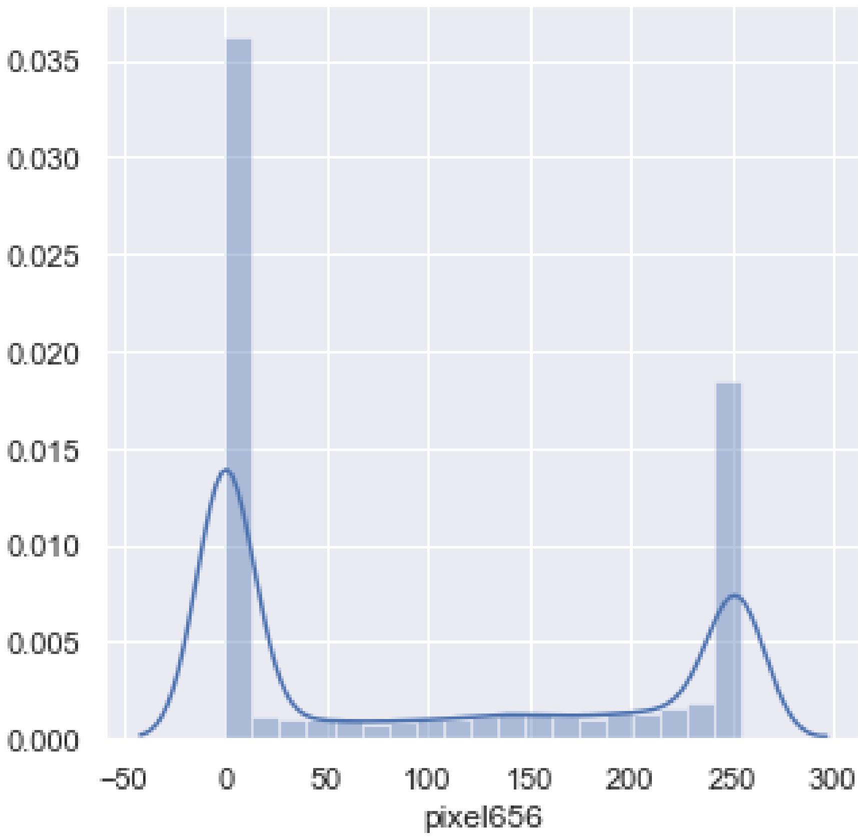

```
y = pd.value_counts(numbers.values.ravel()).sort_index()
N = len(y)
x = range(N)
```

```
width =0.9
plt.figure(figsize=[8,8])
plt.bar(x, y, width, color="blue")
plt.title('Pixel Value Frequency (Log Scale)')
plt.yscale('log')
plt.xlabel('Pixel Value (0-255)')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```

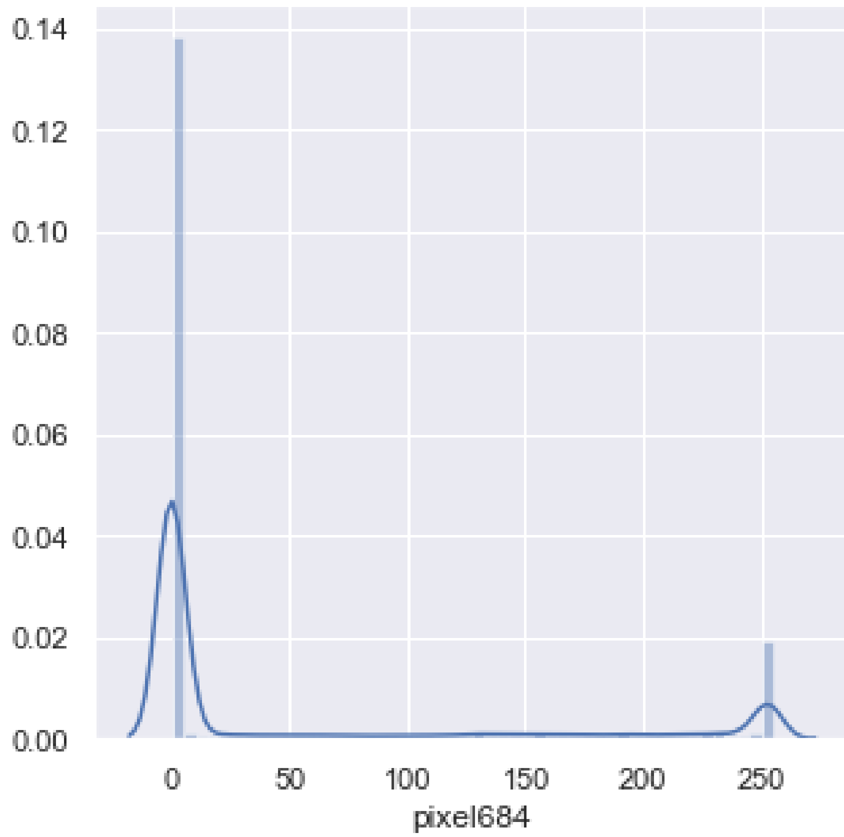



```
plt.figure(figsize=(5,5))
sns.distplot(numbers['pixel656'])
plt.show()
```



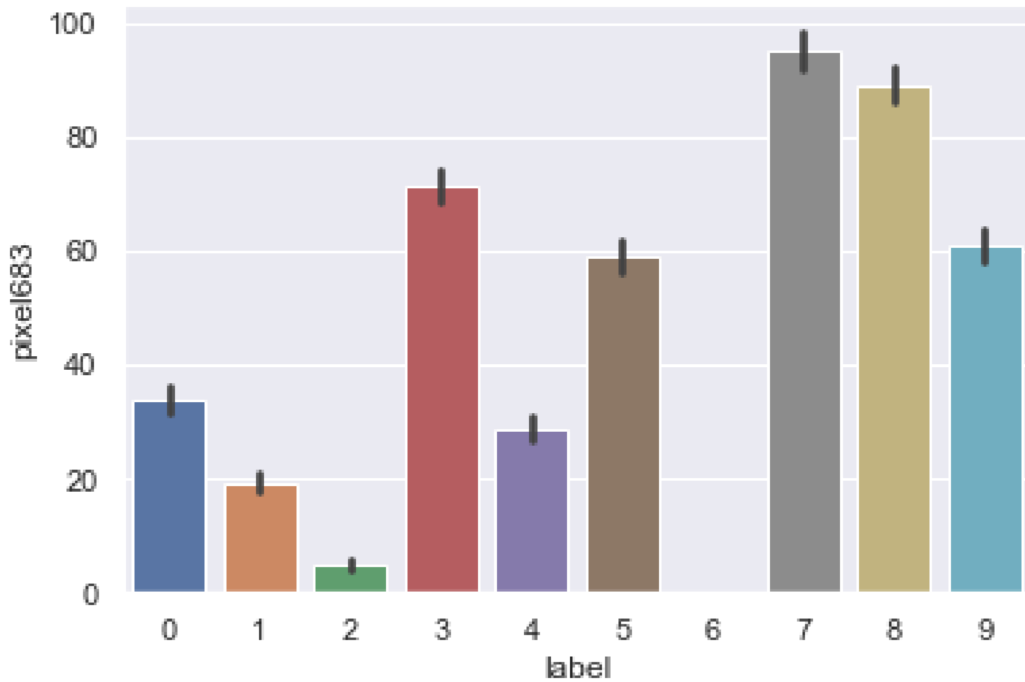

```
plt.figure(figsize=(5,5))
sns.distplot(numbers['pixel684'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2cb900f9978>
```



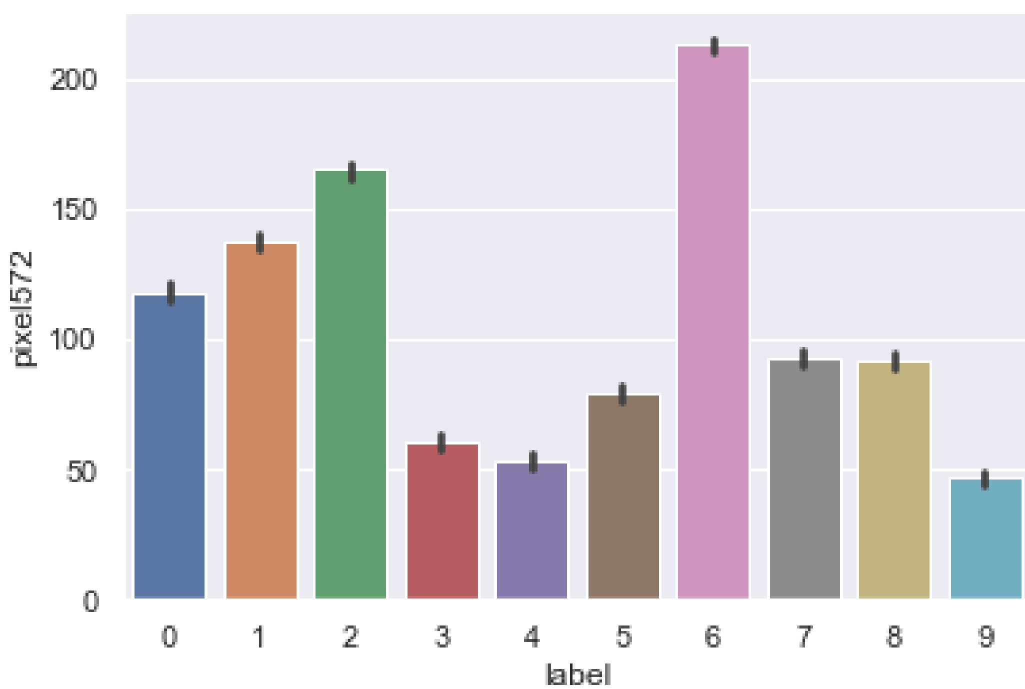

```
#label vs pixel
sns.barplot(x='label', y='pixel683', data=numbers)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2cb9019b940>
```




```
sns.barplot(x='label', y='pixel572', data=numbers)
```

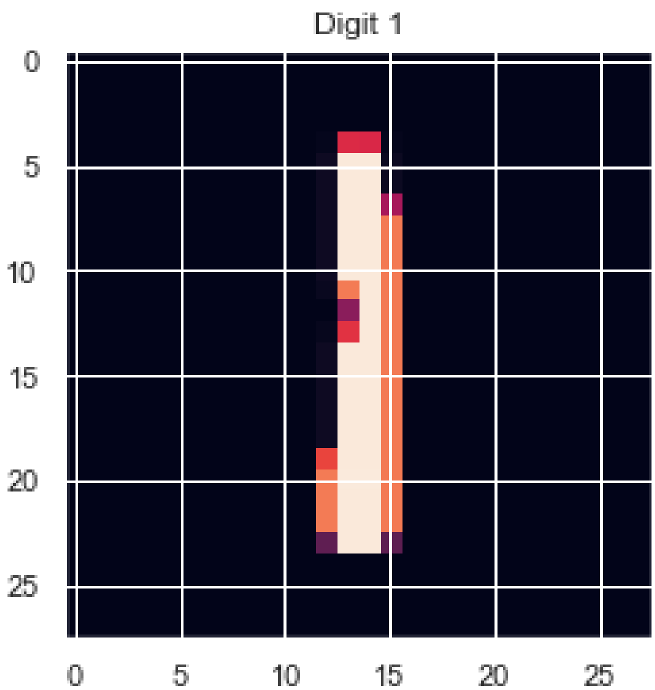
```
<matplotlib.axes._subplots.AxesSubplot at 0x2cb90226b38>
```




```
#visualize numbers
one = numbers.iloc[2, 1:]
one = one.values.reshape(28,28)
```

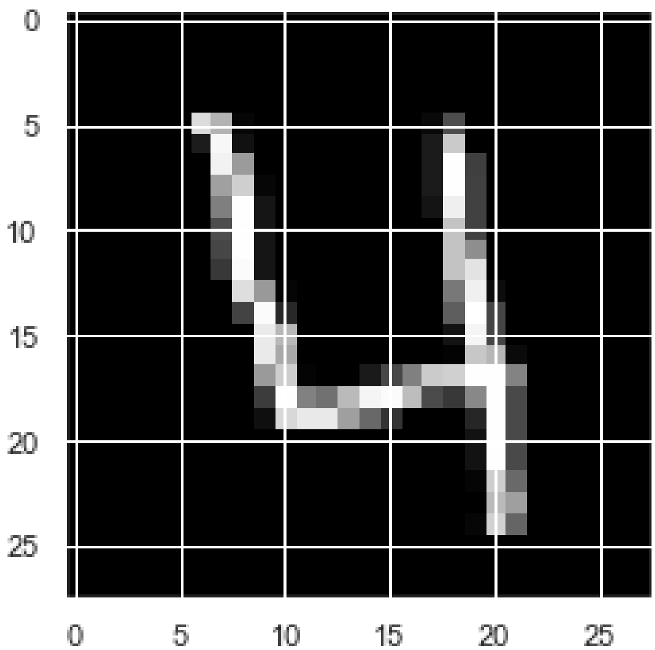
```
plt.imshow(one)
plt.title("Digit 1")
```

```
Text(0.5, 1.0, 'Digit 1')
```




```
four = numbers.iloc[3, 1:]
four.shape
four = four.values.reshape(28, 28)
plt.imshow(four, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x2cb900c4a20>
```




```
# visualise the array
print(four[5:-5, 5:-5])
```

```
[[  0 220 179   6   0   0   0   0   0   0   0   0   9  77   0   0   0   0]
 [  0  28 247  17   0   0   0   0   0   0   0   0  27 202   0   0   0   0]
 [  0   0 242 155   0   0   0   0   0   0   0   0  27 254  63   0   0   0]
 [  0   0 160 207   6   0   0   0   0   0   0   0  27 254  65   0   0   0]
 [  0   0 127 254  21   0   0   0   0   0   0   0  20 239  65   0   0   0]
 [  0   0  77 254  21   0   0   0   0   0   0   0   0 195  65   0   0   0]
 [  0   0  70 254  21   0   0   0   0   0   0   0   0 195 142   0   0   0]
 [  0   0  56 251  21   0   0   0   0   0   0   0   0 195 227   0   0   0]
 [  0   0   0 222 153   5   0   0   0   0   0   0   0 120 240  13   0   0]
 [  0   0   0  67 251  40   0   0   0   0   0   0   0  94 255  69   0   0]
 [  0   0   0   0 234 184   0   0   0   0   0   0   0  19 245  69   0   0]
 [  0   0   0   0 234 169   0   0   0   0   0   0   0   3 199 182  10   0]
 [  0   0   0   0 154 205   4   0   0  26  72 128 203 208 254 254 131   0]
 [  0   0   0   0  61 254 129 113 186 245 251 189  75  56 136 254  73   0]
 [  0   0   0   0  15 216 233 233 159 104  52   0   0   0  38 254  73   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  18 254  73   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  18 254  73   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   5 206 106   0]]
```

```
# missing values - there are none
numbers.isnull().sum()
```

```
label       0
pixel0      0
pixel1      0
pixel2      0
pixel3      0
```

```
pixel4      0
pixel5      0
pixel6      0
pixel7      0
pixel8      0
pixel9      0
pixel10     0
pixel11     0
pixel12     0
pixel13     0
pixel14     0
pixel15     0
pixel16     0
pixel17     0
pixel18     0
pixel19     0
pixel20     0
pixel21     0
pixel22     0
pixel23     0
pixel24     0
pixel25     0
pixel26     0
pixel27     0
pixel28     0
           ..
pixel754    0
pixel755    0
pixel756    0
pixel757    0
pixel758    0
pixel759    0
pixel760    0
pixel761    0
pixel762    0
pixel763    0
pixel764    0
pixel765    0
pixel766    0
pixel767    0
pixel768    0
pixel769    0
pixel770    0
pixel771    0
pixel772    0
pixel773    0
pixel774    0
pixel775    0
pixel776    0
pixel777    0
```

```
pixel778    0
pixel779    0
pixel780    0
pixel781    0
pixel782    0
pixel783    0
Length: 785, dtype: int64
```

```python
# average values/distributions of features
description = numbers.describe()
description
```

```
              label   pixel0   pixel1   pixel2   pixel3   pixel4   pixel5  \
count  42000.000000  42000.0  42000.0  42000.0  42000.0  42000.0  42000.0
mean       4.456643      0.0      0.0      0.0      0.0      0.0      0.0
std        2.887730      0.0      0.0      0.0      0.0      0.0      0.0
min        0.000000      0.0      0.0      0.0      0.0      0.0      0.0
25%        2.000000      0.0      0.0      0.0      0.0      0.0      0.0
50%        4.000000      0.0      0.0      0.0      0.0      0.0      0.0
75%        7.000000      0.0      0.0      0.0      0.0      0.0      0.0
max        9.000000      0.0      0.0      0.0      0.0      0.0      0.0

        pixel6   pixel7   pixel8  ...      pixel774      pixel775  \
count  42000.0  42000.0  42000.0  ...  42000.000000  42000.000000
mean       0.0      0.0      0.0  ...      0.219286      0.117095
std        0.0      0.0      0.0  ...      6.312890      4.633819
min        0.0      0.0      0.0  ...      0.000000      0.000000
25%        0.0      0.0      0.0  ...      0.000000      0.000000
50%        0.0      0.0      0.0  ...      0.000000      0.000000
75%        0.0      0.0      0.0  ...      0.000000      0.000000
max        0.0      0.0      0.0  ...    254.000000    254.000000

           pixel776     pixel777      pixel778      pixel779  pixel780  \
count  42000.000000  42000.00000  42000.000000  42000.000000   42000.0
mean       0.059024      0.02019      0.017238      0.002857       0.0
std        3.274488      1.75987      1.894498      0.414264       0.0
min        0.000000      0.00000      0.000000      0.000000       0.0
25%        0.000000      0.00000      0.000000      0.000000       0.0
50%        0.000000      0.00000      0.000000      0.000000       0.0
75%        0.000000      0.00000      0.000000      0.000000       0.0
max      253.000000    253.00000    254.000000     62.000000       0.0

       pixel781  pixel782  pixel783
count   42000.0   42000.0   42000.0
mean        0.0       0.0       0.0
std         0.0       0.0       0.0
min         0.0       0.0       0.0
25%         0.0       0.0       0.0
50%         0.0       0.0       0.0
75%         0.0       0.0       0.0
```

```
max          0.0       0.0       0.0

[8 rows x 785 columns]
```

```python
# average feature values
pd.set_option('display.max_rows', 999)
round(numbers.drop('label', axis=1).mean(), 2).sort_values(ascending = False)
```

```
pixel407    139.83
pixel435    139.07
pixel408    137.42
pixel434    135.52
pixel211    135.49
pixel210    133.59
pixel602    132.98
pixel212    132.90
pixel436    132.00
pixel601    130.81
pixel380    130.66
pixel406    130.14
pixel409    128.65
pixel381    127.50
pixel463    127.45
pixel575    126.73
pixel603    126.34
pixel209    126.26
pixel629    126.03
pixel462    126.01
pixel213    126.00
pixel574    125.56
pixel183    124.89
pixel379    124.51
pixel237    123.89
pixel238    123.23
pixel433    123.10
pixel628    122.65
pixel600    122.59
pixel240    122.22
pixel239    121.98
pixel184    121.90
pixel630    121.89
pixel241    121.77
pixel437    121.70
pixel464    121.32
pixel236    120.60
pixel182    119.70
pixel576    119.67
pixel547    118.65
pixel353    118.46
pixel548    117.19
```

pixel573	116.96
pixel242	116.12
pixel491	115.44
pixel461	115.36
pixel405	114.96
pixel352	113.93
pixel627	113.00
pixel208	112.99
pixel214	112.65
pixel492	112.60
pixel519	112.59
pixel546	112.43
pixel599	112.37
pixel520	112.15
pixel490	112.10
pixel263	111.90
pixel185	111.65
pixel604	111.62
pixel378	111.46
pixel465	111.36
pixel270	111.29
pixel264	111.12
pixel269	110.75
pixel235	110.21
pixel382	109.51
pixel549	108.68
pixel631	108.46
pixel325	108.00
pixel354	107.83
pixel410	107.68
pixel181	107.54
pixel298	106.52
pixel572	106.25
pixel518	106.08
pixel521	105.99
pixel326	105.68
pixel577	105.26
pixel493	105.24
pixel432	104.61
pixel297	104.54
pixel265	104.09
pixel268	103.87
pixel291	103.60
pixel351	102.12
pixel438	101.92
pixel290	101.85
pixel489	101.21
pixel598	101.02
pixel262	101.01
pixel460	100.51

pixel545	100.33
pixel243	100.19
pixel318	99.96
pixel656	99.90
pixel657	99.82
pixel271	99.14
pixel626	98.92
pixel346	98.91
pixel267	98.87
pixel266	98.53
pixel324	97.82
pixel571	97.62
pixel404	97.16
pixel374	97.15
pixel377	96.88
pixel466	95.93
pixel186	95.59
pixel207	95.42
pixel319	94.94
pixel522	94.60
pixel402	94.56
pixel517	94.35
pixel299	94.19
pixel296	93.76
pixel431	93.73
pixel494	93.53
pixel550	93.43
pixel292	93.18
pixel215	92.73
pixel403	92.68
pixel234	92.12
pixel373	91.76
pixel655	91.62
pixel155	91.59
pixel658	91.57
pixel347	91.42
pixel375	91.12
pixel345	91.09
pixel605	90.95
pixel430	90.78
pixel180	90.47
pixel570	90.21
pixel401	89.73
pixel350	89.51
pixel156	89.45
pixel544	89.26
pixel632	89.06
pixel488	88.92
pixel459	88.88
pixel327	88.70

pixel317	88.65
pixel597	87.45
pixel376	86.84
pixel154	86.30
pixel323	85.60
pixel289	85.24
pixel429	85.14
pixel578	85.09
pixel355	84.27
pixel295	83.99
pixel458	83.55
pixel516	82.62
pixel543	82.16
pixel293	82.03
pixel569	81.35
pixel383	81.27
pixel320	81.14
pixel625	80.79
pixel157	80.30
pixel487	80.26
pixel261	79.87
pixel348	79.76
pixel411	79.75
pixel349	79.70
pixel294	78.92
pixel439	78.86
pixel495	78.66
pixel467	78.36
pixel523	78.10
pixel457	77.70
pixel542	77.61
pixel272	76.66
pixel244	76.63
pixel654	76.57
pixel659	76.47
pixel515	76.27
pixel322	76.01
pixel153	75.01
pixel206	74.79
pixel187	74.60
pixel486	74.59
pixel400	73.58
pixel551	73.22
pixel321	72.99
pixel372	72.83
pixel179	72.00
pixel514	71.46
pixel300	71.39
pixel541	71.28
pixel428	71.10

pixel596	70.45
pixel344	69.60
pixel485	69.40
pixel233	69.18
pixel568	68.99
pixel216	68.90
pixel606	67.71
pixel633	66.84
pixel456	66.20
pixel513	65.86
pixel158	65.76
pixel316	65.19
pixel328	64.59
pixel540	62.61
pixel579	62.30
pixel288	60.88
pixel624	60.69
pixel484	60.62
pixel468	60.52
pixel496	60.41
pixel152	60.18
pixel440	58.85
pixel512	58.56
pixel356	58.37
pixel653	58.29
pixel660	58.16
pixel524	57.81
pixel412	56.31
pixel260	55.56
pixel384	55.37
pixel205	54.74
pixel178	54.01
pixel188	53.74
pixel567	53.74
pixel273	52.10
pixel245	52.02
pixel552	51.85
pixel595	51.39
pixel684	51.24
pixel539	51.12
pixel685	51.04
pixel427	51.02
pixel399	50.57
pixel455	49.76
pixel159	49.65
pixel511	48.30
pixel483	48.02
pixel371	47.93
pixel301	47.89
pixel232	47.38

pixel217	46.21
pixel683	46.20
pixel607	46.19
pixel127	46.09
pixel686	45.93
pixel634	45.73
pixel151	45.36
pixel128	44.54
pixel343	44.02
pixel126	42.71
pixel469	42.63
pixel329	42.46
pixel441	41.73
pixel497	41.52
pixel580	41.47
pixel661	40.59
pixel623	40.24
pixel315	40.07
pixel652	39.84
pixel413	39.24
pixel129	38.95
pixel525	38.40
pixel357	38.36
pixel177	37.88
pixel385	37.48
pixel287	37.33
pixel687	37.29
pixel682	37.14
pixel566	36.90
pixel538	36.88
pixel204	36.60
pixel125	36.08
pixel189	34.96
pixel510	34.72
pixel259	34.52
pixel160	34.39
pixel553	33.29
pixel482	33.28
pixel594	32.56
pixel454	32.25
pixel150	32.22
pixel246	31.16
pixel130	30.96
pixel274	30.89
pixel426	30.78
pixel231	29.64
pixel635	28.86
pixel608	28.82
pixel398	28.52
pixel302	28.21

pixel124	28.04
pixel688	28.02
pixel218	27.63
pixel442	27.17
pixel470	27.05
pixel681	26.40
pixel414	25.75
pixel370	25.70
pixel662	25.69
pixel330	25.37
pixel498	25.25
pixel581	25.23
pixel386	24.29
pixel176	24.13
pixel358	23.99
pixel651	23.71
pixel622	23.26
pixel131	22.91
pixel342	22.85
pixel526	22.68
pixel203	22.36
pixel537	21.59
pixel161	21.46
pixel149	21.42
pixel314	21.09
pixel190	20.70
pixel565	20.66
pixel286	20.48
pixel123	20.33
pixel509	20.15
pixel258	19.54
pixel689	19.44
pixel554	18.95
pixel481	18.16
pixel712	18.13
pixel713	17.90
pixel593	16.95
pixel230	16.91
pixel636	16.69
pixel711	16.56
pixel680	16.42
pixel453	16.35
pixel609	16.23
pixel714	16.11
pixel247	15.89
pixel663	15.21
pixel443	15.10
pixel275	15.04
pixel132	14.87
pixel415	14.86

pixel219	14.79
pixel471	14.34
pixel425	14.31
pixel175	14.11
pixel387	13.90
pixel122	13.80
pixel582	13.66
pixel715	13.64
pixel303	13.50
pixel710	13.48
pixel99	13.40
pixel100	13.07
pixel359	12.95
pixel148	12.93
pixel499	12.82
pixel331	12.66
pixel202	12.39
pixel397	12.36
pixel162	12.23
pixel690	12.16
pixel98	12.07
pixel650	12.01
pixel101	11.57
pixel527	11.29
pixel191	11.12
pixel621	10.91
pixel716	10.89
pixel369	10.70
pixel97	10.05
pixel257	9.88
pixel285	9.81
pixel341	9.56
pixel709	9.44
pixel313	9.44
pixel102	9.30
pixel555	9.13
pixel679	8.93
pixel637	8.91
pixel229	8.70
pixel133	8.69
pixel121	8.68
pixel536	8.62
pixel564	8.58
pixel664	8.41
pixel610	8.40
pixel717	8.06
pixel96	7.75
pixel508	7.52
pixel174	7.37
pixel691	7.31

pixel147	7.15
pixel248	6.91
pixel220	6.78
pixel592	6.76
pixel103	6.71
pixel740	6.61
pixel583	6.50
pixel741	6.44
pixel163	6.37
pixel201	5.98
pixel739	5.98
pixel480	5.89
pixel444	5.89
pixel276	5.85
pixel416	5.84
pixel708	5.84
pixel95	5.71
pixel742	5.68
pixel472	5.59
pixel718	5.33
pixel388	5.26
pixel192	5.25
pixel500	5.05
pixel649	5.03
pixel120	4.95
pixel743	4.66
pixel256	4.65
pixel738	4.64
pixel304	4.63
pixel134	4.55
pixel638	4.55
pixel360	4.46
pixel452	4.45
pixel528	4.45
pixel284	4.31
pixel665	4.27
pixel678	4.15
pixel104	4.14
pixel332	4.13
pixel611	4.12
pixel620	4.03
pixel692	4.00
pixel228	3.97
pixel71	3.80
pixel744	3.77
pixel94	3.77
pixel72	3.74
pixel312	3.57
pixel146	3.56
pixel556	3.54

pixel424	3.48
pixel70	3.39
pixel73	3.33
pixel173	3.27
pixel737	3.24
pixel719	3.19
pixel707	2.99
pixel340	2.98
pixel396	2.94
pixel164	2.91
pixel368	2.81
pixel745	2.75
pixel69	2.70
pixel74	2.68
pixel584	2.63
pixel249	2.48
pixel200	2.45
pixel119	2.45
pixel221	2.38
pixel563	2.29
pixel93	2.29
pixel105	2.27
pixel736	2.16
pixel666	2.11
pixel135	2.10
pixel639	2.10
pixel535	2.04
pixel693	1.99
pixel75	1.99
pixel68	1.98
pixel591	1.91
pixel277	1.84
pixel648	1.81
pixel746	1.80
pixel255	1.80
pixel193	1.77
pixel720	1.72
pixel283	1.71
pixel677	1.64
pixel612	1.63
pixel507	1.51
pixel145	1.44
pixel227	1.40
pixel67	1.35
pixel501	1.33
pixel706	1.31
pixel311	1.29
pixel529	1.25
pixel473	1.24
pixel76	1.20

pixel92	1.19
pixel305	1.16
pixel619	1.16
pixel735	1.15
pixel172	1.15
pixel106	1.09
pixel747	1.09
pixel445	1.06
pixel557	1.03
pixel118	1.03
pixel479	0.97
pixel667	0.95
pixel694	0.93
pixel339	0.90
pixel66	0.87
pixel136	0.84
pixel721	0.84
pixel417	0.83
pixel640	0.80
pixel165	0.79
pixel585	0.75
pixel770	0.68
pixel333	0.65
pixel451	0.65
pixel199	0.62
pixel389	0.62
pixel367	0.61
pixel77	0.60
pixel771	0.60
pixel769	0.56
pixel676	0.56
pixel748	0.56
pixel254	0.55
pixel282	0.54
pixel91	0.54
pixel65	0.53
pixel768	0.51
pixel734	0.50
pixel772	0.49
pixel222	0.49
pixel361	0.48
pixel423	0.48
pixel250	0.48
pixel705	0.47
pixel647	0.47
pixel310	0.44
pixel395	0.43
pixel613	0.43
pixel107	0.42
pixel144	0.41

pixel767	0.41
pixel695	0.40
pixel117	0.39
pixel226	0.38
pixel722	0.38
pixel194	0.35
pixel773	0.34
pixel278	0.33
pixel562	0.32
pixel668	0.31
pixel64	0.30
pixel766	0.30
pixel78	0.29
pixel534	0.28
pixel590	0.27
pixel338	0.27
pixel90	0.24
pixel749	0.24
pixel171	0.23
pixel306	0.22
pixel774	0.22
pixel137	0.20
pixel502	0.20
pixel42	0.20
pixel641	0.20
pixel40	0.19
pixel474	0.19
pixel41	0.19
pixel765	0.18
pixel43	0.17
pixel108	0.17
pixel39	0.17
pixel506	0.17
pixel618	0.17
pixel530	0.16
pixel44	0.16
pixel558	0.16
pixel723	0.15
pixel63	0.15
pixel45	0.15
pixel704	0.15
pixel675	0.14
pixel446	0.14
pixel334	0.14
pixel733	0.14
pixel166	0.13
pixel38	0.13
pixel366	0.13
pixel775	0.12
pixel478	0.11

pixel46	0.11
pixel198	0.11
pixel764	0.11
pixel79	0.10
pixel281	0.10
pixel696	0.10
pixel253	0.10
pixel586	0.10
pixel750	0.09
pixel89	0.09
pixel116	0.09
pixel763	0.08
pixel418	0.07
pixel309	0.07
pixel37	0.07
pixel614	0.07
pixel362	0.07
pixel450	0.07
pixel225	0.07
pixel390	0.07
pixel646	0.06
pixel62	0.06
pixel47	0.06
pixel669	0.06
pixel776	0.06
pixel36	0.05
pixel143	0.05
pixel394	0.05
pixel48	0.05
pixel337	0.04
pixel732	0.04
pixel422	0.04
pixel762	0.04
pixel138	0.04
pixel279	0.03
pixel80	0.03
pixel35	0.03
pixel195	0.03
pixel307	0.03
pixel115	0.02
pixel642	0.02
pixel49	0.02
pixel88	0.02
pixel697	0.02
pixel197	0.02
pixel777	0.02
pixel109	0.02
pixel724	0.02
pixel778	0.02
pixel674	0.02

pixel170	0.02
pixel503	0.02
pixel223	0.02
pixel703	0.02
pixel475	0.02
pixel251	0.02
pixel751	0.02
pixel363	0.01
pixel335	0.01
pixel365	0.01
pixel81	0.01
pixel308	0.01
pixel60	0.01
pixel61	0.01
pixel391	0.01
pixel761	0.01
pixel13	0.01
pixel589	0.01
pixel50	0.01
pixel34	0.01
pixel252	0.01
pixel670	0.01
pixel561	0.01
pixel559	0.01
pixel51	0.01
pixel533	0.01
pixel531	0.01
pixel505	0.01
pixel280	0.01
pixel14	0.01
pixel725	0.01
pixel447	0.01
pixel142	0.01
pixel12	0.00
pixel33	0.00
pixel10	0.00
pixel24	0.00
pixel25	0.00
pixel26	0.00
pixel9	0.00
pixel8	0.00
pixel7	0.00
pixel6	0.00
pixel57	0.00
pixel5	0.00
pixel27	0.00
pixel28	0.00
pixel29	0.00
pixel32	0.00
pixel2	0.00

pixel1	0.00
pixel23	0.00
pixel56	0.00
pixel30	0.00
pixel17	0.00
pixel11	0.00
pixel15	0.00
pixel59	0.00
pixel54	0.00
pixel53	0.00
pixel52	0.00
pixel16	0.00
pixel18	0.00
pixel22	0.00
pixel4	0.00
pixel58	0.00
pixel55	0.00
pixel19	0.00
pixel3	0.00
pixel20	0.00
pixel21	0.00
pixel31	0.00
pixel783	0.00
pixel82	0.00
pixel673	0.00
pixel726	0.00
pixel702	0.00
pixel701	0.00
pixel700	0.00
pixel699	0.00
pixel698	0.00
pixel672	0.00
pixel588	0.00
pixel671	0.00
pixel645	0.00
pixel644	0.00
pixel643	0.00
pixel617	0.00
pixel616	0.00
pixel727	0.00
pixel728	0.00
pixel729	0.00
pixel730	0.00
pixel731	0.00
pixel752	0.00
pixel753	0.00
pixel754	0.00
pixel755	0.00
pixel756	0.00
pixel757	0.00

```
pixel758      0.00
pixel759      0.00
pixel760      0.00
pixel779      0.00
pixel780      0.00
pixel781      0.00
pixel615      0.00
pixel587      0.00
pixel83       0.00
pixel113      0.00
pixel168      0.00
pixel167      0.00
pixel141      0.00
pixel140      0.00
pixel139      0.00
pixel114      0.00
pixel112      0.00
pixel560      0.00
pixel111      0.00
pixel110      0.00
pixel87       0.00
pixel86       0.00
pixel85       0.00
pixel84       0.00
pixel169      0.00
pixel196      0.00
pixel224      0.00
pixel336      0.00
pixel364      0.00
pixel782      0.00
pixel392      0.00
pixel393      0.00
pixel419      0.00
pixel420      0.00
pixel421      0.00
pixel448      0.00
pixel449      0.00
pixel476      0.00
pixel477      0.00
pixel504      0.00
pixel532      0.00
pixel0        0.00
dtype: float64
```

```
# splitting into X and y
X = numbers.drop("label", axis = 1)
y = numbers['label']
```

```
# scaling the features
X_scaled = scale(X)
```

```python
# train test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
train_size=0.2,test_size = 0.8, random_state = 101)

print('X_train shape:',X_train.shape)
print('y_train shape:',y_train.shape)
print('X_test shape:',X_test.shape)
print('y_test shape:',y_test.shape)
```

```
X_train shape: (8400, 784)
y_train shape: (8400,)
X_test shape: (33600, 784)
y_test shape: (33600,)
```

```python
# linear model
model_linear = SVC(kernel='linear')
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)

# confusion matrix and accuracy, precision, recall

# accuracy
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred),
"\n")

# cm
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```

```
accuracy: 0.913125

[[3188    0   10    5   11   20   32    3   15    1]
 [   0 3677   14   11    5    7    4    8   30    4]
 [  36   29 3027   54   55   10   30   42   48   12]
 [  13   12  104 3051    9  181    5   21   54   25]
 [   8   14   33    2 3057    4   25   31    6  110]
 [  30   23   29  136   44 2622   44   12   72   27]
 [  26   11   44    4   28   33 3113    0   18    0]
 [   7   24   36   19   59    9    2 3210    4  134]
 [  13   46   50  120   21  110   30   18 2843   21]
 [  19   17   21   22  172   20    4  161   26 2893]]
```

```python
#precision, recall and f1-score
scores=metrics.classification_report(y_test, y_pred, labels=[0, 1, 2, 3, 4,
5, 6, 7, 8, 9])
print(scores)
```

```
              precision    recall  f1-score   support
```

```
           0       0.95      0.97      0.96      3285
           1       0.95      0.98      0.97      3760
           2       0.90      0.91      0.90      3343
           3       0.89      0.88      0.88      3475
           4       0.88      0.93      0.91      3290
           5       0.87      0.86      0.87      3039
           6       0.95      0.95      0.95      3277
           7       0.92      0.92      0.92      3504
           8       0.91      0.87      0.89      3272
           9       0.90      0.86      0.88      3355

    accuracy                           0.91     33600
   macro avg       0.91      0.91      0.91     33600
weighted avg       0.91      0.91      0.91     33600
```

```
# non-linear model
# using poly kernel, C=1, default value of gamma

# model
non_linear_model_poly = SVC(kernel='poly')

# fit
non_linear_model_poly.fit(X_train, y_train)

# predict
y_pred = non_linear_model_poly.predict(X_test)

#confusion matrix and accuracy, precision, recall

# accuracy
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred),
"\n")

# cm
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```

```
accuracy: 0.87125

[[2893    0    8    3   38   13   28    0  299    3]
 [   1 3684   11    1   12    0    6    0   43    2]
 [  14   18 2489   37  153    1   11   19  588   13]
 [   0   16   24 2846   27   36    3   13  464   46]
 [   3    8   15    0 3080    8    3    5   17  151]
 [   6    1    5   72   73 2358   30    7  442   45]
 [  16    9   10    0  108   43 2901    2  188    0]
 [   1   41    7    7  138    1    0 2862  105  342]
 [   2   16    6   30   20   40    1    4 3131   22]
 [   6   14    1   13  153    8    0   42   88 3030]]
```

```python
# non-linear model
# using rbf kernel, C=1, default value of gamma

# model
non_linear_model = SVC(kernel='rbf')

# fit
non_linear_model.fit(X_train, y_train)

# predict
y_pred = non_linear_model.predict(X_test)

# confusion matrix and accuracy, precision, recall

# accuracy
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred),
"\n")

# cm
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```

```
accuracy: 0.9396428571428571

[[3195    0   19    5    4   11   32    4   14    1]
 [   0 3689   23   12    8    3    7    6    8    4]
 [  15   15 3144   29   31    5   18   37   43    6]
 [   5    8   92 3191    5   73    6   31   43   21]
 [   3    7   57    1 3099    9   19   21    7   67]
 [  15   10   37   66   16 2776   53   15   32   19]
 [  19    5   46    1   12   31 3149    2   12    0]
 [   6   21   66   11   25    3    0 3285    3   84]
 [  14   24   40   63   14   62   22   19 2996   18]
 [  12   10   38   40   80    6    0   97   24 3048]]
```

```python
#precision, recall and f1-score
scores=metrics.classification_report(y_test, y_pred, labels=[0, 1, 2, 3, 4,
5, 6, 7, 8, 9])
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      3285
           1       0.97      0.98      0.98      3760
           2       0.88      0.94      0.91      3343
           3       0.93      0.92      0.93      3475
           4       0.94      0.94      0.94      3290
           5       0.93      0.91      0.92      3039
           6       0.95      0.96      0.96      3277
           7       0.93      0.94      0.94      3504
           8       0.94      0.92      0.93      3272
```

```
           9       0.93      0.91      0.92      3355

    accuracy                           0.94     33600
   macro avg       0.94      0.94      0.94     33600
weighted avg       0.94      0.94      0.94     33600
```

```python
# creating a KFold object with 5 splits
folds = KFold(n_splits = 5, shuffle = True, random_state = 101)

# specify range of hyperparameters
# Set the parameters by cross-validation
hyper_params = [ {'gamma': [0.01, 0.001,0.0001],
                     'C': [1, 10, 100]}]


# specify model
model = SVC(kernel="rbf")

# set up GridSearchCV()
model_cv = GridSearchCV(estimator = model,
                        param_grid = hyper_params,
                        scoring= 'accuracy',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True,n_jobs = -1)

# fit the model
model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed: 22.6min finished

GridSearchCV(cv=KFold(n_splits=5, random_state=101, shuffle=True),
             error_score='raise-deprecating',
             estimator=SVC(C=1.0, cache_size=200, class_weight=None,
coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='auto_deprecated', kernel='rbf', max_iter=-
1,
                           probability=False, random_state=None,
shrinking=True,
                           tol=0.001, verbose=False),
             iid='warn', n_jobs=-1,
             param_grid=[{'C': [1, 10, 100], 'gamma': [0.01, 0.001,
0.0001]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='accuracy', verbose=1)
```

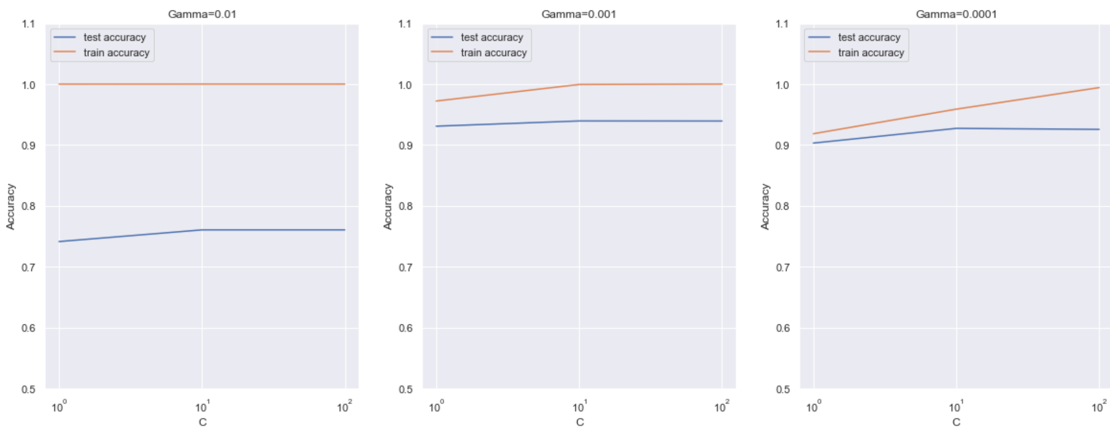
```python
# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time param_C  \
0     130.659739      1.989133        14.666499        0.081432       1
1      32.536529      0.142300         8.515690        0.093374       1
2      49.312719      1.380140        12.081309        0.709100       1
3     155.832136     12.587659        16.657551        2.226980      10
4      27.124150      0.316659         7.750964        0.016862      10
5      20.726886      0.232504         7.159922        0.099636      10
6     133.620725      1.723362        14.977487        0.186436     100
7      27.370743      0.306953         7.755510        0.118800     100
8      15.408296      0.850696         5.484372        0.774388     100

  param_gamma                         params  split0_test_score  \
0        0.01      {'C': 1, 'gamma': 0.01}           0.752381
1       0.001     {'C': 1, 'gamma': 0.001}           0.935119
2      0.0001    {'C': 1, 'gamma': 0.0001}           0.910119
3        0.01     {'C': 10, 'gamma': 0.01}           0.766071
4       0.001    {'C': 10, 'gamma': 0.001}           0.941071
5      0.0001   {'C': 10, 'gamma': 0.0001}           0.933929
6        0.01    {'C': 100, 'gamma': 0.01}           0.766071
7       0.001   {'C': 100, 'gamma': 0.001}           0.939881
8      0.0001  {'C': 100, 'gamma': 0.0001}           0.929762

   split1_test_score  split2_test_score  ...  mean_test_score  std_test_score
\
0           0.750595           0.747024  ...         0.741310        0.010784
1           0.926786           0.935119  ...         0.930833        0.004216
2           0.905952           0.907738  ...         0.903095        0.006075
3           0.772619           0.765476  ...         0.760476        0.009705
4           0.938690           0.945833  ...         0.939405        0.003865
5           0.923214           0.931548  ...         0.927262        0.004678
6           0.772619           0.765476  ...         0.760476        0.009705
7           0.936905           0.946429  ...         0.939286        0.003783
8           0.923810           0.925595  ...         0.925595        0.003409

   rank_test_score  split0_train_score  split1_train_score  \
0                9            1.000000            0.999851
1                3            0.972321            0.971429
2                6            0.916518            0.917708
3                7            1.000000            1.000000
4                1            0.999405            0.999554
5                4            0.957887            0.959970
6                7            1.000000            1.000000
7                2            1.000000            1.000000
8                5            0.994345            0.994494

   split2_train_score  split3_train_score  split4_train_score  \
```

```
0           0.999851           1.000000           1.000000
1           0.971429           0.972173           0.973214
2           0.916518           0.921577           0.919940
3           1.000000           1.000000           1.000000
4           0.999405           0.999107           0.999256
5           0.959375           0.957738           0.959077
6           1.000000           1.000000           1.000000
7           1.000000           1.000000           1.000000
8           0.994940           0.993006           0.994196

   mean_train_score  std_train_score
0          0.999940         0.000073
1          0.972113         0.000663
2          0.918452         0.002001
3          1.000000         0.000000
4          0.999345         0.000152
5          0.958810         0.000865
6          1.000000         0.000000
7          1.000000         0.000000
8          0.994196         0.000645

[9 rows x 22 columns]
```

```python
# converting C to numeric type for plotting on x-axis
cv_results['param_C'] = cv_results['param_C'].astype('int')

# # plotting
plt.figure(figsize=(20,7))

# subplot 1/3
plt.subplot(131)
gamma_01 = cv_results[cv_results['param_gamma']==0.01]

plt.plot(gamma_01["param_C"], gamma_01["mean_test_score"])
plt.plot(gamma_01["param_C"], gamma_01["mean_train_score"])
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title("Gamma=0.01")
plt.ylim([0.50, 1.1])
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')

# subplot 2/3
plt.subplot(132)
gamma_001 = cv_results[cv_results['param_gamma']==0.001]

plt.plot(gamma_001["param_C"], gamma_001["mean_test_score"])
plt.plot(gamma_001["param_C"], gamma_001["mean_train_score"])
plt.xlabel('C')
```

```
plt.ylabel('Accuracy')
plt.title("Gamma=0.001")
plt.ylim([0.50, 1.1])
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')


# subplot 3/3
plt.subplot(133)
gamma_0001 = cv_results[cv_results['param_gamma']==0.0001]

plt.plot(gamma_0001["param_C"], gamma_0001["mean_test_score"])
plt.plot(gamma_0001["param_C"], gamma_0001["mean_train_score"])
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title("Gamma=0.0001")
plt.ylim([0.50, 1.1])
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')
```




```
# printing the optimal accuracy score and hyperparameters
best_score = model_cv.best_score_
best_hyperparams = model_cv.best_params_

print("The best test score is {0} corresponding to hyperparameters
{1}".format(best_score, best_hyperparams))
```

```
The best test score is 0.9394047619047619 corresponding to hyperparameters
{'C': 10, 'gamma': 0.001}
```

```
#BUILDING AND EVALUATING THE FINAL MODEL
# model with optimal hyperparameters

# model
model = SVC(C=10, gamma=0.001, kernel="rbf")

model.fit(X_train, y_train)
```

```python
y_pred = model.predict(X_test)

# metrics
print("accuracy", metrics.accuracy_score(y_test, y_pred), "\n")
print(metrics.confusion_matrix(y_test, y_pred), "\n")
```

```
accuracy 0.9477083333333334

[[3211    0   19    2    2   12   26    3    8    2]
 [   0 3692   26    9    6    3    5    9    7    3]
 [  13   12 3165   29   29    6   20   40   21    8]
 [   4    5   77 3232    4   79    1   23   31   19]
 [   5    8   42    1 3117    5   20   19    9   64]
 [  15    8   33   61   15 2815   35   11   31   15]
 [  19    5   44    1   12   18 3167    1   10    0]
 [   5   17   52   12   29    4    1 3322    4   58]
 [   7   16   42   53   15   51   18   16 3044   10]
 [   9    9   33   20   81   10    0   94   21 3078]]
```

```python
# different class-wise accuracy - #precision, recall and f1-score
scores=metrics.classification_report(y_test, y_pred, labels=[0, 1, 2, 3, 4,
5, 6, 7, 8, 9])
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      3285
           1       0.98      0.98      0.98      3760
           2       0.90      0.95      0.92      3343
           3       0.95      0.93      0.94      3475
           4       0.94      0.95      0.94      3290
           5       0.94      0.93      0.93      3039
           6       0.96      0.97      0.96      3277
           7       0.94      0.95      0.94      3504
           8       0.96      0.93      0.94      3272
           9       0.95      0.92      0.93      3355

    accuracy                           0.95     33600
   macro avg       0.95      0.95      0.95     33600
weighted avg       0.95      0.95      0.95     33600
```

```python
# Let us visualize our final model on unseen training dataset

df = np.random.randint(1,y_pred.shape[0]+1,5)

plt.figure(figsize=(16,4))
for i,j in enumerate(df):
    plt.subplot(150+i+1)
```

```
    d = X_test[j].reshape(28,28)
    plt.title(f'Predicted Label: {y_pred[j]}')
    plt.imshow(d)
plt.show()
```