In [1]:

```
# imports
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import random
import numpy as np
import mltools as ml
```

In [2]:

```
#1 Read the advertising data set (3 attributes, one response variable)
# into table using a pandas data frame.
advertising = pd.read_csv('advertising.csv')
advertising.head(10)
```

Out[2]:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9
5	6	8.7	48.9	75.0	7.2
6	7	57.5	32.8	23.5	11.8
7	8	120.2	19.6	11.6	13.2
8	9	8.6	2.1	1.0	4.8
9	10	199.8	2.6	21.2	10.6

In [3]:

```
#2
# Add two columns to your table:
# TVstd: x’ = (x – μ) / s, where μ is the mean and s is the standard deviation.
# TVnorm: x’ = (x — xmin) / (xmax — xmin)


# add standardization column
TV_mean = advertising['TV'].mean()
TV_standard_dev = advertising['TV'].std()

advertising['TV_std'] = (np.subtract(advertising['TV'], TV_mean))/(TV_standard_dev)

# add normalization column
TV_min = advertising['TV'].min()
TV_max = advertising['TV'].max()

advertising['TV_norm'] = (np.subtract(advertising['TV'], TV_min)) / (np.subtract(TV_max, TV_min))

```

In [4]:

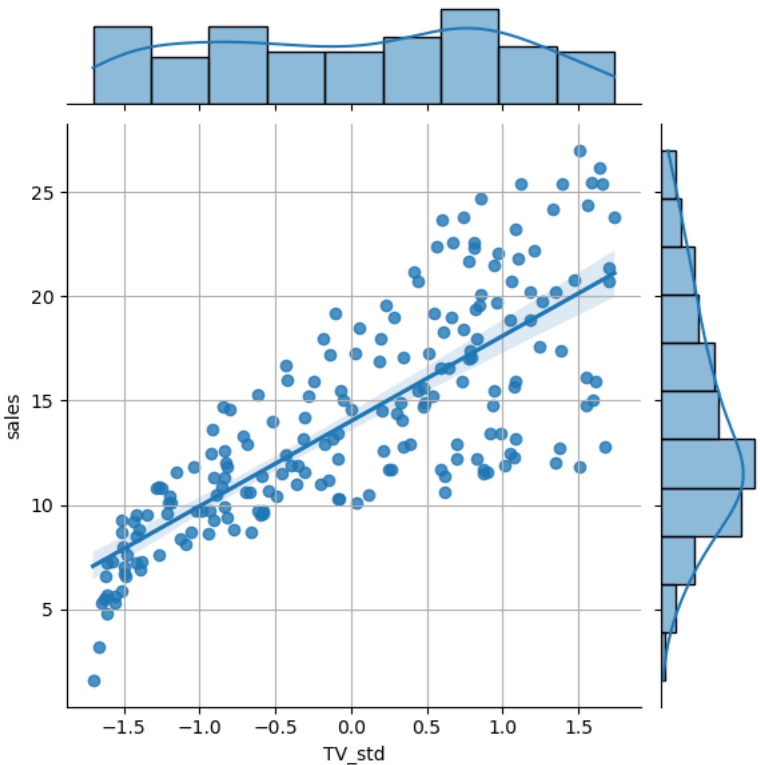
```
advertising.head(10)
```

Out[4]:

	Unnamed: 0	TV	radio	newspaper	sales	TV_std	TV_norm
0	1	230.1	37.8	69.2	22.1	0.967425	0.775786
1	2	44.5	39.3	45.1	10.4	-1.194379	0.148123
2	3	17.2	45.9	69.3	9.3	-1.512360	0.055800
3	4	151.5	41.3	58.5	18.5	0.051919	0.509976
4	5	180.8	10.8	58.4	12.9	0.393196	0.609063
5	6	8.7	48.9	75.0	7.2	-1.611365	0.027054
6	7	57.5	32.8	23.5	11.8	-1.042960	0.192087
7	8	120.2	19.6	11.6	13.2	-0.312652	0.404126
8	9	8.6	2.1	1.0	4.8	-1.612530	0.026716
9	10	199.8	2.6	21.2	10.6	0.614501	0.673318

In [5]:

```
# 3
# Use the seaborn library to produce a joint plot of sales (y axis) vs. TVstd advertising (x- axis).
# Specifying the parameter kind='reg' will produce a linear regression line.
# The shaded area around the line represents a 95% confidence interval.

# seaborn joint plot
sns.jointplot(data = advertising, x='TV_std', y='sales',  kind='reg')
plt.grid()
```



In [6]:

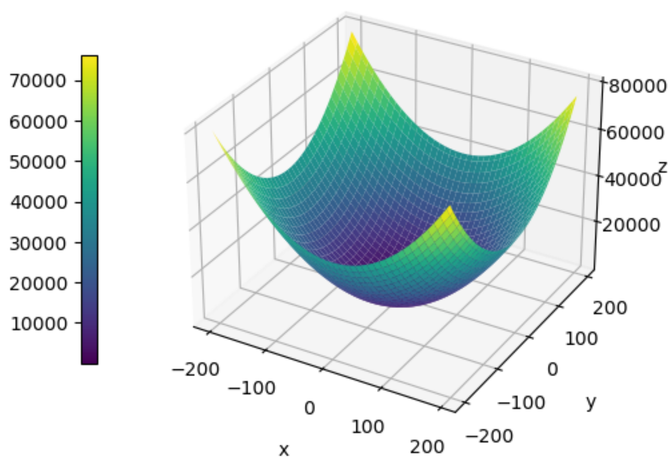
```
# 4 Implement functions for MSE (Mean Squared Error) and MAE (Mean Absolute Error).
# Both functions should accept two parameters (actual response values and model predictions as either a data series

import numpy as np


def MSE(response_values, model_predictions):
    mse = np.mean(((np.subtract(response_values, model_predictions)) ** 2))
    return mse


def MAE(response_values, model_predictions):
    mae = np.mean(np.abs(np.subtract(response_values, model_predictions)))
    return mae

```

In [7]:

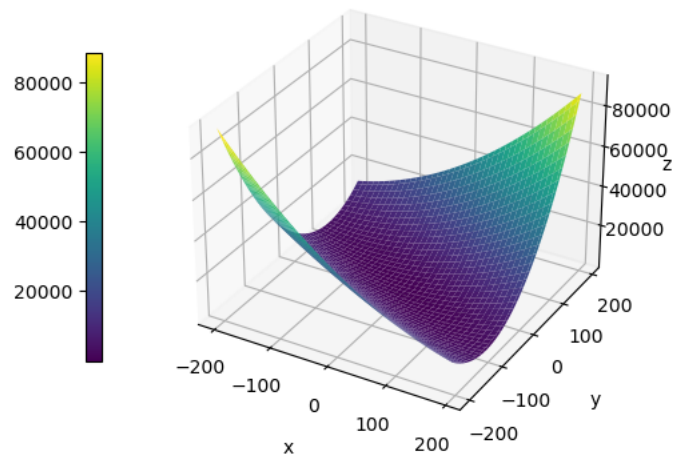
```
# 5
# Suppose that our predicted response is: ypred = b0 + b1 x.
# Create two surface plots depicting MSE as a function of b0 and b1.
# One plot should use x-values taken from TVstd, the other from TVnorm.
# Allow both b0 and b1 to range from -200 to +200 in steps of 5.

# calculate MSE function for plots
def MSE_calc(b, m, x):
    b0_errors = []
    for b0 in b:
        b1_errors = []
        for b1 in m:
            x_errors = []
            for spend in x:
                x_errors.append(b0 + (b1*spend))
            error = MSE(x_errors, x)
            b1_errors.append(error)
        b0_errors.append(b1_errors)
    return np.array(b0_errors)

# calculate MAE function for plots
def MAE_calc(b, m, x):
    b0_errors = []
    for b0 in b:
        b1_errors = []
        for b1 in m:
            x_errors = []
            for spend in x:
                x_errors.append(b0 + (b1*spend))
            error = MAE(x_errors, x)
            b1_errors.append(error)
        b0_errors.append(b1_errors)
    return np.array(b0_errors)

# set surface plots
b0 = np.arange(-200, 200, 5)
b1 = np.arange(-200, 200, 5)
x, y = np.meshgrid(b0, b1)

```

In [8]:

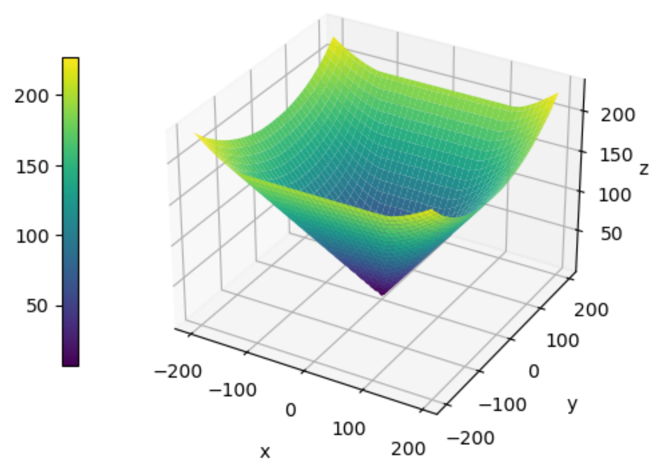
```
# PLOT 1 - MSE, x values from TVstd
z1 = MSE_calc(b0, b1, advertising['TV_std'])
ml.surface_plot(x,y,z1)
```



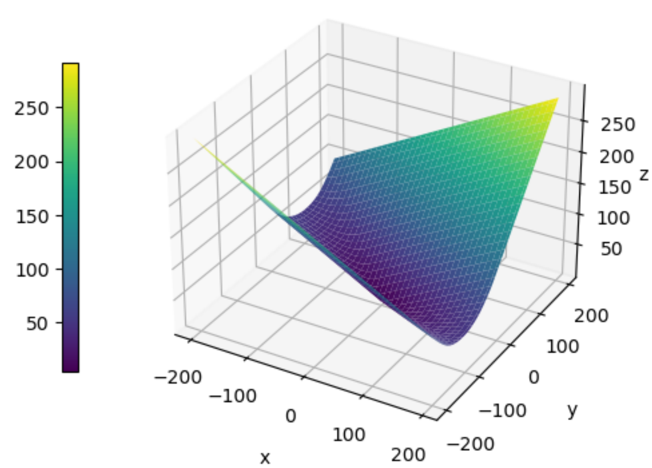
In [9]:
```
# PLOT 2 - MSE, x-values from TVnorm
z1 = MSE_calc(b0, b1, advertising['TV_norm'])
ml.surface_plot(x,y,z1)
```



In [10]:
```
# PLOT 3 - MAE, x-vaues from TVstd
z1 = MAE_calc(b0, b1, advertising['TV_std'])
ml.surface_plot(x,y,z1)
```



In [11]:
```
# PLOT 4 - MAE, x-values from TVnorm
z1 = MAE_calc(b0, b1, advertising['TV_norm'])
ml.surface_plot(x,y,z1)

```



In [12]:

```
# 6.
# Implement a “random step” search algorithm (as a function) to find the best-fit linear model for X = TVstd, y = s
# Here is how the random step algorithm works: Start your search at a random value for b0 and b1 in the range [-200
# Measure the current MSE. Now randomly adjust values for both coefficients by some random amount [-1...+1] and rec
# If the MSE is reduced, keep the new values for b0 and b1. Otherwise revert to the original values.
# Halt the algorithm when no improvements to MSE can be found after k = 1000 random updates.
# Report your final b0 and b1.
import random as rnd

# x = tv_std
tv_std = advertising["TV_std"].tolist()

# y = sales
sales = advertising["sales"].tolist()

# randomly adjust values for both coefficients by some random amount [-1...+1]
#random_adjust = np.arange(-1, 1, .01)

#“random step” search algorithm
def random_search(b0, b1, x, y):

    b0_start = rnd.uniform(-200, 200)
    b1_start = rnd.uniform(-200,200)
    b0_list_randstep = []
    b1_list_randstep = []

    # measure the current MSE
    y_pred = []
    for val in x:
        y_pred.append(b0_start + b0_start * val)

    mse_start = MSE(y, y_pred)

    # randomly adjust values for both coefficients by some random amount and recompute MSE
    # Halt the algorithm when no improvements to MSE can be found after k = 1000 random updates.
    b0_best = b0_start
    b1_best = b1_start
    step_counter = 0
    while step_counter < 1000:

        b0_new = b0_start + rnd.uniform(-1,1)
        b1_new = b1_start + rnd.uniform(-1,1)

        step_counter += 1

        # measure new MSE
        y_pred1 = []
        for val in x:
            y_pred1.append(b0_new + b1_new * val)

        mse_new = MSE(y, y_pred1)

        # If the MSE is reduced, keep the new values for b0 and b1. Otherwise revert to the original values.
        if (mse_new < mse_start):
            b0_start = b0_new
            b1_start = b1_new
            b0_list_randstep.append(b0_start)
            b1_list_randstep.append(b1_start)
            mse_start = mse_new


    return b0_start, b1_start, b0_list_randstep, b1_list_randstep

```

In [13]:
```python
# test random step seach algorithm
random_search_results = random_search(b0,b1,tv_std,sales)

b0_final = random_search_results[0]
b1_final = random_search_results[1]

b0_vals_random_step = random_search_results[2]
b1_vals_random_step = random_search_results[3]


print("b0=",b0_final)
print("b1=", b1_final)

#ignore these lists, printing them for sanity check
print(b0_vals_random_step,b1_vals_random_step)

```

```
b0= 14.125902015138555
b1= 4.154462674754151
[-168.65494864245366, -168.35937931519783, -167.9735758644796, -167.2587276248721, -167.00862089134125, -166.61968761
83269, -166.4925843075241, -165.743643175579, -164.97973437109184, -164.87031163445667, -163.9892699469731, -163.1707
287460509, -162.4118855279324, -161.6922760160491, -162.19321467937363, -161.5418399290197, -160.74948493797285, -16
0.148866307404, -159.43383148583777, -159.3846096136429, -158.66691699247687, -158.36530187519043, -158.5369201498676
8, -158.65895472620664, -157.7532562456838, -156.86087815357402, -156.57885928367986, -155.6507090904378, -155.583298
36209867, -155.20197868353208, -155.81869141589502, -155.58689086367988, -155.25817455715938, -154.5361014273454, -15
3.7665216848481, -154.06832312543503, -153.55271814460048, -153.15320420779753, -152.6713774296831, -152.414147644448
75, -151.48330153849585, -151.90712557485466, -151.90998747770678, -151.0142632256178, -150.1123396365369, -149.59458
014191426, -149.52828792911725, -148.58902882595294, -147.76068868577778, -147.0183266546095, -146.52479967431526, -1
47.01995749270358, -146.4309799070848, -146.2900634328544, -145.3944439952367, -144.55339377834923, -144.090376463857
38, -143.7516984163195, -143.29668544107713, -142.327207487908, -141.82348574774736, -141.17354630143797, -141.062676
84193313, -140.12428368366957, -139.70035031588193, -138.8753816459979, -139.3845075404427, -138.95453812954236, -13
8.11178703399062, -137.95285028518933, -137.1275733967749, -136.6718014866944, -136.56952403934073, -135.978687917833
16, -135.14193863347103, -134.47604923721207, -134.29912116037536, -134.1796484372901, -133.20429546778863, -132.5436
3960828917, -131.91221396986955, -131.02248207326275, -130.9136589513985, -129.97782235256346, -129.4484768549909, -1
29.2095196429882, -128.51522784298263, -128.89342049804955, -128.9582172293882, -128.54820336658966, -127.69941243888
262, -127.35419666353648, -126.44975427053237, -126.85174372478285, -126.55490350891651, -125.56760371448541, -124.94
```

Are your results consistent with your Seaborn joint plot from step 3?

Yes my results are consistent with my Seaborn joint plot from step 3 because at TV_std = 0 is about 14 on the sales scale and the slope of the line seems to be about 4.

In [14]:
```python
# 7. Modify your random step algorithm to instead implement gradient descent.
# Recompute and report your optimal best fit coefficients.

b0 = np.arange(-200, 200, 1)
b1 = np.arange(-200, 200, 1)

# calculate slope for b1
def slope_calc_b1(y, y_pred, x):
    return np.sum(np.subtract(y_pred, y) * x)

#check
trial = []
trial.append(b0[200] +(b1[200]*advertising['TV_std']))
slope_calc_b1(advertising.sales.tolist(), trial, advertising.TV_std.tolist())
```

Out[14]: -812.1631703093049

In [15]:
```python
# calculate slope for b0
def slope_calc_b0(y, y_pred):
    return np.sum(np.subtract(y_pred, y))

#check
trial = []
trial.append(b0[200] +(b1[200]*advertising['TV_std']))
slope_calc_b0(advertising.sales.tolist(), trial)
```

Out[15]: -2804.5

In [16]:

```python
b0 = np.arange(-200, 200, 1)
b1 = np.arange(-200, 200, 1)

# Random step algorithm with gradient descent
def random_search_gradient(b0, b1, x, y):
    alpha = 0.001

    i = random.randint(0,399)
    j = random.randint(0,399)

    b0_list_gradient = []
    b1_list_gradient = []

    b0_start = b0[i]
    b1_start = b1[j]

    # measure the current MSE
    y_pred = []
    for val in x:
        y_pred.append(b0_start + b1_start * val)

    start_slope_b0 = slope_calc_b0(y, y_pred)
    start_slope_b1 = slope_calc_b1(y, y_pred, x)
    mse_start = MSE(y, y_pred)
    mse_new = mse_start - alpha
    b0_best = 0
    b1_best = 0

    while(mse_new <= mse_start):
        b0_start -= start_slope_b0 * alpha
        b1_start -= start_slope_b1 * alpha
        y_pred = []
        for val in x:
            y_pred.append(b0_start + b1_start * val)
        new_slope_b0 = slope_calc_b0(y, y_pred)
        new_slope_b1 = slope_calc_b1(y, y_pred, x)
        mse_new = MSE(y, y_pred)

        if mse_new <= mse_start:
            start_slope_b0 = new_slope_b0
            start_slope_b1 = new_slope_b1
            b0_best = b0_start
            b1_best = b1_start
            b0_list_gradient.append(b0_best)
            b1_list_gradient.append(b1_best)
            mse_start = mse_new
        else:
            b0_start = b0_best
            b1_start = b1_best
    return b0_best, b1_best, b0_list_gradient, b1_list_gradient
```

In [17]:
```
# recompute and report optimal best fit coefficients
gradient_results = random_search_gradient(b0, b1, tv_std, sales)

b0_final_gradient = gradient_results[0]
b1_final_gradient = gradient_results[1]

b0_vals_gradient = gradient_results[2]
b1_vals_gradient = gradient_results[3]

print("b0=", b0_final_gradient)
print("b0=",b1_final_gradient)


#ignore these lists, printed for sanity check
print(b0_vals_gradient,b1_vals_gradient)
```

```
b0= 14.02249999282254
b0= 4.081221976927659
[-54.795500000000004, -41.03190000000001, -30.021020000000007, -21.21231600000001, -14.165352800000008, -8.5277822400
00008, -4.017725792000008, -0.4096806336000074, 2.4767554931199927, 4.785904394495994, 6.633223515596795, 8.111078812
477436, 9.293363049981949, 10.239190439985558, 10.995852351988447, 11.601181881590756, 12.085445505272604, 12.4728564
04218083, 12.782785123374467, 13.030728098699573, 13.229082478959658, 13.387765983167727, 13.514712786534181, 13.6162
70229227345, 13.697516183381875, 13.7625129467055, 13.8145103573644, 13.85610828589152, 13.889386628713215, 13.916009
302970572, 13.937307442376458, 13.954345953901166, 13.967976763120932, 13.978881410496745, 13.987605128397396, 13.994
584102717917, 14.000167282174333, 14.004633825739466, 14.008207060591573, 14.011065648473258, 14.013352518778605, 14.
015182015022884, 14.016645612018307, 14.017816489614646, 14.018753191691717, 14.019502553353373, 14.020102042682698,
14.020581634146158, 14.020965307316926, 14.02127224585354, 14.021517796682831, 14.021714237346265, 14.02187138987701
2, 14.021997111901609, 14.022097689521287, 14.022178151617029, 14.022242521293622, 14.022294017034897, 14.02233521362
7917, 14.022368170902334, 14.022394536721867, 14.022415629377493, 14.022432503501994, 14.022446002801596, 14.02245680
2241276, 14.02246544179302, 14.022472353434416, 14.022477882747532, 14.022482306198025, 14.02248584495842, 14.0224886
75966736, 14.02249094077339, 14.022492752618712, 14.02249420209497, 14.022495361675976, 14.022496289340781, 14.022497
031472625, 14.022497625178099, 14.02249810014248, 14.022498480113983, 14.022498784091185, 14.02249902727294, 14.0224
99221818359, 14.022499377454688, 14.02249950196375, 14.022499601571, 14.0224996812568, 14.02249974500544, 14.02249979
6004352, 14.022499836803481, 14.022499869442784, 14.022499895554228, 14.022499916443381, 14.022499933154705, 14.02249
9946523764, 14.022499957219011, 14.022499965775209, 14.022499972620167, 14.022499978096134, 14.022499982476907, 14.02
2499985981526, 14.02249998878522, 14.022499991028177, 14.02249999282254] [135.38016317030934, 109.25167386972711, 88.
32275393996073, 71.55868907621786, 58.13067312035982, 47.37483233971753, 38.75940387442306, 31.85844567372218, 26.330
778154960775, 21.90311647243289, 18.356559464728054, 15.51576730155648, 13.240292778856048, 11.417637686173002, 9.957
690956933881, 8.788273626813345, 7.851570345386795, 7.101271016964129, 6.5002812548975735, 6.018888455482262, 5.63329
2823150597, 5.324430721652934, 5.0770321783533054, 4.878865945170303, 4.720134792390718, 4.592991139014271, 4.4911490
72659736, 4.409573577509754, 4.344231605894618, 4.291892686630894, 4.249969212300652, 4.216388509362128, 4.1894903663
0837, 4.167944953722309, 4.150687078240875, 4.1368635199802455, 4.125790849813482, 4.116921641009904, 4.1098174047582
39, 4.1041269115206545, 4.099568826437349, 4.095917800285622, 4.092993328338088, 4.090650826308114, 4.08877448218210
4, 4.08727153053717, 4.086067666269578, 4.085103370991237, 4.084330970473286, 4.083712277658408, 4.083216704713689,
4.082819750784971, 4.082501790688067, 4.082247104650446, 4.082043101134313, 4.08187969431789, 4.081748805457935, 4.08
16439634811115, 4.081559985057676, 4.081492718340503, 4.081438837700048, 4.081395679307044, 4.081361109434248, 4.0813
33418966137, 4.081311238901181, 4.081293472669151, 4.081279241917295, 4.081267843085059, 4.081258712620437, 4.0812513
99118275, 4.081245541003044, 4.081240848652744, 4.081237090080153, 4.081234079463508, 4.0812316679595755, 4.081229736
344925, 4.08122818912159, 4.081226949795699, 4.08122595709566, 4.081225161942929, 4.081224525025592, 4.08122401485480
4, 4.081223606208003, 4.081223278881916, 4.08122301669372, 4.081222806680975, 4.081222638460766, 4.081222503716379,
4.081222395786125, 4.081222309333992, 4.0812222400858325, 4.081222184618057, 4.081222140188369, 4.081222104600188, 4.
081222076094056, 4.081222053260644, 4.0812220349710815, 4.081222020321142, 4.08122200858654, 4.0812219991871235, 4.08
1221991658191, 4.081221985627517, 4.081221980796946, 4.081221976927659]
```

In [18]:
```
# 8.
# Determine a linear best-fit using the scikit-learn linear regression model.
# The values obtained from your two algorithms should be very close to the scikit-learn result!

from sklearn import linear_model as lm

regr = lm.LinearRegression()
regr.fit(advertising[["TV_std"]], advertising["sales"])
y_intercept = regr.intercept_
coef = regr.coef_
print("Y-intercept: ", y_intercept)
print("Slope: ", coef)

```

```
Y-intercept:  14.0225
Slope:  [4.08122196]
```

The values obtained from my two above algorithms are very close to the skikit learn result here.

In [19]:

```
# 9.
# Plot the progress of your two algorithms on a single contour plot of the MSE error function.
#random_search_gradient
#random_search

# The 'single contour plot of the MSE error' refers to just a contour plot that visualizes the output
# of the MSE for different values of b0 and b1. For this, you need to compute the output of the MSE
# for each pair of b0, b1 and use these outputs as the Z for the contour plot.


# Note that this does not yet involve any other algorithm such as gradient descent or random stepping.
```

In [20]:

```
# var1,var2,var3,var4 = random_search(b0,b1,tv_std, sales)

#b0_vals_random_step = var3
#b1_vals_random_step = var4
#b0_vals_random_step = random_search(b0,b1,tv_std,sales)[2]
#b1_vals_random_step = random_search(b0,b1,tv_std,sales)[3]

#b0_vals_gradient = random_search_gradient(b0, b1, tv_std, sales)[2]
#b1_vals_gradient = random_search_gradient(b0, b1, tv_std, sales)[3]
```

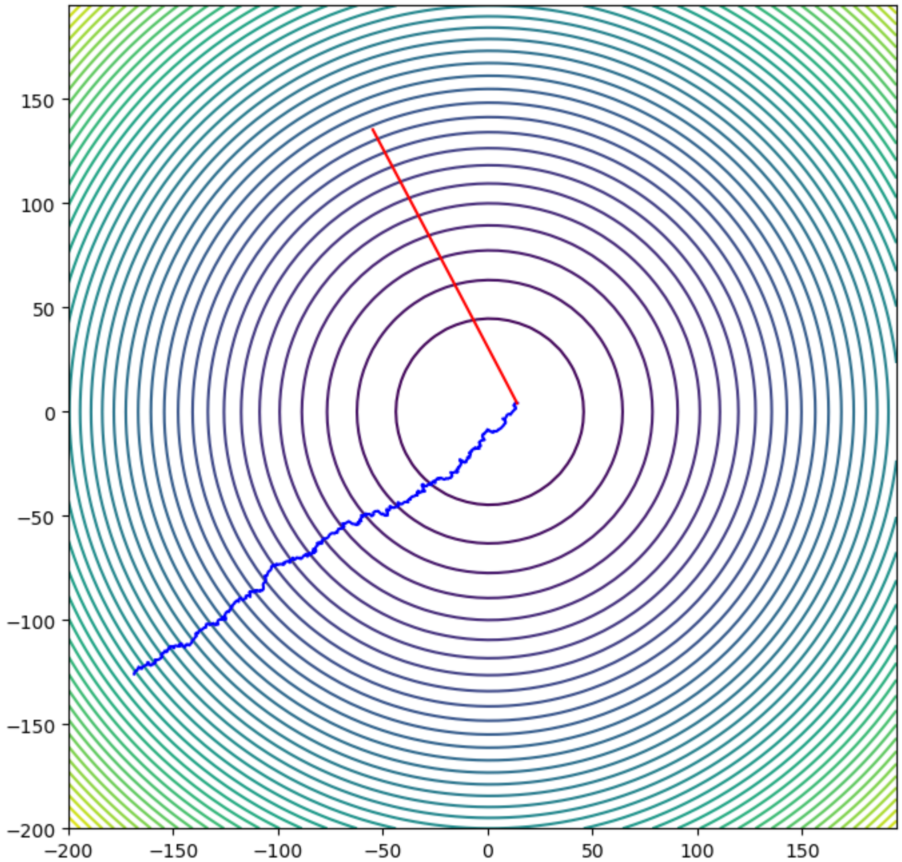
In [21]:

```
#print(b0_vals_random_step)
#print(b1_vals_random_step)
```

In [24]:

```
def contour_plot(x, y, f, dfdx, dfdy, gradient_field = True, ascend = None, alpha=0.01):

    b0 = np.arange(-200, 200,5)
    b1 = np.arange(-200, 200,5)

    X, Y = np.meshgrid(b0,b1)

    prediction = advertising["TV_std"]
    actual = advertising['sales']

    tv_std = list(prediction)
    act_sales = np.array(actual)

    Z = MSE_calc(b0, b1, prediction)


    plt.figure(figsize=(8, 8))
    cp = plt.contour(X,Y,Z, 50)


# store the b0, b1 from gradient descent algorithm and random step algoritms
# 2 different lists of b0, 2 lists of b1, 2 from gradient descnet and 2 from random step algorithsm
    plt.plot(b0_vals_random_step,b1_vals_random_step, color = "blue")
    plt.plot(b0_vals_gradient,b1_vals_gradient, color = "red")

```

In [25]:
```
b0 = np.arange(-200, 200, 5)
b1 = np.arange(-200, 200, 5)
contour_plot(b0, b1, MSE_calc, slope_calc_b0, slope_calc_b1, advertising["TV_std"], advertising["sales"])


```



In [ ]:
```

```