Part 1

In [ ]:
```
import pandas as pd
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version = 1)
mnist.keys()
```

Out[ ]:
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_names', 'DESCR', 'details', 'url'])
```

In [ ]:
```
X, y = mnist["data"], mnist["target"]
X.shape
```

Out[ ]:
```
(70000, 784)
```

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

Out[ ]:
```
(70000,)
```

In [ ]:
```
X.head(2)
```

Out[ ]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

2 rows × 784 columns

In [ ]:
```
y[0]
```

Out[ ]:
```
'5'
```

In [ ]:
```
X = X.to_numpy()
```

In [ ]:
```
import numpy as np
y = y.astype(np.uint8)
y[0]
```

Out[ ]:
```
5
```

In [ ]:
```
import matplotlib.pyplot as plt

some_digit = X[0]
some_digit_image = some_digit.reshape(28,28)
plt.figure(figsize=(3,3))
plt.imshow(some_digit_image, cmap="binary")
plt.axis("off")
plt.show()
```

In [ ]:
```
X_train_RAW, X_test_RAW, y_train_RAW, y_test_RAW = X[:60000], X[60000:], y[:60000], y[60000:]
```

Get a DataFrame column that contain only the 5s and 3s for training and testing

In [ ]:
```
# Train data
y_train_p1 = y_train_RAW[(y_train_RAW == 3) | (y_train_RAW == 5)]

y_train_p1_3 = y_train_p1 == 3

X_train_p1 = X_train_RAW[(y_train_RAW == 3) | (y_train_RAW == 5)]

# Test data
y_test_p1 = y_test_RAW[(y_test_RAW == 3) | (y_test_RAW == 5)]

y_test_p1_3 = y_test_p1 == 3

X_test_p1 = X_test_RAW[(y_test_RAW == 3) | (y_test_RAW == 5)]
```

In [ ]:
```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train_p1, y_train_p1_3)
```

Out[ ]:

```
SGDClassifier
SGDClassifier(random_state=42)
```

In [ ]:

```
sgd_clf.predict([some_digit])
```

Out[ ]:

```
array([False])
```

A) Use cross_val_score() to show the accuracy of prediction under cross validation.

In [ ]:

```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train_p1, y_train_p1_3, cv = 3 , scoring = "accuracy")
```

Out[ ]:

```
array([0.92962867, 0.95299922, 0.94701299])
```

B) Use cross_val_predict() to generate predictions on the training data.

In [ ]:

```
from sklearn.model_selection import cross_val_predict
y_p1_pred = cross_val_predict(sgd_clf, X_train_p1, y_train_p1_3, cv =3)
y_p1_pred
```

Out[ ]:

```
array([False,  True,  True, ..., False,  True, False])
```

Confusion Matrix

In [ ]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_p1_3, y_p1_pred)
```

Out[ ]:

```
array([[4994,  427],
       [ 229, 5902]], dtype=int64)
```

Precision Score

In [ ]:

```
from sklearn.metrics import precision_score, recall_score, f1_score
precision_score(y_train_p1_3, y_p1_pred)
```

Out[ ]:

```
0.9325327855901406
```

Recall Score

In [ ]:

```
recall_score(y_train_p1_3, y_p1_pred)
```

Out[ ]: 0.9626488337954656

F1 Score

In [ ]:
```
f1_score(y_train_p1_3, y_p1_pred)
```

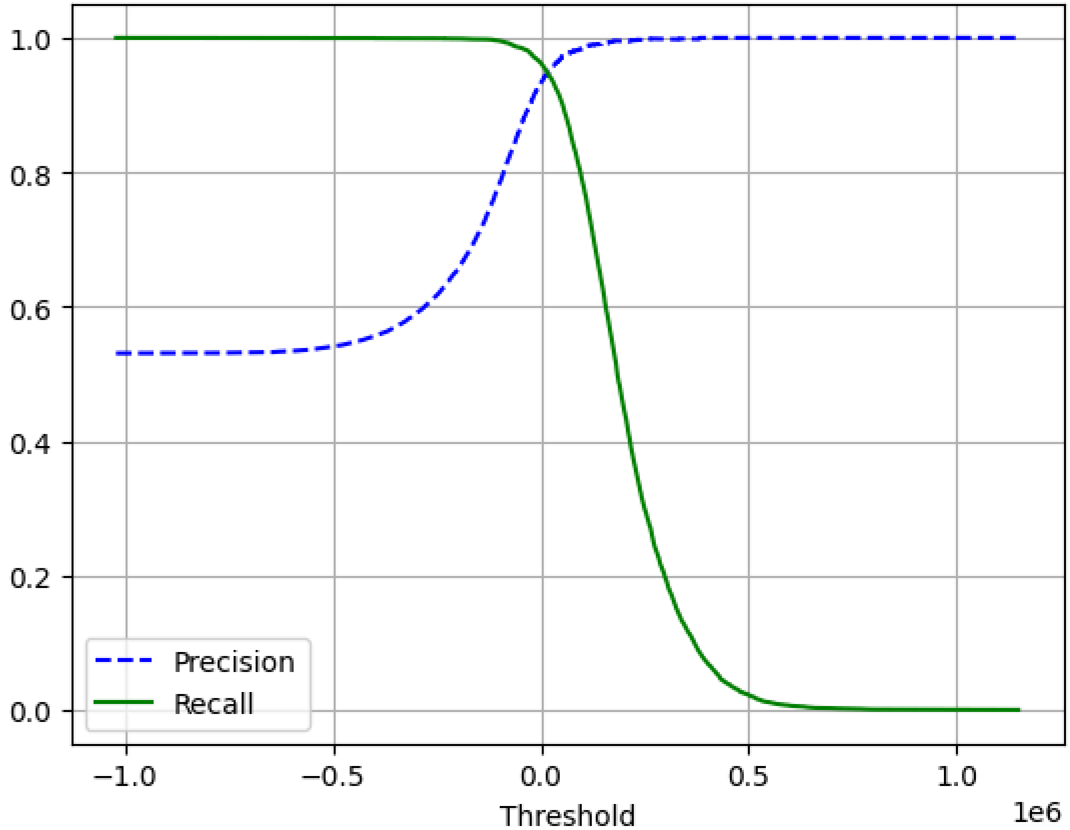
Out[ ]: 0.9473515248796147

C) Use cross_val_predict() to generate the prediction scores on the training set. Then, plot the precision and recall curves as functions of the threshold value.

In [ ]:
```
from sklearn.metrics import precision_recall_curve

y_scores = cross_val_predict(sgd_clf, X_train_p1, y_train_p1_3, cv = 3, method = "decision_function")
precisions, recalls, thresholds = precision_recall_curve(y_train_p1_3, y_scores)
```

In [ ]:
```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label = "Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label = "Recall")
    plt.legend(loc = "best")
    plt.grid()
    plt.xlabel('Threshold')
```

In [ ]:
```
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```



D) Based on the curves, what will be a sensible threshold value to choose? Generate predictions under the chosen threshold value. Evaluate the precision and recall scores using the predictions.

Because these curves intersect around roughly 0, I will use this as the threshold value to maximize both precision and recall.

In [ ]:
```
threshold_00 = thresholds[np.argmax(precisions >= 0)]
y_p1_pred_00 = (y_scores >= threshold_00)

precision_score(y_train_p1_3, y_p1_pred_00)
```

Out[ ]:
```
0.5307306094182825
```

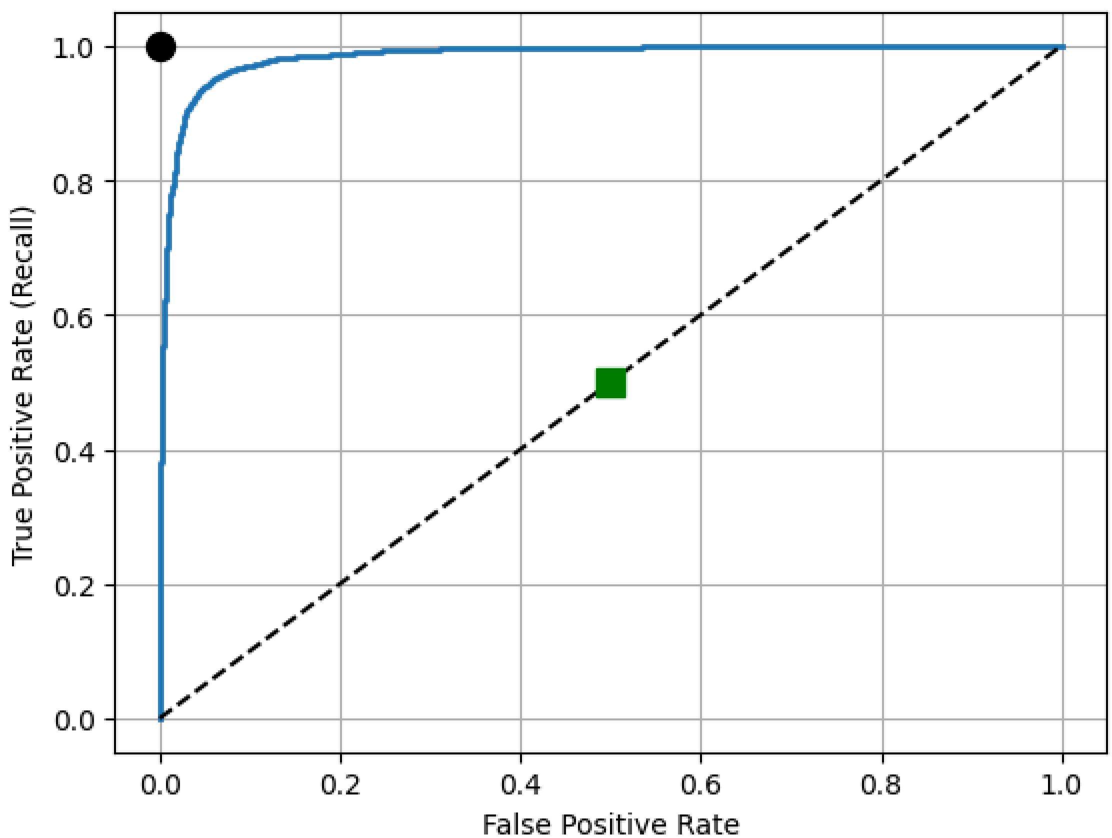
In [ ]:
```
recall_score(y_train_p1_3, y_p1_pred_00)
```

Out[ ]:
```
1.0
```

E) Plot the ROC curve and evaluate the ROC AUC score.

In [ ]:
```
from sklearn.metrics import roc_curve, roc_auc_score

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train_p1_3, y_scores)
```

In [ ]:
```
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.grid()
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate (Recall)')
    plt.plot(0, 1, marker='o', markersize=10, markeredgecolor="black", markerfacecolor="black")
    plt.plot(0.5, 0.5, marker="s", markersize=10, markeredgecolor="green", markerfacecolor="green")


plot_roc_curve(false_positive_rate, true_positive_rate)
plt.show()
```



ROC AUC Score

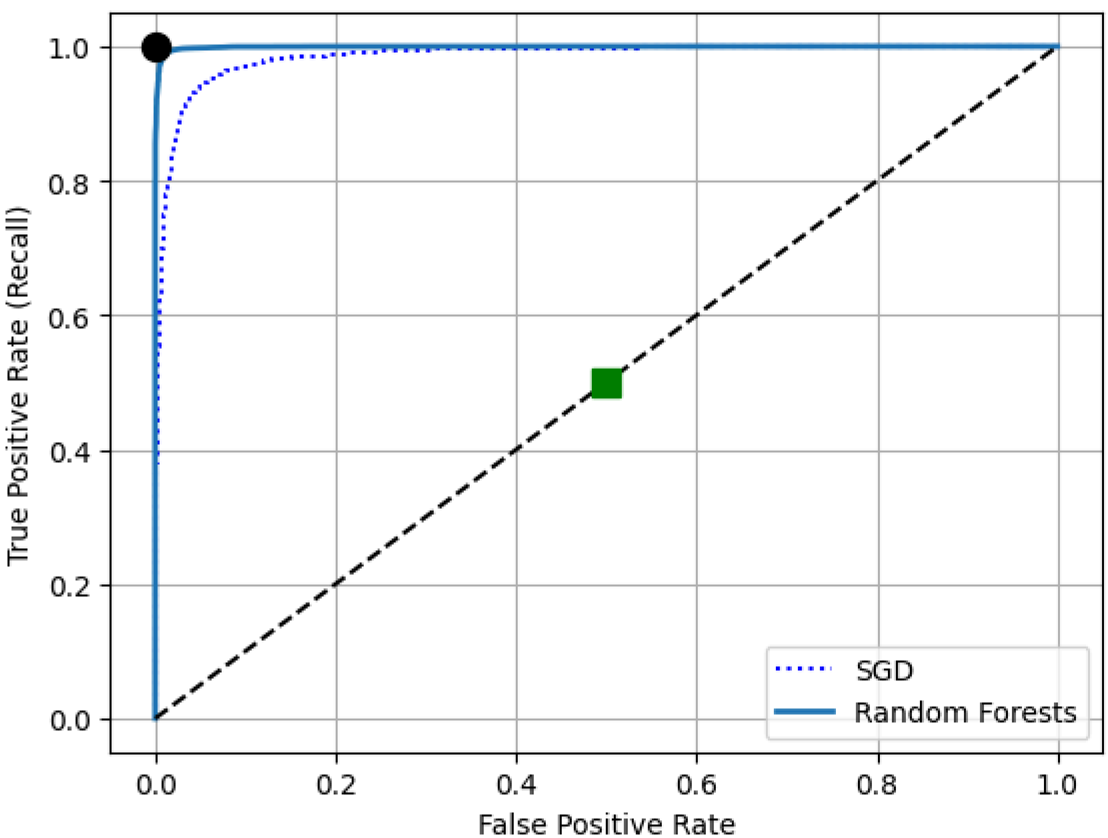
In [ ]:
```
roc_auc_score(y_train_p1_3, y_scores)
```

Out[ ]: 0.9851212013087797

F) RandomForestClassifier

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(random_state=42)

y_probas_forest= cross_val_predict(forest_clf, X_train_p1, y_train_p1_3, cv = 3, method = "predict_proba")

y_scores_forest = y_probas_forest[:, 1]
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_p1_3, y_scores_forest)
```

In [ ]:
```
plt.plot(false_positive_rate, true_positive_rate, "b:", label = "SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forests")
plt.legend(loc = "lower right")
plt.show()
```



In [ ]:
```
roc_auc_score(y_train_p1_3, y_scores_forest)
```

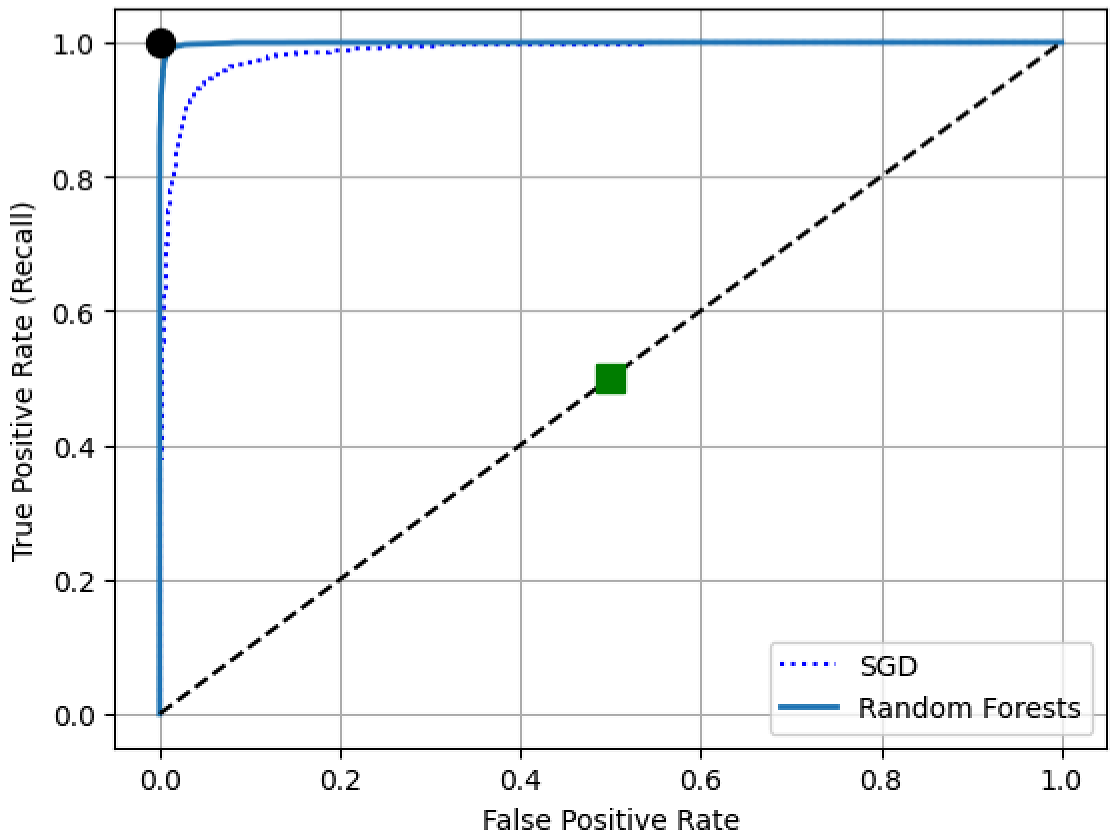
Out[ ]: 0.9992079106873717

G) Standard Scaler on X (features) data before training the model

In [ ]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_p1 = scaler.fit_transform(X_train_p1)
```

In [ ]:
```
forest_clf = RandomForestClassifier(random_state=42)


y_probas_forest= cross_val_predict(forest_clf, X_train_p1, y_train_p1_3, cv = 3, method = "predict_proba")

y_scores_forest = y_probas_forest[:, 1]
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_p1_3, y_scores_forest)
```

In [ ]:
```
plt.plot(false_positive_rate, true_positive_rate, "b:", label = "SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forests")
plt.legend(loc = "lower right")
plt.show()
```



In [ ]:
```
roc_auc_score(y_train_p1_3, y_scores_forest)
```

Out[ ]:
```
0.9992048718276675
```

Part 2

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

from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version = 1)
mnist.keys()

X, y = mnist["data"], mnist["target"]

X = X.to_numpy()
y = y.astype(np.uint8)
```

Get training and testing data from MNIST dataset

In [ ]:
```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

Create a class that will separate the data into 3v5, 3vOther, 5vOther; as well as train the three different classifiers on this data

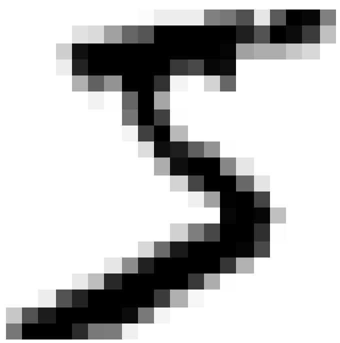
In [ ]:
```
from sklearn.linear_model import SGDClassifier
import random


class SGD_3v5vO_clf:
    X_train = None
    X_train_3v5 = None

    y_train = None
    y_train_3v5_3 = None
    y_train_3vO_3 = None
    y_train_5vO_5 = None

    loss_method = "log_loss"

    sgd_3v5_clf = SGDClassifier(random_state=42, loss=loss_method)
    sgd_3vO_clf = SGDClassifier(random_state=42, loss=loss_method)
    sgd_5vO_clf = SGDClassifier(random_state=42, loss=loss_method)

    def __init__(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train
```

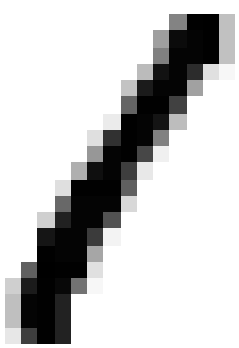
```python
        y_train_3v5 = y_train[(y_train == 3) | (y_train == 5)]

        self.y_train_3v5_3 = y_train_3v5 == 3
        self.X_train_3v5 = X_train[(y_train == 3) | (y_train == 5)]

        self.y_train_3vO_3 = y_train == 3
        self.y_train_5vO_5 = y_train == 5

    def fit(self):
        self.sgd_3v5_clf.fit(self.X_train_3v5, self.y_train_3v5_3)
        self.sgd_3vO_clf.fit(self.X_train, self.y_train_3vO_3)
        self.sgd_5vO_clf.fit(self.X_train, self.y_train_5vO_5)

    def predict(self, X):
        three = 0
        five = 0
        other = 0

        if self.sgd_3v5_clf.predict(X)[0]:
            three += 1
        else:
            five += 1

        if self.sgd_5vO_clf.predict(X)[0]:
            five += 1
        else:
            other += 1

        if self.sgd_3vO_clf.predict(X)[0]:
            three += 1
        else:
            other += 1

        if three == five == other:
            return random.choice(["3", "5", "Other"])

        elif three == 2:
            return "3"
        elif five == 2:
            return "5"
        else:
            return "Other"
```

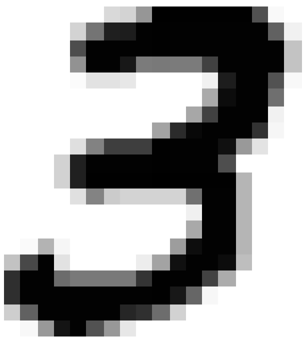
In [ ]:
```python
multi_clf = SGD_3v5vO_clf(X_train, y_train)
multi_clf.fit()
```

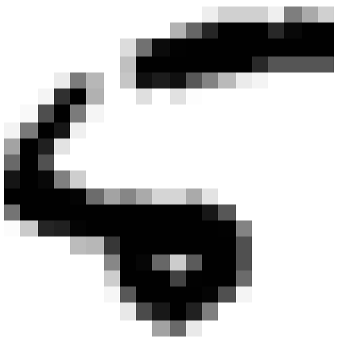
Demonstration of the Predict function

In [ ]:
```
digit_5 = X[0]
digit_0 = X[3]
digit_3 = X[7]
question_digit = X_test[8]

all_digits = [digit_5, digit_0, digit_3, question_digit]

for digit in all_digits:
    digit_image = digit.reshape(28,28)
    plt.figure(figsize=(3,3))
    plt.imshow(digit_image, cmap="binary")
    plt.axis("off")
    plt.show()
```









In [ ]:
```
for digit in all_digits:
    print(f"Prediction: {multi_clf.predict(np.reshape(digit, (1,-1)))}")
```

```
Prediction: 5
Prediction: Other
Prediction: 3
Prediction: Other
```

Part 3

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

from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version = 1)
mnist.keys()

X, y = mnist["data"], mnist["target"]

X = X.to_numpy()
y = y.astype(np.uint8)

X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

Perform a grid search to find the best hyperparameters for the model using n_neighbors and weights.

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

knn_clf = KNeighborsClassifier()
param_grid = [
    {"n_neighbors" : [3,5,10,20,30,50]},
    {"weights" : ["uniform", "distance"]}
]

grid_search = GridSearchCV(knn_clf, param_grid, cv = 5, scoring = 'neg_mean_squared_error', return_train_score= True)

grid_search.fit(X_train, y_train)
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
  ▸ KNeighborsClassifier
```

In [ ]:
```
best_knn = grid_search.best_estimator_

best_knn.fit(X_train, y_train)
```

Out[ ]:

```
KNeighborsClassifier

KNeighborsClassifier(weights='distance')
```

In [ ]:

```python
grid_search.best_params_
```

Out[ ]:

```
{'weights': 'distance'}
```

In [ ]:

```python
from sklearn.model_selection import cross_val_score

cross_val_score(best_knn, X_train, y_train, cv = 3, scoring = "accuracy")
```

Out[ ]:

```
array([0.9688 , 0.96795, 0.96905])
```

In [ ]:

```python
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(best_knn, X_train, y_train, cv = 3)
conf_mx = confusion_matrix(y_train, y_train_pred)
conf_mx
```

Out[ ]:

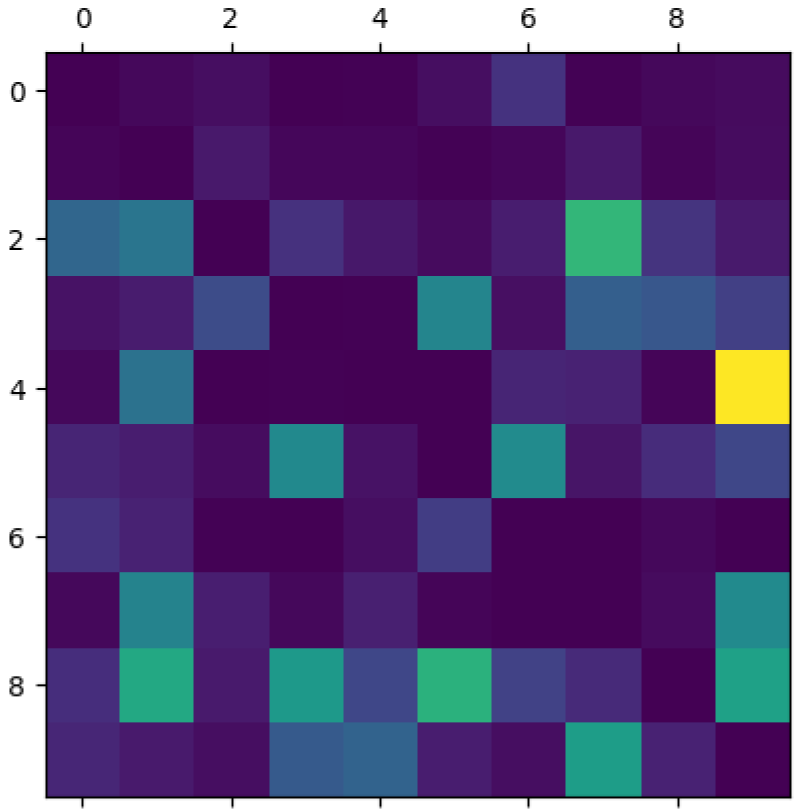
```
array([[5881,    3,    5,    0,    1,    5,   20,    1,    3,    4],
       [   2, 6701,   11,    3,    3,    1,    3,   11,    2,    5],
       [  46,   54, 5691,   20,    9,    4,   11,   92,   21,   10],
       [   7,   11,   33, 5899,    1,   65,    6,   43,   39,   27],
       [   3,   51,    0,    1, 5622,    0,   14,   13,    2,  136],
       [  13,   10,    4,   60,    6, 5217,   61,    7,   16,   27],
       [  20,   13,    1,    0,    5,   25, 5851,    0,    3,    0],
       [   3,   65,   12,    3,   13,    2,    0, 6093,    4,   70],
       [  18,   82,   10,   73,   29,   87,   27,   16, 5431,   78],
       [  15,   10,    5,   39,   44,   11,    5,   77,   13, 5730]],
      dtype=int64)
```

In [ ]:

```python
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
np.fill_diagonal(norm_conf_mx, 0)

plt.matshow(norm_conf_mx)
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x1f6898cc4d0>
```



The model commonly confuses 7 and 2, 9 and 4, and 8 and 5.

Part 4

Step 1

Generate 100 datasets, each with 50 instances. Each data point is drawn from the Gaussian distribution N(67,3.8). That is, the true mean is $\mu = 67$ and the standard deviation is $\sigma = 3.8$. Hint: You can generate a random two-dimensional array with the shape 100×50 in one go.

In [ ]:
```
import numpy as np

dataset = np.random.normal(67, 3.8, (100, 50))
```

Step 2 and 3)

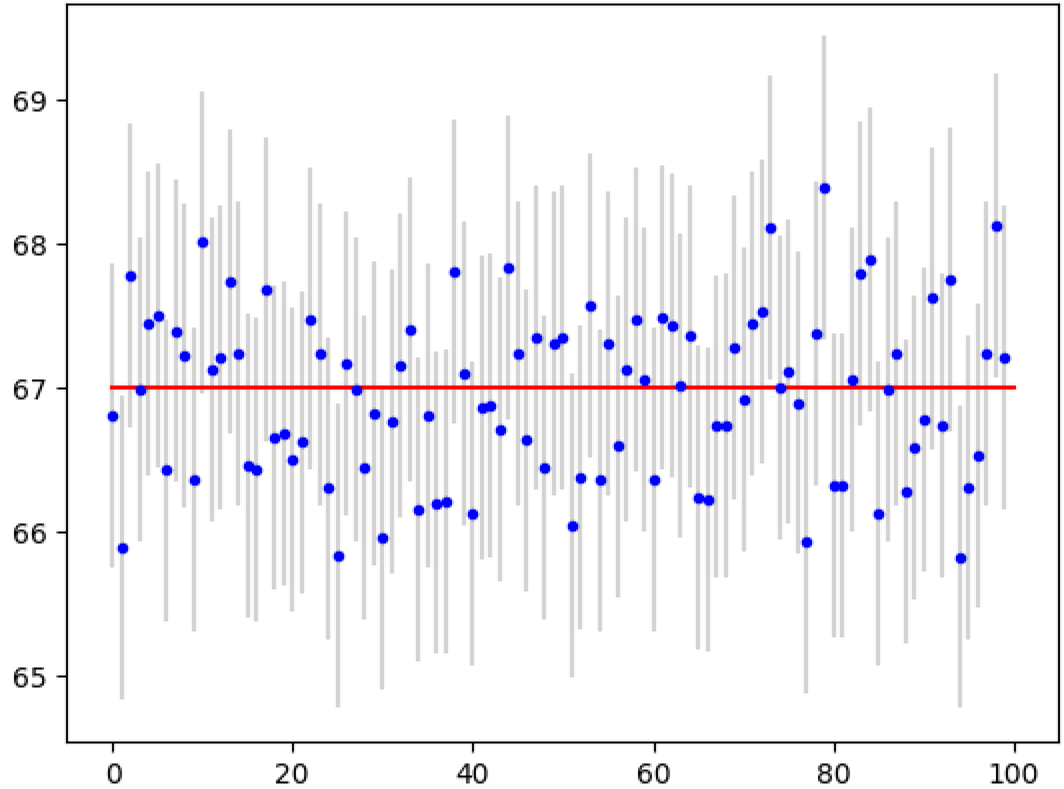
Compute sample mean and sample standard deviation for each data set
Calculate the confidence intervals using the population standard deviation

In [ ]:
```
data_means = np.mean(dataset, axis=1)
data_std = np.std(dataset, axis=1, ddof=1)

data_error = 1.96 * 3.8 / np.sqrt(50)
```

Step 4

Plot and count number of datasets where known mean is outside the confidence interval

In [ ]:
```
import matplotlib.pyplot as plt

plt.errorbar(np.arange(100), data_means, data_error, fmt = '.k', mfc = "blue", mec = "blue",
             ecolor= "lightgray")

plt.plot(np.linspace(0,100,2), np.array([67,67]) , color = "red")
plt.show()
```



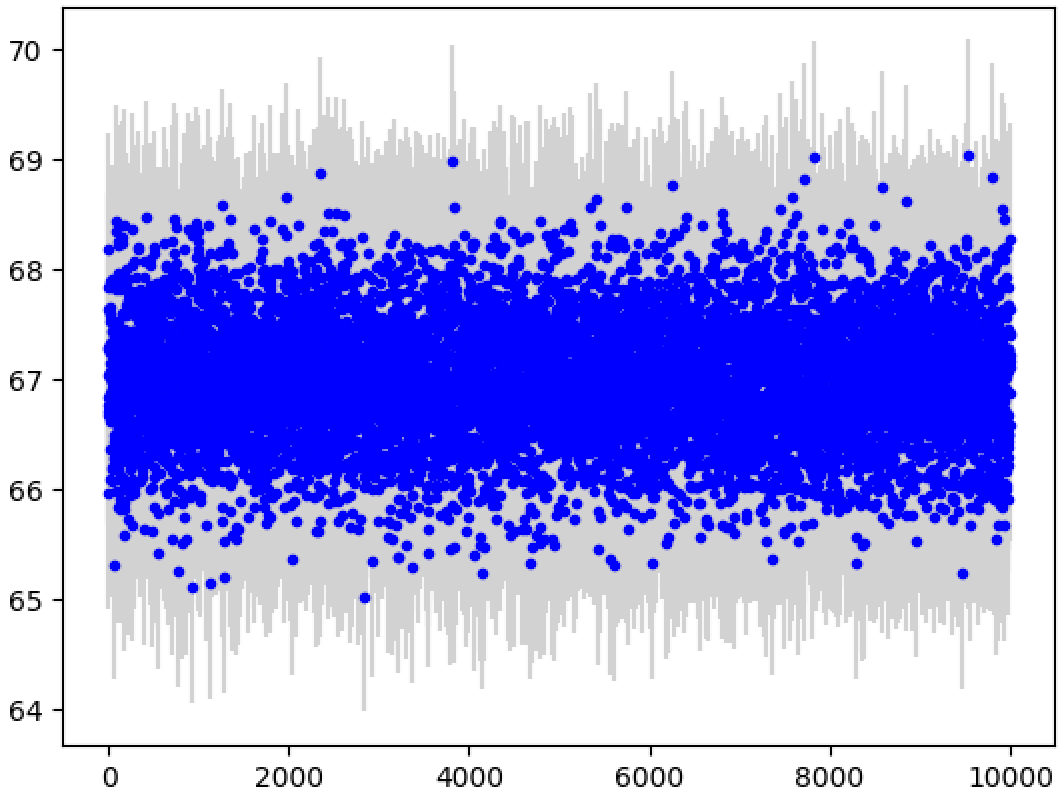
In [ ]:
```
def count_intervals(means, known_mean, known_std, data_in_each):
    outside_confidence_interval = 0
    num_data = len(means)

    for i in range(num_data):
        if (means[i] + 1.96*(known_std / np.sqrt(data_in_each))) < known_mean or (means[i] - 1.96 * (known_std/ np.sqrt(data_in_ea
            outside_confidence_interval += 1

    return outside_confidence_interval

count_intervals(data_means, 67, 3.8, 50)
```

Out[ ]: 7

Step 5

Generate 10,000 datasets, 50 data points each. Count how many lie outside. Compute the percent of intervals that do not contain the true mean. Repeat 10 times. Print the 10 percentages.

In [ ]:
```
num_datasets = 10000
dataset = np.random.normal(67, 3.8, (num_datasets, 50))

data_means = np.mean(dataset, axis=1)
data_std = np.std(dataset, axis=1, ddof=1)

data_error = 1.96 * 3.8 / np.sqrt(50)

plt.errorbar(np.arange(num_datasets), data_means, data_error, fmt = '.k', mfc = "blue", mec = "blue",
             ecolor= "lightgray")

plt.plot(np.linspace(0,num_datasets,2), np.array([67,67]) , color = "red")
plt.show()
```



In [ ]:
```
num_datasets = 10000
percentages = []

for _ in range(10):
    dataset = np.random.normal(67, 3.8, (num_datasets, 50))

    data_means = np.mean(dataset, axis=1)
    data_std = np.std(dataset, axis=1, ddof=1)
```

```
    percentages.append(count_intervals(data_means, 67, 3.8, 50) / num_datasets)

percentages
```

Out[ ]:
```
[0.0484,
 0.0485,
 0.0496,
 0.0534,
 0.0508,
 0.0503,
 0.0484,
 0.0491,
 0.0509,
 0.0515]
```

Step 6

Repeat for 100,000 datasets

In [ ]:
```
num_datasets = 100000
percentages = []

for _ in range(10):
    dataset = np.random.normal(67, 3.8, (num_datasets, 50))

    data_means = np.mean(dataset, axis=1)
    data_std = np.std(dataset, axis=1, ddof=1)

    percentages.append(count_intervals(data_means, 67, 3.8, 50) / num_datasets)

percentages
```

Out[ ]:
```
[0.04942,
 0.05041,
 0.04995,
 0.04902,
 0.04997,
 0.0507,
 0.04877,
 0.04898,
 0.04989,
 0.05042]
```

Step 7

Use Student's t-distribution by using the sample standard deviation to replace the true standard deviation (3.8).

In [ ]:
```
def count_intervals_student(means, stds, known_mean, data_in_each):
    outside_confidence_interval = 0
    num_data = len(means)

    for i in range(num_data):
        if (means[i] + 1.96*(stds[i] / np.sqrt(data_in_each))) < known_mean or (means[i] - 1.96 * (stds[i]/ np.sqrt(data_in_each))
            outside_confidence_interval += 1

    return outside_confidence_interval
```

In [ ]:
```
num_datasets = 100000
num_datapoints = 10
percentages = []

for _ in range(10):
    dataset = np.random.normal(67, 3.8, (num_datasets, num_datapoints))

    data_means = np.mean(dataset, axis=1)
    data_std = np.std(dataset, axis=1, ddof=1)

    percentages.append(count_intervals_student(data_means, data_std, 67, num_datapoints) / num_datasets)

percentages
```

Out[ ]:
```
[0.08202,
 0.08133,
 0.08088,
 0.0817,
 0.08206,
 0.08156,
 0.08106,
 0.08167,
 0.08165,
 0.08258]
```

Step 9

Use Student's t-distribution, replacing our constant of 1.96 with 2.262 due to the look up table and a degree of freedom of 9 (10-1).

In [ ]:
```
def count_intervals_student(means, stds, known_mean, data_in_each):
    outside_confidence_interval = 0
    num_data = len(means)
```

```python
    for i in range(num_data):
        if (means[i] + 2.262*(stds[i] / np.sqrt(data_in_each))) < known_mean or (means[i] - 1.96 * (stds[i]/ np.sqrt(data_in_each)
            outside_confidence_interval += 1

    return outside_confidence_interval
```

In [ ]:

```python
num_datasets = 100000
num_datapoints = 10
percentages = []

for _ in range(10):
    dataset = np.random.normal(67, 3.8, (num_datasets, num_datapoints))

    data_means = np.mean(dataset, axis=1)
    data_std = np.std(dataset, axis=1, ddof=1)

    percentages.append(count_intervals_student(data_means, data_std, 67, num_datapoints) / num_datasets)

percentages
```

Out[ ]:

```
[0.06621,
 0.0662,
 0.06759,
 0.06587,
 0.06715,
 0.06568,
 0.06547,
 0.06597,
 0.06565,
 0.06707]
```