Project 1

For this project you'll have a partner. Those have been automatically assigned and you can find your partner's information on the assignment link. **You should not discuss the project with anyone outside your group. If you are stuck please contact Dr. Post or the TA, Khuzaima Hameed** If you run into issues with your partner, let me know immediately. I can't help if it is the day before the due date and you tell me you haven't heard from them. You'll be stuck doing it yourself then!

For this project you will create a python notebook (`.ipynb` file) and output a `.html` file. Both of these files should then be uploaded to wolfware in the assignment link! **Both partners should submit the files. When submitting, please leave a note about how working with your partner went and if you think they gave equal effort or not.**

This project is meant to assess your ability to program in python. The result should be a report with a narrative throughout, section headings, graphs outputted in appropriate places, etc. To be clear **be sure to include markdown text describing what you are doing, even when not explicitly asked for!** The audience you are writing for is someone that understands programming and very basic statistics, but would need you to provide details and explanation of what you are doing to understand it.

Overall Goal

We will read in a data set, do some basic exploratory data analysis (EDA - numeric summaries and graphs), and then implement some algorithms on the data from scratch. It should be fun :)

Data set

We'll be using a data set containing information about motorcycle sales available at the assignment link in Moodle. The original data can be found on kaggle here. The variables in the data set are:

- `name` (of motorcycle)
- `selling_price`
- `year` (of motorcycle, not sale)
- `seller_type`
- `owner` (# of owners)
- `km_driven`
- `ex_showroom_price`

Report Components: Introduction, EDA, Prediction of `selling_price` (using two methods, each done with two algorithms)

Introduction

First, you should have a small section that introduces the purpose of the report and the data set you'll be working with. You should read in the data and display a small snippet of it. You should also describe the packages (modules) you'll use

EDA

Next, you should have a section where you explore the data. The variable we are most interested is the `selling_price` variable (often called our **response variable**). You'll want to focus your EDA on this variable but feel free to display relationships between other variables as well.

Ideally, I'd leave this part off and let you explore the data (as that is a skill to develop). However, to avoid the question of "is this enough?" I'll provide some basic guidance here:

- You should have numeric summaries of the `selling_price` variable (at different levels/combinations of other variables)
- You should have plots of the `selling_price` variable (again showing relationships with other variables)

- You should create at least one two-way contingency table with corresponding side-by-side/stacked bar plot visual.

Any EDA elements you have should be discussed. Why did you report this? What information does it convey?

Prediction of `selling_price`

We want to be able to predict a value of `selling_price` for a new motorcycle. How to do we figure out what a good prediction would be? We need to quantify the *quality* of the prediction. This is usually done through a **loss function**.

Loss function Suppose there is some prediction (call it c). How do we measure the quality of c? We want something 'close' to all the known values of `selling_price` from our data set. A common metric to use is **squared error loss**.

The squared error loss for a data point (call it y_1) and a prediction (c) is

$$L(y_1, c) = (y_1 - c)^2$$

For a given set of data ($y_1, \ldots, y_n$ - our `selling_price` values), we could consider the mean squared error:

$$\text{Root Mean Square Error} = RMSE(c) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} L(y_i, c)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - c)^2}$$

Ok, we can quantify the quality based off of this equation. We now want to choose the value of c that minimizes this equation (smaller loss means better prediction - closer to all the points). We could use calculus to find the optimal value, but maybe you haven't studied calculus or it is rusty. Instead, we can use numerical methods!

1. Grid search (brute force search essentially)
2. Gradient descent (based on calculus ideas)

We'll implement these two algorithms for two equations representing c:

- One where we don't consider any data other than the $y's$. That is, c is going to just be a constant that minimizes the function. (Note: the calculus based answer for this comes out to be the sample mean, $\bar{y}$)
- One where we consider a linear equation of one other numeric variable. That is, we have a different prediction for each observation (i) given by $c_i = b_0 + b_1 x_i$. Here x_i is one of the other numeric variables from our data set. We'll use the `km_driven` variable. (Note: the calculus based answer for this comes out to be the usual **simple linear regression** estimates, which you can find using `scipy.stats` as done in the notes!)

Using a Grid Search Algorithm

You'll be implementing a grid search to find the optimal value of c based off of our data set.

Just y: Pseudo code for using just the $y's$ and no other variables:

1. Create a *grid* of values for c. Use your EDA to determine reasonable values to consider for c.
2. Create a squared error loss function that takes in y and c that outputs $(y - c)^2$.
3. Create a root mean squared error (objective) function that takes in y and c that outputs $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - c)^2}$ (You can put step 2 and 3 into one function if you want.)
4. Loop over the grid of c values, finding the RMSE for each value of c.
5. Determine which value of c gives the optimal (smallest) RMSE.
6. Report that as the prediction!

- Run your algorithm on the `selling_price` variable and determine the **optimal** prediction.
- Just to make sure your algorithm generalizes, run it using the `km_driven` variable as the *response* (this shouldn't involve any new functions, just new function calls!)

Using y and another numeric variable x: Next, you'll implement the grid search to find the optimal pair of values for b_0 and b_1 using `km_driven` as your x variable and `selling_price` as your y variable. The pseudo code is very similar to that above, but your grid now has two-dimensions! You'll need to populate a grid of b_0 and b_1 values that you want to consider. Create your grid using integers between the range 60000 to 70000 for b_0 and with values of -1 to 1 by 0.01 for b_1.

Use your "best" combination of b_0 and b_1 to predict the `selling_price` for `km_driven` of 10000, 25000, and 35000.

Using a Gradient Descent Algorithm

We can implement our own Gradient Descent Algorithm in python. I'll go through the ideas here first (if you already know the ideas, feel free to skim).

Aside: Slope of the tangent line Consider a function you want to minimize. We could look at the average rate of change of the function between two points (call the function f and the two points x and $x+\Delta$). For example, for the basic parabola below the average rate of change between $x = 0.75$ and $x+\Delta = 0.75+0.25$ is

$$\text{Avg rate of change} = \frac{f(x+\Delta) - f(x)}{((x+\Delta) - x)} = \frac{f(x+\Delta) - f(x)}{\Delta}$$

Plugging in $f(x) = x^2$ with $x = 0.75$ and $\Delta = 0.25$ we get

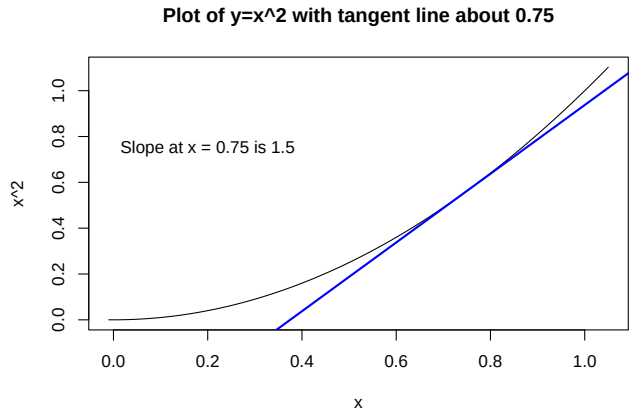
$$\frac{(0.75+0.25)^2 - 0.75^2}{0.25} = 1.75$$

The **slope** of the red line (0.25) represents the average rate of change of $f(x) = x^2$ from 0.75 to 1.

What happens as the difference in x values goes to zero (i.e. Δ goes to zero)?

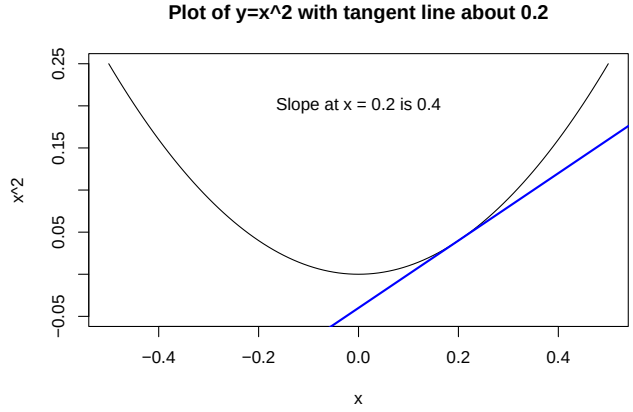
$$f'(x) = lim_{\Delta \to 0} \frac{f(x+\Delta) - f(x)}{(x+\Delta) - x} = lim_{\Delta \to 0} \frac{f(x+\Delta) - f(x)}{\Delta}$$

- Gives the instantaneous Rate of Change at the point x! Also, called the **derivative**.
- The derivative gives the *slope* of the *tangent* line at x (a line that just *touches* the graph at the point x)

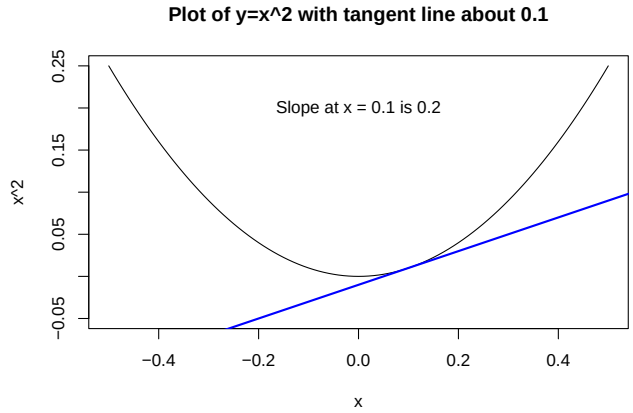


How can we use the derivative for maximization/minimization?

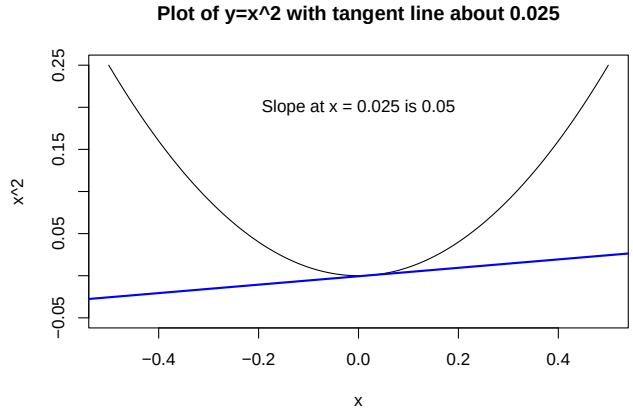
- x^2 has a minimum at $x = 0$
- Given a starting value (say 0.25), we can look at the slope of the tangent line (the derivative) and determine a direction to move.



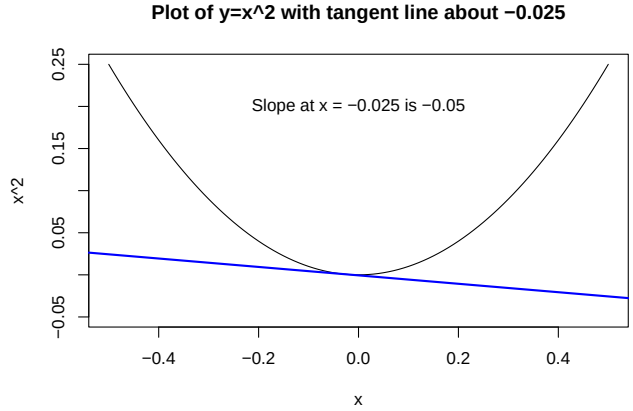
The slope is positive, so we should try a new value that is smaller than 0.25. Perhaps 0.1:



Slope is still positive, so we should try something smaller...



Slope is still positive. We move to the left again, but perhaps we step too far to the left and check x = -0.025.



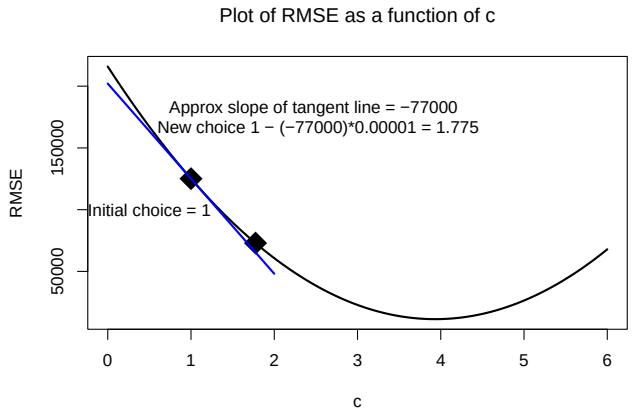
Slope is now negative so we'd try something larger! Ideally, we'd find the point where the derivative is 0 to find a min!

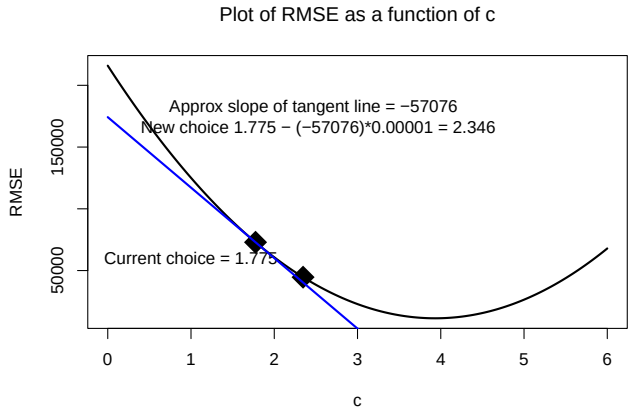
Key: step direction can be determined by the derivative (slope of the tangent line). We should move in the negative direction of the slope to find our minimum.

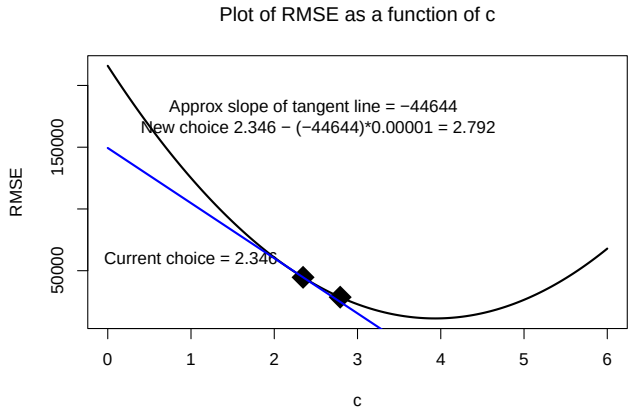
Just y: Ok, now we are ready to write out the pseudo-code for our gradient descent function! First the case where we only consider the $y's$ and no $x's$. Basic idea:

- Start with an initial guess
- Update the guess based on an approximation to the derivative (slope) of the RMSE and a "step-size" (generally some small value) `new_value = old_value - (slope) * step_size`
- Check if the movement (`|new_value - old_value|`) exceeds some small numeric tolerance. If so, we are still moving quite a bit and should repeat the steps above. If not, we've possibly settled very near a minimum and can stop. We should also set a maximum number of iterations so we don't repeat forever!

Visuals corresponding to a similar problem are given here to help:







And again, we'd continue until the absolute value of the difference is very small.

Gradient Descent Pseudo-code:

1. Create a squared error loss function that takes in y and c and outputs $(y-c)^2$.
2. Create a root mean squared error (objective) function that takes in y and c and outputs $RMSE(c) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y-c)^2}$
3. Create a difference quotient function to approximate the slope of the tangent line that takes in the guess c, a small change to c - I'll call it Δ, and y that outputs

$$diff\_quotient = \frac{RMSE(c+\Delta) - RMSE(c)}{\Delta}$$

4. Pick a starting value for `cur_c`.
5. Evaluate the difference quotient at `cur_c`.
6. Update the value by moving (a small step) in the negative direction of the difference quotient
   `new_c = cur_c - diff_quotient * step_size`
7. Check if `abs(new_c - cur_c) < num_tol` where `num_tol` is a small value.
   - If so, update the `cur_c` to be the `new_c` value and stop.
   - If not, update the value of `cur_c` to `new_c` and repeat steps 5-7.
     - Put in a safety that stops the loop after a maximum number of iterations is reached (even if `abs(new_c - cur_c) < num_tol` is not met)
8. Use the last value as the prediction!

- Run your algorithm on the `selling_price` variable and determine the **optimal** prediction.
- Just to make sure your algorithm generalizes, run it using the `km_driven` variable as the *response* (this shouldn't involve any new functions, just new function calls!)

Using y and another numeric variable x: Next, you'll implement the gradient descent idea to find the optimal value of b_0 and b_1 using `km_driven` as your x variable and `selling_price` as your y variable. The pseudo code is very similar to that above, but you have two variables to optimize over (b_0 and b_1) instead of one (c).

Gradient Descent Pseudo-code:

1. Create a squared error loss function that takes in y, x, b_0, and b_1 that outputs $(y-b_0-b_1x)^2$
2. Create a mean squared error (objective) function that takes in the same values that outputs $RMSE(b_0,b_1) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y-b_0-b_1x)^2}$
3. Create a difference quotient function for b_0 to approximate the slope of the tangent line in the direction of b_0. It should take in y, x, b_0, a small change to b_0 - I'll call it Δ_0, and b_1 that outputs

$$diff\_quotient\_b0 = \frac{RMSE(b_0+\Delta_0, b_1) - RMSE(b_0, b_1)}{\Delta_0}$$

4. Create a difference quotient function for b_1 to approximate the slope of the tangent line in the direction of b_1. It should take in y, x, b_0, b_1 and a small change to b_1 - I'll call it Δ_1 that outputs

$$diff\_quotient\_b1 = \frac{RMSE(b_0, b_1+\Delta_1) - RMSE(b_0, b_1)}{\Delta_1}$$

5. Pick starting values for `b_0` and `b_1` (say `cur_b0` and `cur_b1`).
6. Evaluate the difference quotient for `b_0` at the `cur_b0` and `cur_b1` values.
7. Update the value of `b_0` by moving (a small step) in the negative direction of the difference quotient
   `new_b0 = cur_b0 - diff_quotient_b0 * step_size_b0`
8. Evaluate the difference quotient for `b_1` at the `new_b0` and `cur_b1` values.
9. Update the value of `b_1` by moving (a small step) in the negative direction of the difference quotient
   `new_b1 = cur_b1 - diff_quotient_b1 * step_size_b1`
10. Check if the distance between the `cur_b0`, `cur_b1` vector to the `new_b0`, `new_b1` vector is less than some small tolerance. (Hint: Look up the euclidean distance between two vectors, there is a function in numpy to calculate it - or just code it up yourself).

- If so, update the `cur_b0` and `cur_b1` to be the `new_b0` and `new_b1` values and stop.
- If not, update the `cur_b0` and `cur_b1` to be the `new_b0` and `new_b1` values and repeat steps 6-10.
  - Put in a safety that stops the loop after a maximum number of iterations is reached (even if the tolerance is not met)

11. Use the last values as the estimates for `b_0` and `b_1`!

Use your "best" combination of b_0 and b_1 to predict the `selling_price` for `km_driven` of 10000, 25000, and 35000.

Note: For gradient descent in the constant case I used

- delta = 0.005
- step size of 0.5
- with a starting value of 50000

For the SLR case I used:

- delta0 and delta1 = 0.005
- step size for b0 = 1
- step size for b1 = 0.000005
- with starting values of 60000 and 0, respectively

Good luck! Have fun!