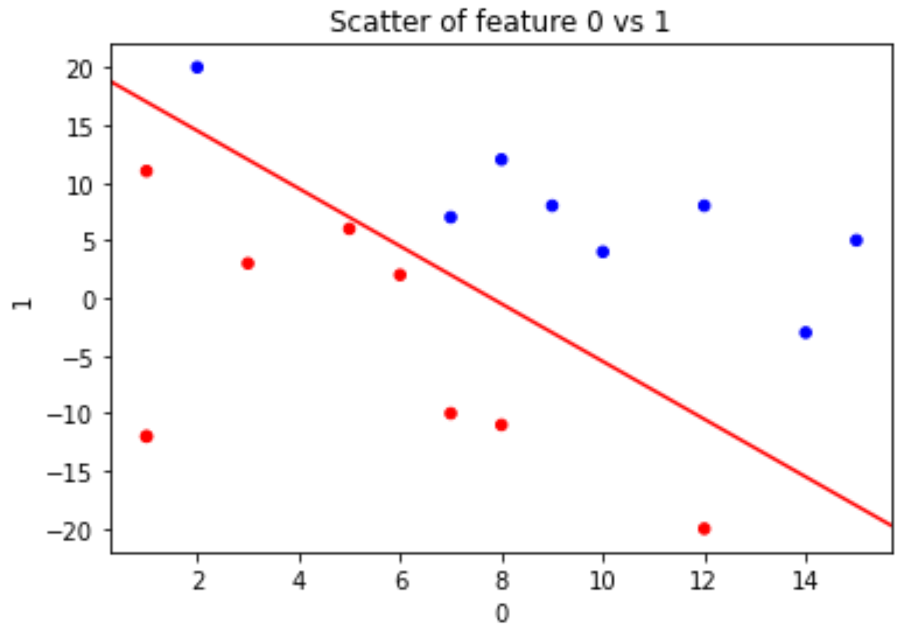
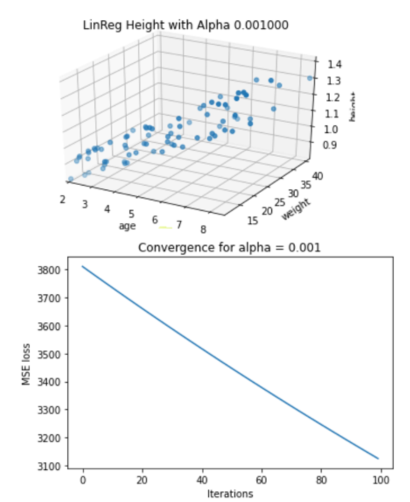
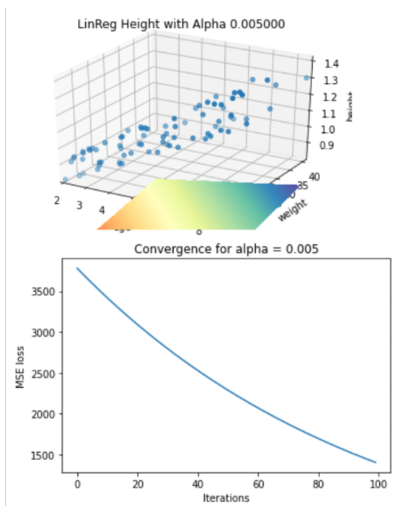
1 **Perceptron** decision boundary:

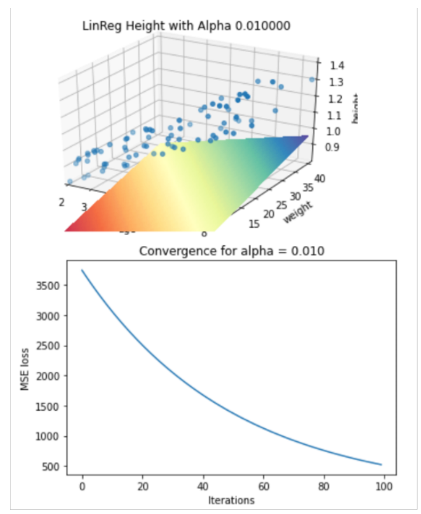


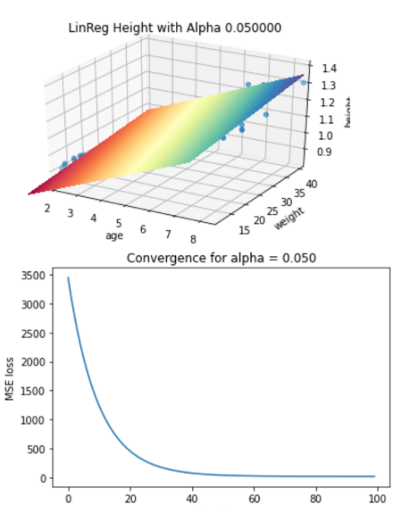
2 **Linear Regression** results (custom choice: alpha = 0.6 for 30 iterations)

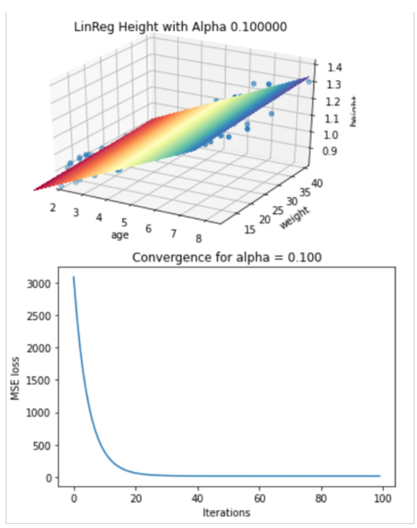
The objective in choosing alpha, the learning rate for the Linear Regression (LR) algorithm, is to select a value that converges on a loss minimum in a reasonable time. If alpha is too high, the algorithm may skip the minima and fail to converge, while if alpha is too low, the algorithm will converge too slowly. The plots below show the Mean Squared Error (MSE) loss against the number of iterations, using the nine provided alpha values (0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, and 10) as well as the custom value that was selected. As instructed, for all of the provided alphas, the algorithm is set to run for 100 iterations. Based on the illustrations, it can be seen that the MSE converges faster as alpha increases, with the fastest convergence occurring for an alpha of 0.5. Although the model still converges at the higher alpha value of 1 (albeit slightly more slowly than for 0.5), the model does not converge at all when alpha becomes too large (5 or 10). When alpha is 0.5 or 1, it can be observed that the model converges in fewer than 30 iterations. The choice for a tenth rate and number of iterations is alpha 0.6 with 30 iterations. From the last set of plots below, it can be seen that this choice converges quickly to a minimum MSE loss in a relatively small number of iterations.

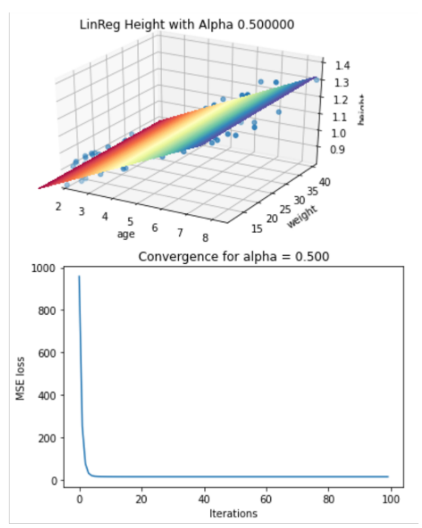


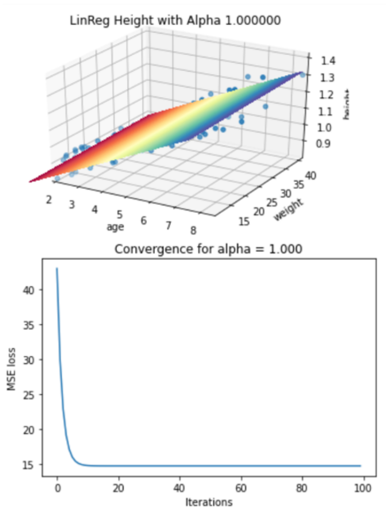


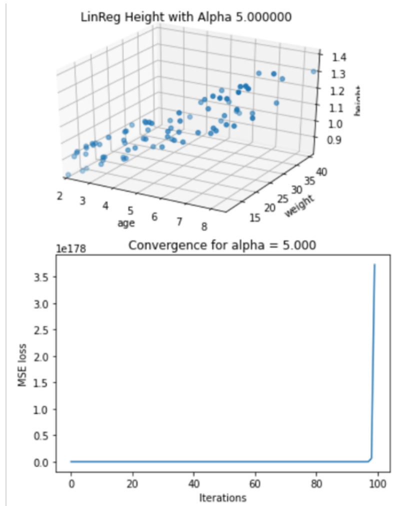


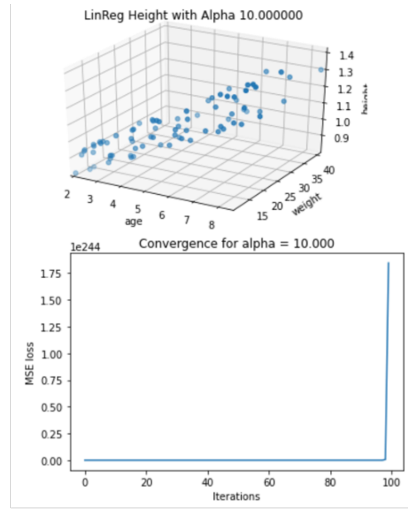


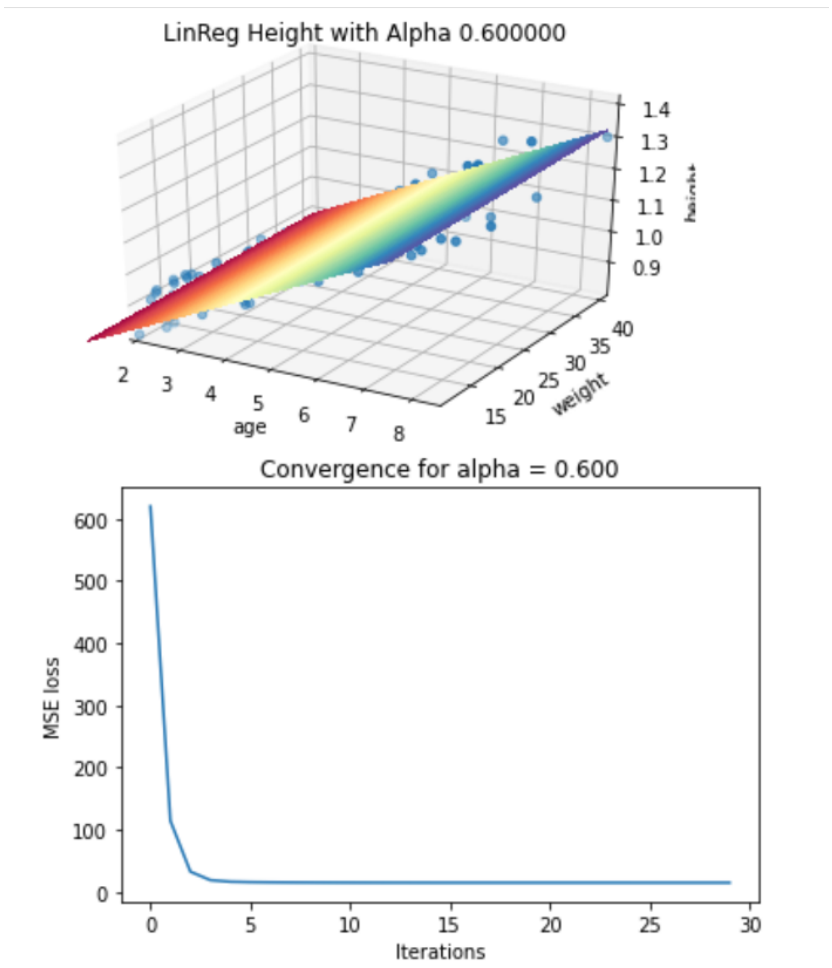





LinReg Height with Alpha 1.000000
age
weight
height
Convergence for alpha = 1.000
MSE loss
Iterations


LinReg Height with Alpha 5.000000
age
weight
height
1e178
Convergence for alpha = 5.000
MSE loss
Iterations


LinReg Height with Alpha 10.000000
age
weight
height
1e244
Convergence for alpha = 10.000
MSE loss
Iterations


LinReg Height with Alpha 0.600000
age
weight
Convergence for alpha = 0.600
MSE loss
Iterations

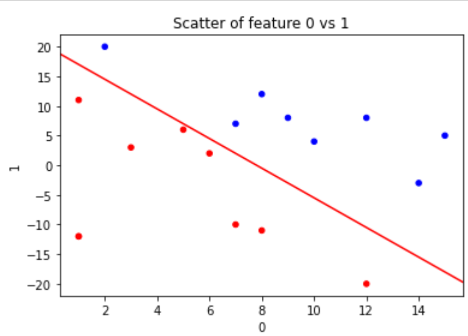
In [ ]:

```python
# plot_db.py

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sys
from matplotlib import cm
import matplotlib.lines as mlines
from mpl_toolkits.mplot3d import Axes3D

"""
Author: Kelsey D'Souza

This file contains two functions for visualizing 2-feature labeled datasets.
Its purpose is to give you ideas on how to vizualize your data and use pandas
and matplotlib, feel free to snippets of any code in here or import the file
into your programs.

This file *does not* need to be included in your submission unless imported.

visualize_scatter
        Assumes binary labels and creates a line between the data using the given
        feature and bias weights.
        Note: weights should be input as [w1, w2, bias]

visualize_3d
        Plots data points in 3D space using feat1 x feat2 on the x-y base, and
        label as the data point height along the z-axis.
        It then creates a 3D surface plot of the continuous label model using
        the given linear regressor weights.

"""


def visualize_scatter(df, feat1=0, feat2=1, labels=2, weights=[-1, -1, 1],
                      title=''):
    """
        Scatter plot feat1 vs feat2.
        Assumes +/- binary labels.
        Plots first and second columns by default.
        Args:
          - df: dataframe with feat1, feat2, and labels
          - feat1: column name of first feature
          - feat2: column name of second feature
          - labels: column name of labels
          - weights: [w1, w2, b]
    """

    # Draw color-coded scatter plot
    colors = pd.Series(['r' if label > 0 else 'b' for label in df[labels]])
    ax = df.plot(x=feat1, y=feat2, kind='scatter', c=colors)

    # Get scatter plot boundaries to define line boundaries
    xmin, xmax = ax.get_xlim()

    # Compute and draw line. ax + by + c = 0  =>  y = -a/b*x - c/b
    a = weights[0]
    b = weights[1]
    c = weights[2]

    def y(x):
        return (-a/b)*x - c/b

    line_start = (xmin, xmax)
    line_end = (y(xmin), y(xmax))
    line = mlines.Line2D(line_start, line_end, color='red')
    ax.add_line(line)


    if title == '':
        title = 'Scatter of feature %s vs %s' %(str(feat1), str(feat2))
    ax.set_title(title)

    plt.show()


def visualize_3d(df, lin_reg_weights=[1,1,1], feat1=0, feat2=1, labels=2,
                 xlim=(-1, 1), ylim=(-1, 1), zlim=(0, 3),
                 alpha=0., xlabel='age', ylabel='weight', zlabel='height',
                 title=''):
    """
    3D surface plot.
    Main args:
      - df: dataframe with feat1, feat2, and labels
      - feat1: int/string column name of first feature
      - feat2: int/string column name of second feature
      - labels: int/string column name of labels
      - lin_reg_weights: [b_0, b_1 , b_2] list of float weights in order
    Optional args:
      - x,y,zlim: axes boundaries. Default to -1 to 1 normalized feature values.
      - alpha: step size of this model, for title only
      - x,y,z labels: for display only
      - title: title of plot
    """
```

```
    # Setup 3D figure
    ax = plt.figure().gca(projection='3d')

    # Add scatter plot
    ax.scatter(df[feat1], df[feat2], df[labels])
    #plt.hold(True)

    # Set axes spacings for age, weight, height
    axes1 = np.arange(xlim[0], xlim[1], step=.05)  # age
    axes2 = np.arange(xlim[0], ylim[1], step=.05)  # weight
    axes1, axes2 = np.meshgrid(axes1, axes2)
    axes3 = np.array( [lin_reg_weights[0] +
                       lin_reg_weights[1]*f1 +
                       lin_reg_weights[2]*f2  # height
                       for f1, f2 in zip(axes1, axes2)] )
    plane = ax.plot_surface(axes1, axes2, axes3, cmap=cm.Spectral,
                            antialiased=False, rstride=1, cstride=1)



    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_zlabel(zlabel)
    ax.set_xlim3d(xlim)
    ax.set_ylim3d(ylim)
    ax.set_zlim3d(zlim)

    if title == '':
        title = 'LinReg Height with Alpha %f' % alpha
    ax.set_title(title)

    plt.show()
```

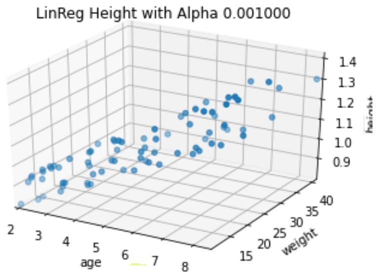
In [ ]:

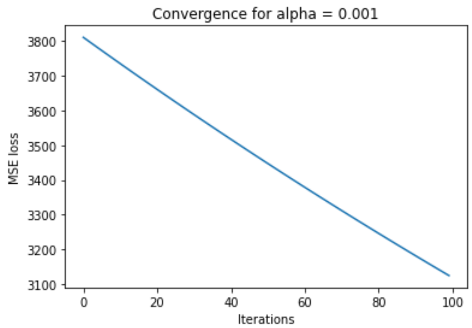
```python
# Perceptron Learning Algorithm (pla.py)

import pandas as pd
import numpy as np
import sys

def perceptron(X, y, output_file):
    num_samples = X.shape[0]
    num_features = X.shape[1]
    output = []

    # Initialize weights and bias to zero
    w = np.zeros((num_features, 1))
    b = 0

    # Iterate until the model converges
    model_not_converged = True
    while model_not_converged:

        # Assume convergence to start
        model_not_converged = False

        # Loop through each sample
        for i in range(num_samples):
            # Get the prediction
            y_pred = b + np.dot(X[[i], :], w)
            y_pred = 1 if y_pred > 0 else -1

            # Check if the true and predicted labels have different signs
            if y[i] * y_pred <= 0:
                # Yes, update weights and bias, and flag model as not converged
                w = w + y[i] * np.transpose(X[[i], :])
                b = b + y[i]
                model_not_converged = True

        # Record weights
        out_line = []
        for w_value in w:
            out_line.append(str(int(w_value[0])))
        out_line.append(str(int(b)))
        output.append(out_line)

    # Write output CSV file
    pd.DataFrame(output).to_csv(output_file, header=False, index=False)

    # Return combined weights vector
    weights = np.zeros((num_features + 1, 1))
    weights[:-1] = w
    weights[-1] = b
    return weights

# Read data
df = pd.read_csv('data1.csv', header=None)
M = df.to_numpy()
X = M[:, [0, 1]]
y = M[:, 2]

# Output file
output_file = str('output1.csv')
weights = perceptron(X, y, output_file)

# Visualize
visualize_scatter(df, weights = weights)
```

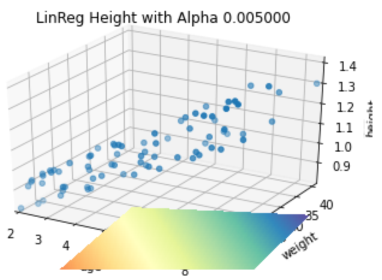


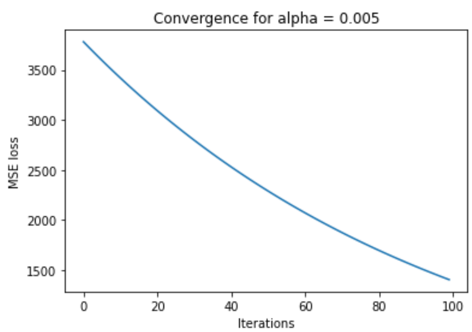
In [ ]:

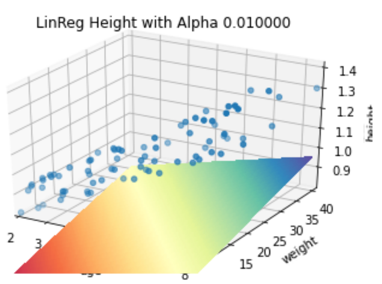
```python
# Linear Regression Model (lr.py)

import pandas as pd
import numpy as np
import sys

def GradientDescent(alpha, X, y, means, stds, df, output, iterations = 100):
    n = len(X)
    beta = np.zeros(3)
    loss_list = np.zeros(iterations)

    # Iterate: update beta values and compute loss
    for i in range(iterations):
        beta = beta - alpha*(1.0/n)*np.transpose(X).dot(X.dot(beta)-y)
        loss_list[i] = (1.0/2*n) * np.sum(np.square(X.dot(beta)-y))

    # De-normalize weights
    weights = np.zeros(3)
    weights[1] = beta[1] / stds[0]
    weights[2] = beta[2] / stds[1]
    weights[0] = beta[0] - (weights[1] * means[0]) - (weights[2] * means[1])

    # Plot model
    xlim = (min(df[0]), max(df[0]))
    ylim = (min(df[1]), max(df[1]))
    zlim = (min(df[2]), max(df[2]))
    visualize_3d(df, lin_reg_weights = weights, alpha = alpha,
                 xlim = xlim, ylim = ylim, zlim = zlim)

    # Plot loss versus iterations
    plt.plot(loss_list)
    plt.title(f'Convergence for alpha = {alpha:.3f}')
    plt.xlabel('Iterations')
    plt.ylabel('MSE loss')
    plt.show()

    # File content
    output.append([alpha, iterations, f"{weights[0]:0.8f}", f"{weights[1]:0.8f}", f"{weights[2]:0.8f}"])

    # Done
    return weights

# Alpha values
alphas = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10]

"""
Custom alpha and iterations
When alpha is too high it can skip the minima for convergence, but if alpha is too low then it will
converge too slowly. We want to find a balance of the alpha and the number of iterations in order to
converge on the minima with the lowest error rates.
"""

my_alpha = 0.6
my_iterations = 30

# Import data and get output file
df = pd.read_csv('data2.csv', header=None)
output_file = str('output2.csv')

# Convert to numpy
data = df.to_numpy()
X = data[:, [0, 1]]
y = data[:, 2]

# Normalize features
# Track the means and std devs to de-normalize later when plotting
means = X.mean(axis=0)
stds = X.std(axis=0)
for i in range(2):
  X[:, i] = (X[:, i] - means[i]) / stds[i]

# Add a column of ones at the start
X = np.hstack([np.ones((X.shape[0], 1)), X])

# Process the pre-set alpha values
output = []
for alpha in alphas:
    GradientDescent(alpha, X, y, means, stds, df, output)

# Process the custom alpha value and iterations count
GradientDescent(my_alpha, X, y, means, stds, df, output, my_iterations)

# Write output CSV file
pd.DataFrame(output).to_csv(output_file, header=False, index=False)
```

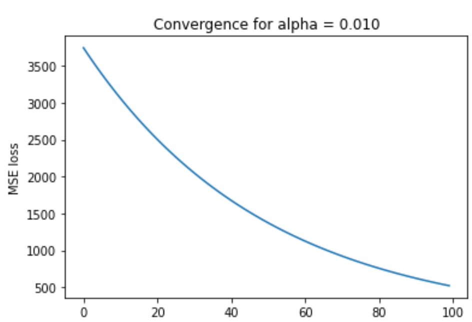

LinReg Height with Alpha 0.001000
age
weight
height

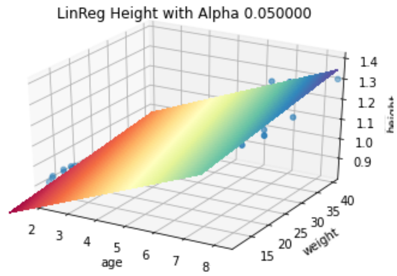

Convergence for alpha = 0.001
MSE loss
Iterations

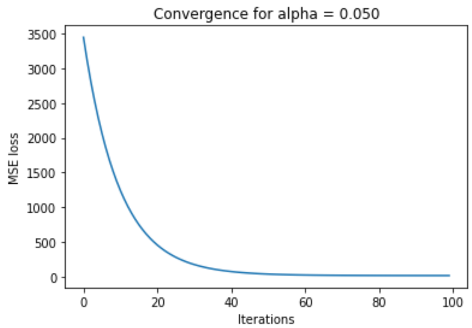

LinReg Height with Alpha 0.005000
weight
height

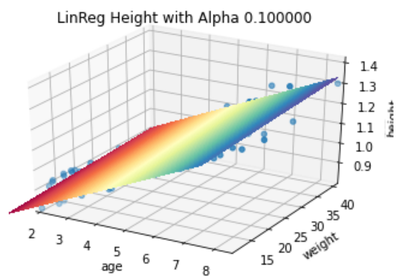

Convergence for alpha = 0.005
MSE loss
Iterations

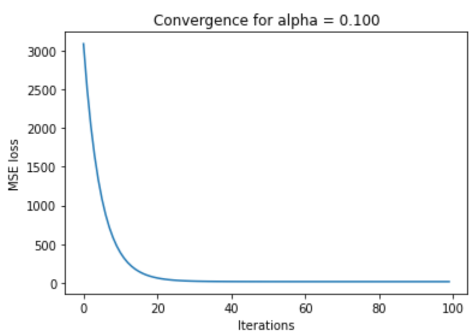

LinReg Height with Alpha 0.010000
weight
height

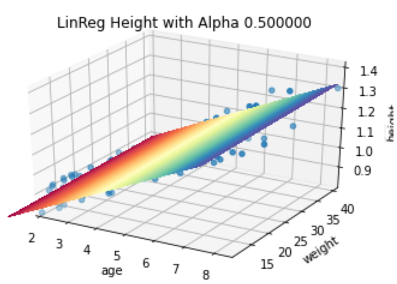

Convergence for alpha = 0.010
MSE loss
Iterations

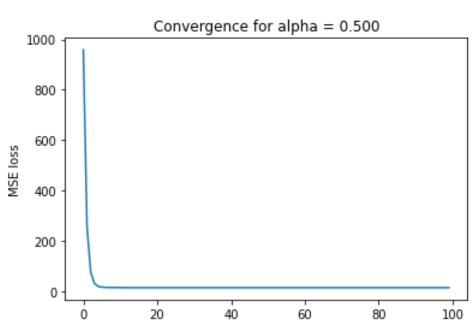

LinReg Height with Alpha 0.050000
age
weight

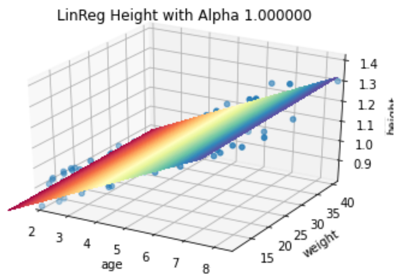

Convergence for alpha = 0.050
MSE loss
Iterations

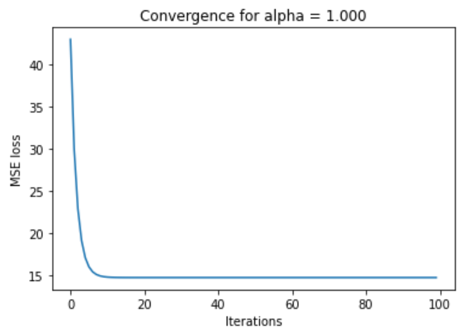

LinReg Height with Alpha 0.100000
age
weight

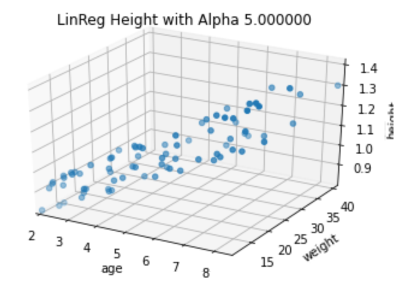

Convergence for alpha = 0.100
MSE loss
Iterations

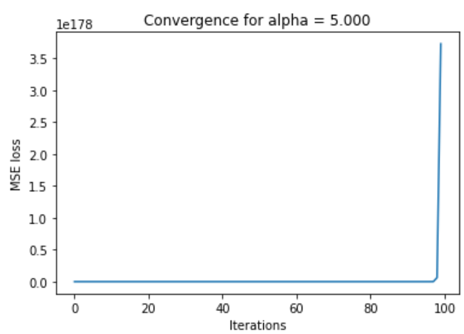

LinReg Height with Alpha 0.500000
age
weight

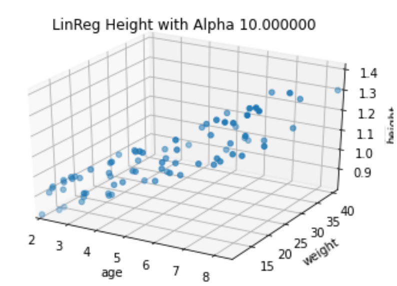

Convergence for alpha = 0.500
MSE loss
Iterations

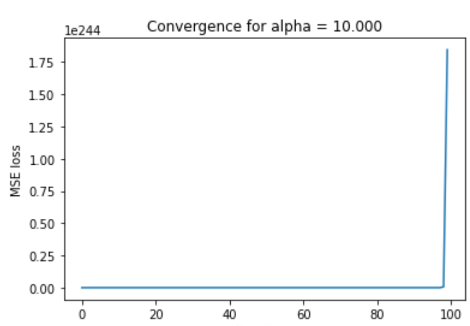

LinReg Height with Alpha 1.000000
age
weight
height

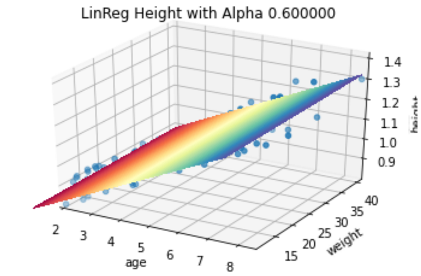

Convergence for alpha = 1.000
MSE loss
Iterations

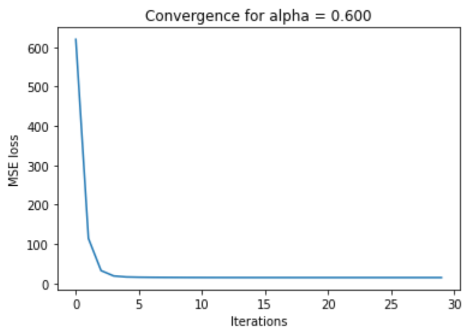

LinReg Height with Alpha 5.000000
age
weight
height


Convergence for alpha = 5.000
1e178
MSE loss
Iterations


LinReg Height with Alpha 10.000000
age
weight
height


Convergence for alpha = 10.000
1e244
MSE loss
Iterations


LinReg Height with Alpha 0.600000
age
weight
height


Convergence for alpha = 0.600
MSE loss
Iterations