What did I do?

I opened colab:

I observed previous examples created in class.
I then borrowed the imports we created.

```
import numpy as np
import pandas as pd
```

I then considered what preprocessing this dataset may need?

As per previous bodies of work I:
Extract values and data and verify normalisation.

I referred to my presentation from the last model and tried simple attempts at preprocessing the data.

Preprocessing attempts made were:

- Normalisation: The pixel values in the MNIST dataset are grayscale values ranging from 0 to 255. Normalisation is a common preprocessing step that scales the pixel values to a common range, typically between 0 and 1, to make it easier for the algorithm to learn from the data.

- Flattening: The MNIST dataset consists of images of size 28x28 pixels. Some algorithms, such as the Bayes Classifier and KNN, require the input data to be in the form of a vector. Flattening the images into a vector of length 784 (28x28) can make it easier for these algorithms to process the data.

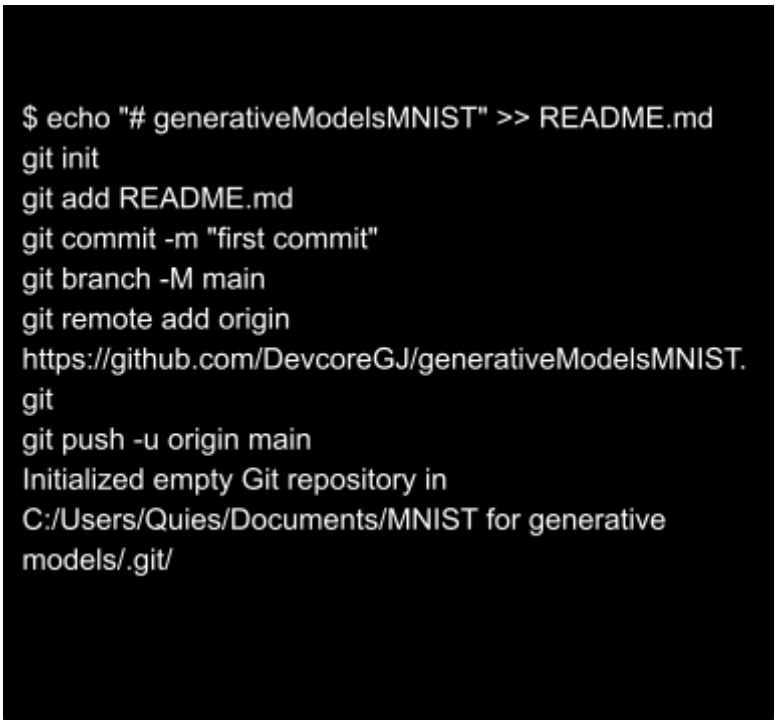
- Dimensionality Reduction: If the dataset is very large, it may be computationally expensive to perform classification directly on the raw images. In such cases, dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-SNE can be applied to reduce the number of features and simplify the data.

- Data Augmentation: Data augmentation is a technique that can be used to increase the size of the dataset by generating new images with minor variations such as rotations, scaling, or translations. This can help improve the generalization performance of the model.

From the results being thrown back at my I discovered the file most likely didn't need flattening or it's data augmented.

After Normalisation and Dimensionality reduction had been implemented, that is about the time I downloaded the notebook. Once the notebook was downloaded locally, I opened Git Bash and navigated to the file and initialised local repository.

I then continued to make a GitHub repo publically available.



I then considered the task at hand, using google classroom.

> Create a use case using the MNIST dataset and Implement a Bayes Classifier or KNN to recognize handwritten numbers.

Performing PCA

```
from sklearn.decomposition import PCA
```

the dimensionality of the data will be reduced to 50 features. The fit_transform method is then used to fit the PCA model to the data and transform it into the reduced feature space.

Split the data in to training and validation

```
from sklearn.model_selection import train_test_split
```

Implemented classification

Bayes classifier

- Implement GaussianNB classifier
- Initilise an instance of bayes object
- Call fit method on bayes object this trains out classifier with c_train and y_train
- Print the accuracy.
- Initially implemented with the scikit-learn's GaussianNB class due to time constraints with the project.The hard way
- I revisited my class example of the naive bayse for calculating the probability of sexes.I then refactored it for my model.

I replaced this:

```
# Split the dataset into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

With this:

```
# Train a Naive Bayes classifier
class_means = np.zeros((10, X_train.shape[1]))
class_vars = np.zeros((10, X_train.shape[1]))
class_counts = np.zeros(10)
```

```
class GaussNB:

    def __init__(self, epsilon=1e-6):
        self.epsilon = epsilon

    def fit(self, X_train, y_train):
        self.likelihood = {}
        self.priors = {}
        self.classes = set(y_train.astype(int))
        for class_label in self.classes:
            X_class = X_train[y_train == class_label, :]
            self.likelihood[class_label] = {"mean": X_class.mean(axis=0), "cov": np.cov(X_class.T)
+ self.epsilon * np.eye(X_train.shape[1])}
            self.priors[class_label] = len(X_class) / len(X_train)

    def predict(self, X_val):
        N, D = X_val.shape
        P_hat = np.zeros((N, len(self.classes)))
        for class_label, likelihood in self.likelihood.items():
            P_hat[:, class_label] = multivariate_normal.logpdf(X_val, likelihood["mean"],
likelihood["cov"]) + np.log(self.priors[class_label])
        return P_hat.argmax(axis=1)
```

and replace this:

```
# Create a Bayes classifier
bayes = GaussianNB()

# Train the classifier on the training data
bayes.fit(X_train, y_train)

# Evaluate the classifier on the validation data
score = bayes.score(X_val, y_val)

# Print the accuracy score
print('Bayes accuracy:', score)
```

With this:

```
gnb = GaussianNB()

gnb.fit(X_train, y_train)
y_hat_train = gnb.predict(X_train)
acc = accuracy_score(y_train, y_hat_train)

y_hat = gnb.predict(X_train)

acc = accuracy_score(y_train, y_hat)

print(acc)
```

I then did a very similar thing with the KNN implementation.

Title: Recognizing Handwritten Numbers using Bayes Classifier or KNN with the MNIST dataset

Goal: To implement a machine learning model that can accurately recognize handwritten numbers using the MNIST dataset, either through a Bayes Classifier or KNN algorithm.

Actors: Junior Data Scientist

Preconditions:

- Access to the MNIST dataset.
- A programming environment, such as Google Colab or Jupyter Notebook, that can execute Python code.
- Basic knowledge of machine learning concepts and techniques, including data preprocessing, dimensionality reduction, and classification algorithms.
- Flow of Events:
- The Data Scientist opens a programming environment and imports necessary libraries, including NumPy and Pandas.
- The Data Scientist considers what preprocessing the dataset may need, including normalisation, flattening, dimensionality reduction, and data augmentation.
- The Data Scientist preprocesses the data by applying necessary techniques, such as normalisation and dimensionality reduction using PCA.
- The Data Scientist splits the data into training and validation sets using scikit-learn's train_test_split method.
- The Data Scientist implements a classification algorithm, such as the Bayes Classifier or KNN, using scikit-learn's GaussianNB or KNeighborsClassifier classes.

- The Data Scientist trains the classifier on the training data and evaluates its accuracy on the validation data.
- If necessary, the Data Scientist refactors the algorithm for better performance and re-evaluates its accuracy.
- The Developer downloads the notebook and initializes a local Git repository.
- The Developer makes the notebook publicly available on GitHub for others to access and use.

Postconditions:

- A machine learning model has been developed that can accurately recognize handwritten numbers using the MNIST dataset.
- The model is available for others to access and use through the publicly available GitHub repository.