Grayson Bass
ID Number: 801074960
Homework 0
Github: https://github.com/gbass2/IntroToML

Problem 1:

The first step in finding the linear regression model for X1, X2, and X3 was to first read the values from the csv file. This was done using pandas and then converted into a NumPy array. X1, X2, and X3 were setup as separate variable corresponding to it's column in the csv file. The ground truth (y) was also read from the csv in the same manner. Next the X1, X2, X3 was reshaped to be a 2-D array where the first column is filled with ones. This allows theta which is a 2x1 to be multiplied by X which is a mx2 matrix where m is the number of training samples. Next the cost is calculated. Theta was initialized as an array of 2 zeros. Which corresponds to theta0 and theta1. The initial cost came out to be 5.244 for X1, X2, and X3. This is the case because the starting theta values were 0. Once the cost is calculated the gradient descent is calculated. The gradient descent function returns theta for the regression model, the history of the cost for each iteration. The array holding the history of theta is used to plot the regression model over the number of iterations. The source code, graphs, and results can be found at the bottom of the report.

Part 1:

The following is the linear regression models with their respective learning rates and iteration numbers.

For X1:
alpha = .02
iterations = 600
h(X1) = 5.52168752 - 1.88021764 * (X1)

For X2:
alpha = .01
iterations = 50
h(X1) = 0.32061638 + 0.68376091 * (X2)

For X3:
alpha = .01
iterations = 50
h(X1) = 2.69330966 - 0.45004043 * (X3)

Part 2:

Figure 4, 5, and 6 shows the plot of the regression model over the number of iterations.

Figures 7, 8, and 9 shows the cost vs the iterations for X1, X2, X3

Part 3:

The explanatory variable with the lowest cost that explains y is X1. The values in X1 best explained the output y and that is why it showed the lowest loss.

Part 4:

Both the learning rate and the iterations play a role in reducing the final loss of a linear regression model. The higher the learning rate the faster the model can learn but a smaller learning rate can sometimes be better because the model will learn better. The number of iterations goes hand in hand with the learning rate. There is a point at which the model will not learn anymore on the same data. This can be seen in figures 7, 8, and 9 which shows a plot of their gradient descents. You want to have more iterations if you choose a small learning rate, and you want a smaller number of iterations if chose a higher learning rate. For X2 in problem 2 a smaller learning rate and number of iterations were used. This was since the loss converged quicker than X1 and X2 when calculating the gradient descent. The following learning rates and iterations is what I found best for X1, X2, and X3:

For X1:
alpha = .02
iterations = 600

For X2:
alpha = .01
iterations = 50

For X3:
alpha = .01
iterations = 600

Problem 2:

Problem 2 was completed in a similar manner to problem 1 with a couple of changes. The first change was combining X1, X2, and X3 into one array. This allows a single model to learn on multiple variables which can lower the loss of the model. The array containing all 1's was also added to the first column of X. Once X was created, theta was initialized to an array of 4 zeros corresponding to theta0 through theta3. 2 more thetas were added compared to problem 1 because the 2 more input variables were added. The cost and gradient descent were calculated using the same functions as in problem 1.

Part 1:

The following is the best linear regression model that I found with it's respective learning rate and number of iterations.

For X:
alpha = .02

iterations = 600

h(X) = 4.84189465 - 1.93700296 * (X1) + 0.61060345 * (X2) - 0.19637631 * (X3)

Part 2:

The plot of the loss can be seen in figure 10.

Part 3:

Both the learning rate and the iterations play a role in reducing the final loss of a linear regression model. The higher the learning rate the faster the model can learn but a smaller learning rate can sometimes be better because the model will learn better. The number of iterations goes hand in hand with the learning rate. There is a point at which the model will not learn anymore on the same data. This can be seen in figures 10 which shows a plot of the gradient descent for X. You want to have more iterations if you choose a small learning rate, and you want a smaller number of iterations if chose a higher learning rate. For X I chose a smaller learning rate and a larger number of iterations. This allowed me to get a lower cost function. I only ran the gradient descent over 600 iterations with an alpha of .02. Adding more iterations did not lower the cost enough to justify having more iterations. The cost only decreased by .2.

Part 4:

The predicted values were calculated using the following function:

h = theta[0] + theta[1]*X[0] + theta[2]*X[1] + theta[3]*X[2]

h is the predicted value and X is the explanatory variable to be predicted on. The following are the predicted values.

For X = (1, 1, 1)
Y = 3.525163184517707

For X = (2, 0, 4)
Y = 0.2317926141038631

For X = (3, 2, 1)
Y = 0.09306401330393993

assignment1

September 17, 2021

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
df = pd.read_csv("D3.csv", header=None)

# Getting each input variable from the columns
X1 = np.array(df.values[:,0])
X2 = np.array(df.values[:,1])
X3 = np.array(df.values[:,2])

# Getting the output variable from the last column
y = np.array(df.values[:,3])

m = len(y) # Training samples
```

[3]:
```
def calculate_cost(X,y,theta):
    """Computes the cost function for linear regression"""

    h = X.dot(theta)
    errors = np.subtract(h,y)
    sqrErrors = np.square(errors)
    J = 1/(2*m) * np.sum(sqrErrors)
    return J

# The gradient descent works with any number of input variables
def gradient_descent(X,y,theta,alpha,iterations):
    """Computes the gradient descent for linear regression"""

    cost_history = np.zeros(iterations,)
    theta_history = np.zeros([iterations, theta.size])

    for i in range(iterations):
        h = X.dot(theta)
        errors = np.subtract(h,y)
        sum_delta = (alpha/m) * X.transpose().dot(errors);
        theta = theta - sum_delta;
        cost_history[i] = calculate_cost(X,y,theta)
        theta_history[i] = theta

    return cost_history, theta, theta_history
```

[4]:
```
#Problem 1
# Plotting the inputs vs output
plt.figure()
plt.rcParams['figure.figsize'] = [12, 7]
plt.scatter(X1, y,  color='blue')
plt.xlabel("X1")
plt.ylabel("y")
plt.title("Figure 1: y vs X1")
plt.grid()

plt.figure()
plt.scatter(X2, y,  color='green')
plt.xlabel("X2")
plt.ylabel("y")
plt.title("Figure 2: y vs X2")
plt.grid()

plt.figure()
plt.scatter(X3, y,  color='orange')
plt.xlabel("X3")
plt.ylabel("y")
plt.title("Figure 3: y vs X3")
plt.grid()
```

Figure 1: y vs X1

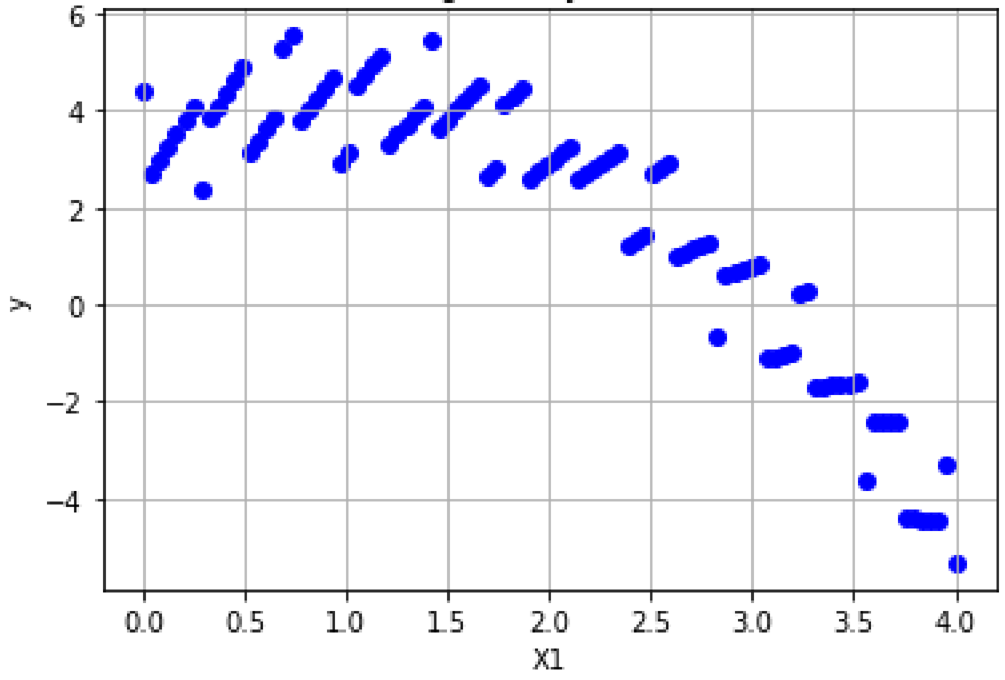
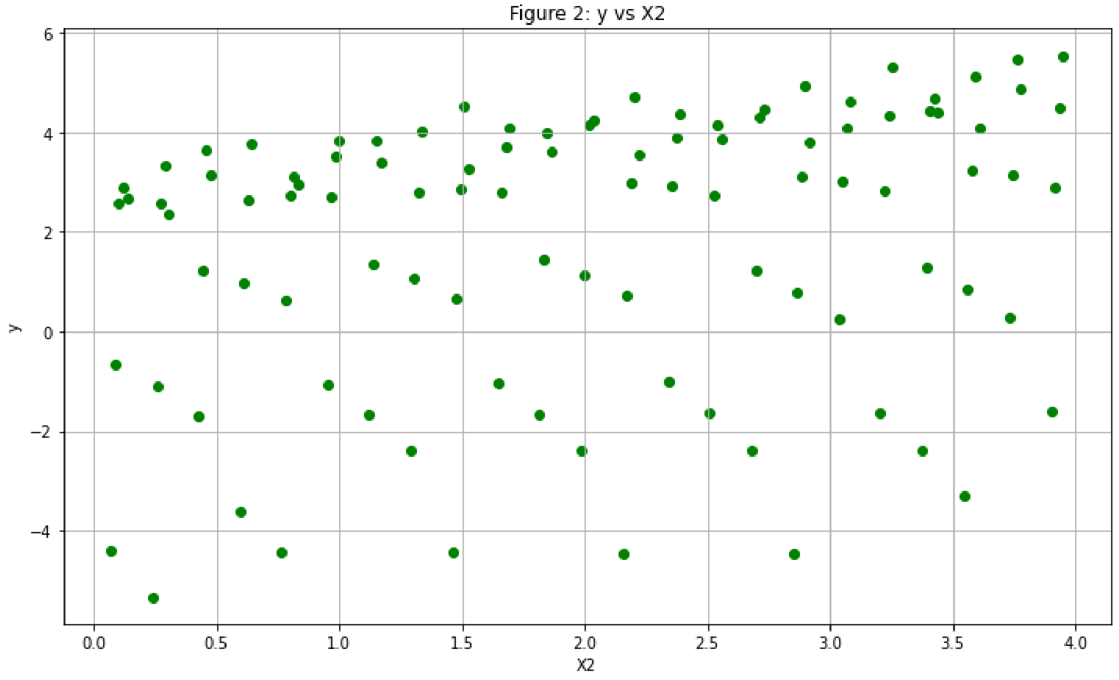
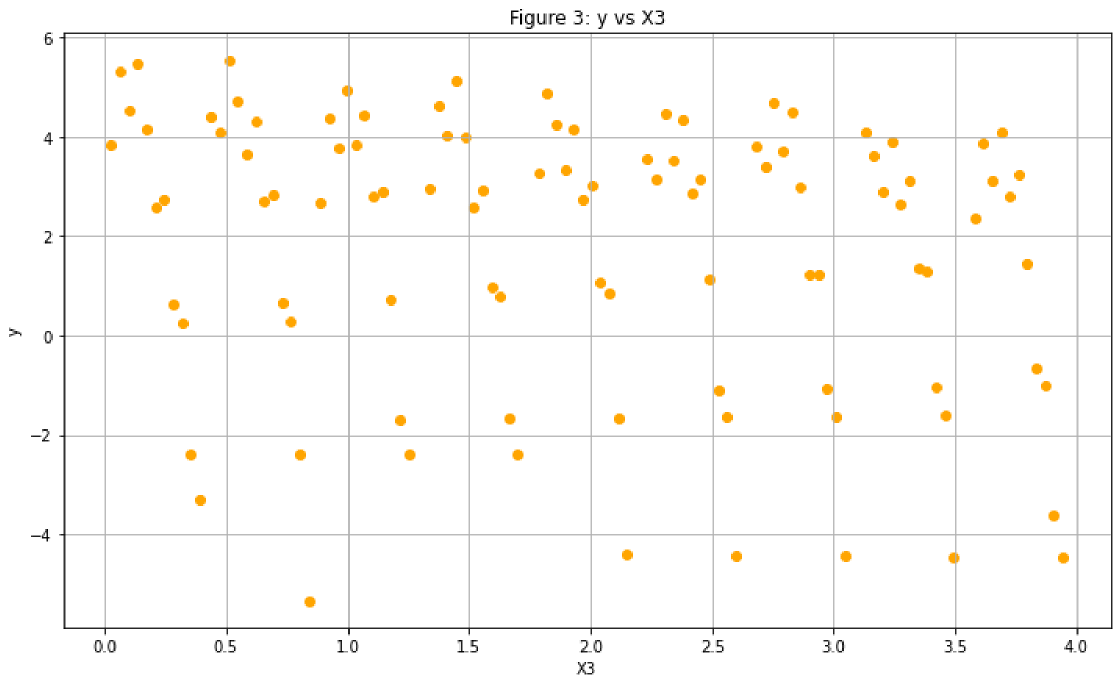


Figure 2: y vs X2





[5]:
```
# Calculating linear regression model for X1, X2, X3 seperately
# Matrix with a single column of ones
X0 = np.ones((m,1))

# Reshaping X1, X2, and X3 to be a 2-D Array
X1=X1.reshape(m,1)
X2=X2.reshape(m,1)
X3=X3.reshape(m,1)

# Combining X0 with X1, X2, and X3 seperately to stack them horizontally
X1=np.hstack((X0,X1))
X2=np.hstack((X0,X2))
X3=np.hstack((X0,X3))
```

[6]:
```
theta = np.zeros(2) # Theta_0 and theta_1

# Computing the inital cost for X1, X2, and X3
cost1 = calculate_cost(X1,y,theta)
print("The cost for X1: ", cost1)
cost2 = calculate_cost(X2,y,theta)
print("The cost for X2: ", cost2)
cost3 = calculate_cost(X3,y,theta)
print("The cost for X3: ", cost3)
```

```
The cost for X1:  5.524438459185473
The cost for X2:  5.524438459185473
The cost for X3:  5.524438459185473
```

[7]:
```
# Computing the gradient descent for X1, X2, and X3 seperately
alpha = .02
iterations = 600

cost1_history, theta1, theta1_history =␣
 ↪gradient_descent(X1,y,theta,alpha,iterations)
print("Theta values for X1: ", theta1)

alpha = .01
iterations = 50
cost2_history, theta2, theta2_history =␣
 ↪gradient_descent(X2,y,theta,alpha,iterations)
print("Theta values for X2: ", theta2)

alpha = .02
iterations = 600
cost3_history, theta3, theta3_history =␣
 ↪gradient_descent(X3,y,theta,alpha,iterations)
print("Theta values for X3: ", theta3)
```

```
Theta values for X1:  [ 5.52168752 -1.88021764]
Theta values for X2:  [0.32061638 0.68376091]
Theta values for X3:  [ 2.69330966 -0.45004043]
```

[8]:
```
plt.rcParams['figure.figsize'] = [14, 9]

# Plotting the model for X1 in each iterion
plt.figure()
for t in theta1_history:
    plt.plot(X1[:,1],X1.dot(t))

plt.xlabel("X1")
plt.ylabel("h(x)")
plt.title("Figure 4: Regression Model for X1")
plt.grid()

# Plotting the model for X2 in each iterion
plt.figure()

for t in theta2_history:
    plt.plot(X2[:,1],X2.dot(t))

plt.xlabel("X2")
plt.ylabel("h(x)")
plt.title("Figure 5: Regression Model for X2")
plt.grid()

# Plotting the model for X3 in each iterion
plt.figure()

for t in theta3_history:
    plt.plot(X3[:,1],X3.dot(t))

plt.xlabel("X3")
plt.ylabel("h(x)")
plt.title("Firgure 6: Regression Model for X3")
plt.grid()


# Plotting the cost historty vs the number of iterations
plt.figure()
plt.plot(cost1_history[0:len(cost1_history)], color='blue', linewidth=1)
plt.xlabel("Iterations")
plt.ylabel("J1")
plt.title("Figure 7: J1 vs. Iterations")
plt.grid()

plt.figure()
plt.plot(cost2_history[0:len(cost2_history)], color='blue', linewidth=1)
plt.xlabel("Iterations")
plt.ylabel("J2")
plt.title("Figure 8: J2 vs. Iterations")
plt.grid()

plt.figure()
plt.plot(cost3_history[0:len(cost3_history)], color='blue', linewidth=1)
plt.xlabel("Iterations")
plt.ylabel("J3")
plt.title("Figure 9: J3 vs. Iterations")
plt.grid()
```

Figure 4: Regression Model for X1

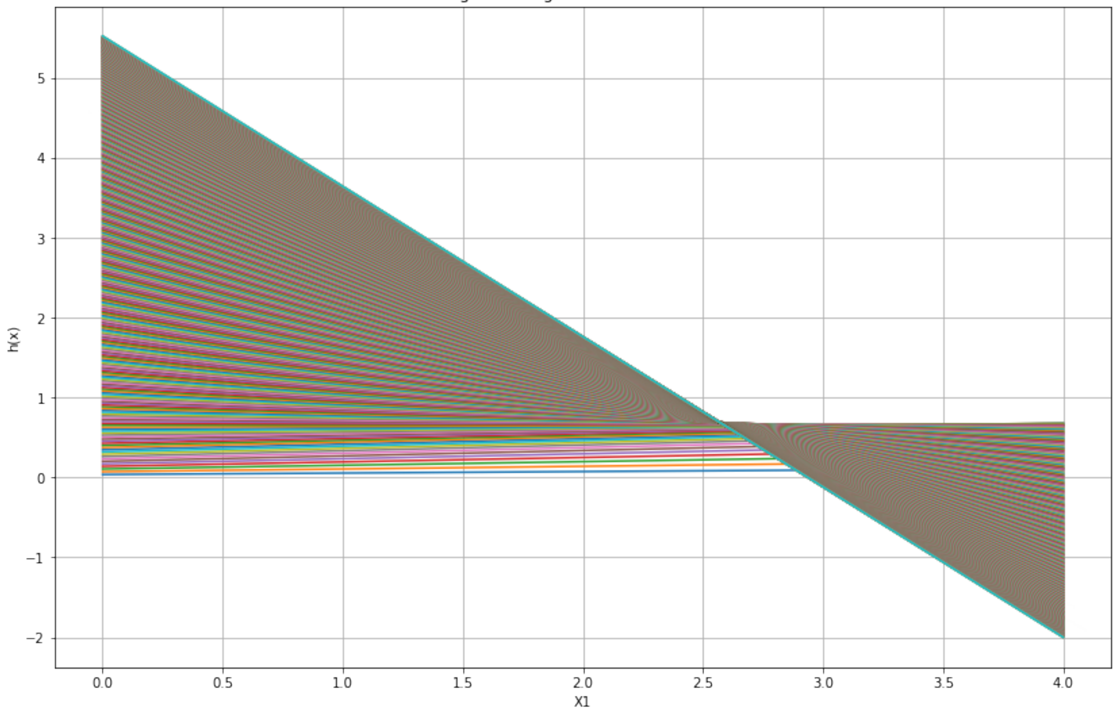
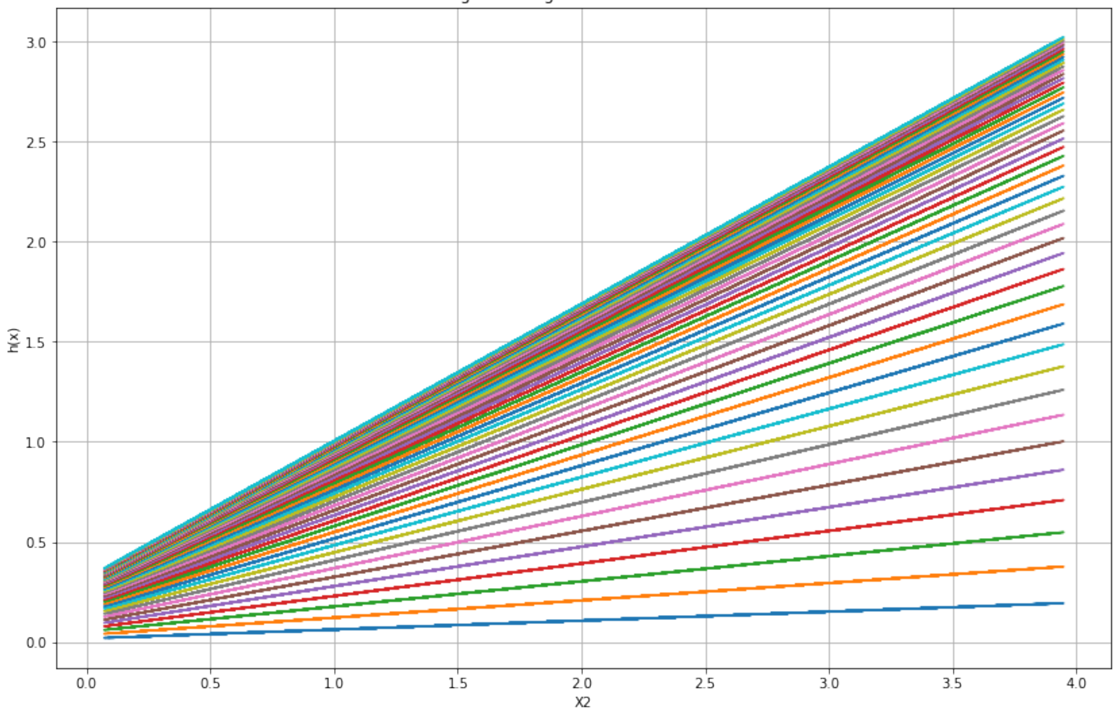


Figure 5: Regression Model for X2



Firgure 6: Regression Model for X3

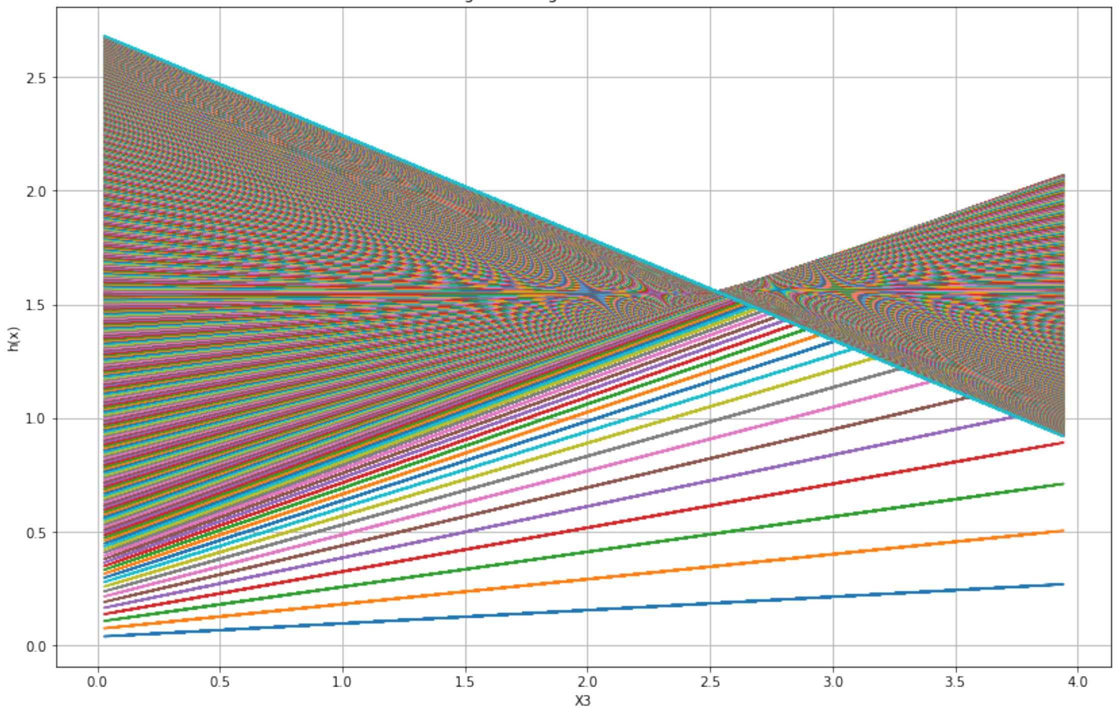


Figure 7: J1 vs. Iterations

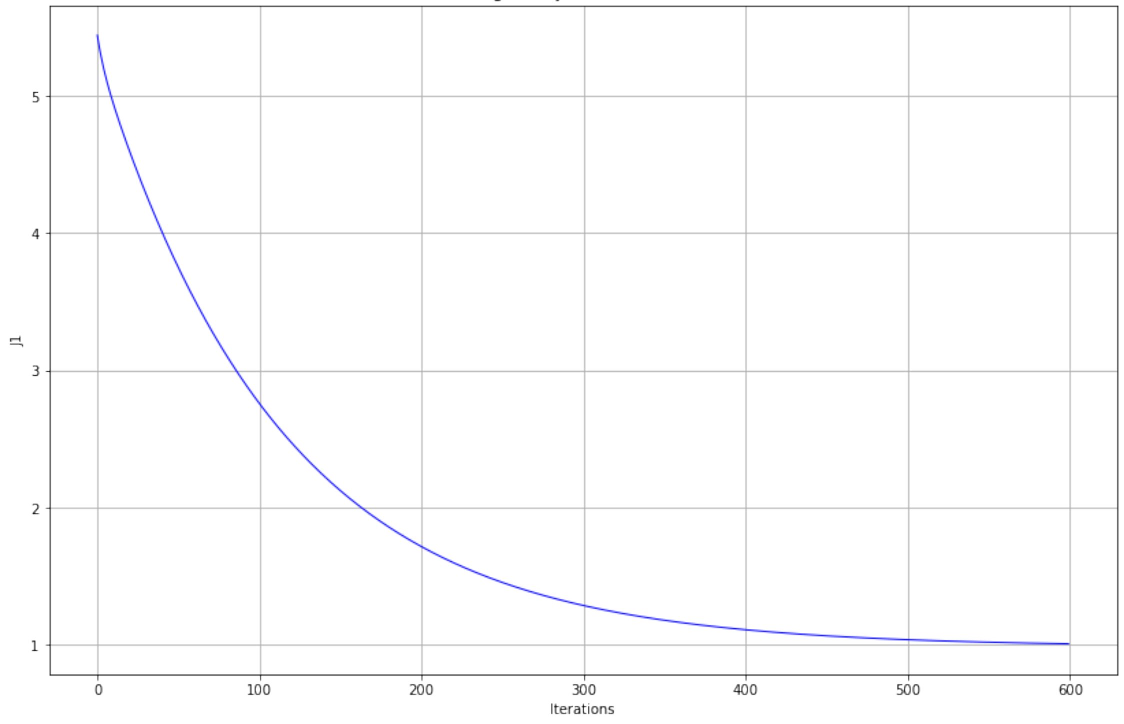


Figure 8: J2 vs. Iterations

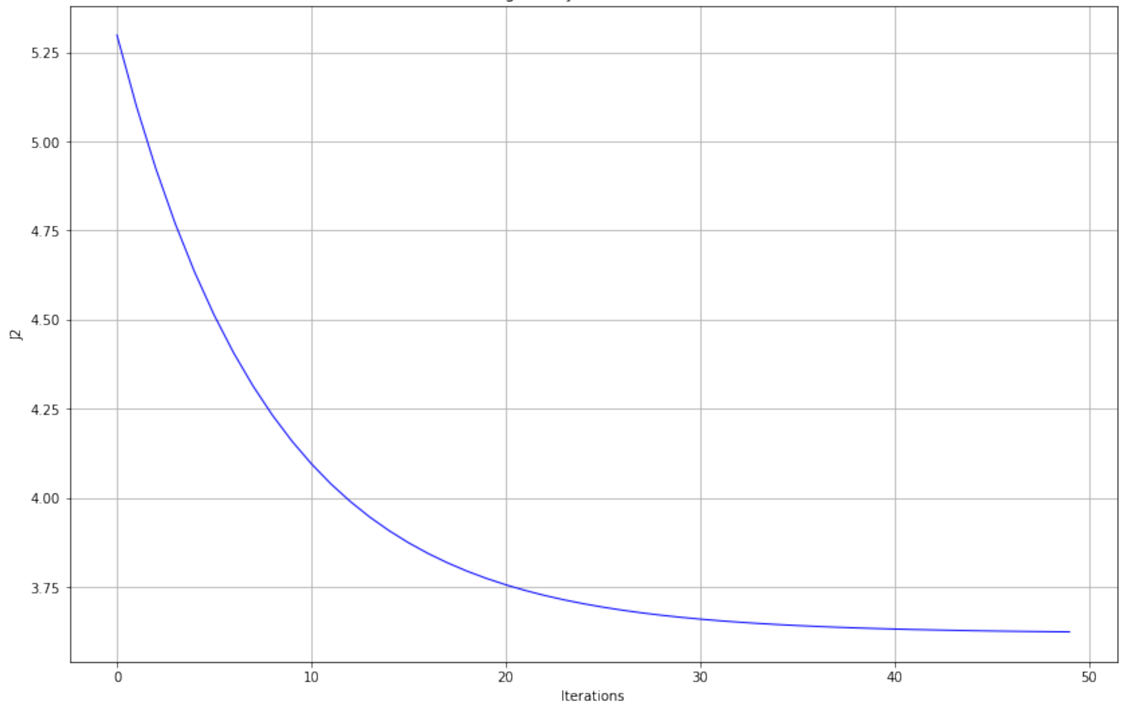
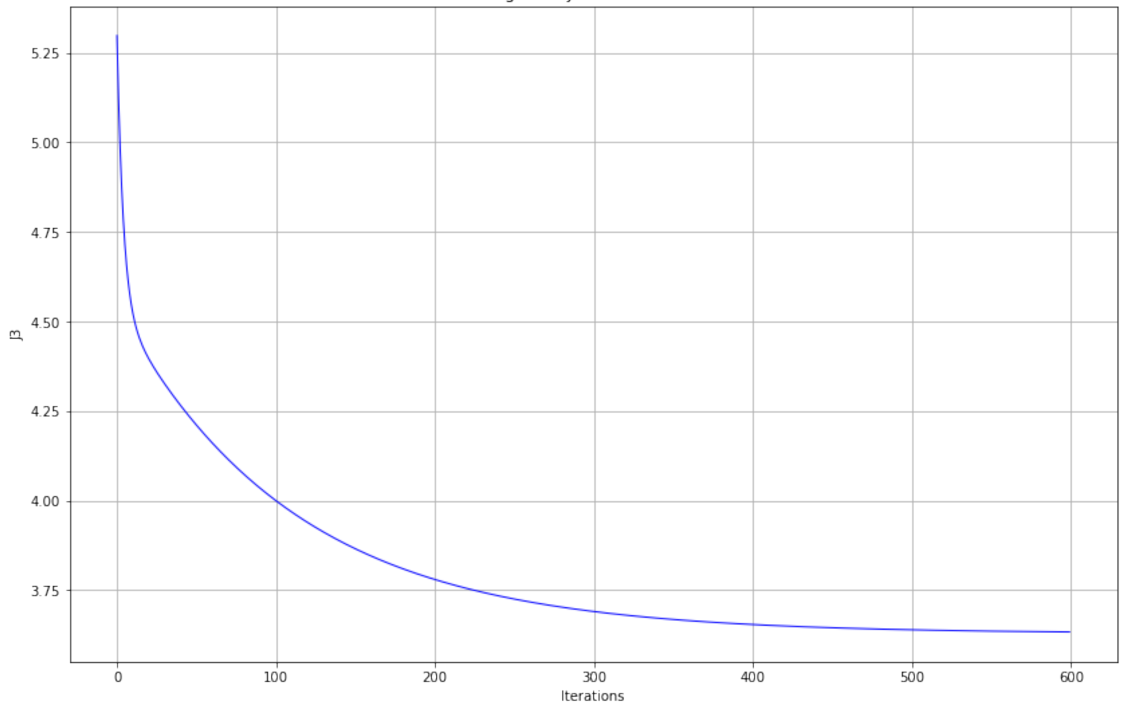


Figure 9: J3 vs. Iterations



[9]:
```
# Problem 2

df = pd.read_csv("D3.csv", header=None)

# Getting each input variable from the columns
X1 = np.array(df.values[:,0])
X2 = np.array(df.values[:,1])
X3 = np.array(df.values[:,2])

# Getting the output variable from the last column
y = np.array(df.values[:,3])

m = len(y) # Training samples

# Calculating linear regression model for X1, X2, X3 seperately
# Matrix with a single column of ones
X0 = np.ones((m,1))

# Reshaping X1, X2, and X3 to be a 2-D Array
X1=X1.reshape(m,1)
X2=X2.reshape(m,1)
X3=X3.reshape(m,1)

# Combining X0 with X1, X2, and X3 to stack them horizontally
X = np.hstack((X0,X1,X2,X3))

# Calculating the intial cost
theta = np.zeros(4) # Theta_0 and theta_1
cost = calculate_cost(X,y,theta)
print("Cost for X: ", cost)
```

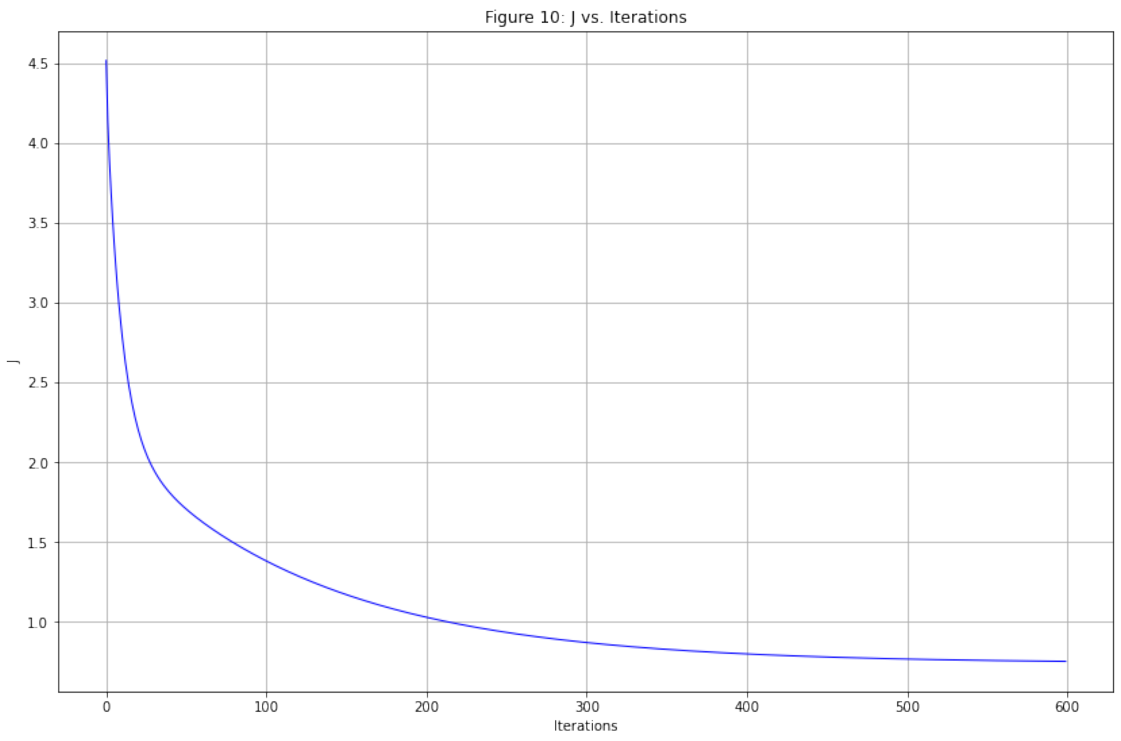
```
Cost for X:  5.524438459185473
```

[10]:
```
# Calculating the gradient descent
alpha = .04
iterations = 600

cost_history, theta, theta_history =␣
 ↪gradient_descent(X,y,theta,alpha,iterations)
print("Theta values for X: ", theta)
```

```
Theta values for X:  [ 4.84189465 -1.93700296  0.61060345 -0.19637631]
```

[11]:
```
# Plotting the Cost vs Iterations
plt.rcParams['figure.figsize'] = [14, 9]
plt.figure()
plt.rcParams['figure.figsize'] = [11, 5]
plt.plot(cost_history[0:len(cost_history)], color='blue', linewidth=1)
plt.xlabel("Iterations")
plt.ylabel("J")
plt.title("Figure 10: J vs. Iterations")
plt.grid()
```



[12]:
```
def predict(X, theta):
    return theta[0] + theta[1]*X[0] + theta[2]*X[1] + theta[3]*X[2]
```

[13]:
```
# Predicting for the new values

# new_X = (1,1,1)
new_X = np.array([1,1,1])
y_pred = predict(new_X, theta)
print(y_pred)

# new_X = (2,0,4)
new_X = np.array([2,0,4])
y_pred = predict(new_X, theta)

print(y_pred)

# new_X = (3,2,1)
new_X = np.array([3,2,1])
```

```python
y_pred = predict(new_X, theta)

print(y_pred)
```

```
3.3191188264354814
0.18238348312061892
0.05571635303650194
```