Load Libraries

In [341…
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Load dataset & data preprocessing

In [342…
```
file_path = "MNIST_train.csv"
df_train = pd.read_csv(file_path)
```

In [343…
```
df_train.head
```

Out[343…
```
<bound method NDFrame.head of        Unnamed: 0  index  labels  0  1  2  3  4
5  6  ...  774  775  776  \
0               0      0       5  0  0  0  0  0  0  0  ...    0    0    0
1               1      1       0  0  0  0  0  0  0  0  ...    0    0    0
2               2      2       4  0  0  0  0  0  0  0  ...    0    0    0
3               3      3       1  0  0  0  0  0  0  0  ...    0    0    0
4               4      4       9  0  0  0  0  0  0  0  ...    0    0    0
...           ...    ...     ... .. .. .. .. .. .. .. ..  ...  ...  ...  ...
59995       59995  59995       8  0  0  0  0  0  0  0  ...    0    0    0
59996       59996  59996       3  0  0  0  0  0  0  0  ...    0    0    0
59997       59997  59997       5  0  0  0  0  0  0  0  ...    0    0    0
59998       59998  59998       6  0  0  0  0  0  0  0  ...    0    0    0
59999       59999  59999       8  0  0  0  0  0  0  0  ...    0    0    0

       777  778  779  780  781  782  783
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
...    ...  ...  ...  ...  ...  ...  ...
59995    0    0    0    0    0    0    0
59996    0    0    0    0    0    0    0
59997    0    0    0    0    0    0    0
59998    0    0    0    0    0    0    0
59999    0    0    0    0    0    0    0

[60000 rows x 787 columns]>
```

In [344…
```
file_path = "MNIST_test.csv"
df_test = pd.read_csv(file_path)
```

In [345…
```
df_test.head
```

Out[345…

```
<bound method NDFrame.head of       Unnamed: 0  index  labels  0  1  2  3  4  5
6  ...  774  775  776  777  \
0              0      0       7  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
2              2      2       1  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
4              4      4       4  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
9996        9996   9996       3  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
9998        9998   9998       5  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

      778  779  780  781  782  783
0       0    0    0    0    0    0
1       0    0    0    0    0    0
2       0    0    0    0    0    0
3       0    0    0    0    0    0
4       0    0    0    0    0    0
...   ...  ...  ...  ...  ...  ...
9995    0    0    0    0    0    0
9996    0    0    0    0    0    0
9997    0    0    0    0    0    0
9998    0    0    0    0    0    0
9999    0    0    0    0    0    0

[10000 rows x 787 columns]>
```

In [346…

```
df_train.shape
```

Out[346…

```
(60000, 787)
```

In [347…

```
df_test.shape
```

Out[347…

```
(10000, 787)
```

In [348…

```
df_test.info
```

Out[348…

```
<bound method DataFrame.info of       Unnamed: 0  index  labels  0  1  2  3  4
5  6  ...  774  775  776  777  \
0              0      0       7  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
2              2      2       1  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
4              4      4       4  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
9996        9996   9996       3  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
9998        9998   9998       5  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

      778  779  780  781  782  783
0       0    0    0    0    0    0
1       0    0    0    0    0    0
2       0    0    0    0    0    0
3       0    0    0    0    0    0
4       0    0    0    0    0    0
...   ...  ...  ...  ...  ...  ...
9995    0    0    0    0    0    0
9996    0    0    0    0    0    0
9997    0    0    0    0    0    0
9998    0    0    0    0    0    0
9999    0    0    0    0    0    0

[10000 rows x 787 columns]>
```

In [349…

```
df_test.describe
```

Out[349…

```
<bound method NDFrame.describe of       Unnamed: 0  index  labels  0  1  2  3
4  5  6  ...  774  775  776  777  \
0              0      0       7  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
2              2      2       1  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
4              4      4       4  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
9996        9996   9996       3  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
9998        9998   9998       5  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

      778  779  780  781  782  783
0       0    0    0    0    0    0
1       0    0    0    0    0    0
2       0    0    0    0    0    0
3       0    0    0    0    0    0
4       0    0    0    0    0    0
...   ...  ...  ...  ...  ...  ...
9995    0    0    0    0    0    0
9996    0    0    0    0    0    0
9997    0    0    0    0    0    0
9998    0    0    0    0    0    0
9999    0    0    0    0    0    0

[10000 rows x 787 columns]>
```

In [350…

```
df_train
```

Out[350…

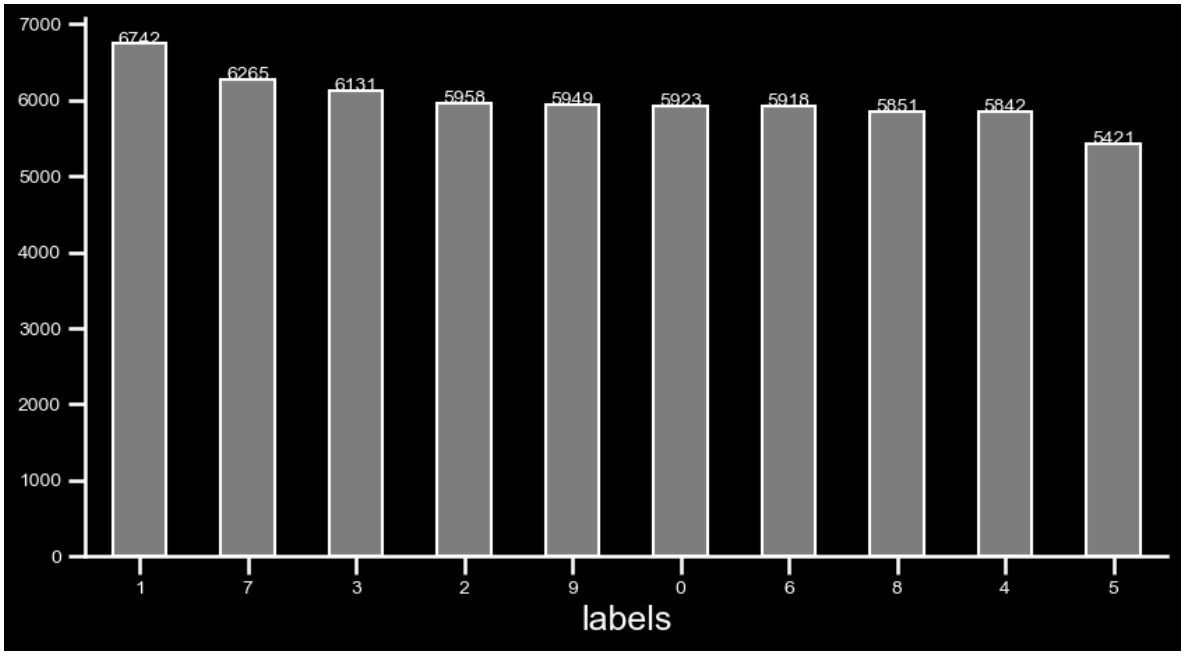
	Unnamed: 0	index	labels	0	1	2	3	4	5	6	…	774	775	776	777	778
0	0	0	5	0	0	0	0	0	0	0	…	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	…	0	0	0	0	0
2	2	2	4	0	0	0	0	0	0	0	…	0	0	0	0	0
3	3	3	1	0	0	0	0	0	0	0	…	0	0	0	0	0
4	4	4	9	0	0	0	0	0	0	0	…	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
59995	59995	59995	8	0	0	0	0	0	0	0	…	0	0	0	0	0
59996	59996	59996	3	0	0	0	0	0	0	0	…	0	0	0	0	0
59997	59997	59997	5	0	0	0	0	0	0	0	…	0	0	0	0	0
59998	59998	59998	6	0	0	0	0	0	0	0	…	0	0	0	0	0
59999	59999	59999	8	0	0	0	0	0	0	0	…	0	0	0	0	0

60000 rows × 787 columns

Checking Target Imbalance

In [351…

```
train_x = df_train.drop('labels',axis=1)
train_y = df_train['labels']

sns.set(style="ticks", context="talk",font_scale = 1)
plt.style.use("dark_background")
plt.figure(figsize = (10,5))
ax = train_y.value_counts().sort_values(ascending=False).plot(kind='bar',
                                                              grid = F
                                                              fontsize
                                                              color='g

plt.xticks(rotation=0)
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+ p.get_width() / 2., height, height, ha = 'center', size =
sns.despine()
```



Observation

All numbers appear to be balanced. 1 is the most and 5 is the least.

In [352…
```
df_train = df_train.to_numpy()
```

In [353…
```
df_train
```

Out[353…
```
array([[    0,     0,     5, ...,     0,     0,     0],
       [    1,     1,     0, ...,     0,     0,     0],
       [    2,     2,     4, ...,     0,     0,     0],
       ...,
       [59997, 59997,     5, ...,     0,     0,     0],
       [59998, 59998,     6, ...,     0,     0,     0],
       [59999, 59999,     8, ...,     0,     0,     0]],
      shape=(60000, 787))
```

In [354…
```
y_train = df_train[:,2]
```

In [355…
```
y_train.shape
```

Out[355…
```
(60000,)
```

In [356…
```
K = set(y_train)
```

In [357…
```
K
```

Out[357…
```
{np.int64(0),
 np.int64(1),
 np.int64(2),
 np.int64(3),
 np.int64(4),
 np.int64(5),
 np.int64(6),
 np.int64(7),
 np.int64(8),
 np.int64(9)}
```

In [358…
```
X_train = df_train[:,3:]
```

In [359…
```
X_train
```

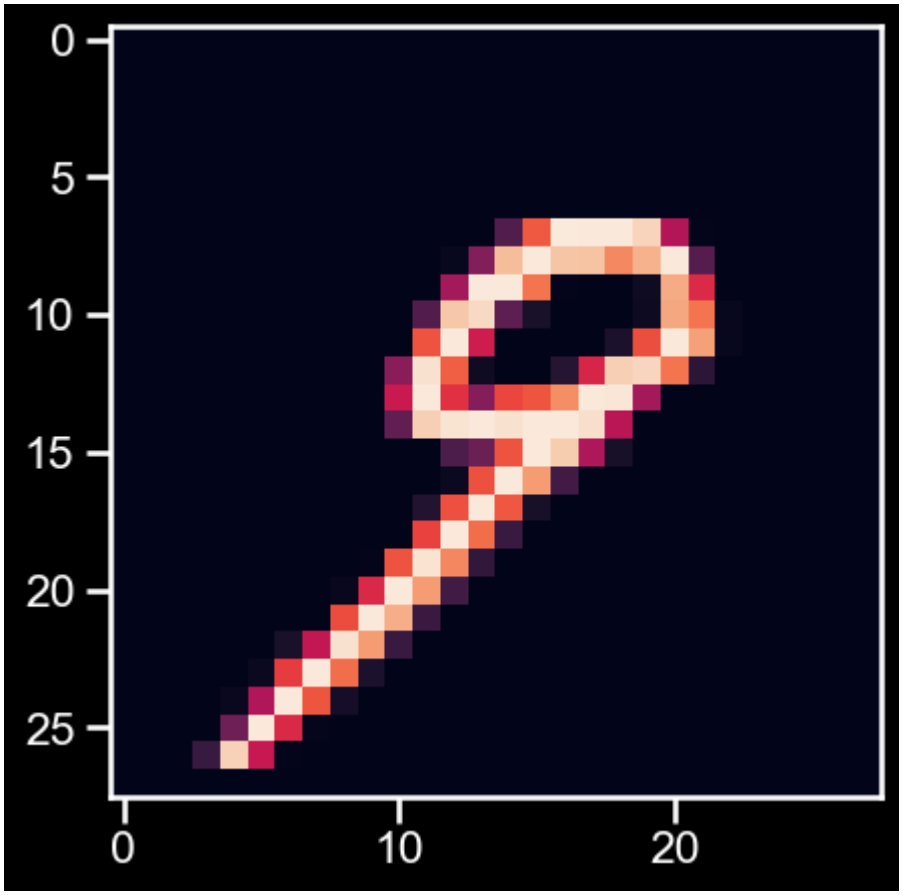
Out[359…
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], shape=(60000, 784))
```

In [360…
```
X_train.shape
```

Out[360…
```
(60000, 784)
```

In [361…
```
X_train1 = X_train[501]

img = X_train1.reshape(28,28)
#sns.set(style="ticks", context="talk",font_scale = 1)
#plt.style.use("dark_background")
fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot(111)
ax.imshow(img)
```

Out[361…
```
<matplotlib.image.AxesImage at 0x21c8d02a240>
```



Observation

Each number image in MNIST is only a list of 28 * 28 dimensional numbers.

In [362…
```
df_test = df_test.to_numpy()
```

In [363…
```
df_test
```

Out[363…
```
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]], shape=(10000, 787))
```

In [364…
```
y_test = df_test[:,2]
```

In [365…
```
X_test = df_test[:,3:]
```

In [366…
```
df_test
```

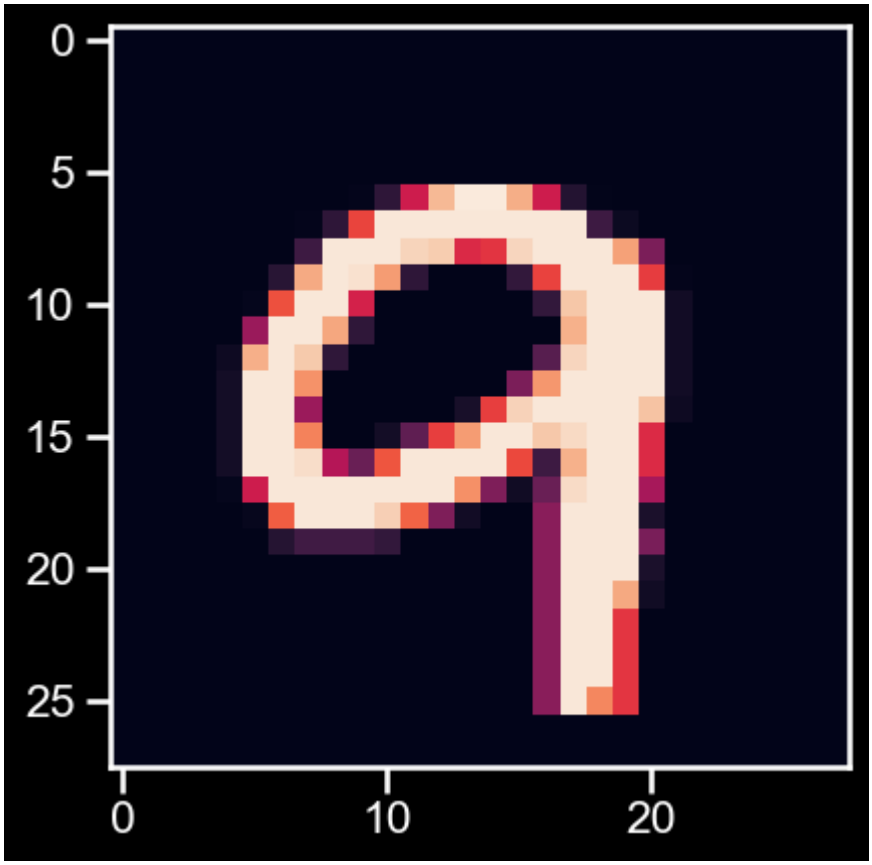
Out[366…
```
array([[   0,    0,    7, ...,    0,    0,    0],
       [   1,    1,    2, ...,    0,    0,    0],
       [   2,    2,    1, ...,    0,    0,    0],
       ...,
       [9997, 9997,    4, ...,    0,    0,    0],
       [9998, 9998,    5, ...,    0,    0,    0],
       [9999, 9999,    6, ...,    0,    0,    0]], shape=(10000, 787))
```

In [367…
```
X_test1 = X_test[501]
```

In [368…
```
x_test_reshape= X_test1.reshape(28,28)
```

In [369…
```
plt.imshow(x_test_reshape)
```

Out[369…
```
<matplotlib.image.AxesImage at 0x21c8cfe0f20>
```

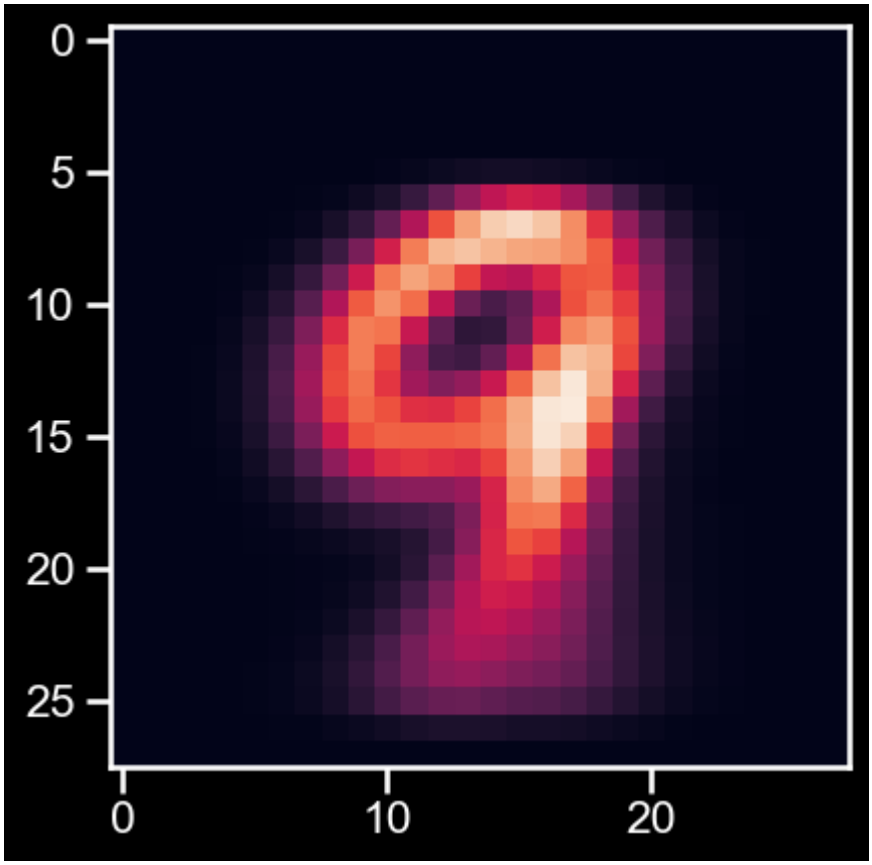


In [370…
```
y_test[501]
```

Out[370…
```
np.int64(9)
```

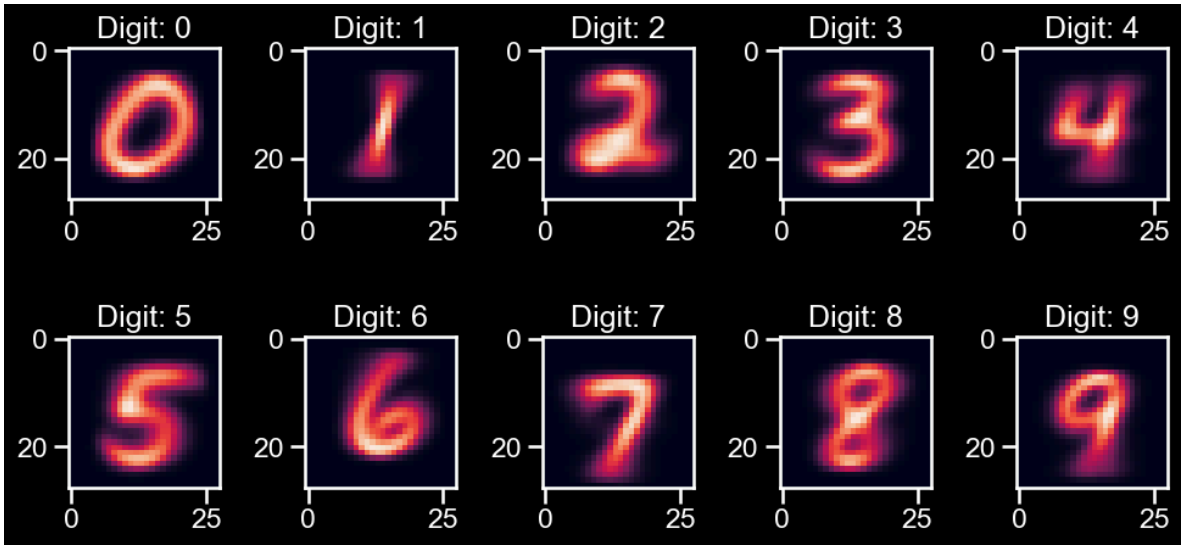
In [371…
```
def show_me(X):
  plt.imshow(X.reshape(28,28))

def show_me_all_mean(X,y,k):
  show_me(sum(X[y==k, :]/len(X[y==k,:])))
```

In [372…
```
show_me_all_mean(X_train,y_train,9)
```



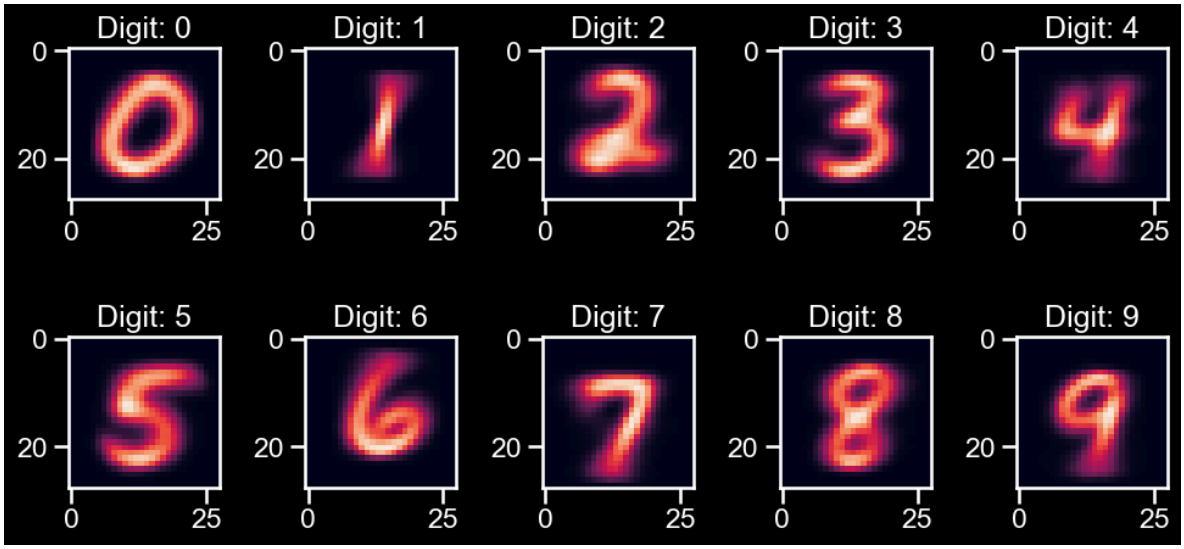
In [373…

```
plt.figure(figsize=(10, 5))
for digit in range(10):
    plt.subplot(2, 5, digit + 1)
    show_me_all_mean(X_train, y_train, digit)
    plt.title(f'Digit: {digit}')
plt.tight_layout()
plt.show()
```



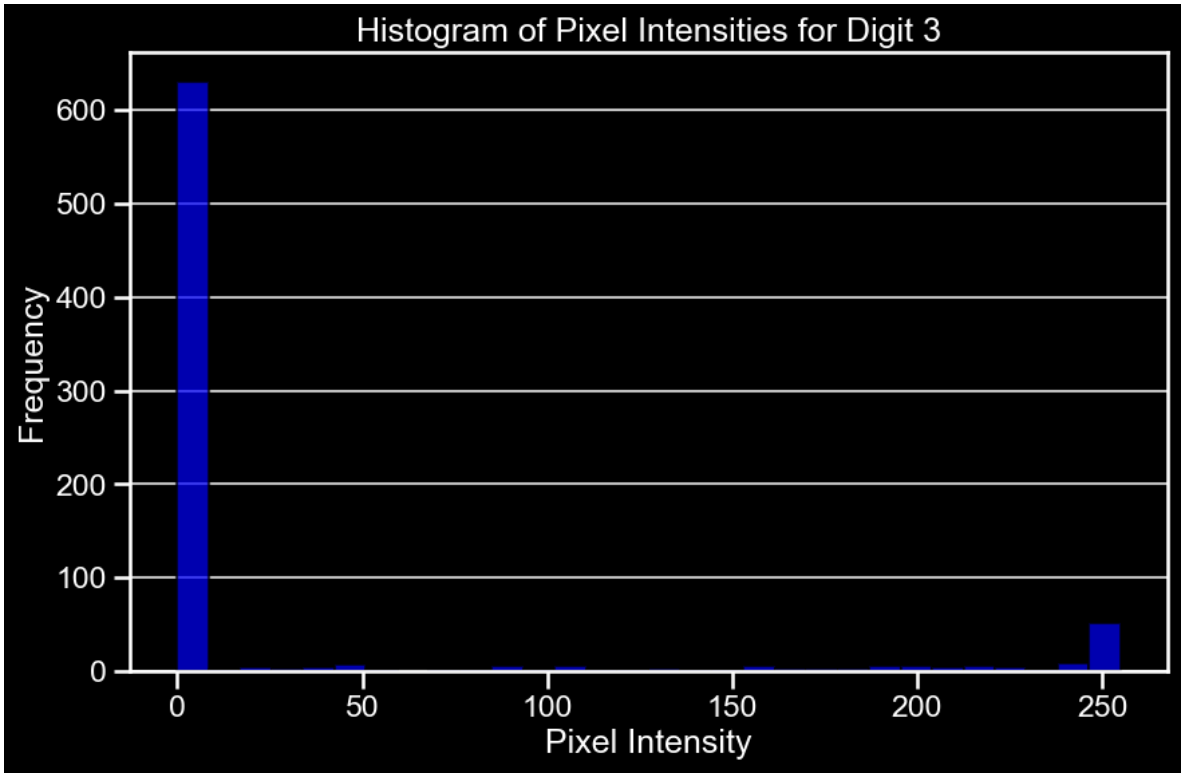
In [374…

```
plt.figure(figsize=(10, 5))
for digit in range(10):
    plt.subplot(2, 5, digit + 1)
    show_me_all_mean(X_test, y_test, digit)
    plt.title(f'Digit: {digit}')
plt.tight_layout()
plt.show()
```



In [375…
```
sample_image_train = X_train[581]
```

In [376…
```
plt.figure(figsize=(10, 6))
plt.hist(sample_image_train, bins=30, color='blue', alpha=0.7, edgecolor='black'
plt.title(f"Histogram of Pixel Intensities for Digit {y_train[581]}")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.grid(axis='y', alpha=0.75)
plt.show()
```



In [377…
```
def CoinFlip(y):
  y_hat=np.zeros(len(y))
  for i in range(len(y)):
    flip = np.random.randn(1)
    if flip>0:
      y_hat[i]=1

  return y_hat
```

Naive Buyes Classifier

In [378…
```
from scipy.stats import multivariate_normal as mvn
```

In [379…
```
class GauseNB():
    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,:]
            self.likelihoods[k]= {"mean":X_k.mean(axis=0),"cov":X_k.var(axis=0)+
            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.prior

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

In [380…
```
def accuracy(y, y_hat):
    return np.mean(y==y_hat)
```

Train model

In [381…
```
X_train = X_train/255
X_test = X_test/255
```

In [382…
```
gnb_train = GauseNB()
```

In [384…
```
gnb_train.fit(X_train,y_train)
```

In [385…
```
y_train_hat = gnb_train.predict(X_train)
```

In [386…
```
#plt.figure(figsize = (10,6))
#plt.scatter(X_train[:,0], X_train[:,1],c=y_train,alpha =0.5,s=10)
```

In [387…
```
#plt.figure(figsize = (10,6))
#plt.scatter(X_train[:,0], X_train[:,1],c=y_train_hat,alpha =0.5,s=10)
```

In [388…
```
accuracy(y_train,y_train_hat)
```

Out[388…
```
np.float64(0.7682333333333333)
```

test data

In [390…
```
y_test_hat = gnb_train.predict(X_test)
```

In [391…
```
accuracy(y_test,y_test_hat)
```

Out[391…
```
np.float64(0.7746)
```

Non Naive Gausian Bayes

In [400…
```
class GauseBayes():
    def fit(self, X,y, epsilon = 1e-1):
        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":X_k.mean(axis=0),
                                  "cov":(1/(N_k-1))*np.matmul((X_k-mu_k).T,X_k-m
            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.prior

        return P_hat.argmax(axis=1)

    def confusion_matrix_manual(self, y_true, y_pred):
        classes = np.unique(y_true)
        matrix = np.zeros((len(classes), len(classes)), dtype=int)

        class_to_index = {cls: idx for idx, cls in enumerate(classes)}
        for actual, predicted in zip(y_true, y_pred):
            matrix[class_to_index[actual], class_to_index[predicted]] += 1

        return matrix
```

In [401…
```
gbays_naive_train = GauseBayes()
```

In [402…
```
gbays_naive_train.fit(X_train,y_train)
```

In [403…
```
y_hat_train_gbays = gbays_naive_train.predict(X_train)
```

In [404…
```
accuracy(y_train,y_hat_train_gbays)
```

Out[404…
```
np.float64(0.9549333333333333)
```

In [405…
```
con_matrix_gb1 = gbays_naive_train.confusion_matrix_manual(y_train, y_hat_train_
print(con_matrix_gb1)
```

```
[[5866   14    1    1    2    5   15    1   13    5]
 [   0 6683   33    4    5    0    1    3    9    4]
 [  27  114 5588   27   27    0   18   52   86   19]
 [  14   69   45 5775    0   29    2   52   70   75]
 [   6   69   10    0 5523    0   19   16    6  193]
 [  31   35    8   56    3 5102   66    1   44   75]
 [  20   45    1    0    3   53 5783    1   12    0]
 [  10   75   16    3   12    2    0 5999   11  137]
 [  30  259   24   84   11   31   23   15 5272  102]
 [  21   53    9   48   11    6    1   69   26 5705]]
```

Test Data

In [406…
```
y_hat_test_gbays = gbays_naive_train.predict(X_test)
```

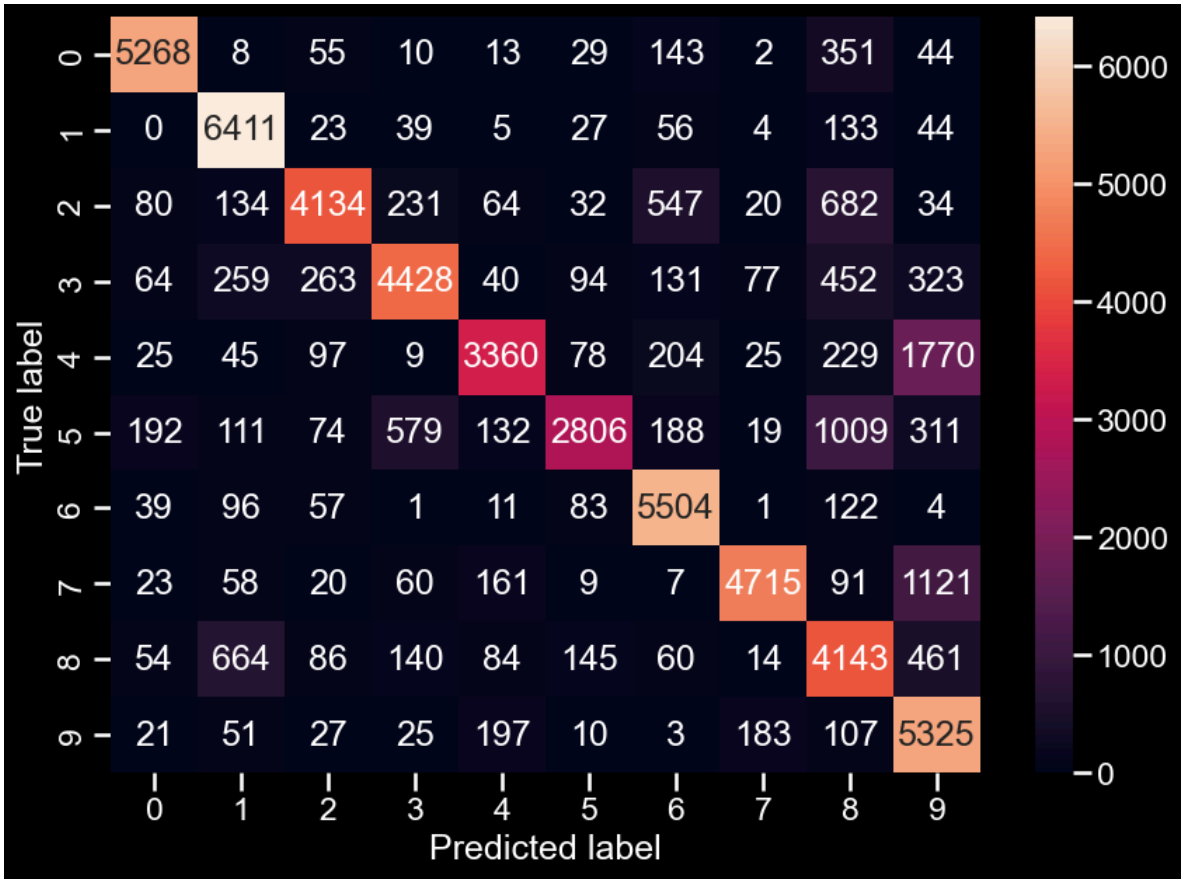
In [407…
```
accuracy(y_test,y_hat_test_gbays)
```

Out[407…
```
np.float64(0.9542)
```

Confusion Matrix

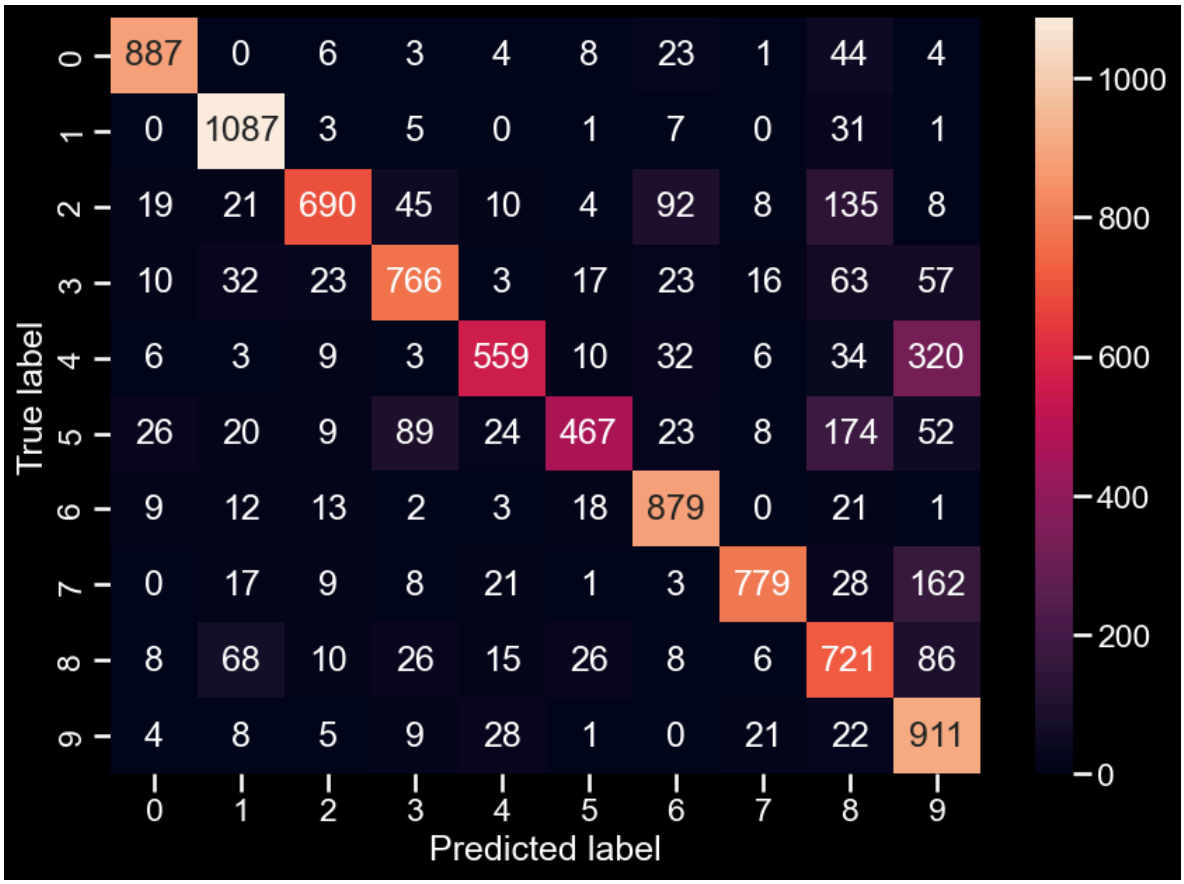
In [408…
```
plt.figure(figsize=(10,7))
y_actu = pd.Series(y_train, name='Actual')
y_pred = pd.Series(y_train_hat, name='Predicted')
cm = pd.crosstab(y_actu, y_pred)
ax = sns.heatmap(cm, annot=True, fmt="d")
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Out[408…
```
Text(0.5, 34.083333333333314, 'Predicted label')
```



In [409…
```
plt.figure(figsize=(10,7))
ytest_actu = pd.Series(y_test, name='Actual')
ytest_pred = pd.Series(y_test_hat, name='Predicted')
cm = pd.crosstab(ytest_actu, ytest_pred)
ax = sns.heatmap(cm, annot=True, fmt="d")
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Out[409… `Text(0.5, 34.083333333333314, 'Predicted label')`



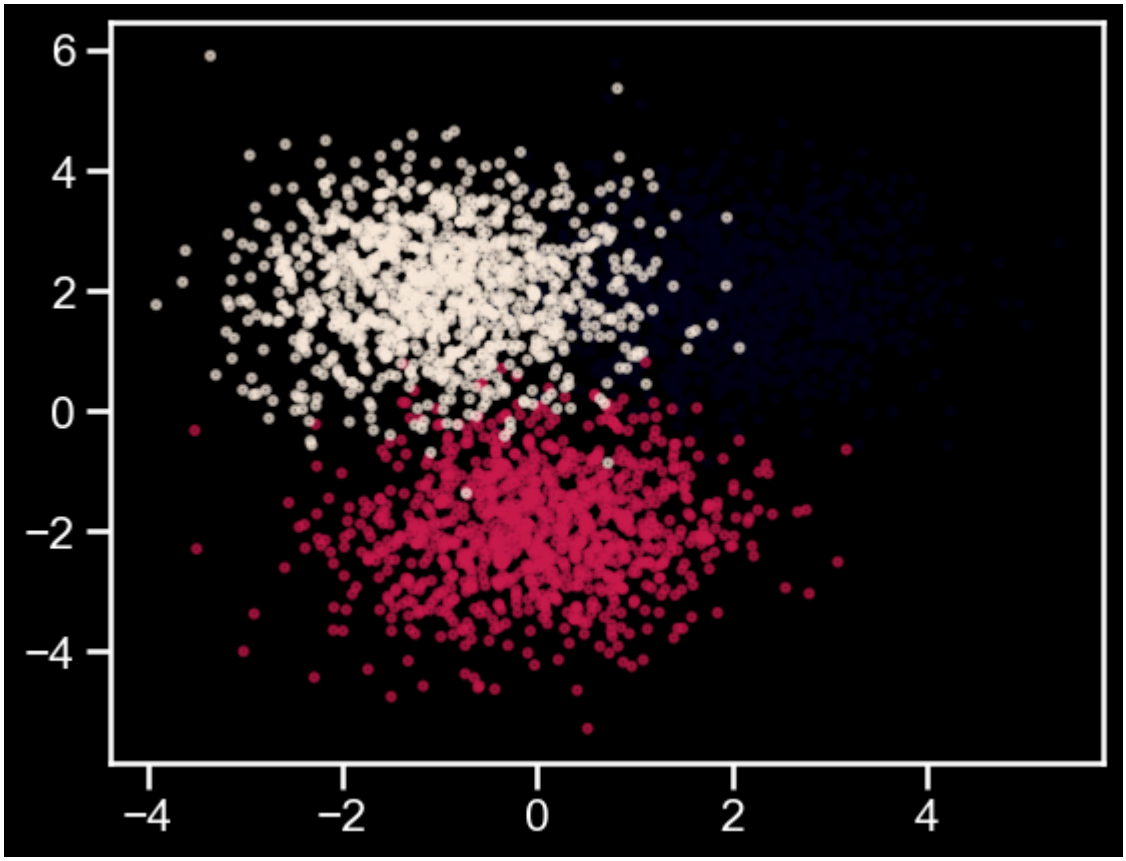
K Nearest Neighbors

Synthetic Dataset Generation

In [410…
```
D = 2 #Dimensions
C = 3 # Classes
N = int(C*1e3) # Number of points/rows

X0 = np.random.randn((N//C), D) + np.array([2,2])
X1 = np.random.randn((N//C), D)+ np.array([0,-2])
X2 = np.random.randn((N//C), D)+ np.array([-1,2])

X = np.vstack((X0, X1, X2))
y = np.array([0]*(N//C)+ [1]*(N//C)+ [2]*(N//C))
```

In [411…
```
plt.figure()
plt.scatter(X[:,0],X[:,1],c=y,alpha=0.6,s=8)
```

Out[411… `<matplotlib.collections.PathCollection at 0x21c8dd45fa0>`



KNN Classifier Class

In [412…

```
class KNNClassifier():
    def fit(self, X, y):# lazy learner just perform operation
        self.X = X
        self.y = y

    def predict(self, X, K , epsilon=1e-3):
        N = len(X) #number of observations
        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] # give a list of index from lowest dist
            gamma_k = 1/(np.sqrt(dist2[idxt] + epsilon))

            y_hat[i] =np.bincount(self.y[idxt],weights=gamma_k).argmax() # to gi

        return y_hat # return outside for loop
```

In [413…

```
knn_instance = KNNClassifier()
```

In [414…

```
mini_train_data, mini_train_labels = X_train[:10000], y_train[:10000]
```

In [415…

```
mini_train_data
```

Out[415…
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], shape=(10000, 784))
```

In [442…
```
mini_test_data, mini_test_labels = X_test[:10000], y_test[:10000]
```

In [417…
```
knn_instance.fit(mini_train_data,mini_train_labels)
```

In [423…
```
y_hat_knn = knn_instance.predict(y_test,K=200)
```

In [433…
```
mini_train_data[:,0]
```

Out[433…
```
array([0., 0., 0., ..., 0., 0., 0.], shape=(10000,))
```

In [435…
```
y_hat_knn
```

Out[435…
```
array([0., 0., 0., ..., 0., 0., 0.], shape=(10000,))
```

In [436…
```
accuracy(mini_train_data[:,0],y_hat_knn)
```

Out[436…
```
np.float64(0.902)
```

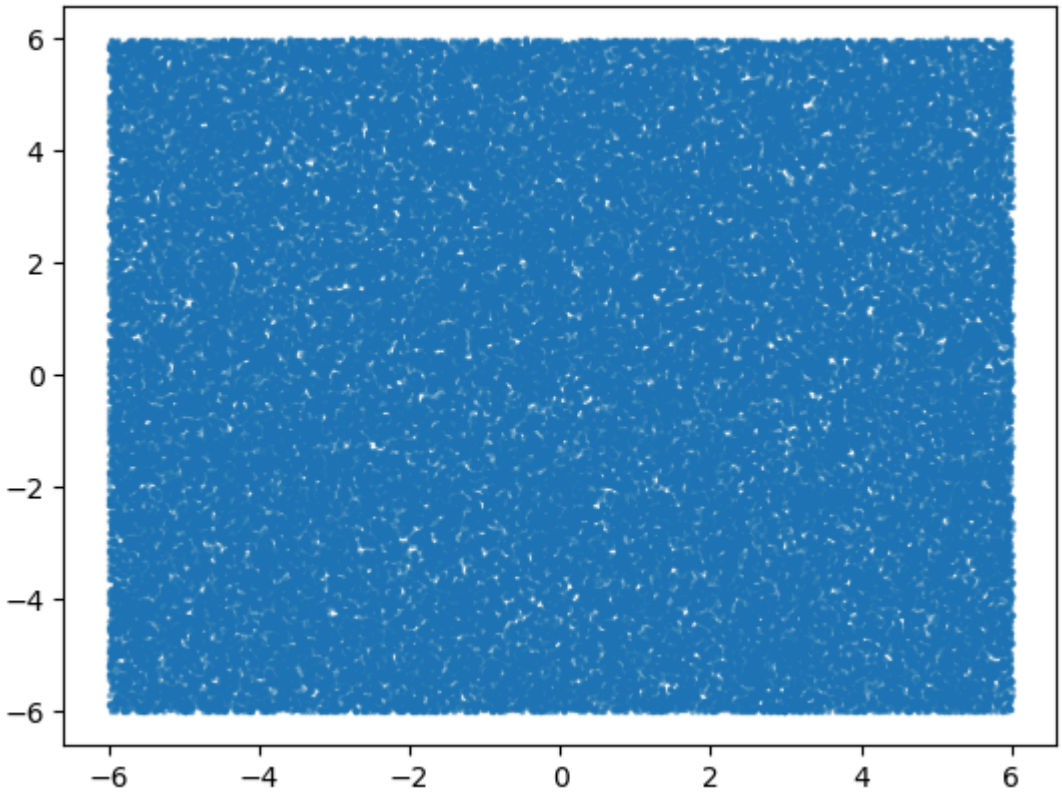
In [443…
```
accuracy(mini_test_data[:,0],y_hat_knn)
```

Out[443…
```
np.float64(0.902)
```

In [104…
```
X_vis = np.random.uniform(-6,6,(int(N*30),D))
```

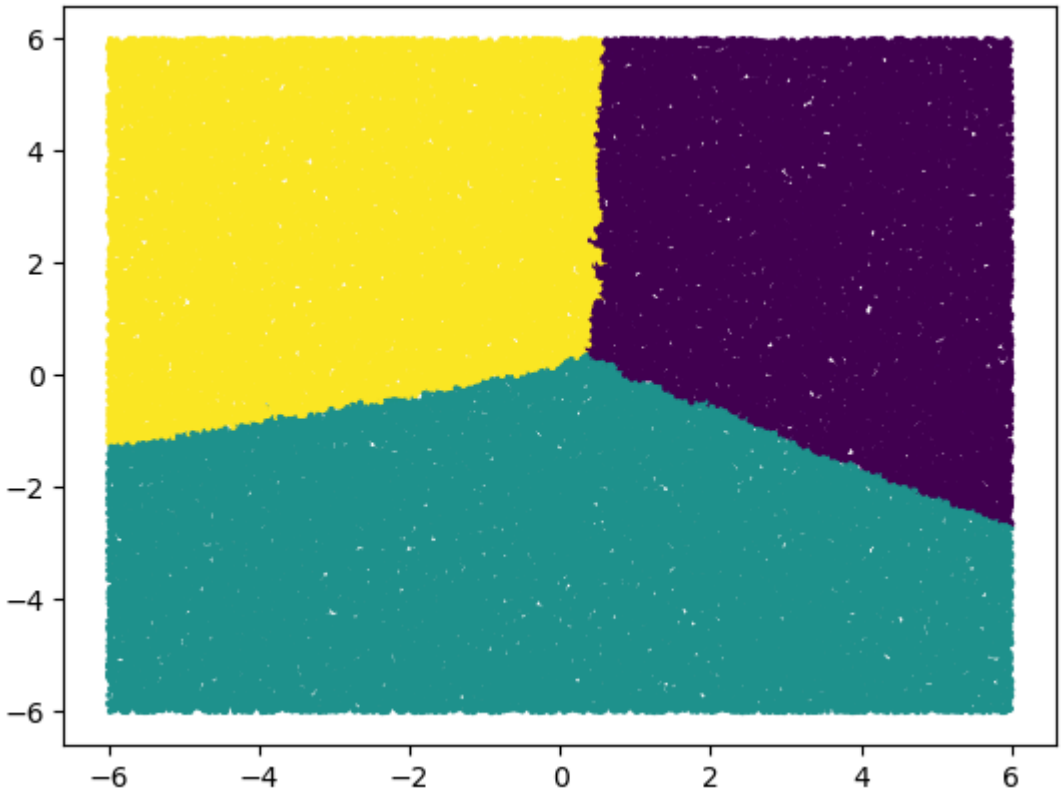
In [105…
```
plt.scatter(X_vis[:,0],X_vis[:,1],s=2,alpha=0.5)
```

Out[105…
```
<matplotlib.collections.PathCollection at 0x21c4915b0b0>
```



In [106…
```
y_hat_vis = knn_instance.predict(X_vis,K=200)
plt.figure()
plt.scatter(X_vis[:,0],X_vis[:,1],c=y_hat_vis,s=2)
```

Out[106… `<matplotlib.collections.PathCollection at 0x21c490c8d40>`



In [ ]: