Problem Set 1: Linear Regression

To run and solve this assignment, one must have a working IPython Notebook installation. The easiest way to set it up for both Windows and Linux is to install Anaconda. Then save this file to your computer, run Anaconda and choose this file in Anaconda's file explorer. Use `Python 3` version. The statements below assume that you have already followed these instructions. If you are new to Python or its scientific library, Numpy, there are some nice tutorials here and here.

To run code in a cell or to render Markdown+LaTeX press `Ctr+Enter` or `[>|]` (like "play") button above. To edit any code or text cell double click on its content. To change cell type, choose "Markdown" or "Code" in the drop-down menu above. Here are some useful resources for Markdown guide and LaTeX tutorial if you are not familiar with the basic syntax.

If certain output is given for some cells, that means that you are expected to get similar results.

We need to submit both PDF file and notebook for ps1 submission. To print this notebook to a pdf file, you can go to "File" -> "Download as" -> "PDF via LaTex(.pdf)" or simply use "print" in browser.

Total: 185 points.

1. Numpy Tutorial

1.1 [5pt] Modify the cell below to return a 5x5 matrix of ones. Put some code there and press `Ctrl+Enter` to execute contents of the cell. You should see something like the output above. [1] [2]

```
import numpy as np

print(np.ones((5, 5)))
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

1.2 [5pt] Vectorizing your code is very important to get results in a reasonable time. Let A be a 10x10 matrix and x be a 10-element column vector. Your friend writes the following code. How would you vectorize this code to run without any for loops? Compare execution speed for different values of `n` with `%timeit`.

```
n = 10
def compute_something(A, x):
    v = np.zeros((n, 1))
    for i in range(n):
        for j in range(n):
            v[i] += A[i, j] * x[j]
    return v

A = np.random.rand(n, n)
x = np.random.rand(n, 1)
print(compute_something(A, x))
```

```
[[3.14905778]
 [2.37508986]
 [2.22022954]
 [1.80044602]
 [1.6969819 ]
 [2.40725336]
 [2.57264633]
 [2.27361538]
 [2.05413954]
 [2.01757259]]
```

```
def vectorized(A, x):
    return A @ x

print(vectorized(A, x))
assert np.max(abs(vectorized(A, x) - compute_something(A, x))) < 1e-3
```

```
[[3.14905778]
 [2.37508986]
 [2.22022954]
 [1.80044602]
 [1.6969819 ]
 [2.40725336]
 [2.57264633]
 [2.27361538]
 [2.05413954]
 [2.01757259]]
```

```
for n in [5, 10, 100, 500]:
    A = np.random.rand(n, n)
    x = np.random.rand(n, 1)
    %timeit -n 5 compute_something(A, x)
    %timeit -n 5 vectorized(A, x)
    print('---')
```

```
68.8 µs ± 2.74 µs per loop (mean ± std. dev. of 7 runs, 5 loops each)
1.56 µs ± 981 ns per loop (mean ± std. dev. of 7 runs, 5 loops each)
---
268 µs ± 20.7 µs per loop (mean ± std. dev. of 7 runs, 5 loops each)
1.58 µs ± 987 ns per loop (mean ± std. dev. of 7 runs, 5 loops each)
---
25.9 ms ± 476 µs per loop (mean ± std. dev. of 7 runs, 5 loops each)
The slowest run took 5.75 times longer than the fastest. This could mean that an intermediate result is being cached.
161 µs ± 127 µs per loop (mean ± std. dev. of 7 runs, 5 loops each)
---
652 ms ± 12.9 ms per loop (mean ± std. dev. of 7 runs, 5 loops each)
The slowest run took 5.06 times longer than the fastest. This could mean that an intermediate result is being cached.
164 µs ± 116 µs per loop (mean ± std. dev. of 7 runs, 5 loops each)
---
```

2. Linear regression with one variable

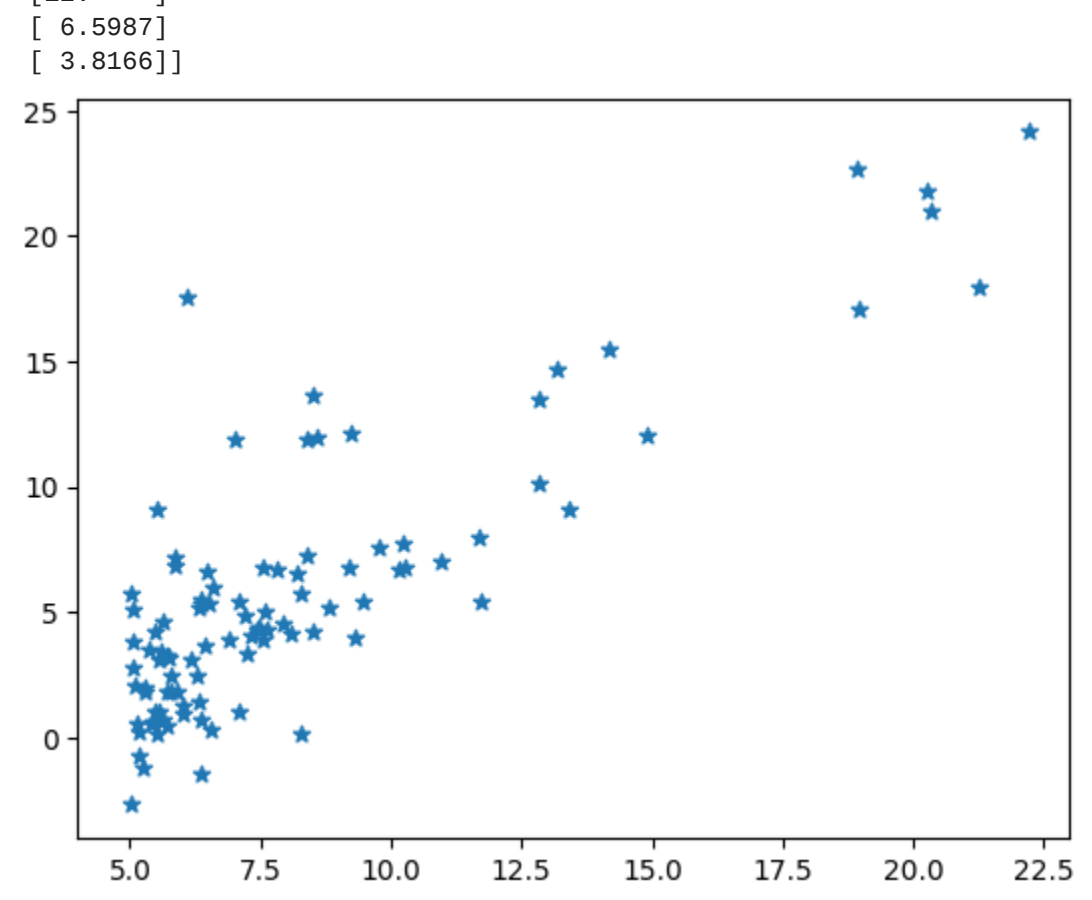
In this part of this exercise, you will implement linear regression with one variable to predict profits for a food truck. Suppose you are the CEO of a restaurant franchise and are considering different cities for opening a new outlet. The chain already has trucks in various cities and you have data for profits and populations from the cities. You would like to use this data to help you select which city to expand to next. The file ex1data.txt contains the dataset for our linear regression problem. The first column is the population of a city and the second column is the profit of a food truck in that city. A negative value for profit indicates a loss.

2.1 [10pt] Get a plot similar to below : [1] [2] [3]

Before starting on any task, it is often useful to understand the data by visualizing it. For this dataset, you can use a scatter plot to visualize the data, since it has only two properties to plot (profit and population). Many other problems that you will encounter in real life are multi-dimensional and can't be plotted on a 2-d plot.

```
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt('ex1data1.txt', delimiter=',')
X, y = data[:, 0, np.newaxis], data[:, 1, np.newaxis]
n = data.shape[0]
print(X.shape, y.shape, n)
print(X[:10], '\n', y[:10])

plt.scatter(X, y, marker='*')
plt.xlim(4, 23)

plt.show()
```

```
(97, 1) (97, 1) 97
[[6.1101]
 [5.5277]
 [8.5186]
 [7.0032]
 [5.8598]
 [8.3829]
 [7.4764]
 [8.5781]
 [6.4862]
 [5.0546]]
 [[17.592 ]
 [ 9.1302]
 [13.662 ]
 [11.854 ]
 [ 6.8233]
 [11.886 ]
 [ 4.3483]
 [12.    ]
 [ 6.5987]
 [ 3.8166]]
```



2.2 Gradient Descent

In this part, you will fit the linear regression parameter θ to our dataset using gradient descent.

The objective of linear regression is to minimize the cost function

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h(x^{(i)};\theta) - y^{(i)}\right)^2$$

where the hypothesis $h(x;\theta)$ is given by the linear model (x' has an additional fake feature always equal to '`1`')

$$h(x;\theta) = \theta^T x' = \theta_0 + \theta_1 x$$

Recall that the parameters of your model are the θ_j values. These are the values you will adjust to minimize cost J(θ). One way to do this is to use the gradient descent algorithm. In batch gradient descent algorithm, each iteration performs the update.

$$\theta_j^{(k+1)} = \theta_j^{(k)} - \eta\frac{1}{m}\sum_i\left(h(x^{(i)};\theta) - y^{(i)}\right)x_j^{(i)}$$

With each step of gradient descent, your parameter θ_j come closer to the optimal values that will achieve the lowest cost J(θ).

2.2.1 [5pt] Where does this update rule comes from?

2.2.2 [30pt] Cost Implementation

As you perform gradient descent to learn to minimize the cost function, it is helpful to monitor the convergence by computing the cost. In this section, you will implement a function to calculate $J(\theta)$ so you can check the convergence of your gradient descent implementation.

In the following lines, we add another dimension to our data to accommodate the intercept term and compute the prediction and the loss. As you are doing this, remember that the variables X and y are not scalar values, but matrices whose rows represent the examples from the training set. In order to get x' add a column of ones to the data matrix `X`.

You should expect to see a cost of approximately 32.

```
import numpy as np
def add_column(X):
    assert len(X.shape) == 2 and X.shape[1] == 1
    return np.insert(X, 0, 1, axis=1)

def predict(X, theta):
    assert len(X.shape) == 2 and X.shape[1] == 1
    assert theta.shape == (2, 1)

    X_prime = add_column(X)
    pred = X_prime @ theta
    return pred

def loss(X, y, theta):
    assert X.shape == (n, 1)
    assert y.shape == (n, 1)
    assert theta.shape == (2, 1)

    X_prime = add_column(X)
    assert X_prime.shape == (n, 2)
    loss = ((predict(X, theta) - y)**2).mean()/2
    return loss

theta_init = np.zeros((2, 1))
print(loss(X, y, theta_init))
```

```
32.072733877455676
```

2.2.3 [40pt] GD Implementation

Next, you will implement gradient descent. The loop structure has been written for you, and you only need to supply the updates to θ within each iteration.

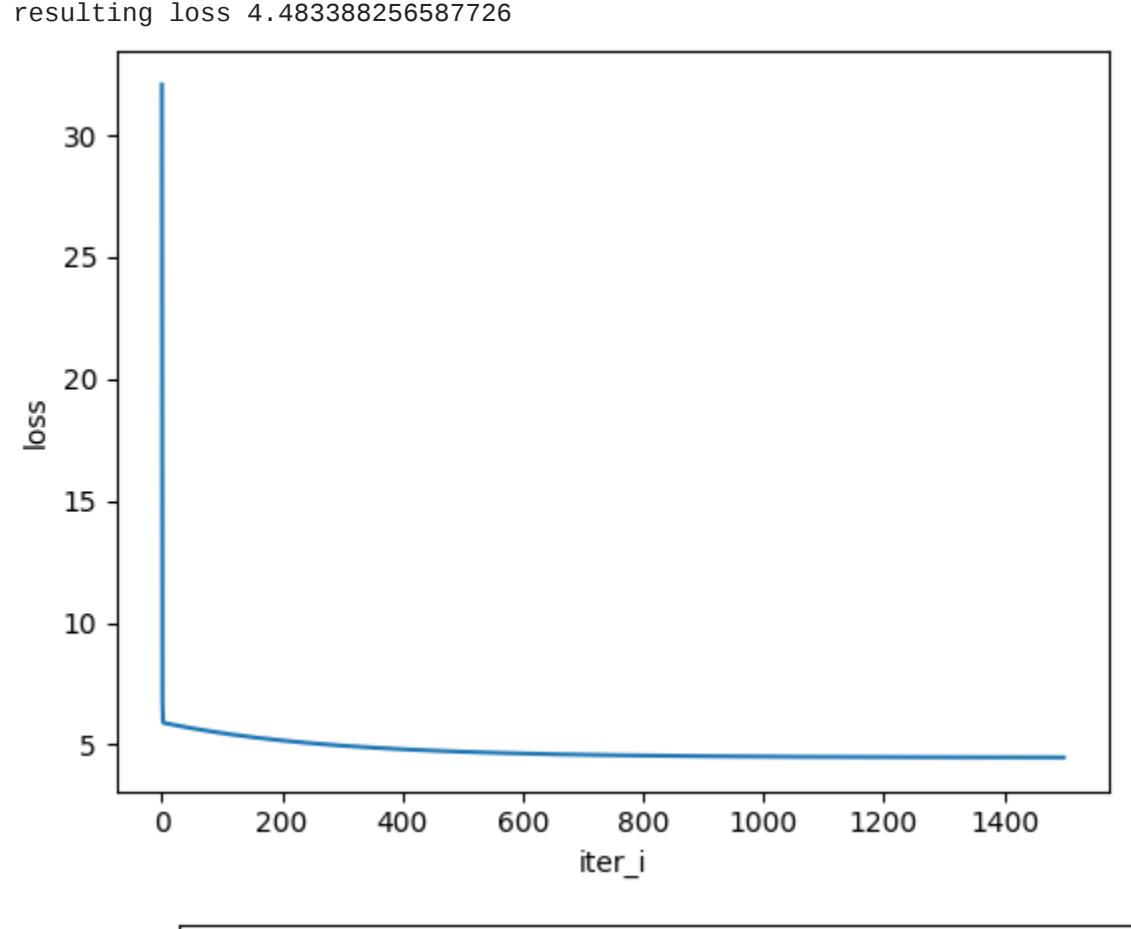
As you program, make sure you understand what you are trying to optimize and what is being updated. Keep in mind that the cost is parameterized by the vector θ not X and y. That is, we minimize the value of $J(\theta)$ by changing the values of the vector θ, not by changing X or y.

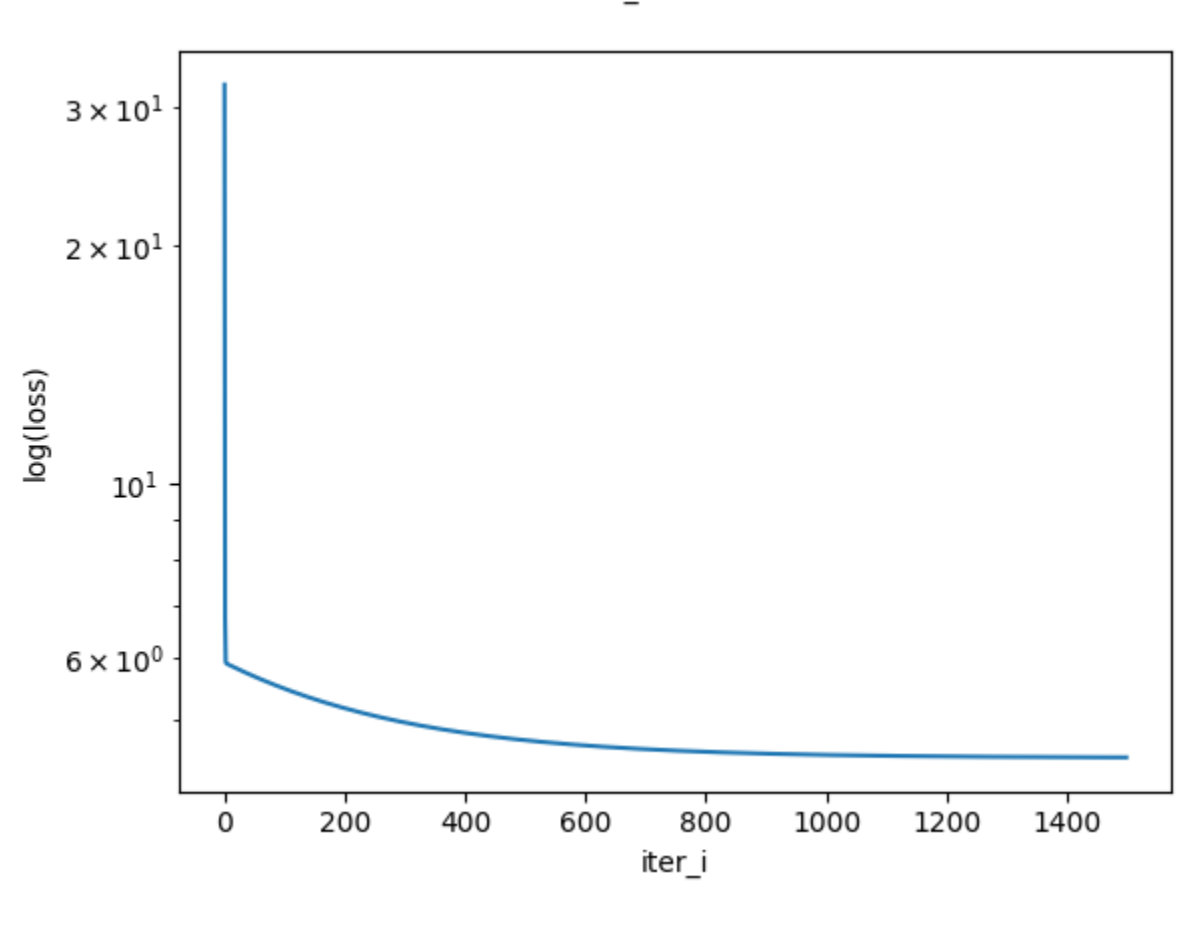
A good way to verify that gradient descent is working correctly is to look at the value of and check that it is decreasing with each step. Your value of $J(\theta)$ should never increase, and should converge to a steady value by the end of the algorithm. Another way of making sure your gradient estimate is correct is to check it againts a finite difference approximation.

We also initialize the initial parameters to 0 and the learning rate alpha to `0.01`.

```
import scipy.optimize
from functools import partial

def loss_gradient(X, y, theta):
    X_prime = add_column(X)
    loss_grad = ((predict(X, theta) - y)*X_prime).mean(axis=0)[:, np.newaxis]
    return loss_grad

assert loss_gradient(X, y, theta_init).shape == (2, 1)

def finite_diff_grad_check(f, grad, points, eps=1e-10):
    errs = []
    for point in points:
        point_errs = []
        grad_func_val = grad(point)
        for dim_i in range(point.shape[0]):
            diff_v = np.zeros_like(point)
            diff_v[dim_i] = eps
            dim_grad = (f(point+diff_v) - f(point-diff_v))/(2*eps)
            point_errs.append(abs(dim_grad - grad_func_val[dim_i]))
        errs.append(point_errs)
    return errs

test_points = [np.random.rand(2, 1) for _ in range(10)]
finite_diff_errs = finite_diff_grad_check(
    partial(loss, X, y), partial(loss_gradient, X, y), test_points
)

print('max grad comp error', np.max(finite_diff_errs))
assert np.max(finite_diff_errs) < 1e-3, "grad computation error is too large"

def run_gd(loss, loss_gradient, X, y, theta_init, lr=0.01, n_iter=1500):
    theta_current = theta_init.copy()
    loss_values = []
    theta_values = []

    for i in range(n_iter):
        loss_value = loss(X, y, theta_current)
        theta_current -= lr*loss_gradient(X, y, theta_current)
        loss_values.append(loss_value)
        theta_values.append(theta_current)

    return theta_current, loss_values, theta_values

result = run_gd(loss, loss_gradient, X, y, theta_init)
theta_est, loss_values, theta_values = result

print('estimated theta value', theta_est.ravel())
print('resulting loss', loss(X, y, theta_est))
plt.ylabel('loss')
plt.xlabel('iter_i')
plt.plot(loss_values)
plt.show()

plt.ylabel('log(loss)')
plt.xlabel('iter_i')
plt.semilogy(loss_values)
plt.show()
```

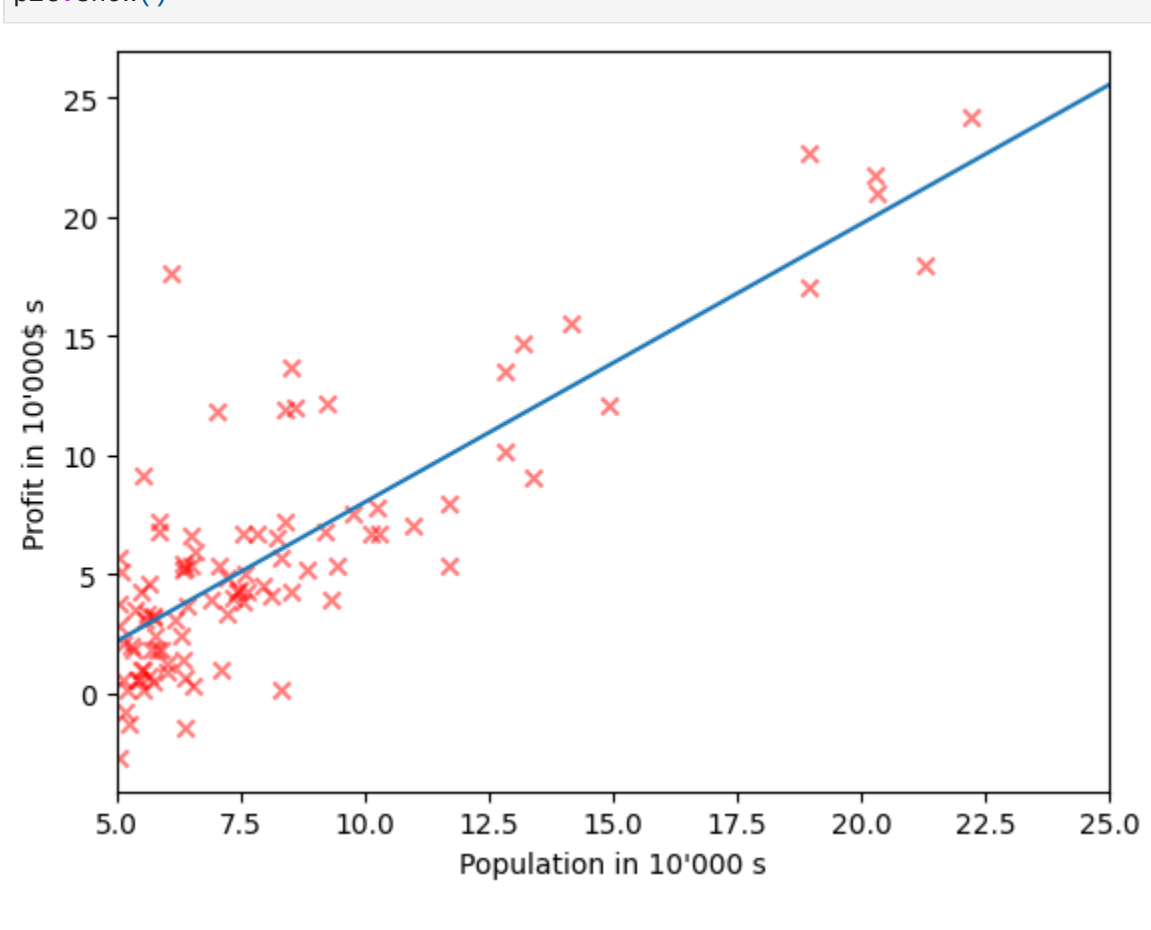
```
max grad comp error 3.288442607196629e-05
estimated theta value [-3.63029144  1.16636235]
resulting loss 4.483388256587726
```





2.2.4 [10pt] After you are finished, use your final parameters to plot the linear fit. The result should look something like on the figure below. Use the `predict()` function.

```
plt.scatter(X, y, marker='x', color='r', alpha=0.5)
x_start, x_end = 5, 25
plt.xlim(x_start, x_end)
X_test = np.array([[x_start], [x_end]])
y_test = predict(X_test, theta_est)
plt.plot(X_test, y_test)
plt.xlabel('Population in 10\'000 s')
plt.ylabel('Profit in 10\'000$ s')
plt.show()
```



Now use your final values for θ and the `predict()` function to make predictions on profits in areas of 35,000 and 70,000 people.

```
print(predict(np.array([[35000], [70000]]), theta_est))
```

```
[[40819.05197031]
 [81641.73423205]]
```

To understand the cost function better, you will now plot the cost over a 2-dimensional grid of values. You will not need to code anything new for this part, but you should understand how the code you have written already is creating these images.

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.cm as cm
limits = [(-10, 10), (-1, 4)]
space = [np.linspace(*limit, 100) for limit in limits]
theta_1_grid, theta_2_grid = np.meshgrid(*space)
theta_meshgrid = np.vstack([theta_1_grid.ravel(), theta_2_grid.ravel()])
loss_test_vals_flat = (((add_column(X) @ theta_meshgrid - y)**2).mean(axis=0)/2)
loss_test_vals_grid = loss_test_vals_flat.reshape(theta_1_grid.shape)
print(theta_1_grid.shape, theta_2_grid.shape, loss_test_vals_grid.shape)

plt.gca(projection='3d').plot_surface(theta_1_grid, theta_2_grid,
                                      loss_test_vals_grid, cmap=cm.viridis,
                                      linewidth=0, antialiased=False)
xs, ys = np.hstack(theta_values).tolist()
zs = np.array(loss_values)
plt.gca(projection='3d').plot(xs, ys, zs, c='r')
plt.xlim(*limits[0])
plt.ylim(*limits[1])
plt.show()

plt.contour(theta_1_grid, theta_2_grid, loss_test_vals_grid, levels=np.logspace(-2, 3, 20))
plt.plot(xs, ys)
plt.scatter(xs, ys, alpha=0.005)
plt.xlim(*limits[0])
plt.ylim(*limits[1])
plt.show()
```

```
(100, 100) (100, 100) (100, 100)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [60], line 11
      8 loss_test_vals_grid = loss_test_vals_flat.reshape(theta_1_grid.shape)
      9 print(theta_1_grid.shape, theta_2_grid.shape, loss_test_vals_grid.shape)
---> 11 plt.gca(projection='3d').plot_surface(theta_1_grid, theta_2_grid,
     12                                       loss_test_vals_grid, cmap=cm.viridis,
     13                                       linewidth=0, antialiased=False)
     14 xs, ys = np.hstack(theta_values).tolist()
     15 zs = np.array(loss_values)

TypeError: gca() got an unexpected keyword argument 'projection'
```

3. Linear regression with multiple input features

3.1 [20pt] Copy-paste your `add_column`, `predict`, `loss` and `loss grad` implementations from above and modify your code of linear regression with one variable to support any number of input features (vectorize your code.)

```
data = np.loadtxt('ex1data2.txt', delimiter=',')
X, y = data[:, :-1], data[:, -1, np.newaxis]
n = data.shape[0]
print(X.shape, y.shape, n)
print(X[:10], '\n', y[:10])
```

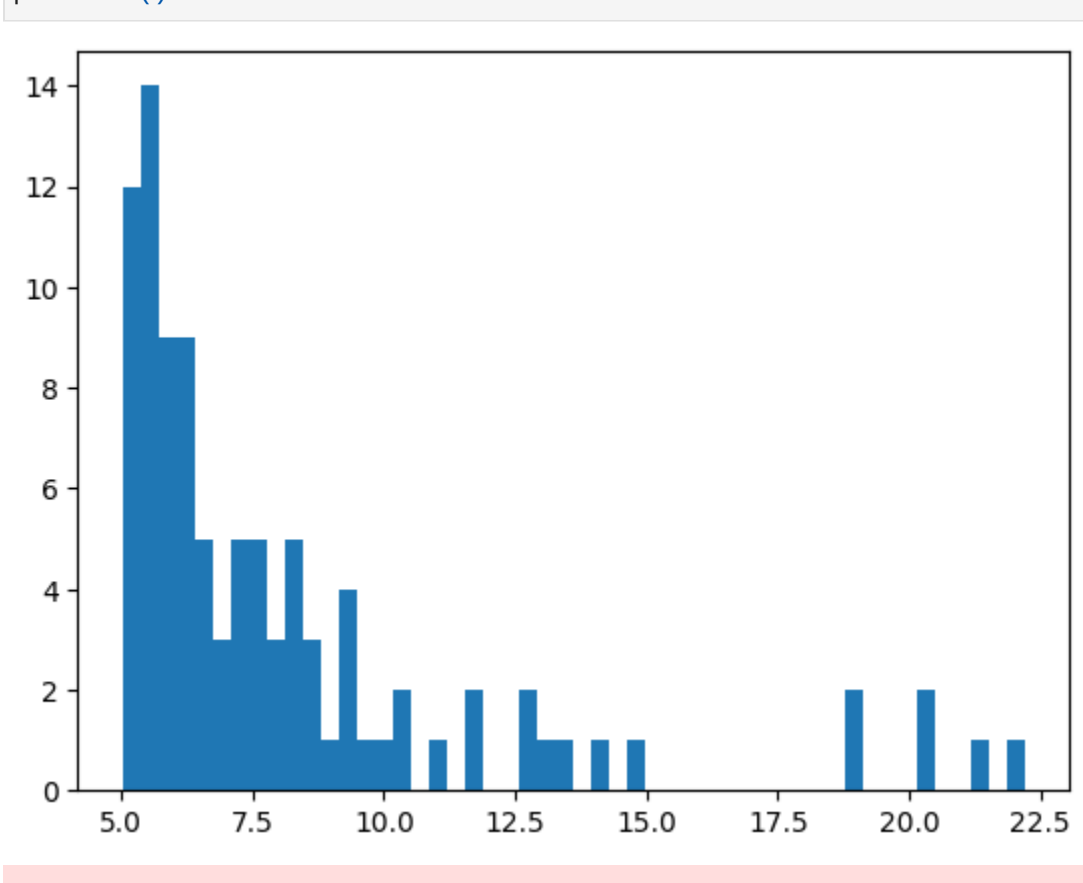
```
(47, 2) (47, 1) 47
[[  2.10400000e+03   3.00000000e+00]
 [  1.60000000e+03   3.00000000e+00]
 [  2.40000000e+03   3.00000000e+00]
 [  1.41600000e+03   2.00000000e+00]
 [  3.00000000e+03   4.00000000e+00]
 [  1.98500000e+03   4.00000000e+00]
 [  1.53400000e+03   3.00000000e+00]
 [  1.42700000e+03   3.00000000e+00]
 [  1.38000000e+03   3.00000000e+00]
 [  1.49400000e+03   3.00000000e+00]]
 [[ 399900.]
 [ 329900.]
 [ 369000.]
 [ 232000.]
 [ 539900.]
 [ 299900.]
 [ 314900.]
 [ 198999.]
 [ 212000.]
 [ 242500.]]
```

```
def add_column(X):
    assert len(X.shape) == 2 and X.shape[1] == 1
    return np.insert(X, 0, 1, axis=1)

def predict(X, theta):
    assert len(X.shape) == 2 and X.shape[1] == 1
    assert theta.shape == (2, 1)

    X_prime = add_column(X)
    pred = X_prime @ theta
    return pred

def loss(X, y, theta):
    assert X.shape == (n, 1)
    assert y.shape == (n, 1)
    assert theta.shape == (2, 1)

    X_prime = add_column(X)
    assert X_prime.shape == (n, 2)
    loss = ((predict(X, theta) - y)**2).mean()/2
    return loss

def loss_gradient(X, y, theta):
    X_prime = add_column(X)
    loss_grad = ((predict(X, theta) - y)*X_prime).mean(axis=0)[:, np.newaxis]
    return loss_grad


theta_init = np.zeros((3, 1))
result = run_gd(loss, loss_gradient, X, y, theta_init, n_iter=10000, lr=1e-10)
theta_est, loss_values, theta_values = result
plt.plot(loss_values)
plt.show()

# raise NotImplementedError("Put your multivariate regression code here")
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In [61], line 30
     26     return loss_grad
     29 theta_init = np.zeros((3, 1))
---> 30 result = run_gd(loss, loss_gradient, X, y, theta_init, n_iter=10000, lr=1e-10)
     31 theta_est, loss_values, theta_values = result
     32 plt.plot(loss_values)

Cell In [57], line 38, in run_gd(loss, loss_gradient, X, y, theta_init, lr, n_iter)
     35 theta_values = []
     37 for i in range(n_iter):
---> 38     loss_value = loss(X, y, theta_current)
     39     theta_current -= lr*loss_gradient(X, y, theta_current)
     40     loss_values.append(loss_value)

Cell In [61], line 16, in loss(X, y, theta)
     14 assert X.shape == (n, 1)
     15 assert y.shape == (n, 1)
---> 16 assert theta.shape == (2, 1)
     18 X_prime = add_column(X)
     19 assert X_prime.shape == (n, 2)

AssertionError:
```

3.2 [20pt] Draw a histogam of values for the first and second feature. Why is feature normalization important? Normalize features and re-run the gradient decent. Compare loss plots that you get with and without feature normalization.

```
plt.hist(X[:, 0], 50)
plt.show()

plt.hist(X[:, 1], 50)
plt.show()
```



```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [69], line 4
      1 plt.hist(X[:, 0], 50)
      2 plt.show()
----> 4 plt.hist(X[:, 1], 50)
      5 plt.show()

IndexError: index 1 is out of bounds for axis 1 with size 1
```

```
theta_init = np.zeros((3, 1))
X_normed = np.zeros_like(X)
X_normed[:, 0] = X[:, 0] / X[:, 0].max()
X_normed[:, 1] = X[:, 1] / X[:, 1].max()
result = run_gd(loss, loss_gradient, X_normed, y, theta_init, n_iter=10000, lr=1e-3)
theta_est, loss_values, theta_values = result

plt.plot(loss_values)
plt.show()
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [67], line 4
      2 X_normed = np.zeros_like(X)
      3 X_normed[:, 0] = X[:, 0] / X[:, 0].max()
----> 4 X_normed[:, 1] = X[:, 1] / X[:, 1].max()
      5 result = run_gd(loss, loss_gradient, X_normed, y, theta_init, n_iter=10000, lr=1e-3)
      6 theta_est, loss_values, theta_values = result

IndexError: index 1 is out of bounds for axis 1 with size 1
```

3.3 [10pt] How can we choose an appropriate learning rate? See what will happen if the learning rate is too small or too large for normalized and not normalized cases?

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [83], line 1
----> 1 plt(loss_gradient(X, y, theta_est)[0])

TypeError: 'module' object is not callable
```

4. Written part

These problems are extremely important preparation for the exam. Submit solutions to each problem by filling the markdown cells below.

4.1 [10 pt] Maximum Likelihood Estimate for Coin Toss

The probability distribution of a single binary variable that takes value with probability is given by the Bernoulli distribution

$$\mathrm{Bern}(x|\mu) = \mu^x(1-\mu)^{1-x}$$

For example, we can use it to model the probability of seeing 'heads' ($x = 1$) or 'tails' ($x = 0$) after tossing a coin, with μ being the probability of seeing 'heads'. Suppose we have a dataset of independent coin flips $D = \{x^{(1)}, \ldots, x^{(m)}\}$ and we would like to estimate μ using Maximum Likelihood. Recall that we can write down the likelihood function as

$$\mathcal{L}(x^{(i)}|\mu) = \mu^{x^{(i)}}(1-\mu)^{1-x^{(i)}}$$

$$P(D|\mu) = \prod_i \mathcal{L}(x^{(i)}|\mu)$$

The log of the likelihood function is

$$\ln P(D|\mu) = \sum_i x^{(i)}\ln\mu + (1-x^{(i)})\ln(1-\mu)$$

Show that the ML solution for μ is given by $\mu_{ML} = \frac{h}{m}$ where h is the total number of 'heads' in the dataset. Show all of your steps.

Take the derivative of

$$\ln P(D|\mu)$$

, we derive

$$\frac{d}{d\mu}\ln P(D|\mu) = \sum_{i=1}^{m} x^{(i)}\ln\mu + (1-x^{(i)})\ln(1-\mu) = 0$$

, then simplify it to

$$\sum_{i=1}^{m} x^{(i)}\frac{1}{\mu} + (x^{(i)}-1)\frac{1}{ln(1-\mu)} = 0$$

, therefore we get

$$\sum_{i=1}^{m} x^{(i)}\frac{1}{\mu} = \sum_{i=1}^{m}(1-x^{(i)})\frac{1}{ln(1-\mu)}.$$

By substitution,

$$\frac{h}{\mu} = \frac{m-h}{1-\mu}.$$

In conclusion,

$$\mu_{ML} = \frac{h}{m}.$$

4.2 [10 pt] Localized linear regression

Suppose we want to estimate localized linear regression by weighting the contribution of the data points by their distance to the query point x_q, i.e. using the cost

$$E(x_q) = \frac{1}{2}\sum_i^m \frac{(y^{(i)} - h(x^{(i)}|\theta))^2}{||x^{(i)} - x_q||^2}$$

where $\frac{1}{||x^{(i)} - x_q||} = w^{(i)}$ is the inverse Euclidean distance between the training point $x^{(i)}$ and query (test) point x_q.

Derive the modified normal equations for the above cost function $E(x_q)$. Hint: first, re-write the cost function in matrix/vector notation, using a diagonal matrix to represent the weights $w^{(i)}$.

Matrix/vector notation:

$$E(x_q) = \frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \theta^{\boldsymbol{T}}x^{(i)})^2(w^{(i)})^2.$$

Then simplify it:

$$E(x_q) = \frac{1}{2}\sum_{i=1}^{m}(w^{(i)}y^{(i)} - w^{(i)}\theta^{\boldsymbol{T}}x^{(i)})^2 = (\boldsymbol{WY} - \boldsymbol{WX\theta})^2.$$

4.3 [10 pt] Betting on Trick Coins

A game is played with three coins in a jar: one is a normal coin, one has "heads" on both sides, one has "tails" on both sides. All coins are "fair", i.e. have equal probability of landing on either side. Suppose one coin is picked randomly from the jar and tossed, and lands with "heads" on top. What is the probability that the bottom side is also "heads"? Show all your steps.

$$p(bottom-head|top-head) = \frac{p(bottom-head-and-top-head)}{p(top-head)}$$

$$= \frac{p(bottom-head-and-top-head)}{p(top-head|head-head)p(head-head) + p(top-head|tail-head)p(tail-head) + p(top-head|tail-tail)p(tail-tail)}$$

$$= \frac{\frac{1}{3}}{\frac{1}{3}*1 + \frac{1}{3}*0 + \frac{1}{3}*\frac{1}{2}}$$

$$= \frac{2}{3}$$