STOR 235 — Lab 3

In this lab, you will apply support vector classifiers to real data. You will also learn a bit about handling tabular data using the `pandas` package and data visualization using the `seaborn` package.

Instructions

There are two parts to this lab:

1. The wine dataset;
2. The MNIST handwriting recognition dataset.

Each part will be composed of a number of tasks for you to complete. The tasks are labeled in bold as **Problem 1**, **Problem 2**, and so on. Please make sure that you do not forget to complete any of the problems. Each problem will be worth 4 points.

There is a single code block in under this introductory material that imports various things you will use later. Please run it.

Important Information About Submitting Your Assignment

On Gradescope, this assignment is broken into 4 questions, representing each of the four parts. Please upload your assignment as a PDF file, and make sure to select to all relevant pages for each part.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_openml
from sklearn.neighbors import KNeighborsClassifier
```

Part 1: Wine Dataset

The `wine` dataset comes with the scikit-learn package, which gives a special function to load it. The following code block loads the dataset and converts it to a `pandas` dataframe, which is essentially a spreadsheet-type data structure. Each row is a datapoint, and each column is an attribute.

By doing this, we can use several useful features of the `pandas` package to explore the data.

```
# Load the wine dataset
wine = load_wine()

# Convert the dataset to a pandas DataFrame for easier manipulation
wine_df = pd.DataFrame(data=wine.data, columns=wine.feature_names)

#initially, the dataframe just has the 13 wine attributes
#the following line adds a new column with the class ("target") label
wine_df['target'] = wine.target
```

The following command prints basic information about the data. It tells us how many columns there are and what kind of data they contain.

```
print(wine_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  target                        178 non-null    int64
dtypes: float64(13), int64(1)
memory usage: 19.6 KB
None
```

The following command prints the first six rows. If you want to see more, you can of course change the number 6 to something else.

```
print(wine_df.head(11))
```

```
    alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0     14.23        1.71  2.43               15.6      127.0           2.80
1     13.20        1.78  2.14               11.2      100.0           2.65
2     13.16        2.36  2.67               18.6      101.0           2.80
3     14.37        1.95  2.50               16.8      113.0           3.85
4     13.24        2.59  2.87               21.0      118.0           2.80
5     14.20        1.76  2.45               15.2      112.0           3.27
6     14.39        1.87  2.45               14.6       96.0           2.50
7     14.06        2.15  2.61               17.6      121.0           2.60
8     14.83        1.64  2.17               14.0       97.0           2.80
9     13.86        1.35  2.27               16.0       98.0           2.98
10    14.10        2.16  2.30               18.0      105.0           2.95

    flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0         3.06                  0.28             2.29             5.64  1.04
1         2.76                  0.26             1.28             4.38  1.05
2         3.24                  0.30             2.81             5.68  1.03
3         3.49                  0.24             2.18             7.80  0.86
4         2.69                  0.39             1.82             4.32  1.04
5         3.39                  0.34             1.97             6.75  1.05
6         2.52                  0.30             1.98             5.25  1.02
7         2.51                  0.31             1.25             5.05  1.06
8         2.98                  0.29             1.98             5.20  1.08
9         3.15                  0.22             1.85             7.22  1.01
10        3.32                  0.22             2.38             5.75  1.25

    od280/od315_of_diluted_wines  proline  target
0                           3.92   1065.0       0
1                           3.40   1050.0       0
2                           3.17   1185.0       0
3                           3.45   1480.0       0
4                           2.93    735.0       0
5                           2.85   1450.0       0
6                           3.58   1290.0       0
7                           3.58   1295.0       0
8                           2.85   1045.0       0
9                           3.55   1045.0       0
10                          3.17   1510.0       0
```

We next look at basic summary statistics, like minimum values, maximum values, and quantiles.

```
print(wine_df.describe())
```

```
          alcohol  malic_acid         ash  alcalinity_of_ash   magnesium  \
count  178.000000  178.000000  178.000000         178.000000  178.000000
mean    13.000618    2.336348    2.366517          19.494944   99.741573
std      0.811827    1.117146    0.274344           3.339564   14.282484
min     11.030000    0.740000    1.360000          10.600000   70.000000
25%     12.362500    1.602500    2.210000          17.200000   88.000000
50%     13.050000    1.865000    2.360000          19.500000   98.000000
75%     13.677500    3.082500    2.557500          21.500000  107.000000
max     14.830000    5.800000    3.230000          30.000000  162.000000

       total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  \
count     178.000000  178.000000            178.000000       178.000000
mean        2.295112    2.029270              0.361854         1.590899
std         0.625851    0.998859              0.124453         0.572359
min         0.980000    0.340000              0.130000         0.410000
25%         1.742500    1.205000              0.270000         1.250000
50%         2.355000    2.135000              0.340000         1.555000
75%         2.800000    2.875000              0.437500         1.950000
max         3.880000    5.080000              0.660000         3.580000

       color_intensity         hue  od280/od315_of_diluted_wines      proline  \
count       178.000000  178.000000                    178.000000   178.000000
```

```
mean          5.058090    0.957449                         2.611685    746.893258
std           2.318286    0.228572                         0.709990    314.907474
min           1.280000    0.480000                         1.270000    278.000000
25%           3.220000    0.782500                         1.937500    500.500000
50%           4.690000    0.965000                         2.780000    673.500000
75%           6.200000    1.120000                         3.170000    985.000000
max          13.000000    1.710000                         4.000000   1680.000000

           target
count  178.000000
mean     0.938202
std      0.775035
min      0.000000
25%      0.000000
50%      1.000000
75%      2.000000
max      2.000000
```

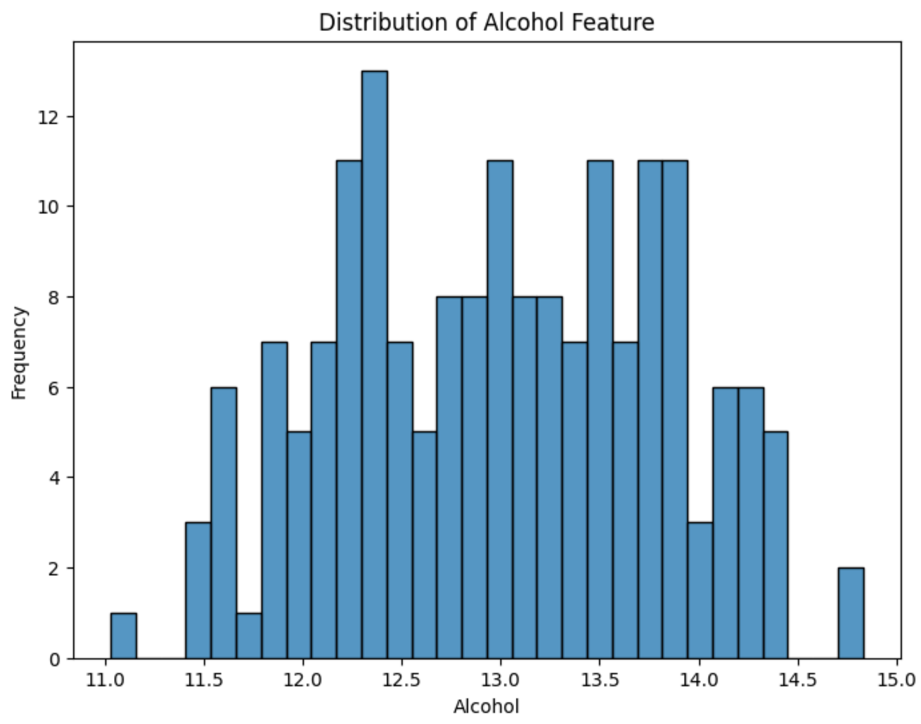
The following code counts the number of wines in each of the three classes.

```
print(wine_df['target'].value_counts())
```

```
target
1    71
0    59
2    48
Name: count, dtype: int64
```

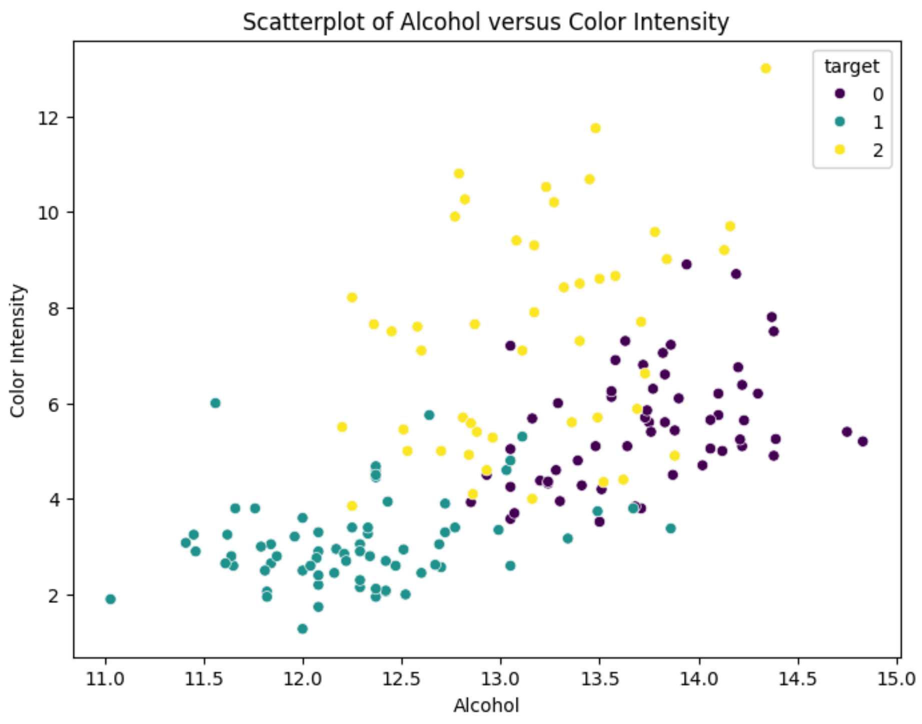
The following code plots a histogram of the `alcohol` values using the function `hist_plot` from `seaborn`. This package provides many fun possibilities for data visualization. See the seaborn gallery for more.

```
plt.figure(figsize=(8, 6))
sns.histplot(wine_df['alcohol'],  bins=30)
plt.title('Distribution of Alcohol Feature')
plt.xlabel('Alcohol')
plt.ylabel('Frequency')
plt.show()
```



Next, we produce a scatterplot using `seaborn` based on two of the features.

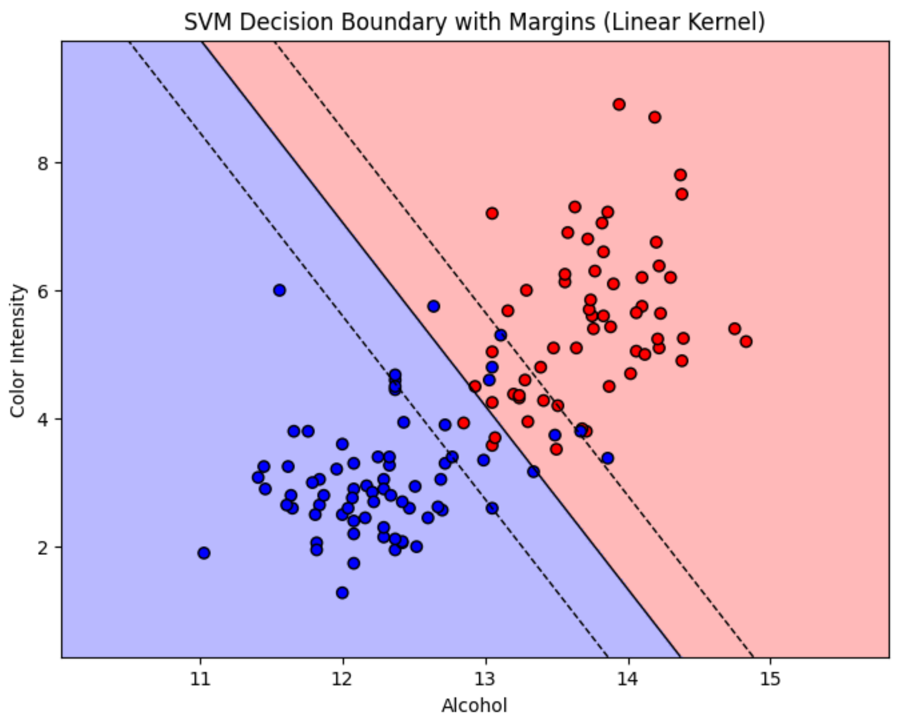
```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='alcohol', y='color_intensity', hue='target', data=wine_df, palette='viridis')
plt.title('Scatterplot of Alcohol versus Color Intensity')
plt.xlabel('Alcohol')
plt.ylabel('Color Intensity')
plt.show()
```



In the next code block, I've written some code that trains a SVC on just two of the classes and two of the features, to make visualization easy. Then I've added some custom plotting code that shows the decision boundary and the margin. Note that the dots for the two classes are the same as those in the previous figure (perhaps with a little rescaling).

The plotting code is messy and you can ignore it. But you should pay careful attention to how the SVC is trained. First, `svm_binary` is created with certain options, then it is fit on the data. Later, `predict` is called to get the predictions from it. (Ignore the `ravel` stuff; I just did this to get the plotting to work. A clearer example of prediction is below.)

```
# Filter the dataset to include only classes 0 and 1
wine_filtered = wine_df[wine_df['target'] != 2]

# Select features for training (using 'alcohol' and 'color_intensity' as features)
X = wine_filtered[['alcohol', 'color_intensity']].values
y = wine_filtered['target'].values

# Train the SVM with a linear kernel
svm_binary = SVC(C=1, kernel='linear')
svm_binary.fit(X, y)

# Create a mesh to plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# Predict class labels for each point in the mesh (for solid color regions)
Z = svm_binary.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Compute decision function values (for margin and decision boundary)
decision_values = svm_binary.decision_function(np.c_[xx.ravel(), yy.ravel()])
decision_values = decision_values.reshape(xx.shape)

# Plot the decision boundary with solid regions and margin lines
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))  # Solid colors
plt.contour(xx, yy, decision_values, levels=[-1, 0, 1],
            linestyles=['--', '-', '--'], colors='k', linewidths=1)  # Margins and decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(['red', 'blue']), edgecolors='k')  # Data points
plt.title('SVM Decision Boundary with Margins (Linear Kernel)')
plt.xlabel('Alcohol')
plt.ylabel('Color Intensity')
plt.show()
```



Now we use all features to train the SVC. (Since this is 13-dimensional, unfortunately I could not produce a simple visualization.)

We implement a test-train split using a built-in function from `scikit-learn`, then call the `fit` and `predict` functions as before.

The last three lines show how you can print the data points from the training set and their associated predicted labels.

The first line of code looks a little complicated, but it just removes the `target` value, leading the 13 numerical attributes.

```
# Use all features and all classes
x_full = wine_df.drop(columns=['target']).values
y_full = wine_df['target'].values

# Split into 70% training and 30% testing
x_train, x_test, y_train, y_test = train_test_split(x_full, y_full, test_size=0.3)

# Train SVM with a linear kernel
svm_full = SVC(C = 1, kernel='linear')
svm_full.fit(x_train, y_train)

y_pred = svm_full.predict(x_test)

print(x_test[0]) #attributes for first member of test set
print(y_test[0]) #label for first member of test set
print(y_pred[0]) #predicted label for first member of test set
```

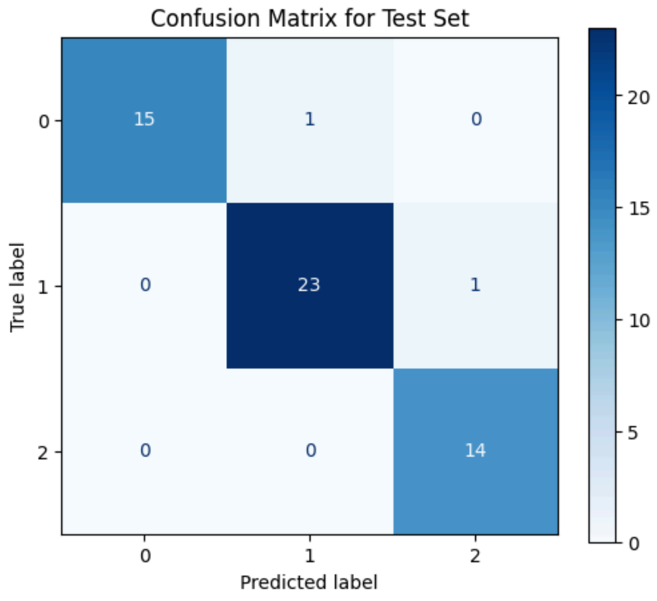
```
[ 12.17   1.45   2.53  19.   104.     1.89   1.75   0.45   1.03   2.95
   1.45   2.23 355.  ]
1
1
```

We can use some built-in functions to compute the accuracy and print a confusion matrix. This dataset is not so challenging, so we reach almost perfect accuracy. (Depending on the randomness in the train-test split, you might even get 100% on the test set.) This indicates that our classifer is doing a great job.

```
# Compute accuracy on the test set
test_accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {test_accuracy}")

# Compute predictions and confusion matrix
conf_matrix_test = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
```

```
disp_test = ConfusionMatrixDisplay(conf_matrix_test, display_labels=svm_full.classes_)
fig, ax = plt.subplots(figsize=(6, 5))
disp_test.plot(ax=ax, cmap="Blues")
plt.title("Confusion Matrix for Test Set")
plt.show()
```

```
Test Accuracy: 0.9629629629629629
```

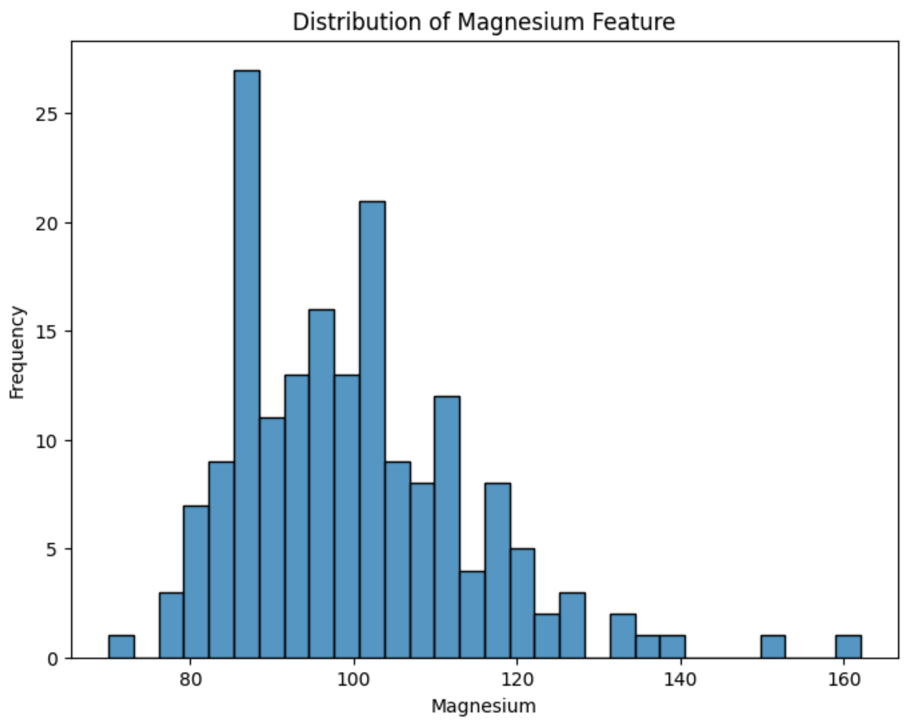


We now consider some problems based on the previous examples.

Problem 1. Above, we printed a histogram of the distribution of the `alcohol` feature. Pick a different feature and plot a histogram of it.

In this problem and all future problems, change the axis labels appropriately to match the new data being used. (For example, in this problem, the plot should no longer be label `alcohol', but instead labeled to match the new feature you choose.)

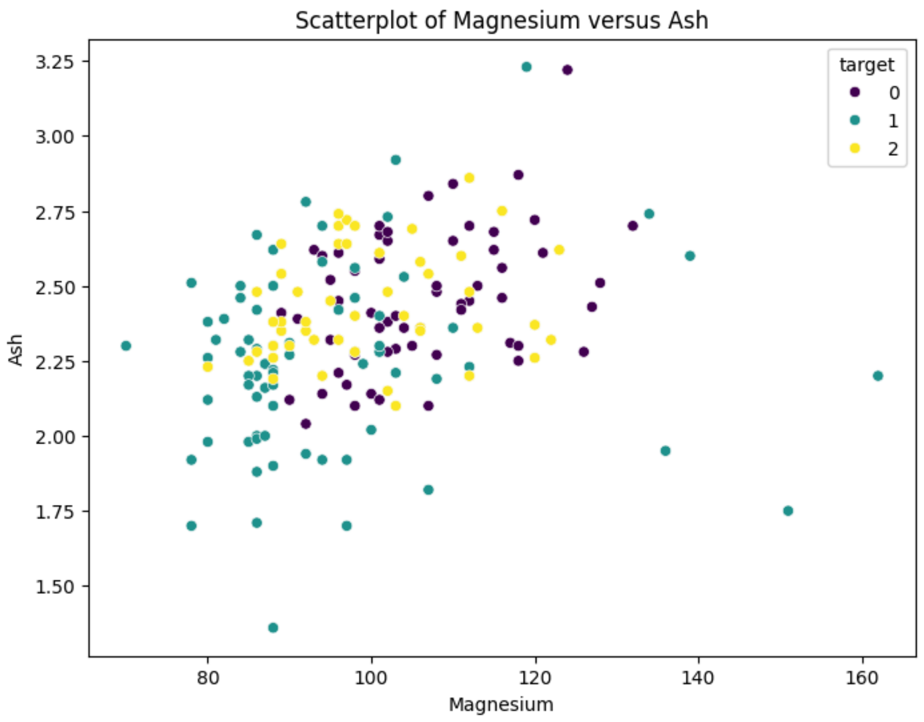
```
plt.figure(figsize=(8, 6))
sns.histplot(wine_df['magnesium'],  bins=30)
plt.title('Distribution of Magnesium Feature')
plt.xlabel('Magnesium')
plt.ylabel('Frequency')
plt.show()
```



Problem 2. Pick a pair of features different from the `alcohol` and `color_intensity` pair used above. Plot a scatterplot with these two class features and class labels by adapting the code from above.

Make sure to change the plot labels. Try to pick a pair where the classes display a decent amount of separation, since you will run a SVC on it in the next problem.

```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='magnesium', y='ash', hue='target', data=wine_df, palette='viridis')
plt.title('Scatterplot of Magnesium versus Ash')
plt.xlabel('Magnesium')
plt.ylabel('Ash')
plt.show()
```

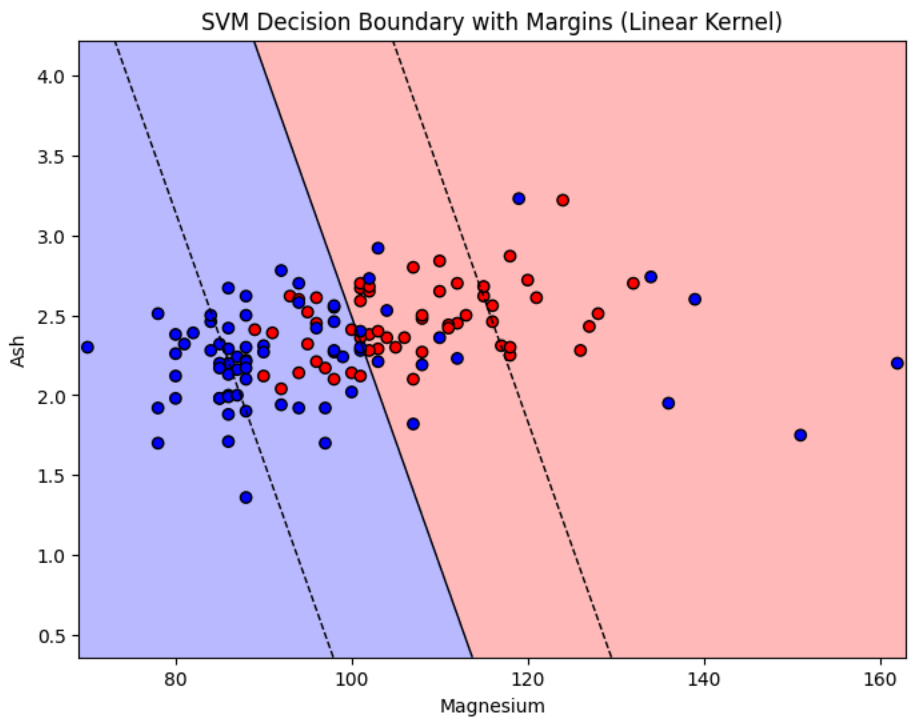


Problem 3. Copy the code from above that gave the red/blue plot with the margins and adapt it to use the features you selected in the previous problem. Change the plot labels as appropriate.

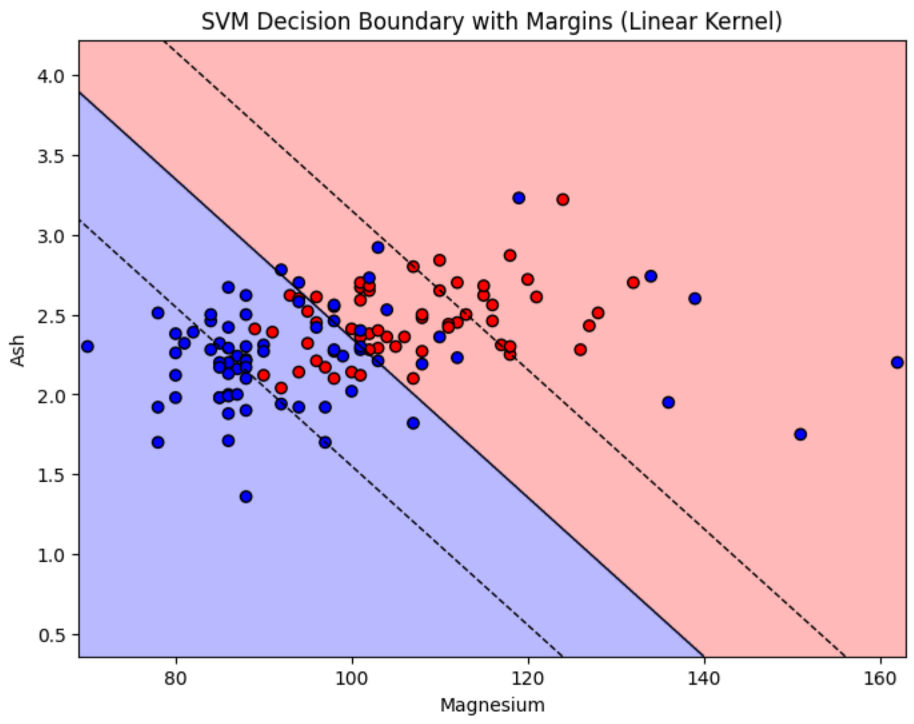
Give three plots, one for each of the choices C=0.1, C=1, and C=100. Keep the linear kernel. (You may need three different code blocks to get these plots to all display at once).

Notice how the margin varies with the value of C, as discussed in class.

```
# Filter the dataset to include only classes 0 and 1
wine_filtered = wine_df[wine_df['target'] != 2]

# Select features for training (using 'alcohol' and 'color_intensity' as features)
X = wine_filtered[['magnesium', 'ash']].values
y = wine_filtered['target'].values

# Train the SVM with a linear kernel
svm_binary = SVC(C=0.1, kernel='linear')
svm_binary.fit(X, y)

# Create a mesh to plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# Predict class labels for each point in the mesh (for solid color regions)
Z = svm_binary.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Compute decision function values (for margin and decision boundary)
decision_values = svm_binary.decision_function(np.c_[xx.ravel(), yy.ravel()])
decision_values = decision_values.reshape(xx.shape)

# Plot the decision boundary with solid regions and margin lines
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))  # Solid colors
plt.contour(xx, yy, decision_values, levels=[-1, 0, 1],
            linestyles=['--', '-', '--'], colors='k', linewidths=1)  # Margins and decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(['red', 'blue']), edgecolors='k')  # Data points
plt.title('SVM Decision Boundary with Margins (Linear Kernel)')
plt.xlabel('Magnesium')
plt.ylabel('Ash')
plt.show()
```

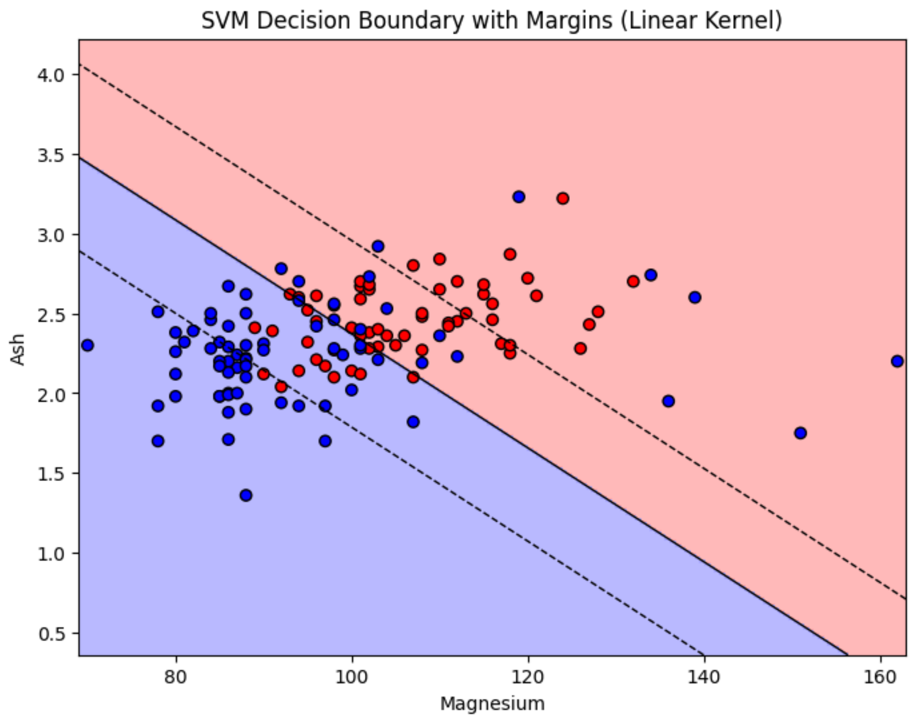


```
# Filter the dataset to include only classes 0 and 1
wine_filtered = wine_df[wine_df['target'] != 2]

# Select features for training (using 'alcohol' and 'color_intensity' as features)
X = wine_filtered[['magnesium', 'ash']].values
y = wine_filtered['target'].values

# Train the SVM with a linear kernel
svm_binary = SVC(C=1, kernel='linear')
svm_binary.fit(X, y)

# Create a mesh to plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# Predict class labels for each point in the mesh (for solid color regions)
Z = svm_binary.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Compute decision function values (for margin and decision boundary)
decision_values = svm_binary.decision_function(np.c_[xx.ravel(), yy.ravel()])
decision_values = decision_values.reshape(xx.shape)

# Plot the decision boundary with solid regions and margin lines
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))  # Solid colors
plt.contour(xx, yy, decision_values, levels=[-1, 0, 1],
            linestyles=['--', '-', '--'], colors='k', linewidths=1)  # Margins and decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(['red', 'blue']), edgecolors='k')  # Data points
plt.title('SVM Decision Boundary with Margins (Linear Kernel)')
plt.xlabel('Magnesium')
plt.ylabel('Ash')
plt.show()
```



```
# Filter the dataset to include only classes 0 and 1
wine_filtered = wine_df[wine_df['target'] != 2]

# Select features for training (using 'alcohol' and 'color_intensity' as features)
X = wine_filtered[['magnesium', 'ash']].values
y = wine_filtered['target'].values

# Train the SVM with a linear kernel
svm_binary = SVC(C=100, kernel='linear')
svm_binary.fit(X, y)
```

```
# Create a mesh to plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# Predict class labels for each point in the mesh (for solid color regions)
Z = svm_binary.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Compute decision function values (for margin and decision boundary)
decision_values = svm_binary.decision_function(np.c_[xx.ravel(), yy.ravel()])
decision_values = decision_values.reshape(xx.shape)

# Plot the decision boundary with solid regions and margin lines
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))  # Solid colors
plt.contour(xx, yy, decision_values, levels=[-1, 0, 1],
            linestyles=['--', '-', '--'], colors='k', linewidths=1)  # Margins and decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(['red', 'blue']), edgecolors='k')  # Data points
plt.title('SVM Decision Boundary with Margins (Linear Kernel)')
plt.xlabel('Magnesium')
plt.ylabel('Ash')
plt.show()
```



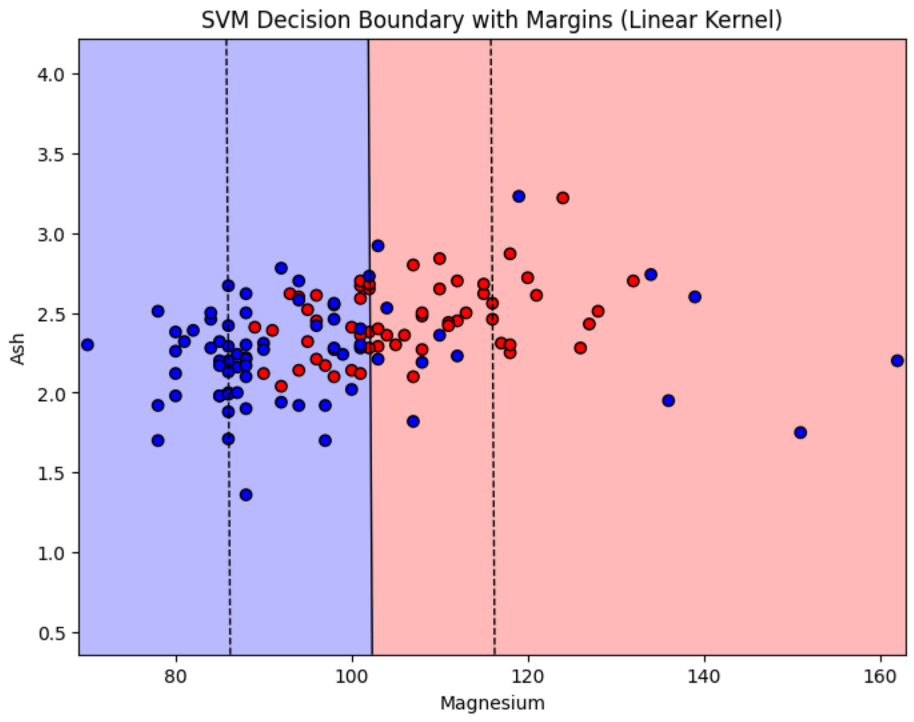
Problem 4. Copy your C=1 code from the last problem (again using your selected features) and change the linear kernel to a polynomial kernel with degree 2. You may wish to consult the documentation to see how to do this.

Run your code, which will produce a plot with the new (nonlinear) decision boundary and margin. Change the title to match the new kernel.

Note. More examples of SVM visualizations (with example code) can be found in the scikit-learn documentation.

```
# Filter the dataset to include only classes 0 and 1
wine_filtered = wine_df[wine_df['target'] != 2]

# Select features for training (using 'alcohol' and 'color_intensity' as features)
X = wine_filtered[['magnesium', 'ash']].values
y = wine_filtered['target'].values

# Train the SVM with a linear kernel
svm_binary = SVC(C=1, kernel='poly', degree=2)
svm_binary.fit(X, y)

# Create a mesh to plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
```

```
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# Predict class labels for each point in the mesh (for solid color regions)
Z = svm_binary.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Compute decision function values (for margin and decision boundary)
decision_values = svm_binary.decision_function(np.c_[xx.ravel(), yy.ravel()])
decision_values = decision_values.reshape(xx.shape)

# Plot the decision boundary with solid regions and margin lines
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))  # Solid colors
plt.contour(xx, yy, decision_values, levels=[-1, 0, 1],
            linestyles=['--', '-', '--'], colors='k', linewidths=1)  # Margins and decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(['red', 'blue']), edgecolors='k')  # Data points
plt.title('SVM Decision Boundary with Margins (Linear Kernel)')
plt.xlabel('Magnesium')
plt.ylabel('Ash')
plt.show()
```



Part 2: MNIST Digit Recognition

In the previous part, you saw an example of how a SVC was trained. In this part, you will train one yourself.

We begin by importing the MNIST dataset using a special data import function from `scikit-learn`, whose details are not important for us.

```
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
X, y = mnist.data, mnist.target

# Convert labels from strings to integers
y = y.astype(np.uint8)
```

Recall that in class, we said that this dataset consists of black and white images (represented as matrices of numbers) and labels (from 0 through 9). Let's check this using the `shape` command, which tells us the shape of an array.

```
print(y.shape)
print(y[0])
print(y[1])
```

```
(70000,)
5
0
```

We see that `y` is a collection of 70,000 integer labels. The first two are 5 and 0.

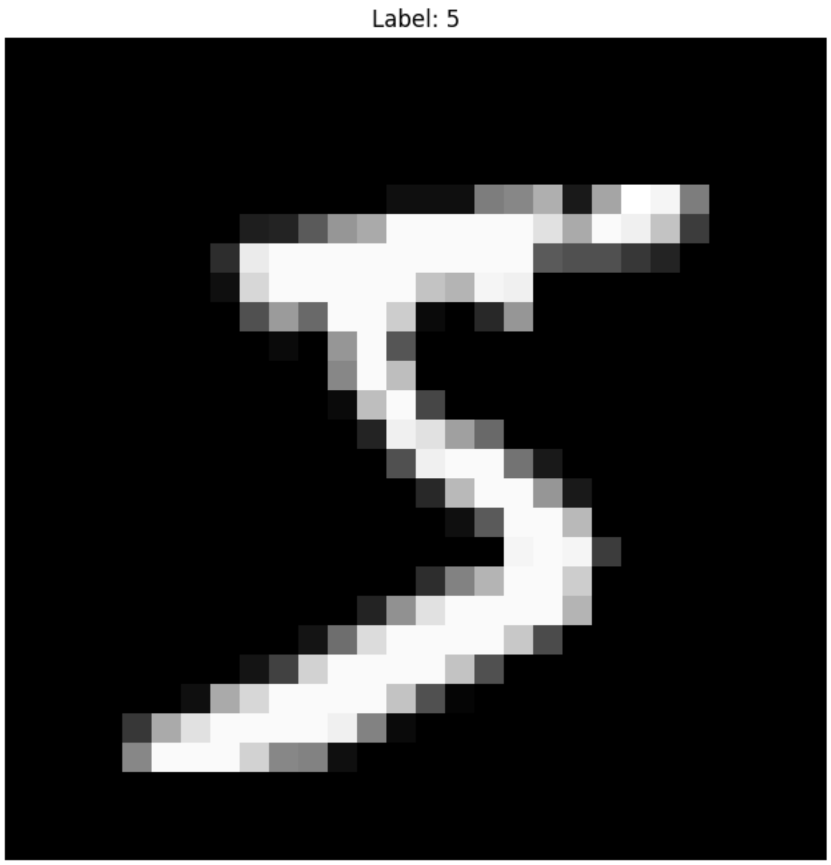
We now check `X`.

```
print(X.shape)
print(X[0])
```

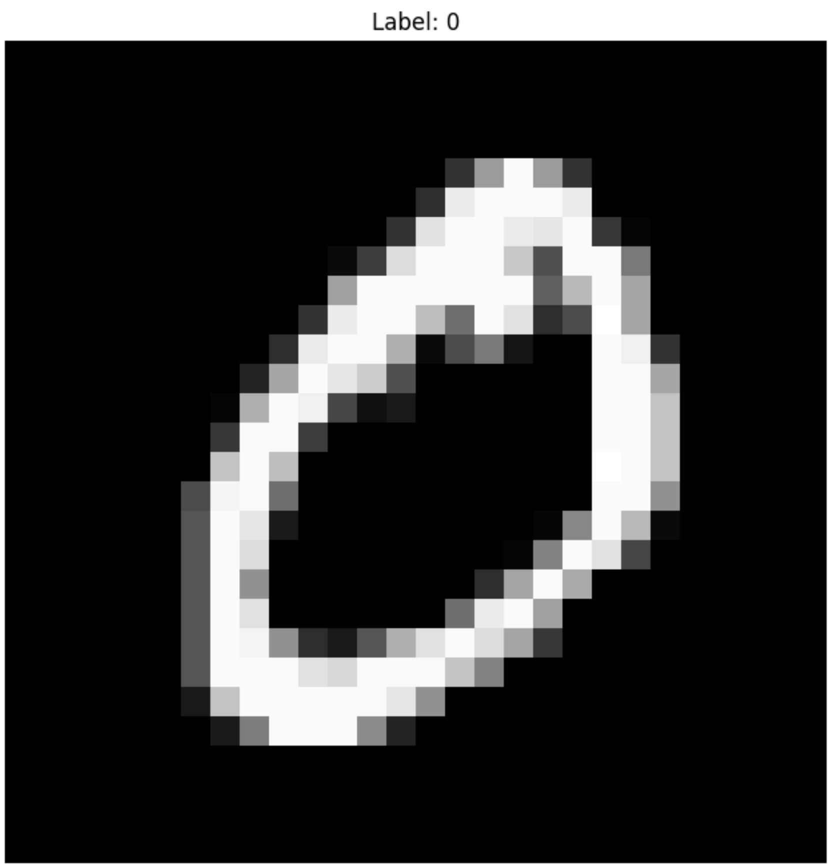
```
(70000, 784)
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   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
   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
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   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  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   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
   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
   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]
```

We see that we have a collection of 70,000 vectors of length 784, representing the pixel values for the 28 by 28 images. It would be nice to visualize these, so let's do that.

```
plt.figure(figsize=(8, 8))
plt.imshow(X[0].reshape(28, 28), cmap='gray')
plt.title(f"Label: {y[0]}")
plt.axis('off')
plt.show()
```



```
plt.figure(figsize=(8, 8))
plt.imshow(X[1].reshape(28, 28), cmap='gray')
plt.title(f"Label: {y[1]}")
plt.axis('off')
plt.show()
```



These match the labels we saw earlier, which is a good sign. You can try visualizing other datapoints, if you want.

To make this lab go faster, we are going to make the dataset smaller and just use the first 10,000 values. **Your solutions to the problems below should use only this smaller version.**

```
x_mnist = X[:10000]
y_mnist = y[:10000]
```

Problem 1. Train a support vector classifier on `x_mnist` and `y_mnist` with a linear kernel and C=1. Use a random train-test split with 20% of the data in the testing set (using the built-in function mentioned above, which chooses the split randomly). Print the label for the first element in your test set, and print the corresponding prediction from your classifier. (They may not match!)

Training may take anywhere from 10 to 30 seconds.

```
# Split into 80% training and 20% testing
x_train, x_test, y_train, y_test = train_test_split(x_mnist, y_mnist, test_size=0.2)

# Train SVM with a linear kernel
svm_full = SVC(C = 1, kernel='linear')
svm_full.fit(x_train, y_train)

y_pred = svm_full.predict(x_test)

print(y_test[0]) #label for first member of test set
print(y_pred[0]) #predicted label for first member of test set
```

```
2
2
```

Problem 2.

Part A. Calculate and print the accuracy of your classifier on the test set. Additionally, provide a confusion matrix.

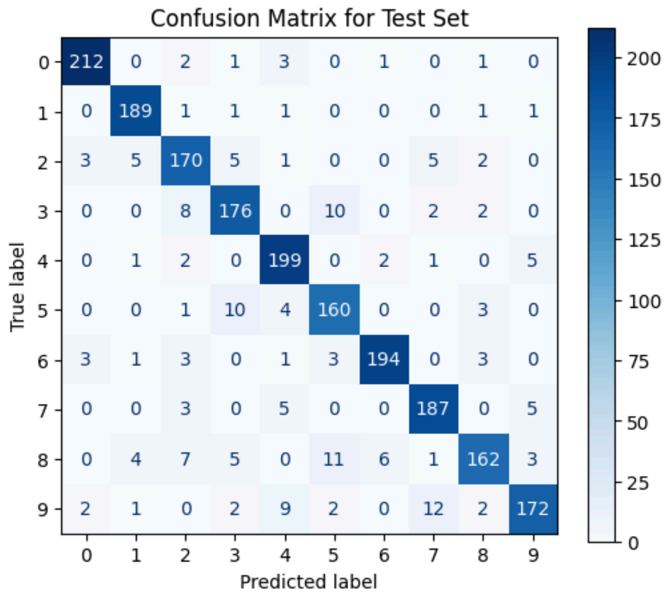
Part B. You should notice that your classifier is fairly good at distinguishing digits that have distinct shapes, like 0 and 1. However, your confusion matrix should indicate that it has trouble on certain pairs of digits with similar shapes. What is one such pair that you observe?

Write your answer below.

My classifier is having trouble with the following pair: *your answer here*

```
test_accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {test_accuracy}")

conf_matrix_test = confusion_matrix(y_test, y_pred)

disp_test = ConfusionMatrixDisplay(conf_matrix_test, display_labels=svm_full.classes_)
fig, ax = plt.subplots(figsize=(6, 5))
disp_test.plot(ax=ax, cmap="Blues")
plt.title("Confusion Matrix for Test Set")
plt.show()
```

```
Test Accuracy: 0.9105
```

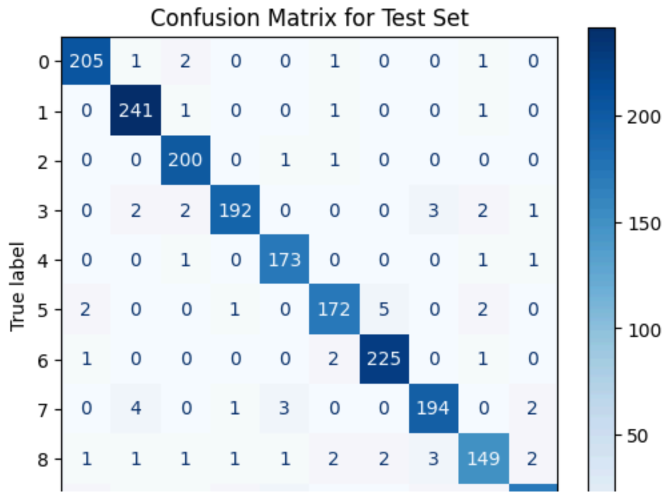


Problem 3. Train an SVM again on the same data, but with a different choice of C or kernel (or both). Provide the corresponding accuracy score and confusion matrix. A list of kernels can be found in the documentation linked in the previous part of this assignment. (If you use the `rbf` kernel, you can also try tweaking the `gamma` parameter.)

To get full credit on this problem, you must achieve at least 94% accuracy on the test set.

```
# Split into 80% training and 20% testing
x_train, x_test, y_train, y_test = train_test_split(x_mnist, y_mnist, test_size=0.2)

# Train SVM with a linear kernel
svm_full = SVC(C = 2, kernel='poly')
svm_full.fit(x_train, y_train)

y_pred = svm_full.predict(x_test)

print(y_test[0]) #label for first member of test set
print(y_pred[0]) #predicted label for first member of test set

test_accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {test_accuracy}")

conf_matrix_test = confusion_matrix(y_test, y_pred)

disp_test = ConfusionMatrixDisplay(conf_matrix_test, display_labels=svm_full.classes_)
fig, ax = plt.subplots(figsize=(6, 5))
disp_test.plot(ax=ax, cmap="Blues")
plt.title("Confusion Matrix for Test Set")
plt.show()
```

```
1
1
Test Accuracy: 0.961
```



Problem 4. Look at the scikit-learn documentation for k-nearest neighbors. By replacing `SVC(C = 1, kernel='linear')` with `KNeighborsClassifier(n_neighbors=5)` in your previous code, create a classifier using the 5 nearest neighbors in the training set, then fit it on the data from the last few problems. Print the accuracy and display a confusion matrix for the k-nearest neighbors classifier.

Generate a slider using jupyter widgets Close

```
# Split into 80% training and 20% testing
x_train, x_test, y_train, y_test = train_test_split(x_mnist, y_mnist, test_size=0.2)

# Train SVM with a linear kernel
svm_full = KNeighborsClassifier(n_neighbors=5)
svm_full.fit(x_train, y_train)
```