In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Import Data

In [3]:
```
df = pd.read_csv('/content/drive/MyDrive/Project_Docs/MNIST_Dataset/MNIST_test.csv')
df
```

Out[3]:

	Unnamed: 0	index	labels	0	1	2	3	4	5	6	...	774	775	776	777	778	779	780	781
0	0	0	7	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
1	1	1	2	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
2	2	2	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
3	3	3	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	4	4	4	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9995	9995	2	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
9996	9996	9996	3	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
9997	9997	9997	4	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
9998	9998	9998	5	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
9999	9999	9999	6	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

10000 rows × 787 columns

In [4]:
```
df.dtypes
```

Out[4]:

	0
Unnamed: 0	int64
index	int64
labels	int64
0	int64
1	int64
...	...
779	int64
780	int64
781	int64
782	int64
783	int64

787 rows × 1 columns

dtype: object

In [5]:
```
df = df.to_numpy()
df
```

Out[5]:
```
array([[   0,    0,    7, ...,    0,    0,    0],
       [   1,    1,    2, ...,    0,    0,    0],
       [   2,    2,    1, ...,    0,    0,    0],
       ...,
       [9997, 9997,    4, ...,    0,    0,    0],
       [9998, 9998,    5, ...,    0,    0,    0],
       [9999, 9999,    6, ...,    0,    0,    0]])
```

In [6]:
```
y=df[:,2]
```

In [7]:
```
y.shape
```

Out[7]:
```
(10000,)
```

In [8]:
```
K=set(y)
K
```

Out[8]:
```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [9]:
```
X=df[:,3:]
X.shape
```

Out[9]:
```
(10000, 784)
```

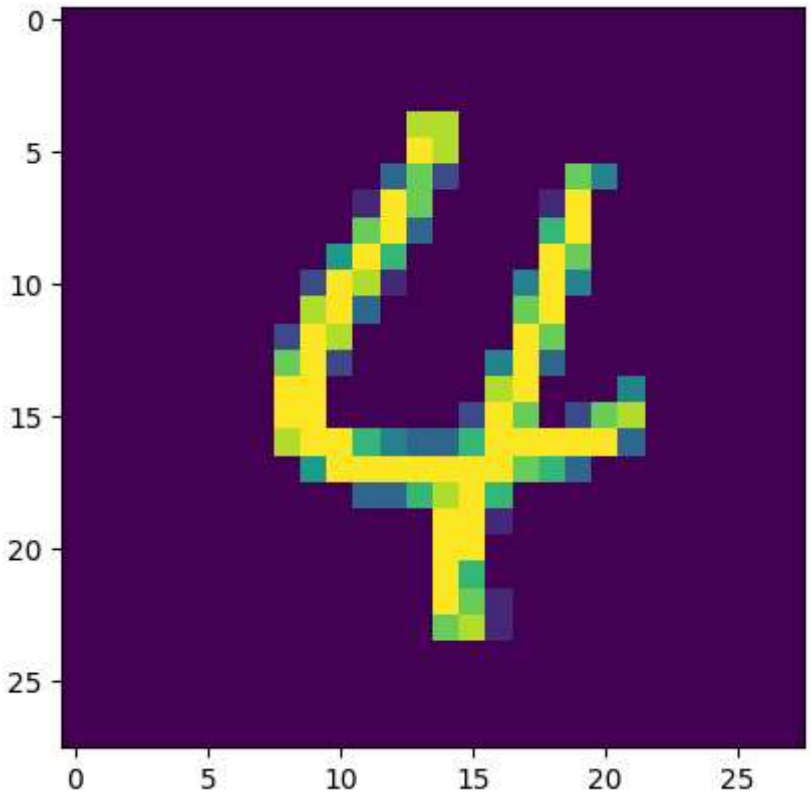
In [10]:
```
# Flattens each 28x28 image in X to a 784-element vector And converts data type to a 3
# Then scales the pixel values to a range of 0 to 1 by dividing by 255
X = X.reshape(-1, 784).astype(np.float32) / 255.0
```

In [11]:
```
X.shape
```

Out[11]:
```
(10000, 784)
```

In [12]:
```
plt.figure()
plt.imshow(X[109].reshape(28,28))
```

Out[12]:
```
<matplotlib.image.AxesImage at 0x7f3069049f60>
```



In [13]:
```
from scipy.stats import multivariate_normal as mvn
```

Naives Bayes Classifier

In [14]:
```
class GaussNB():
  def fit(self, X, y, epsilon = 1e-2):
    self.likelihoods = dict()
    self.priors = dict()
    self.K = set(y.astype(int))

    for k in self.K:
      X_k = X[y==k]

      self.likelihoods[k] = {"mean": X_k.mean(axis=0), "cov":X_k.var(axis=0)+epsilon}
      self.priors[k] = len(X_k)/len(X)

  def predict(self, X):
    N , D = X.shape
    P_hat = np.zeros((N,len(self.K)))

    for k,l in self.likelihoods.items():
```

```
      P_hat[:,k] = mvn.logpdf(X, l["mean"], l["cov"])+np.log(self.priors[k])

    return P_hat.argmax(axis=1)
```

In [15]:
```
gnb = GaussNB()
```

In [16]:
```
gnb.fit(X,y)
```

In [17]:
```
y_hat = gnb.predict(X)
```

In [18]:
```
def accuracy(y,y_hat):
  return np.mean(y==y_hat)
```

In [19]:
```
accuracy(y,y_hat)
```

Out[19]:
```
0.821
```

Non Naive Basessian Classifier Assuming Normal Distribution

In [20]:
```
class GaussBayes():

  def fit(self, X, y, epsilon = 1e-3):
    self.likelihoods = dict()
    self.priors = dict()
    self.K = set(y.astype(int))

    for k in self.K:
      X_k = X[y==k,:]
      N_k, D = X_k.shape
      mu_k = X_k.mean(axis=0)

      self.likelihoods[k] = {"mean":mu_k, "cov":(1/(N_k-1))*np.matmul(np.transpose(X_k
      self.priors[k] = len(X_k)/len(X)

  def predict(self, X):
    N , D = X.shape
    P_hat = np.zeros((N,len(self.K)))
    for k , l in self.likelihoods.items():
      P_hat[:,k] = mvn.logpdf(X, l["mean"], l["cov"])+np.log(self.priors[k])

    return P_hat.argmax(axis=1)
```

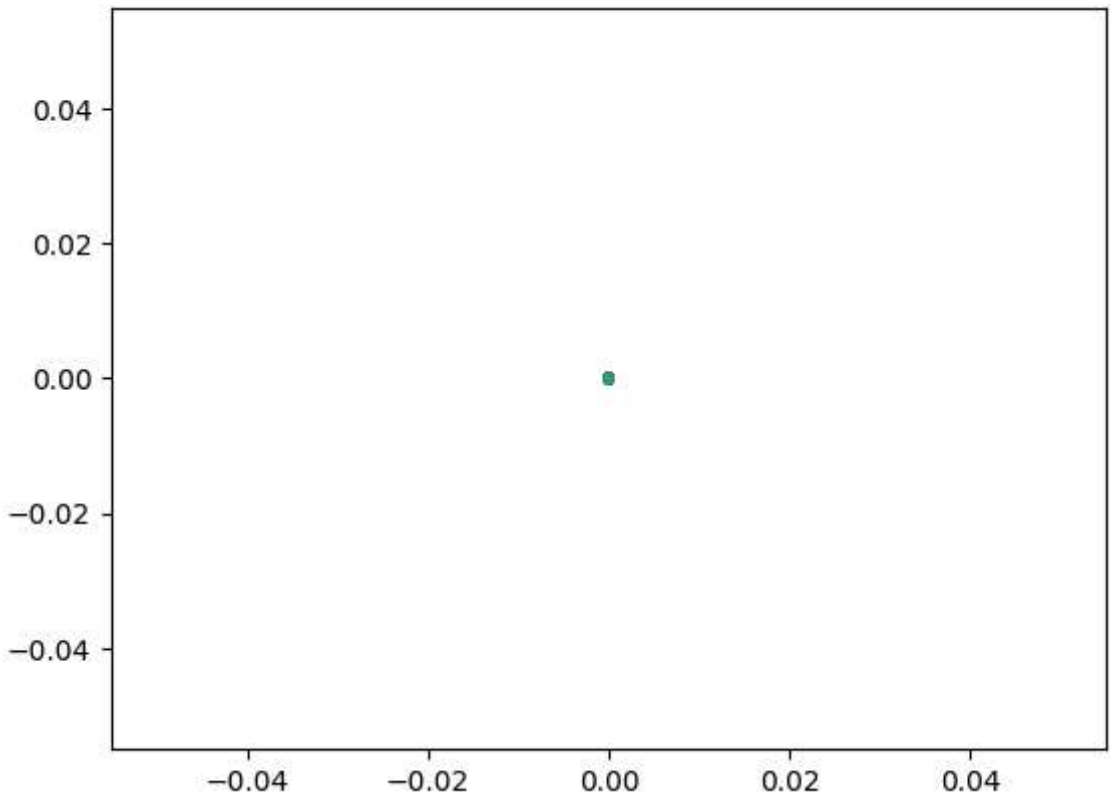
In [21]:
```
gB = GaussBayes()
```

In [22]:
```
gB.fit(X,y)
```

In [23]:
```
y_hat = gB.predict(X)
```

In [24]:
```
plt.figure()
plt.scatter(X[:,0],X[:,1], c=y_hat, s=10, alpha=0.4)
```

Out[24]: <matplotlib.collections.PathCollection at 0x7f305b378e80>



In [25]:
```
accuracy(y,y_hat)
```

Out[25]: 0.9948

K Nearest Neighbors

In [26]:
```
class KNNClassifier():
  def fit(self, X,y):
    self.X = X
    self.y = y

  def predict(self, X, k, epsilon=1e-3):
    N = len(X)
    y_hat = np.zeros(N)

    for i in range(N):
      dist2 = np.sum(((self.X-X[i])**2), axis=1)
      idxt = np.argsort(dist2)[:k]
      gamma_k = 1/(np.sqrt(dist2[idxt]+epsilon))
      y_hat[i] = np.bincount(self.y[idxt], weights=gamma_k).argmax()

    return y_hat
```

In [27]:
```
knn = KNNClassifier()
```

In [28]:
```
knn.fit(X,y)
```

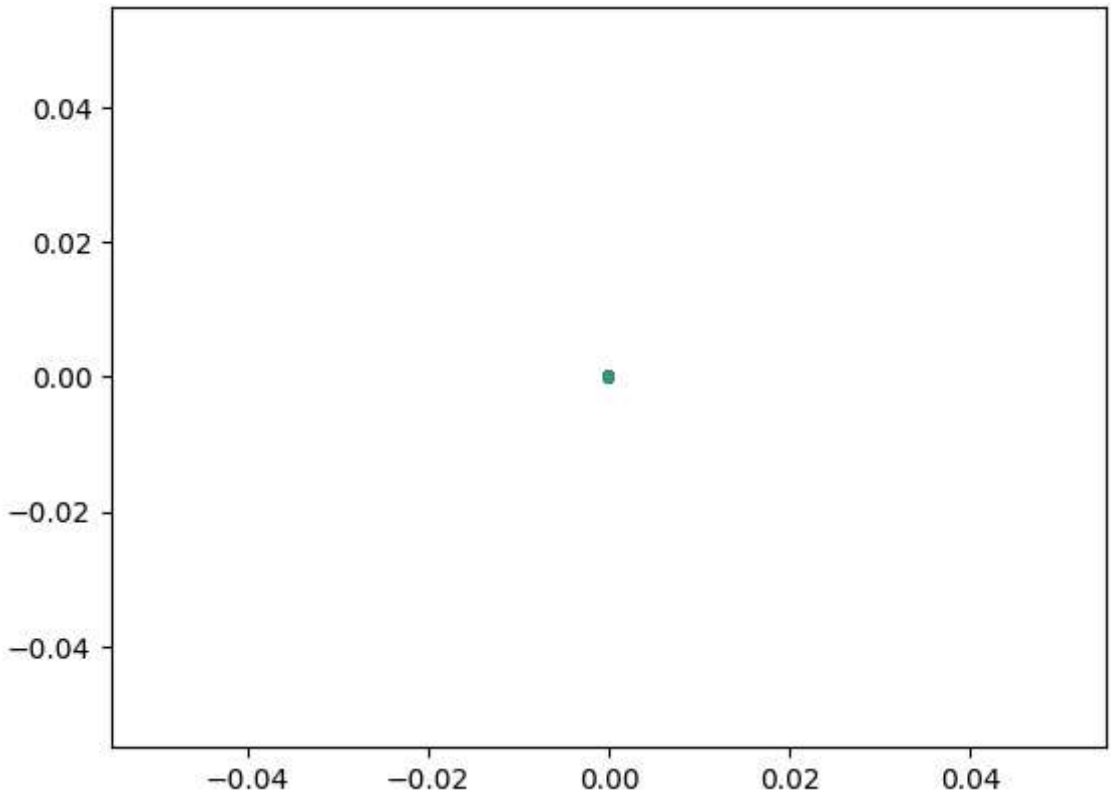
In [29]:
```
X.shape
```

Out[29]:
```
(10000, 784)
```

In [30]:
```
y_hat = knn.predict(X,2)
```

In [31]:
```
plt.figure()
plt.scatter(X[:,0],X[:,1], c=y_hat, s=10, alpha=0.4)
```

Out[31]:
```
<matplotlib.collections.PathCollection at 0x7f305aa56e30>
```



In [32]:
```
accuracy(y,y_hat)
```

Out[32]:
```
1.0
```

In [33]:
```
!jupyter nbconvert --to html MNIST_Test.ipynb
```

```
[NbConvertApp] WARNING | pattern 'MNIST_Test.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include
the error message in the cell output (the default behaviour is to abort conversion).
This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default
basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_form
at=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_form
at=notebook --FilesWriter.build_directory= --ClearOutputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.e
xclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
```

```
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporter.
exclude_input=True --TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the syst
em.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful for t
he HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITIC
AL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'p
ython', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
```

```
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each notebook. To rec
over
                                    previous default behaviour (outputting to the curre
nt
                                    working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-sli
deshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'note
book', 'pdf', 'python', 'rst', 'script', 'slides', 'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file
```

```
> jupyter nbconvert mynotebook.ipynb --stdout

PDF is generated via latex

> jupyter nbconvert mynotebook.ipynb --to pdf

You can get (and serve) a Reveal.js-powered slideshow

> jupyter nbconvert myslides.ipynb --to slides --post serve

Multiple notebooks can be given at the command line in a couple of
different ways:

> jupyter nbconvert notebook*.ipynb
> jupyter nbconvert notebook1.ipynb notebook2.ipynb

or you can specify the notebooks list in a config file, containing::

    c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

> jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```

In [33]: