Adam Gordon(2275253)
Jaden Harris(2169997)

Question 4a:

i)

```
#i)
#Sample 150 x-values from a Normal distribution using a mean of 0 and standard deviation of 10.
#np.random.seed(1)
data=np.random.normal(loc=0,scale=10,size=150)
print(data)
```

✓ 0.2s Python

ii)

```
#ii)

design_matrix=[]
for i in data:
    row=[1,i,i**2]
    design_matrix.append(row)

design_matrix=np.array(design_matrix)
print(design_matrix)
```

✓ 0.4s Python

iii)

```
#iii
#true_theta=np.random.normal(loc=0,scale=1,size=3)
true_theta=np.random.uniform(size=3)
print(true_theta)
```

✓ 0.3s Python

```
[0.26636259 0.98627482 0.77943831]
```

iv)

```
#iv
y=lambda x: true_theta[0]+true_theta[1]*x+true_theta[2]*x**2
y_vals=[]
count=0
for i in data:
    y_vals.append(y(i))

y_vals=np.array(y_vals)
y_vals = np.random.normal(y_vals, 2)
print(y_vals)
```

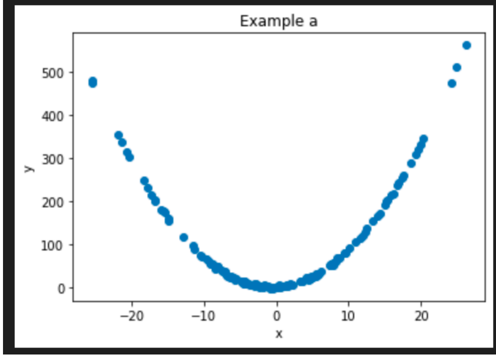
✓ 0.3s Python

v)

```
#v
plt.scatter(data,y_vals)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Example a")
plt.show
```

✓ 0.1s

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



vi)

```
#vi
split_ratio = 3/4
validation_split_ratio = 4/5
#take size of dataset and multiply by split ratio to get index of testing data
testing_split_index = round(data.shape[0] * split_ratio)
#use validation ratio to split into testing and validation
validation_split_index = round(testing_split_index * validation_split_ratio)
# training set
#using inexes for splits done above generate datasets
training_x_values = data[:validation_split_index]#index form 0 to index from above
training_y_values = y_vals[:validation_split_index]#same for y vals
training_set=[training_x_values,training_y_values]#combine into one dataset
# validation set
validation_x_values = data[validation_split_index:testing_split_index]#index from val index to the testing ind
validation_y_values = y_vals[validation_split_index:testing_split_index]#same for y vals
validation_set=[validation_x_values,validation_y_values]
# testing set
testing_x_values = data[testing_split_index:]#index from testingf splt to the end (112-150)
testing_y_values = y_vals[testing_split_index:]#same for y
testing_set=[testing_x_values,testing_y_values]

"""
print("Trainging set: ",training_set)
print("Validation set: ", validation_set)
print("Training set: ",training_set)
"""
```

✓ 0.2s Python

Question 4b:
i)

```python
#i
def calc_design_matrix(x):
    design_matrix=[]
    for i in x:
        row=[1,i,i**2]
        design_matrix.append(row)

    design_matrix=np.array(design_matrix)
    return design_matrix

def closed_form_soln(training_set):
    training_design_matrix=calc_design_matrix(training_set[0])
    theta=np.linalg.inv(training_design_matrix.transpose().dot(training_design_matrix)).dot(training_design_ma
    return theta

closed_form_soln_thetas=closed_form_soln(training_set)
print(closed_form_soln(training_set))
```

✓ 0.2s Python

```
[0.35464189 1.00606658 0.77927576]
```

ii)

```
[0.26636259 0.98627482 0.77943831]-True values
[0.35464189 1.00606658 0.77927576]-Closed Form Solution
Therefore very close
```

iii)

Training error: 236.18885145109022
Validation error: 43.377042627265425
Testing error: 79.45700171332469

```python
#iii
def calc_error(y_true, y_hat):
    store_errors=[]
    for i in range(len(y_true)):
        calc_errors=(y_true[i]-y_hat[i])**2
        store_errors.append(calc_errors)
    error=(sum(store_errors))/2
    return error

def calc_error_gd(y_true, y_hat):
    store_errors=[]
    for i in range(len(y_true)):
        calc_errors=(y_true[i]-y_hat[i])**2
        store_errors.append(calc_errors)
    error=(sum(store_errors))
    return error

y_hat_training=[]
for i in training_set[0]:
    predicted_val=closed_form_soln_thetas[0]+(closed_form_soln_thetas[1]*i)+(closed_form_soln_thetas[2]*i**2)
    y_hat_training.append(predicted_val)

error_t=calc_error(training_set[1],y_hat_training)
print("Training error: ",error_t)

y_hat_validation=[]
for i in validation_set[0]:
     predicted_val=closed_form_soln_thetas[0]+(closed_form_soln_thetas[1]*i)+(closed_form_soln_thetas[2]*i**2)
     y_hat_validation.append(predicted_val)

error_v=calc_error(validation_set[1],y_hat_validation)
print("Validation error: ", error_v)

y_hat_testing=[]
for i in testing_set[0]:
    predicted_val=closed_form_soln_thetas[0]+(closed_form_soln_thetas[1]*i)+(closed_form_soln_thetas[2]*i**2)
    y_hat_testing.append(predicted_val)
error_test=calc_error(testing_set[1],y_hat_testing)
print("Testing error: ",error_test)
```

✓ 0.3s Python

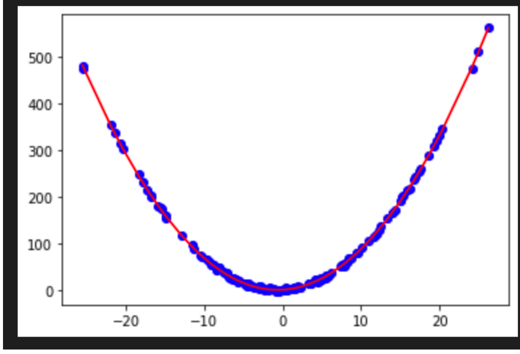
```
Training error:  236.18885145109022
Validation error:  43.377042627265425
Testing error:  79.45700171332469
```

iv)

```python
#iv
sorted_x = np.sort(data)
plt.scatter(data, y_vals,color="blue")
to_plot_y=[]
for i in sorted_x:
    predicted_val=closed_form_soln_thetas[0]+(closed_form_soln_thetas[1]*i)+(closed_form_soln_thetas[2]*i**2)
    to_plot_y.append(predicted_val)
plt.plot(sorted_x,np.array(to_plot_y),color="red")
```

✓ 0.1s Python

```
[<matplotlib.lines.Line2D at 0x7faa5b3707f0>]
```



v)

```python
def gradient_decent_2(design_matrix,y_vals,alpha,theta,stop):
    theta0=theta[0]
    theta1=theta[1]
    theta2=theta[2]
    #predicted_vals_final=[]
    store_error_totals=[]
    for i in range(stop):
        predicted_vals=[]
        for j in range(len(design_matrix)):
            predicted_val=theta0+(theta1*design_matrix[j])+(theta2*(design_matrix[j]**2))
            loss=predicted_val-y_vals[j]
            predicted_vals.append(predicted_val)
            theta0=theta0-alpha*(loss)
            theta1=theta1-alpha*(loss)*(design_matrix[j])
            theta2=theta2-alpha*(loss)*(design_matrix[j]**2)

        store_error_totals.append(calc_error(y_vals,predicted_vals))


    theta=[theta0,theta1,theta2]
    return [theta,store_error_totals]

training_design_matrix=calc_design_matrix(training_x_values)
theta_s,error_gd=gradient_decent_2(training_x_values,training_y_values,0.000005,[0.1,0.1,0.1],500)

print("Theta values: ",theta_s)
```

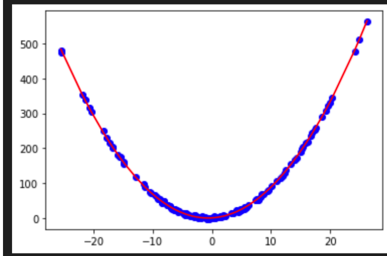
✓ 0.1s Python

```
[0.1713580046885465, 1.0289963555229389, 0.781931825668917]
```

```python
sorted_data=np.sort(data)
plt.scatter(data,y_vals,color="blue")
plt.plot(sorted_data,theta_s[0]+theta_s[1]*sorted_data+theta_s[2]*(sorted_data**2),color="red")
```

✓ 0.2s Python

```
[<matplotlib.lines.Line2D at 0x7faa60789400>]
```




```python
training_pred=theta_s[0]+theta_s[1]*training_x_values+theta_s[2]*(training_x_values**2)
training_error=calc_error(training_y_values,training_pred)
#errors[0,1]=training_error
print("Training error: ",training_error)

validation_pred=theta_s[0]+theta_s[1]*validation_x_values+theta_s[2]*(validation_x_values**2)
validation_error=calc_error(validation_y_values,validation_pred)
#errors[1,1]=validation_error
print("Validation error: ",validation_error)

testing_pred=theta_s[0]+theta_s[1]*testing_x_values+theta_s[2]*(testing_x_values**2)
testing_error=calc_error(testing_y_values,testing_pred)
#errors[2,1]=testing_error
print("Testing error: ",testing_error)
```

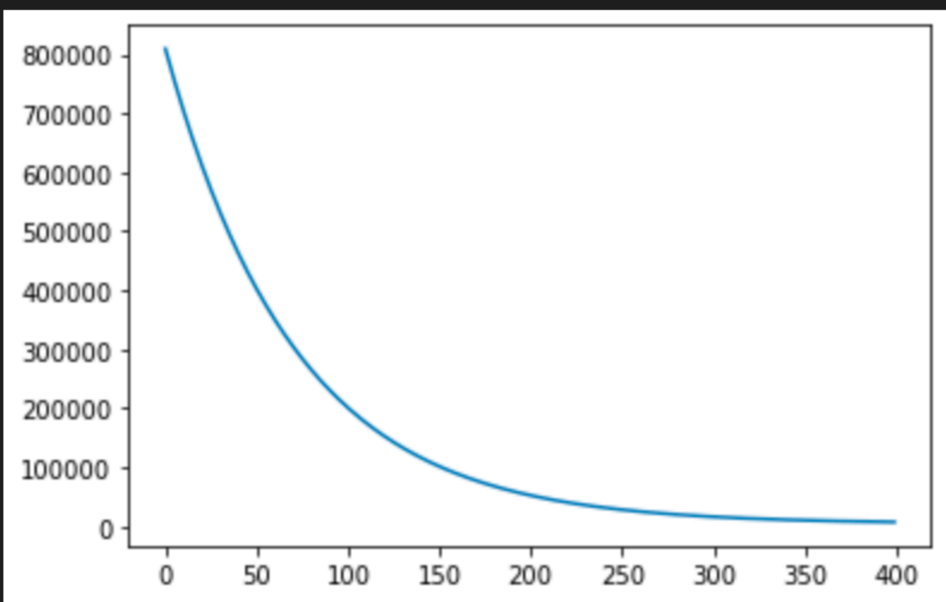
✓ 0.2s

```
Training error:  247.02274298085337
Validation error:  37.13214882224015
Testing error:  83.06954258281709
```

```python
plt.plot(error_gd)
```

✓ 0.1s

```
[<matplotlib.lines.Line2D at 0x7faa61f1f760>]
```



Question 4c:

i)

```python
def calc_design_matrix_3d(x):
    design_matrix=[]
    for i in x:
        row=[1,i,i**2,i**3]
        design_matrix.append(row)

    design_matrix=np.array(design_matrix)
    return design_matrix
```

iv)

- The third order gradient descent model achieve the lowest training error (it overfits the data)
- The second order gradient descent model trains the fastest
- The third order gradient descent model has a better validation error
- Yes, the third order model tries to fit each data point much more than the second order model.
- Yes the Third order model achieved a lower Training error than the closed form solution model, but not valuation error.

v)

- The Closed form model obtains the lowest testing error
- Yes, the regularisation improved the test error performance of the third order gradient descent
- The Third order gradient descent model with regularisation achieved the best test error
- Use a model high order model with regularisations as this way we can test more functions and have more flexibly when fitting the data, it also achieved the lowest test error of the gradient descent methods.