ECGR_4105_Assignment_0_Source_Code

June 18, 2025

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

[3]:
```
file_path = '/content/univariate_profits_and_populations_from_the_cities.csv'
df = pd.DataFrame(pd.read_csv(file_path))

# df = pd.read_csv('../Datasets/New folder/
 ↪univariate_profits_and_populations_from_the_cities.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)
M
```

[3]: 97

[4]:
```
df.describe()
```

[4]:

	population	profit
count	97.000000	97.000000
mean	8.159800	5.839135
std	3.869884	5.510262
min	5.026900	-2.680700
25%	5.707700	1.986900
50%	6.589400	4.562300
75%	8.578100	7.046700
max	22.203000	24.147000

[7]:
```
# Separate features and labels
X = df.values[:, 0]  # get input values from first column -- X is a list here
y = df.values[:, 1]  # get output values from second column -- Y is the list␣
 ↪here
m = len(y)  # Number of training examples
n = len(X)  # Number of training examples

# Display first 5 records and the total number of training examples
```

```python
print('X = ', X[: 5])
print('y = ', y[: 5])
print('m = ', m)
print('n = ', n)
```

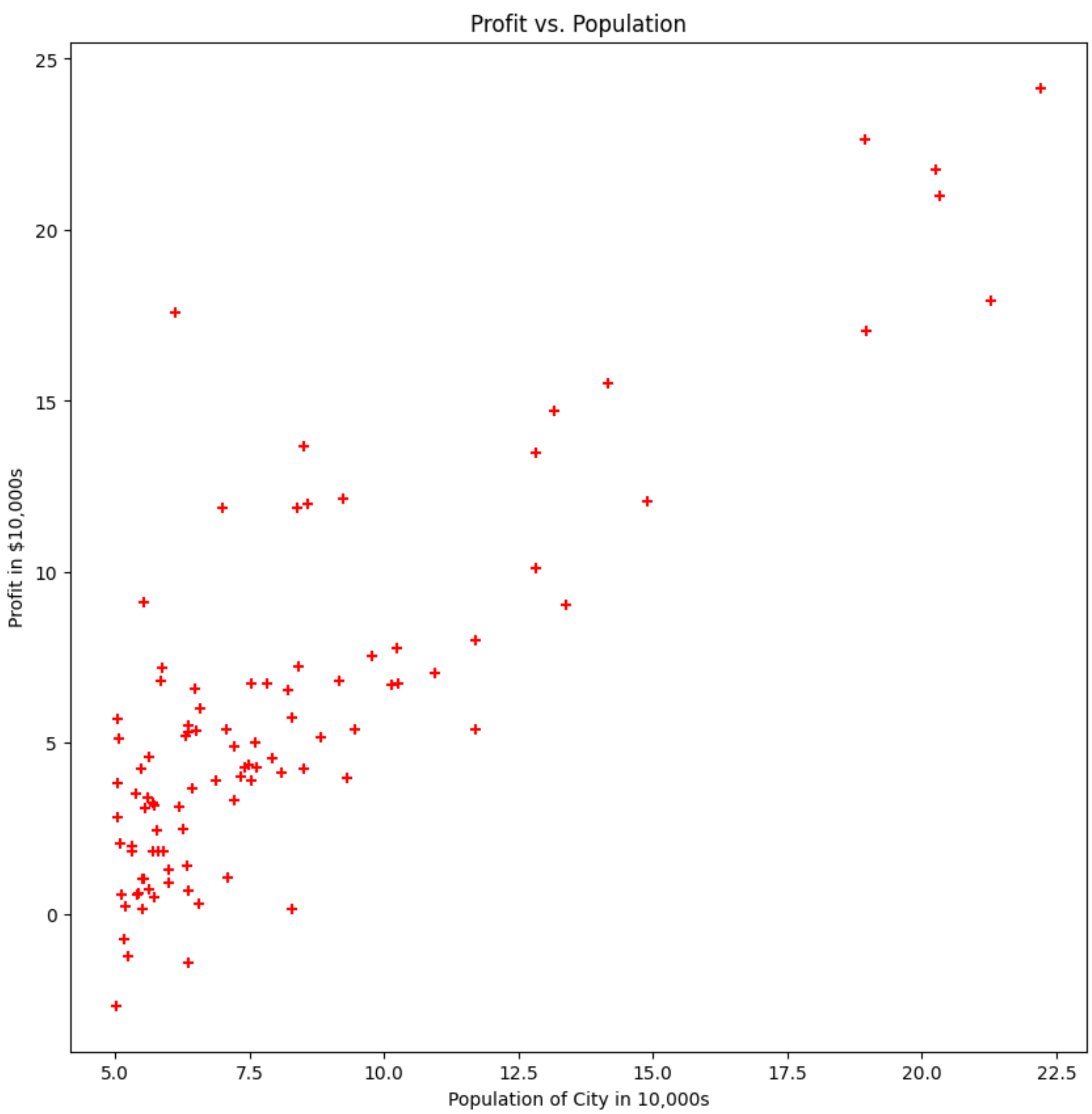
```
X =  [6.1101 5.5277 8.5186 7.0032 5.8598]
y =  [17.592   9.1302 13.662  11.854   6.8233]
m =  97
n =  97
```

[9]:

```python
# Scatter plot
plt.scatter(X, y, color='red', marker='+')

# Grid, labels, and title
# plt.grid(True)
plt.rcParams["figure.figsize"] = (10, 10)
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
plt.title('Profit vs. Population')

# Show the plot
plt.show()
```

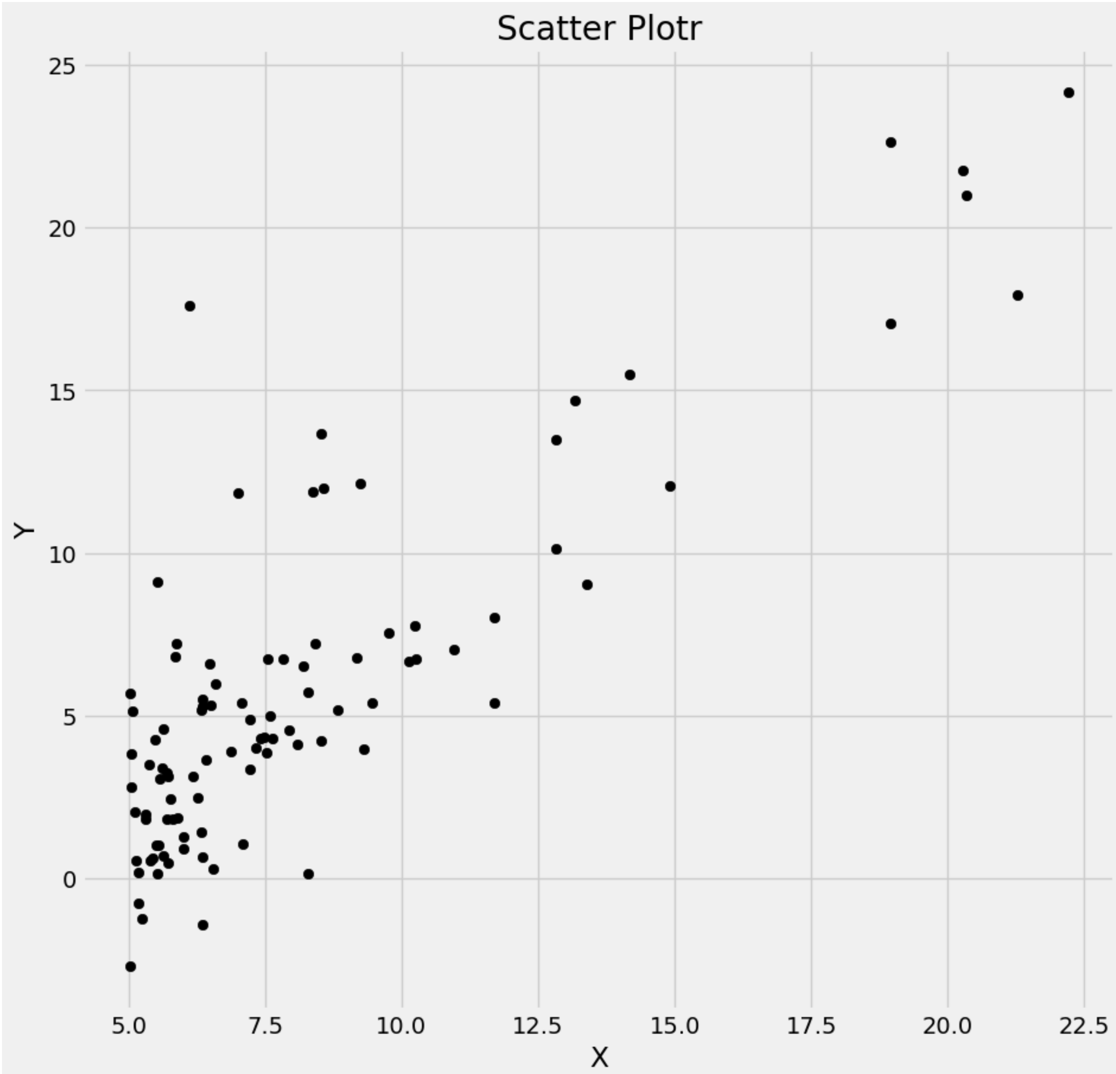


[10]:
```
# Another way to plot the data

import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

plt.scatter(X, y, color='black')
plt.xlabel('X')
plt.ylabel('Y')
plt.gca().set_title("Scatter Plotr")
```

[10]:
```
Text(0.5, 1.0, 'Scatter Plotr')
```



[12]:
```
#We walk through the initial steps of building a linear regression model from␣
↪scratch using NumPy. Let's break down what you're doing:
#X_0 = np.ones((m, 1)): We're creating a column vector of ones. This will be␣
↪used as the "bias" term for the linear regression model.
#X_1 = X.reshape(m, 1): You're reshaping features (X) to make it a 2D array␣
↪suitable for matrix operations.
#X = np.hstack((X_0, X_1)): We're horizontally stacking X_0 and X_1 to create␣
↪final feature matrix X.
```

[13]:
```
m = len(y)  # Number of training examples
n = len(X)  # Number of training examples
```

[14]:
```python
X_0 = np.ones((m, 1)) #We're creating a column vector of ones. This will be␣
↪used as the "bias" term for the linear regression model.
X_0[:5]
```

[14]:
```
array([[1.],
       [1.],
       [1.],
       [1.],
       [1.]])
```

[15]:
```python
X_1 = X.reshape(m, 1) # You're reshaping features (X) to make it a 2D array␣
↪suitable for matrix operations.
X_1[:10]
```

[15]:
```
array([[6.1101],
       [5.5277],
       [8.5186],
       [7.0032],
       [5.8598],
       [8.3829],
       [7.4764],
       [8.5781],
       [6.4862],
       [5.0546]])
```

[17]:
```python
# Lets use hstack() function from numpy to stack X_0 and X_1 horizontally (i.e.␣
↪column
# This will be our final X matrix (feature matrix)
X = np.hstack((X_0, X_1)) # We're horizontally stacking X_0 and X_1 to create␣
↪final feature matrix X.
X[:5]
```

[17]:
```
array([[1.    , 6.1101],
       [1.    , 5.5277],
       [1.    , 8.5186],
       [1.    , 7.0032],
       [1.    , 5.8598]])
```

[18]:
```python
theta = np.zeros(2)
theta
```

[18]:
```
array([0., 0.])
```

[19]:
```python
def compute_cost(X, y, theta):
    """
    Compute cost for linear regression.
    Input Parameters
```

```
    ----------------
    X : 2D array where each row represent the training example and each column␣
  ↪represent
    m= number of training examples
    n= number of features (including X_0 column of ones)
    y : 1D array of labels/target value for each traing example. dimension(1 x m)
    theta : 1D array of fitting parameters or weights. Dimension (1 x n)
    Output Parameters
    -----------------
    J : Scalar value.
    """
    predictions = X.dot(theta)
    errors = np.subtract(predictions, y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)

    return J

def gradient_descent(X, y, theta, alpha, iterations):
    """
    Compute cost for linear regression.
    Input Parameters
    ----------------
    X : 2D array where each row represent the training example and each column␣
  ↪represent
    m= number of training examples
    n= number of features (including X_0 column of ones)
    y : 1D array of labels/target value for each traing example. dimension(m x 1)
    theta : 1D array of fitting parameters or weights. Dimension (1 x n)
    alpha : Learning rate. Scalar value
    iterations: No of iterations. Scalar value.
    Output Parameters
    -----------------
    theta : Final Value. 1D array of fitting parameters or weights. Dimension (1␣
  ↪x n)
    cost_history: Conatins value of cost for each iteration. 1D array.␣
  ↪Dimansion(m x 1)
    """
    cost_history = np.zeros(iterations)

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

```
    return theta, cost_history
```

[20]:
```
# Lets compute the cost for theta values
cost = compute_cost(X, y, theta)
print('The cost for given values of theta_0 and theta_1 =', cost)
```

```
The cost for given values of theta_0 and theta_1 = 32.072733877455676
```

[21]:
```
theta = [0., 0.]
iterations = 150;
alpha = 0.0001;

theta, cost_history = gradient_descent(X, y, theta, alpha, iterations)
print('Final value of theta =', theta)
print('cost_history =', cost_history)
```

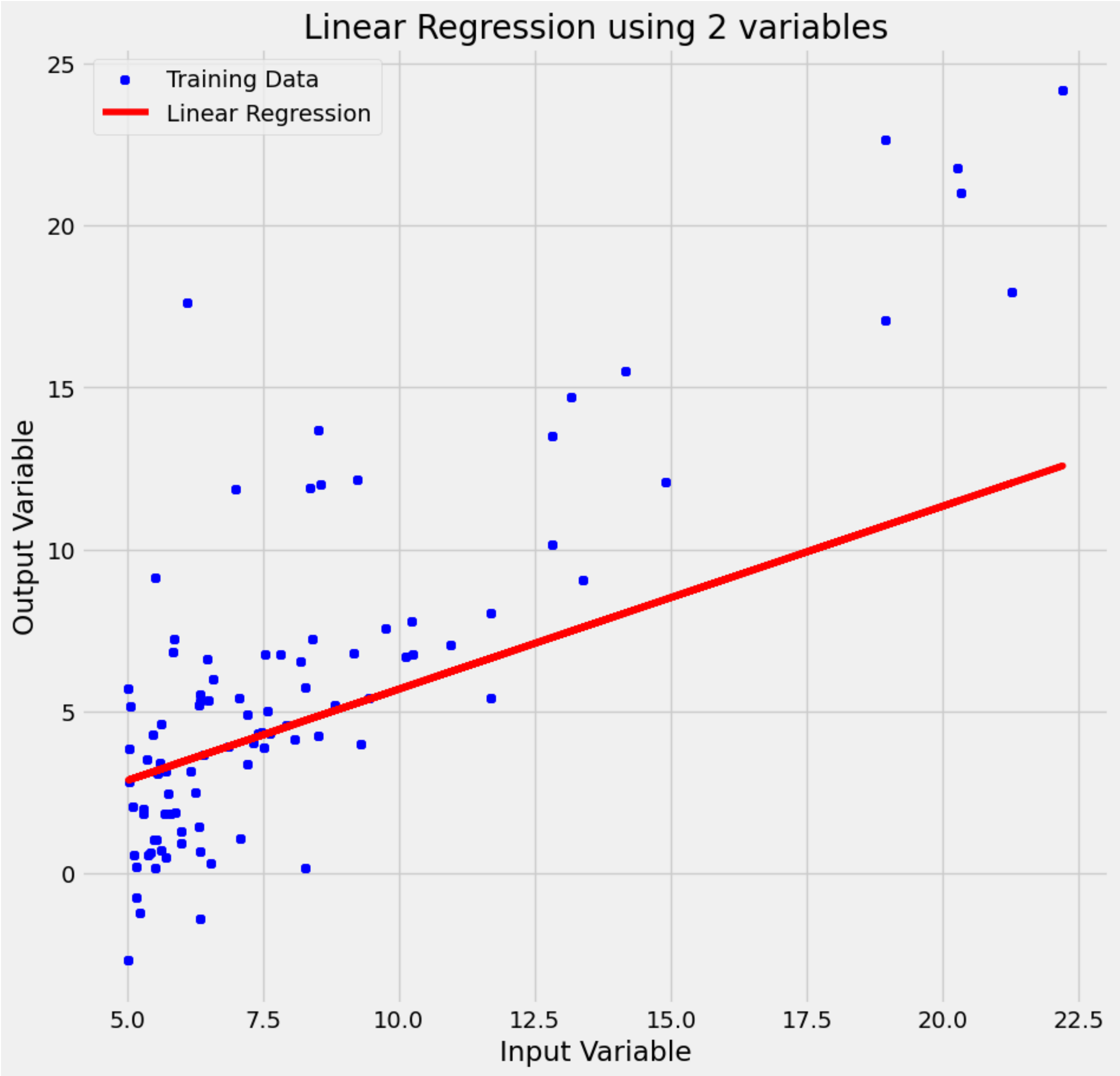
```
Final value of theta = [0.04588732 0.56470353]
cost_history = [31.64430687 31.22289541 30.80838462 30.40066148 29.99961482
29.60513531
 29.2171154  28.83544929 28.46003292 28.09076395 27.72754171 27.37026715
 27.01884287 26.67317307 26.3331635  25.99872146 25.66975578 25.34617676
 25.02789619 24.7148273  24.40688473 24.10398453 23.80604411 23.51298226
 23.22471907 22.94117596 22.66227563 22.38794203 22.11810039 21.85267713
 21.59159989 21.3347975  21.08219995 20.83373838 20.58934505 20.34895333
 20.1124977  19.87991368 19.65113787 19.42610791 19.20476244 18.98704113
 18.77288463 18.56223455 18.35503346 18.15122489 17.95075327 17.75356396
 17.55960319 17.3688181  17.18115667 16.99656775 16.81500103 16.636407
 16.46073698 16.28794308 16.11797821 15.95079603 15.78635096 15.62459819
 15.46549362 15.30899388 15.15505631 15.00363895 14.85470052 14.70820044
 14.56409876 14.4223562  14.28293414 14.14579456 14.01090009 13.87821396
 13.7477     13.61932264 13.49304689 13.36883832 13.24666309 13.1264879
 13.00827998 12.89200712 12.77763764 12.66514035 12.5544846  12.44564023
 12.33857758 12.23326745 12.12968116 12.02779047 11.92756761 11.82898527
 11.73201658 11.63663512 11.54281488 11.45053031 11.35975625 11.27046796
 11.18264111 11.09625177 11.01127639 10.92769182 10.84547529 10.76460437
 10.68505705 10.60681164 10.52984682 10.45414162 10.37967541 10.30642789
 10.23437912 10.16350945 10.09379958 10.02523052  9.95778357  9.89144037
  9.82618284  9.76199319  9.69885394  9.63674788  9.5756581   9.51556795
  9.45646106  9.39832133  9.34113292  9.28488024  9.22954798  9.17512106
  9.12158465  9.06892417  9.01712527  8.96617385  8.91605602  8.86675813
  8.81826676  8.77056869  8.72365094  8.67750072  8.63210547  8.58745281
  8.5435306   8.50032686  8.45782983  8.41602794  8.3749098   8.33446421
  8.29468016  8.25554682  8.21705353  8.1791898   8.14194533  8.10530997]
```

[22]:
```
# Since X is list of list (feature matrix) lets take values of column of index␣
↪1 only
plt.scatter(X[:,1], y, color='b', marker= '+', label= 'Training Data')
```

```python
plt.plot(X[:,1],X.dot(theta), color='r', label='Linear Regression')
plt.rcParams["figure.figsize"] = (10,6)

plt.xlabel('Input Variable')
plt.ylabel('Output Variable')
plt.title('Linear Regression using 2 variables')
plt.legend()
```

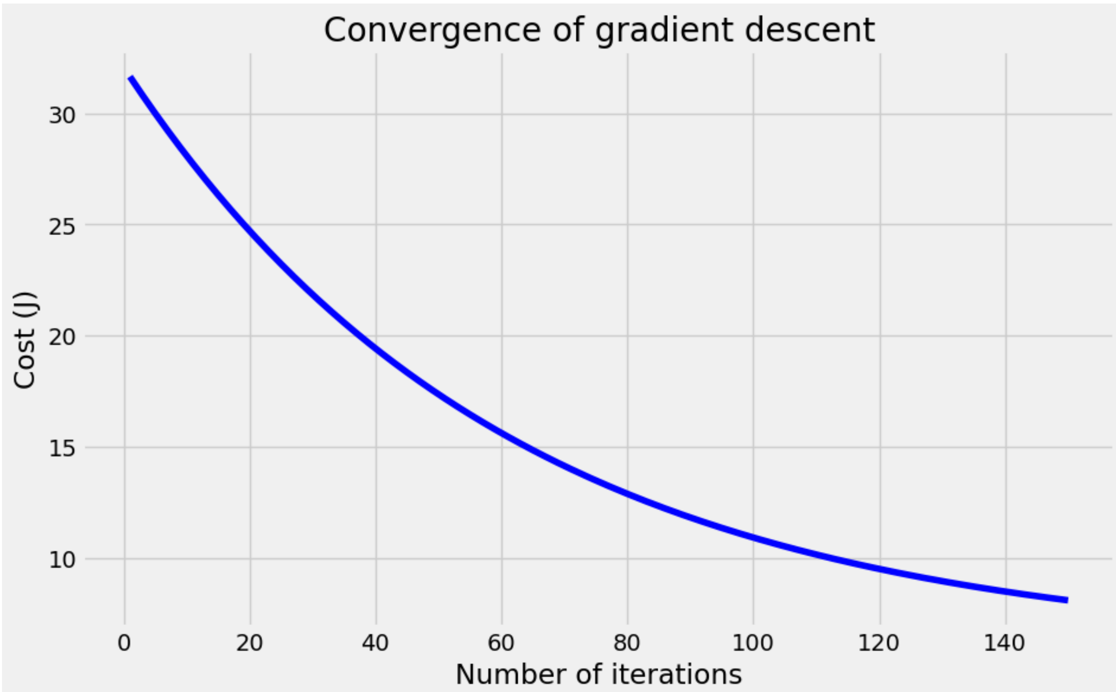
[22]: <matplotlib.legend.Legend at 0x7d2176b23710>



[23]:
```python
plt.plot(range(1, iterations + 1),cost_history, color='blue')
plt.rcParams["figure.figsize"] = (10,6)
# plt.grid()
plt.xlabel('Number of iterations')
```

```
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent')
```

[23]: Text(0.5, 1.0, 'Convergence of gradient descent')



0.1 Next change the iterations and learning rate (alpha) and try to take loss to minimum

[52]:
```
# Load the dataset
data = pd.read_csv('univariate_profits_and_populations_from_the_cities.csv')

# Prepare X_data and y_data
X_data = data['population'].values
y_data = data['profit'].values
m = len(y_data)

# Add a column of ones to X for the bias term
X = np.c_[np.ones(m), X_data]
y = y_data.reshape((m, 1))
```

[30]:
```
# Cost function
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
```

```
    error = predictions - y
    cost = (1 / (2 * m)) * np.dot(error.T, error)
    return cost.item()

# Gradient descent
def gradient_descent(X, y, theta, learning_rate, iterations):
    m = len(y)
    cost_history = []

    for _ in range(iterations):
        prediction = X.dot(theta)
        error = prediction - y
        theta -= (learning_rate / m) * X.T.dot(error)
        cost = compute_cost(X, y, theta)
        cost_history.append(cost)

    return theta, cost_history
```

[51]:

```
# Define parameter combinations to test the impact of Hyperparameters
combinations = [
    {"theta": np.zeros((2, 1)), "alpha": 0.01, "iters": 1500},
    {"theta": np.zeros((2, 1)), "alpha": 0.001, "iters": 1500},
    {"theta": np.array([[1.0], [1.0]]), "alpha": 0.01, "iters": 500},
    {"theta": np.array([[5.0], [-1.0]]), "alpha": 0.1, "iters": 100},
    {"theta": np.zeros((2, 1)), "alpha": 1.0, "iters": 100},
]
```

[56]:

```
# Run and display results for five different combinations
for i, combo in enumerate(combinations):
    theta_final, cost_history = gradient_descent(X, y, combo["theta"].copy(),␣
 ↪combo["alpha"], combo["iters"])
    print(f"Run {i+1}:")
    print(f"  Initial Theta: {combo['theta'].flatten()}")
    print(f"  Learning Rate: {combo['alpha']}")
    print(f"  Iterations: {combo['iters']}")
    print(f"  Final Theta: {theta_final.flatten()}")
    print(f"  Final Hypothesis: h(x) = {theta_final[0][0]:.4f} +␣
 ↪{theta_final[1][0]:.4f}x")
    print(f"  Min Cost: {cost_history[-1]:.4f}")
    print("-" * 50)
```

```
Run 1:
  Initial Theta: [0. 0.]
  Learning Rate: 0.01
  Iterations: 1500
  Final Theta: [-3.63029144  1.16636235]
  Final Hypothesis: h(x) = -3.6303 + 1.1664x
```

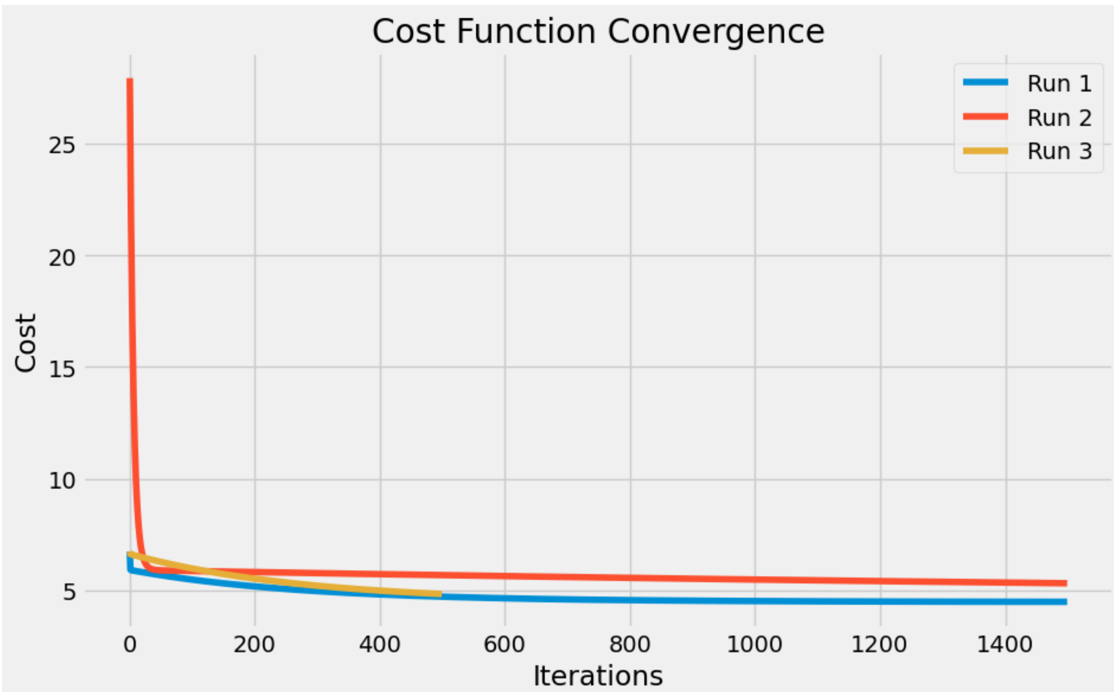
```
  Min Cost: 4.4834
--------------------------------------------------
Run 2:
  Initial Theta: [0. 0.]
  Learning Rate: 0.001
  Iterations: 1500
  Final Theta: [-0.86221218  0.88827876]
  Final Hypothesis: h(x) = -0.8622 + 0.8883x
  Min Cost: 5.3148
--------------------------------------------------
Run 3:
  Initial Theta: [1. 1.]
  Learning Rate: 0.01
  Iterations: 500
  Final Theta: [-1.92155972  0.99470171]
  Final Hypothesis: h(x) = -1.9216 + 0.9947x
  Min Cost: 4.8318
--------------------------------------------------
Run 4:
  Initial Theta: [ 5. -1.]
  Learning Rate: 0.1
  Iterations: 100
  Final Theta: [-9.53514166e+84 -9.49140084e+85]
  Final Hypothesis: h(x) = -95351416618956775173122346563764956802293203588882600984516564258991479118652820684800.0000 + -949140084308055207983363409023548036688789228409346230419484635745058916361987938058240.0000x
  Min Cost: 37410081749385192008695159225999902677487408909606484878676156486702594371899249904863541348596689845566465240150919301730108908506326364741556485863818376392630063966715904 0.0000
--------------------------------------------------
Run 5:
  Initial Theta: [0. 0.]
  Learning Rate: 1.0
  Iterations: 100
  Final Theta: [-7.41129347e+189 -7.37729543e+190]
  Final Hypothesis: h(x) = -74112934660000881446299905228854649783797350719412780900549953216595462083560309808765574313771516590923117410904522313210481566787892665417466172603347197840555929430763923809819871244451840.0000 + -73772954346989387697187673601349170523047054676099760988937914603013100511095612631120363835306141056861823254871714235373171872009031064212601695675809303129884817981108680789043594959781888.0000x
  Min Cost: inf
--------------------------------------------------
```

[58]:

```
# Plot Cost Function Convergence
for i, combo in enumerate(combinations[:3]):
```

```
    _, cost_history = gradient_descent(X, y, combo["theta"].copy(),␣
↪combo["alpha"], combo["iters"])
    plt.plot(range(len(cost_history)), cost_history, label=f"Run {i+1}")

plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Cost Function Convergence')
plt.legend()
plt.grid(True)
plt.show()
```



[59]:

```
import pandas as pd

results = []

for i, combo in enumerate(combinations):
    theta_final, cost_history = gradient_descent(X, y, combo["theta"].copy(),␣
↪combo["alpha"], combo["iters"])
    results.append({
        "Run": i+1,
        "Initial Theta": combo["theta"].flatten().tolist(),
        "Learning Rate": combo["alpha"],
        "Iterations": combo["iters"],
        "Final Theta": theta_final.flatten().tolist(),
```

```
        "Hypothesis": f"h(x) = {theta_final[0][0]:.4f} + {theta_final[1][0]:.
↪4f}x",
        "Min Cost": cost_history[-1]
    })

results_df = pd.DataFrame(results)
results_df
```

[59]:

```
   Run Initial Theta  Learning Rate  Iterations  \
0    1    [0.0, 0.0]          0.010        1500
1    2    [0.0, 0.0]          0.001        1500
2    3    [1.0, 1.0]          0.010         500
3    4   [5.0, -1.0]          0.100         100
4    5    [0.0, 0.0]          1.000         100

                                          Final Theta  \
0          [-3.6302914394043593, 1.1663623503355818]
1          [-0.8622121795348757, 0.8882787638284045]
2            [-1.92155971570547, 0.9947017143871797]
3   [-9.535141661895678e+84, -9.491400843080552e+85]
4  [-7.411293466000088e+189, -7.377295434698939e+…

                                        Hypothesis       Min Cost
0                           h(x) = -3.6303 + 1.1664x   4.483388e+00
1                           h(x) = -0.8622 + 0.8883x   5.314765e+00
2                           h(x) = -1.9216 + 0.9947x   4.831802e+00
3  h(x) = -95351416618956775173122346563764956802…  3.741008e+173
4  h(x) = -74112934660000881446299905228854649783…            inf
```