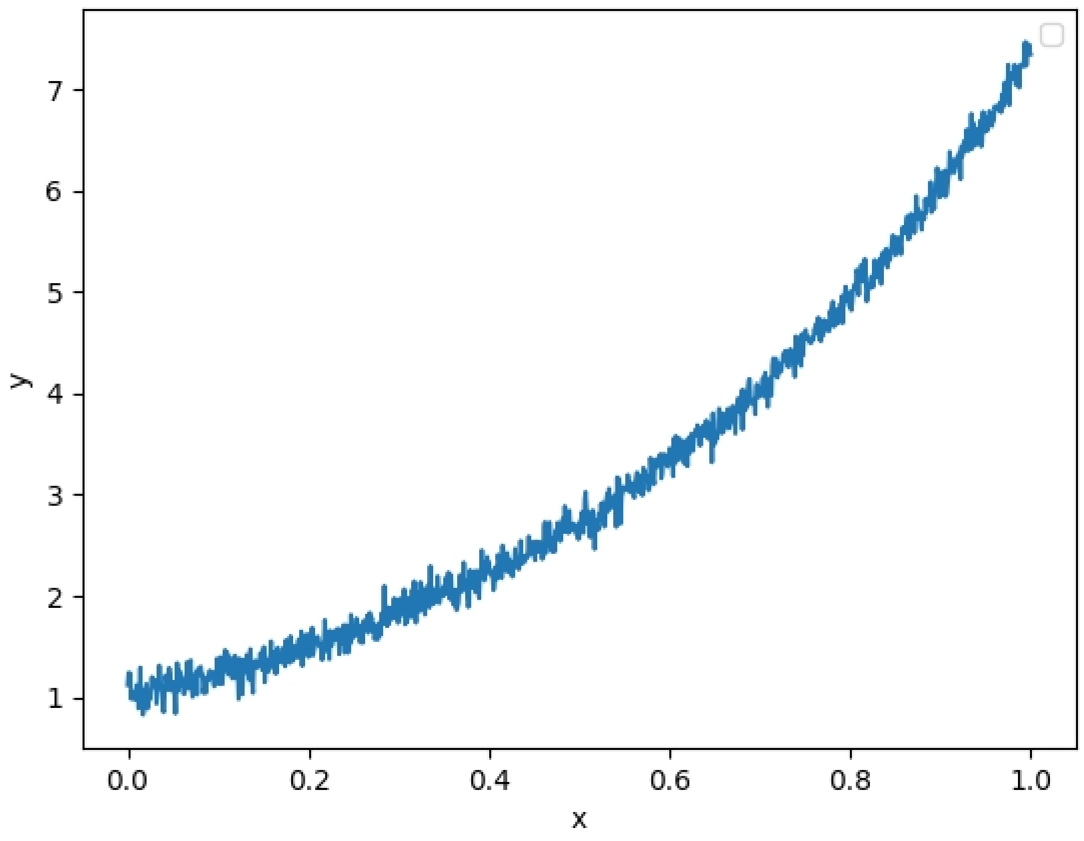
資料三 110703065 詹松霖

All models are implmented by myself.
All the codes are in Github Repo
Nice looking markdown version

1.

1.1 Plot the data using plot function.

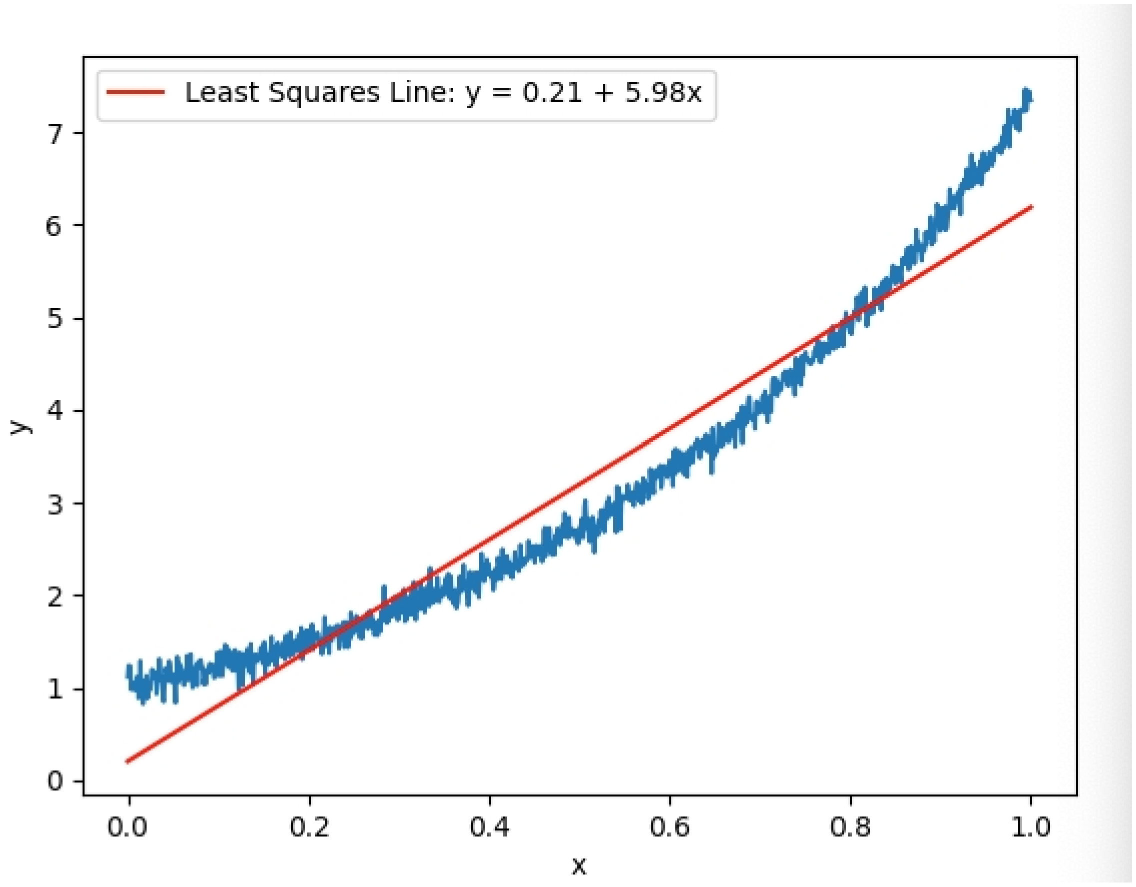
```python
import scipy.io
import matplotlib.pyplot as plt

# load data
mat_data = scipy.io.loadmat('data.mat')
x = mat_data['x'].flatten()
y = mat_data['y'].flatten()

# plot data
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



1.2

code
theta_0 = 0.2070272
theta_1 = 5.98091717

mse = 0.20580596682517396

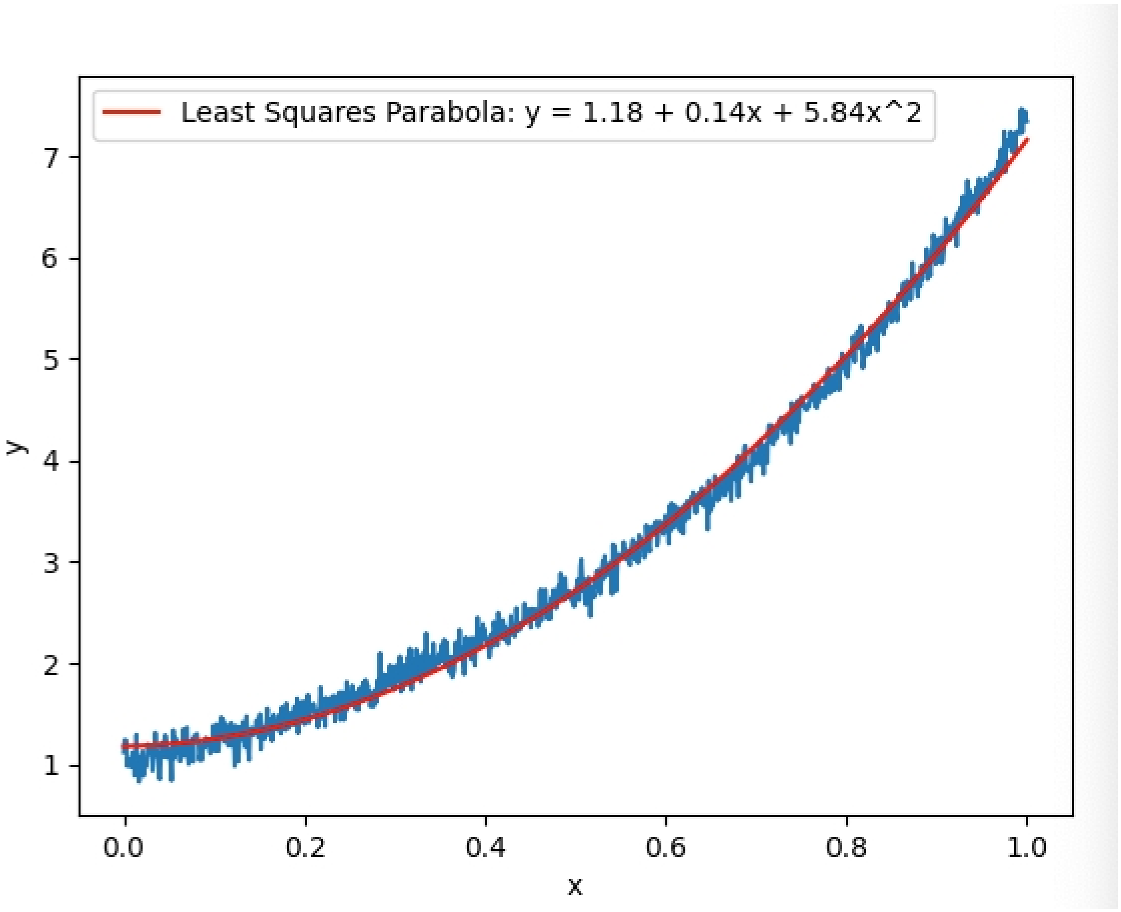


1.3

code

y = 1.18 + 0.14x + 5.84x^2

mse = 0.015744919931207565

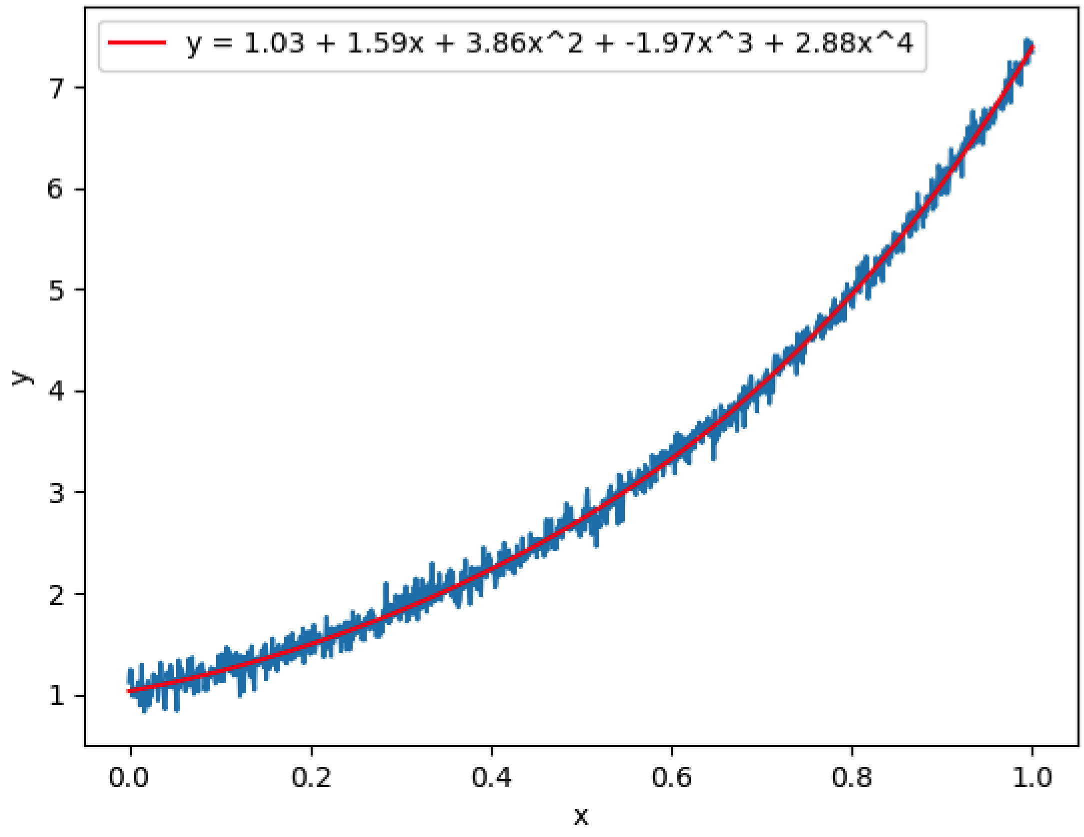


1.4

code

y = 1.03 + 1.59x + 3.86x^2 + -1.97x^3 + 2.88x^4

mse = 0.010413051044174356

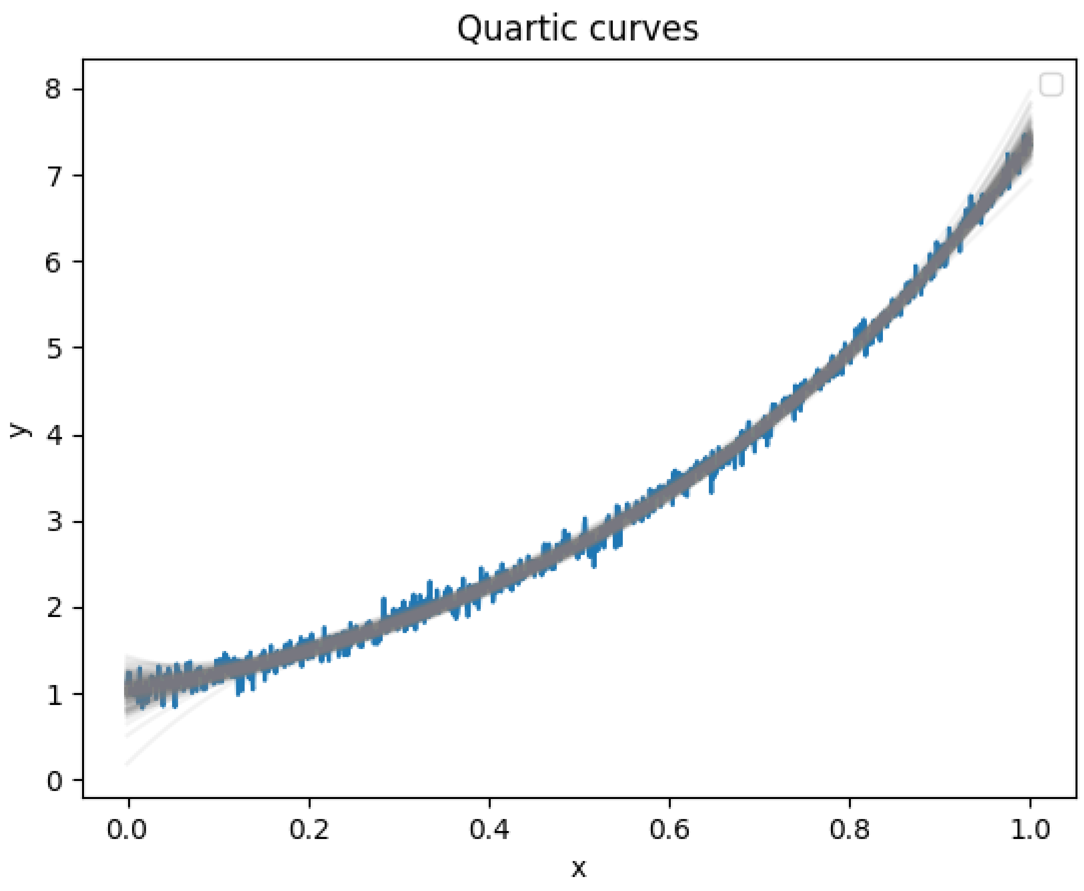


1.5

The model with the smallest mean square error fits the dataset the best, which makes it more suitable than the others. The quartic curve has the smallest mean square error with 0.01, so it's the most suitable.

2

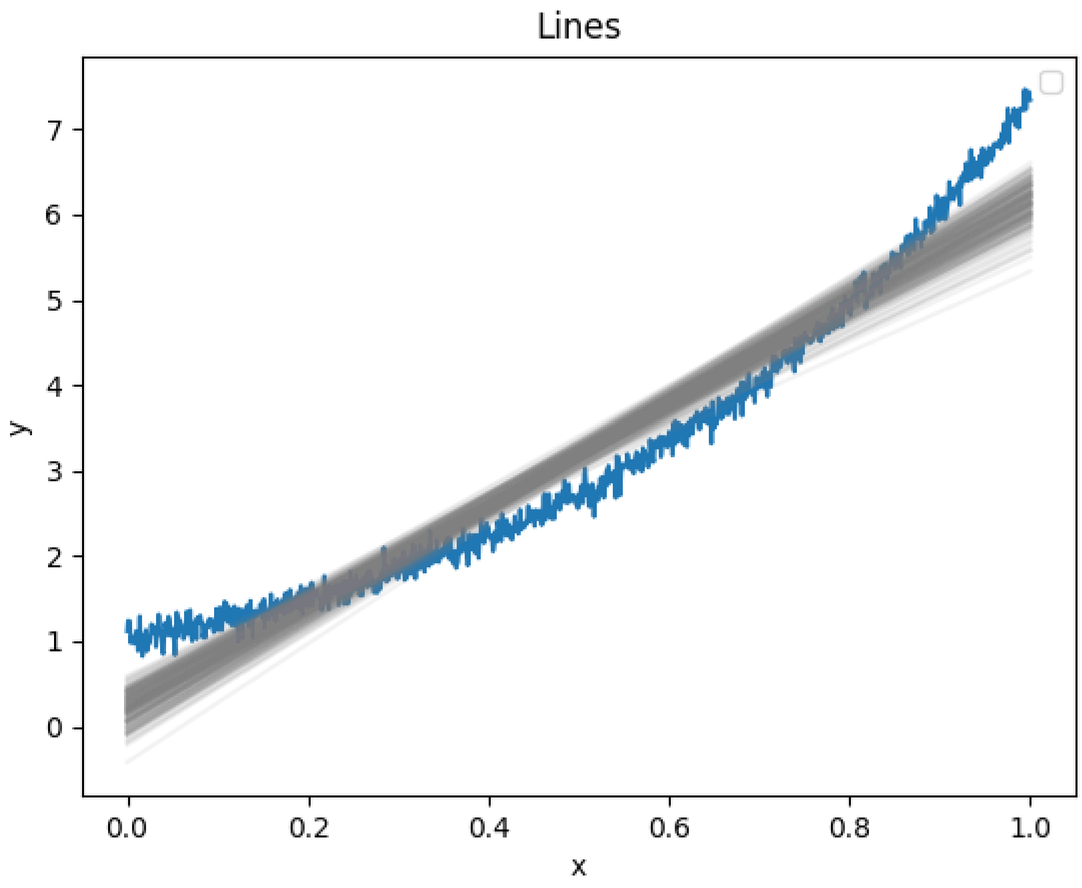
Curves



Bias:

The quartic model is more complex and can capture more intricate patterns in the data. It is able to fit the data more closely, indicating **lower** bias.

Lines



Bias:

The linear model has a simpler form, which may not be able to capture the complexity of the underlying data if it follows a more curved pattern. This leads to **higher** bias, as the model is less flexible and might not fit the data well.

Variance

I'm not quite sure how to determine which model has a higher variance, because quartic curves are densely concentrated in the middle portion, so if we only focus on the middle, the variance appears to be smaller than the lines. However, if we take a step back and examine the edges, quartic curves are more spread out. If we only consider the tails, then the variance is greater for quartic curves compared to the lines. After doing some research online, I've come to understand that this is a concept known as "local smoothness" versus "global smoothness". Different models may perform differently in different parts of the data. When it comes to choosing models, it might be necessary to consider the purpose of the model and focus on the segments of interest, such as the middle portion.

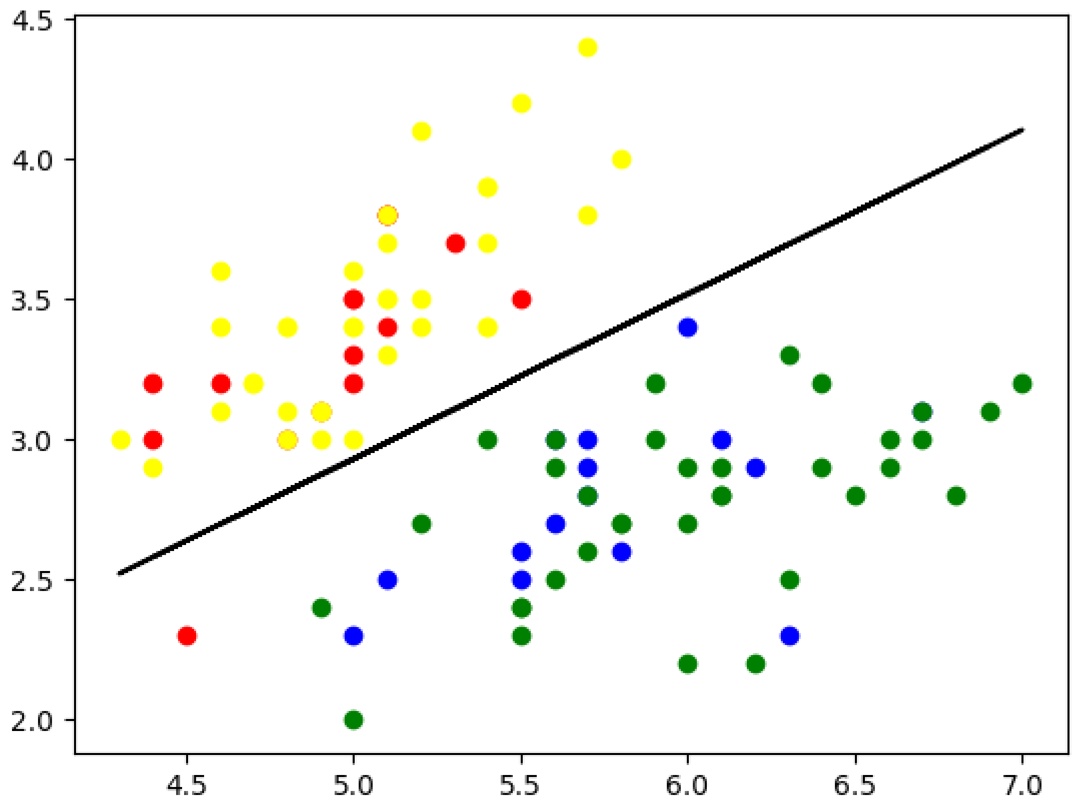
3

---

code
Test Accuracy: 0.97

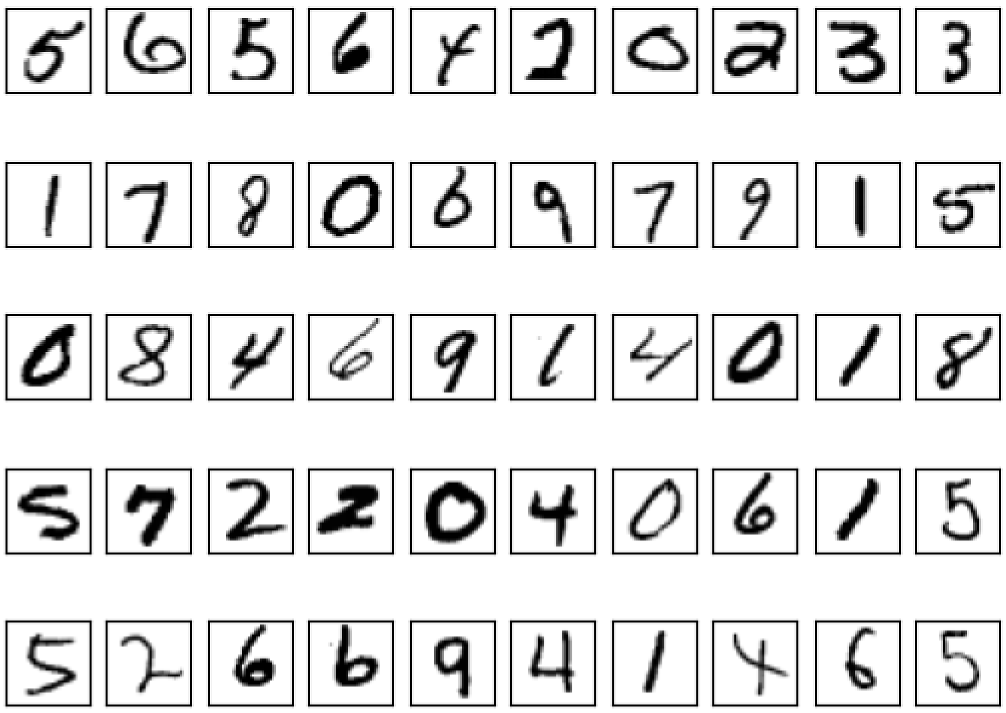
decision boundary:
theta0: -0.060600000000000084

theta1: 0.10332857142857141



4

4.1



code

4.2

full code

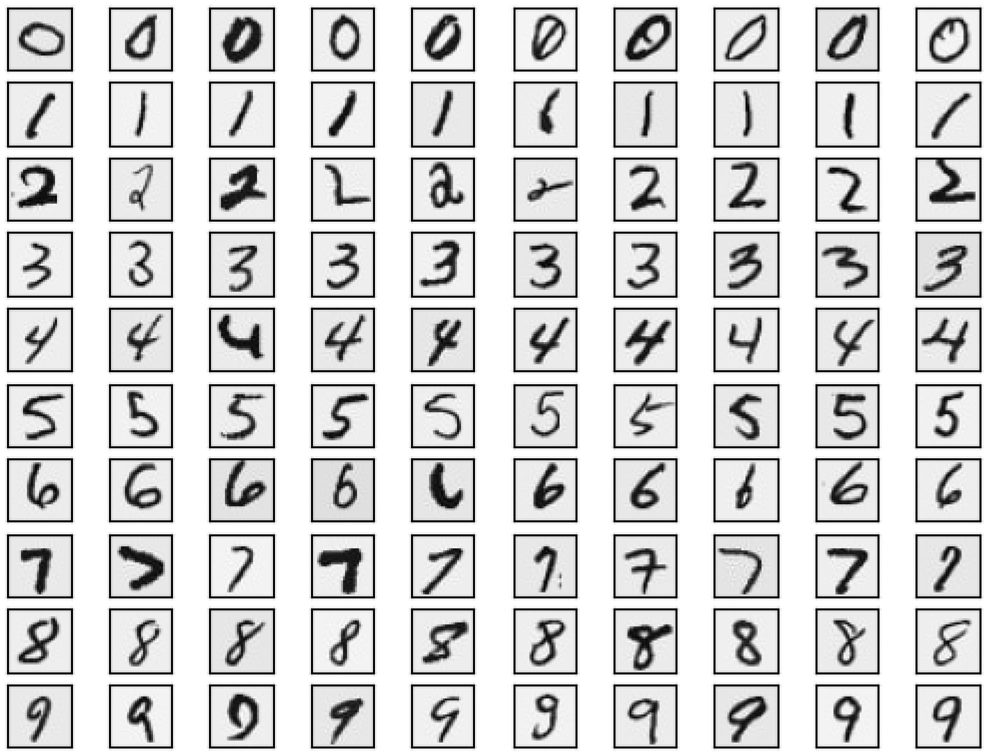
```
# 28*28 => 1*784
x_selected = x_selected.reshape(5000, -1)
# Normalize the data
mean = np.mean(x_selected, axis=0)
std_dev = np.std(x_selected, axis=0)

normalized_images = (x_selected - mean) / (std_dev + 1e-10 )

# Step 3: Compute Covariance Matrix
cov_matrix = np.cov(normalized_images, rowvar=False)

# Step 4: Compute Eigenpairs
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Step 5: Sort Eigenpairs
sorted_indices = np.argsort(eigenvalues)[::-1]
sorted_eigenvalues = eigenvalues[sorted_indices]
sorted_eigenvectors = eigenvectors[:, sorted_indices]
```

4.3

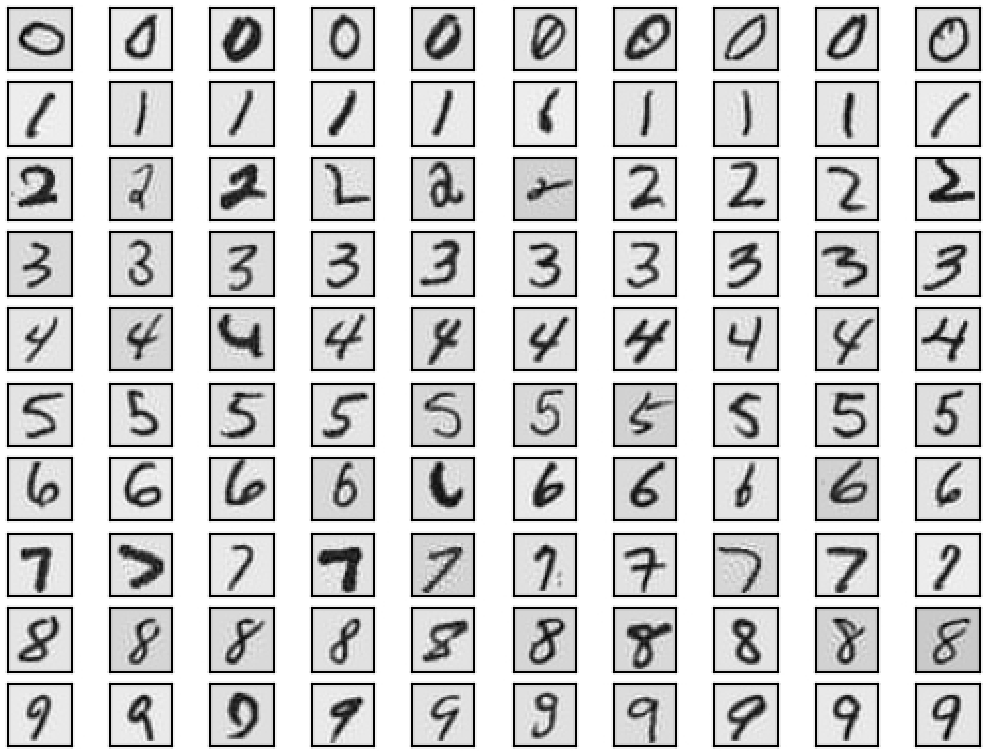
full code

```
def doPCA(dimension: int):
  # Step 1: Select the top {dimension} eigenvectors
  top_eigenvectors = sorted_eigenvectors[:, :dimension]

  # Step 2: Project the normalized data onto the top {dimension}
eigenvectors
  reduced_data = np.dot(normalized_images, top_eigenvectors)
  # Step 1: Project back to the original space
  decoded_data = np.dot(reduced_data, top_eigenvectors.T)

  # Step 2: Add back the mean
  decoded_data = (decoded_data * std_dev) + mean
  decoded_data = decoded_data.reshape(5000, 28, 28).astype('float64')

  lines = 10
  columns = 10

  fig = plt.figure()
  fig.suptitle(f'PCA with {dimension} dimensions')
  for i in range(lines):
    for j in range(columns):
      ax = fig.add_subplot(lines,columns, 1+i*10+j)
      plt.imshow(decoded_data[i*500+j,:,:], cmap='binary')
      plt.sca(ax)
      ax.set_xticks([], [])
      ax.set_yticks([], [])


doPCA(500)
doPCA(300)
doPCA(100)
doPCA(50)
plt.show()
```

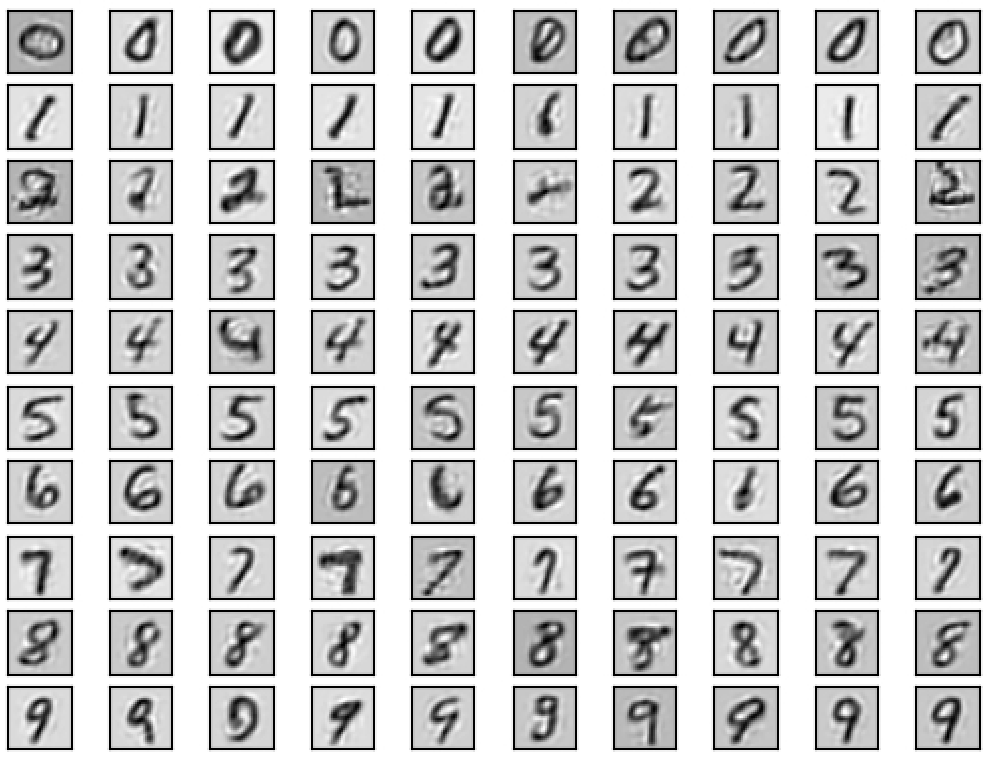
PCA with 500 dimensions



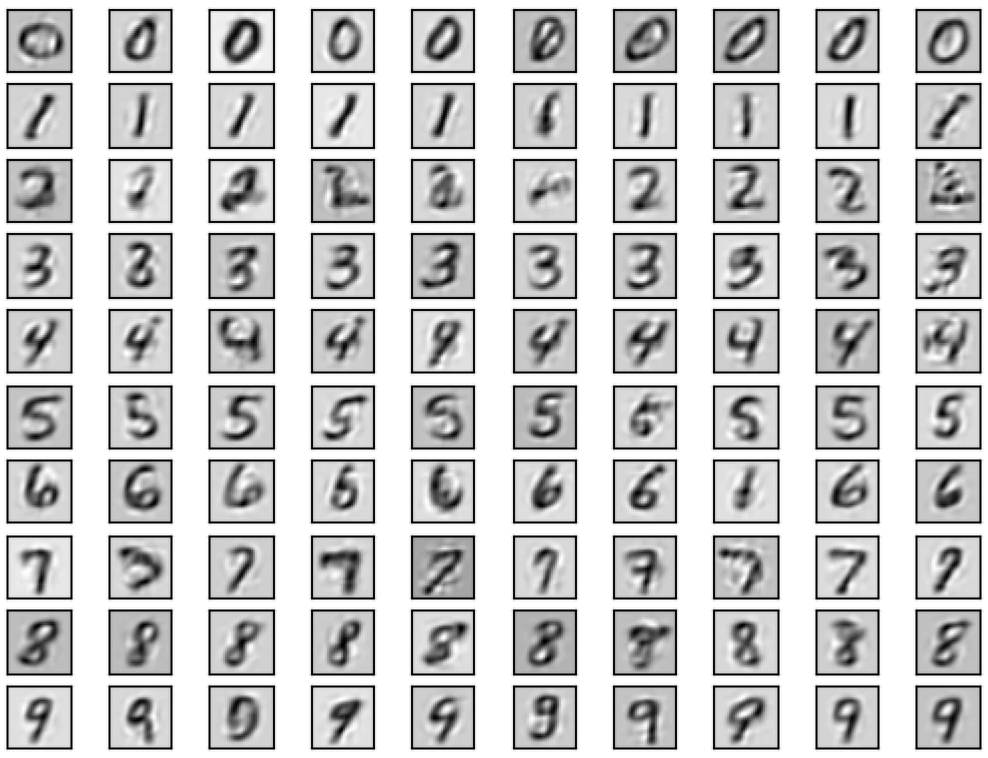
PCA with 300 dimensions



PCA with 100 dimensions



PCA with 50 dimensions



As the dimension being reduced, we are effectively representing the data with fewer features. This could cause detail loss. The decoded images with lower dimensions will likely be more blurred or have less distinct features compared to those with higher dimensions. With extremely low dimensions (like 50), some digits might become difficult to distinguish, as there's less information available for reconstruction(decoding).