COSC 130 – HW 06 Instructions

General Instructions

Create a new script named **`HW_06_YourLastName.py`** and a new notebook named **`HW_06_YourLastName.ipynb`**. You will use the script to define three new classes, and you will use the notebook to apply these classes to various test cases.

Instructions for the Script

You will be asked to create three classes named **`LinReg`**, **`ATM`**, and **`Student`**. Descriptions of these classes are provided below.

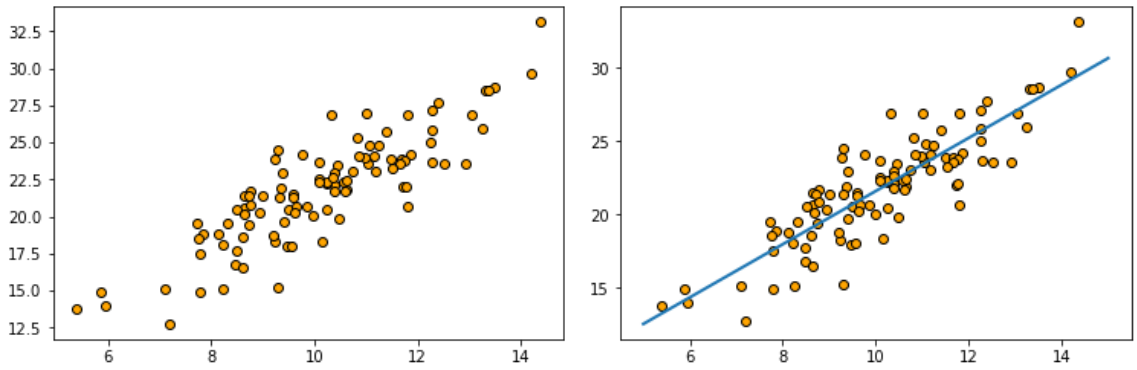
Script, Part 1: The LinReg Class

The purpose of this class is to implement a linear regression algorithm. Instances of the class are intended to represent linear regression models. We will start by providing some background into linear regression.

Assume that we have the following information:

- We have n values of a variable x denoted by $x_1, x_2, \dots, x_n$.
- We have n values of a variable y denoted by $y_1, y_2, \dots, y_n$.
- The values of x and y are paired together into points of the following form: (x_i, y_i).

If we plot all of these points, we might get a scatter plot that looks something like this:



If the points seem to fall near a line, as is the case in the plot above, then we might try to use such a line to explain the relationship between the variables x and y. In fact, we could use this line to try to predict a value of y for a given value of x. We will use the symbol $\hat{y}$ to denote a predicted value of y.

A **linear model** is an equation of the form $\hat{y} = b_0 + b_1 x$. Assuming we know the values of b_0 and b_1, we can plug in a value x and get back out a predicted value of y, denoted as $\hat{y}$. Any values of the coefficients b_0 and b_1 will give us a linear model. The goal of **linear regression** is to find the model that provides the best fit for our data. There are multiple ways of deciding which fitted model is the "best". We won't discuss these various methods in this lab, but using the most common metric for scoring the models, we obtain the following formulas for calculating b_0 and b_1:

1. Let $\bar{x}$ be the mean (average) of the x-values and let $\bar{y}$ be the mean (average) of the y-values.
2. Let $Sxx = \sum(x_i - \bar{x})^2$ and let $Sxy = \sum(x_i - \bar{x})(y_i - \bar{y})$.
3. Then $b_1 = \frac{Sxy}{Sxx}$ and $b_0 = \bar{y} - b_1\bar{x}$.

Your **LinReg** class should contain the following four methods:

- **__init__()** – This is the constructor. This method will accept a list of x-values and a list of y-values, and will calculate the slope and intercept of the regression line, which will be stored as attributes.
- **display_model()** – This method will print out the slope and intercept of the regression line.
- **predict()** – This method accepts a collection of x-values as a parameter, and will return a collection of predicted y-values.
- **score()** – This method accepts as parameters a collect of x-values and a collection of y-values. It calculates a score for how well the model describes the data provided. The particular score calculated is called the r-square value.

These four methods are explained in more detail below.

__init__()

The constructor should accept three parameters: **self**, **x**, and **y**.

- **x** is a list or array of x-values for the points used to create the linear regression model.
- **y** is a list or array of y-values for the points used to create the linear regression model.

The constructor will calculate values for two attributes, **self.b0** and **self.b1**, which are the intercept and slope of the linear regression model, respectively. Your constructor should perform the following steps:

1. Convert **x** and **y** to arrays (in case they were provided as lists), and store them in attributes called **self.x** and **self.y**.
2. Use loops to calculate the mean of the x-values and mean of the y-values.
3. Use loops to calculate Sxx and Sxy.
4. Calculate b_1 and b_0, storing the results in attributes.

display_model()

This method accepts a single parameter named **self** and should display the slope and intercept of the regression line in the following format:

```
Intercept: xxxx
Slope:     xxxx
```

predict()

This method accepts two parameters, **self** and **x**, and returns an array of predicted y-values for the supplied x-values. The method should first convert **x** to an array (in case it was provided as a list). The predictions are calculated by multiplying the x-values by the slope **b1** and then adding the intercept **b0**. This calculation can be performed in a single line of code using numpy. The method should return the array of predicted y-values.

score()

This method should accept three parameters: **self**, **x**, and **y**.

- **x** is a list or array of x-values for a collection of points used to score the linear regression model.
- **y** is a list or array of y-values for a collection of points used to score the linear regression model.

The method should calculate an r-squared value for the set of points, as follows:

1. Use the **predict()** method and **x** to generate predicted values of y.
2. Create an array called **errors** by subtracting the predicted y-values from the actual y-values that were supplied to the method. Mathematically, the formula for the error terms is given by $e_i = y_i - \hat{y}_i$.
3. Calculate a variable called **SSE** (sum of squared errors) by squared each element of **errors** and then summing the results. In mathematical notation, we have $SSE = \sum e_i^2$.
4. Calculate a variable called **SST** by subtracting the mean of the supplied y-values from the y-values themselves, squaring the resulting differences, and then summing the squared differences. The mathematical formula for **SST** is: $SST = \sum (y_i - \bar{y})^2$.
5. Finally, r-squared is calculated as follows: **r2 = 1 - SSE / SST**. This value should be returned.

Script, Part 2: The ATM Class

In this section you will create a class named **ATM**. Instances of this classes are intended to represent individual automatic teller machines.

Your **ATM** class should contain four methods:

- **`__init__()`** – This is the constructor. It will be used to set the name of the owner of the account, and the balance of the account.
- **`check_balance()`** – This method will print a message stating the current balance of the account.
- **`deposit()`** – This method can be used to process a deposit into the account.
- **`withdrawal()`** – This method can be used to process a withdrawal from the account.

These four methods are explained in more detail below.

`__init__()`

The constructor should accept four parameters: **`self`**, **`first`**, **`last`**, and **`balance`**. The parameters **`first`** and **`last`** are expected to be strings containing the name of the person owning the account. The parameter **`balance`** is the initial balance of the account. The constructor should simply store the parameter values as attributes with the same names.

`__check_balance__()`

This method should accept only a single parameter named **`self`**. It should print the following message, with the bracketed expressions replaced with their appropriate values:

```
Hello, [FirstName]. Your current balance is: $[CurrentBalance]
```

`__deposit__()`

This method should accept two parameters: **`self`** and **`amount`**. The method should increase the current balance of the account by **`amount`** and print the message shown below, with the bracketed expression replaced with the appropriate value:

```
Deposit of $[Amount] complete.
```

`__withdrawal__()`

This method should accept two parameters: **`self`** and **`amount`**. If the current balance of the account is less than **`amount`**, then the method should do nothing other than print the following message, with the bracketed expression replaced with the appropriate value:

```
Withdrawal of $[Amount] failed. Insufficient funds.
```

Otherwise, the method should decrease the balance by amount, and should print the following message, with the bracketed expression replaced with the appropriate value:

```
Withdrawal of $[Amount] complete.
```

Script, Part 3: The Student Class

The **Student** class will be used to contain personal and grade information for a student at a university. Each instance of the class will represent a single student.

Your **`Student`** class should contain three methods:

- **`__init__()`** – This is the constructor. It will be used to set the name and student id of the student.
- **`add_grades()`** – This method will add a list of grades to the student's record.
- **`gpa()`** – This method will calculate the student's current GPA.

These three methods are explained in more detail below.

`__init__()`

The constructor should accept four parameters: **`self`**, **`first`**, **`last`**, and **`sid`**. The parameters **`first`** and **`last`** store the name of a student. The parameter **`sid`** stores the student's student ID number. The constructor should store these parameter values as attributes with the same names. It should also create two additional attributes named **`self.credits`** and **`self.grades`**. These two attributes should initially be set to be empty lists.

`__add_grades__()`

This method should accept three parameters: **`self`**, **`credits`**, and **`grades`**. The **`credits`** parameter is expected to be a list of credit hours for a set of courses taken by the student, and the **`grades`** parameter is expected to be a list of letter grades earned in those same courses. The method should store the contends of these lists in the attribute lists **`self.credits`** and **`self.grades`**.

`__gpa__()`

This method should accept only a single parameter named **`self`**. The method should calculate and return the student's current GPA based on the information stored in **`self.credits`** and **`self.grades`**.

Instructions for the Notebook

You will now create a notebook to test the classes you have created. Make sure that your script and notebook are contained in the same directory.

Any set of instructions you see in this rest of this document with an orange bar to the left will indicate a place where you should create a markdown cell. For each new problem, create a markdown cell that indicates the title of that problem as a level 2 header.

Any set of instructions you see with a blue bar to the left will provide instructions for creating a single code cell.

Assignment Header

Create a markdown cell with a level 1 header that reads: "COSC 130 - Homework 06". Add your name as a level 3 header

We will start by running the script.

Use the Magic command **`%run`** to run the contents of your script.

Problem 1: Testing the LinReg Class

In this problem, you will test your **LinReg** class. We will start by generating some data to use.

In the cell below, perform the following steps:

1. Copy the code below to create two lists **`x`** and **`y`**.

```
x = [8.3, 14.4, 0.0, 6.0, 2.9, 1.8, 3.7, 6.9, 7.9, 10.8, 8.4, 13.7, 4.1, 17.6, 0.5, 13.4,
     8.3, 11.2, 2.8, 4.0, 16.0, 19.4, 6.3, 13.8, 17.5, 17.9, 1.7, 0.8, 3.4, 17.6, 2.0, 8.4,
     19.2, 10.7, 13.8, 6.3, 13.7, 16.7, 0.4, 15.0, 19.8, 15.0, 5.6, 15.8, 2.1, 9.0, 18.2,
     5.9, 5.8, 2.6, 0.4, 13.6, 4.2, 5.3, 9.8, 1.1, 11.5, 2.9, 11.8, 14.0, 2.0, 8.3, 13.9,
     8.3, 1.0, 10.7, 13.3, 10.3, 18.9, 11.7, 18.1, 2.7, 2.8, 16.1, 8.0, 3.3, 18.6, 7.0,
     15.0, 14.5]

y = [95.5, 79.7, 113.9, 120.6, 121.9, 113.5, 118.2, 95.8, 110.8, 105.1, 125.8, 46.1, 85.7,
     44.2, 126.3, 83.7, 96.0, 46.7, 109.3, 122.6, 62.4, 56.9, 96.1, 64.6, 55.4, 57.4, 122.0,
     124.6, 101.0, 58.3, 119.7, 108.7, 64.7, 84.2, 61.9, 89.3, 75.1, 57.1, 119.0, 63.7,
     33.0, 74.3, 95.3, 79.3, 123.8, 97.9, 32.2, 104.0, 113.7, 99.8, 120.1, 69.5, 86.3,
     108.7, 98.2, 107.8, 83.4, 92.7, 75.9, 41.4, 135.3, 97.8, 67.3, 78.8, 142.2, 112.5,
     40.5, 102.5, 72.7, 82.1, 30.9, 128.3, 111.4, 48.5, 72.9, 120.9, 60.8, 86.8, 71.3, 46.7]
```

2. Display a scatter plot of the 80 points determined by x and y. Set the point color to black, the fill color to a color of your choice, the point size to 40, and the alpha level of 0.6. Set the limits of the x-axis to be -2 to 22, and the limits of the y-axis to be 0 to 150.

We will now create the model.

Use the arrays x and y to create an instance of the **`LinReg`** class named **`model`**. Then called the **`display_model()`** method of your model. Also call the **`score()`** method of the model, printing the result.

We will now score the model on a new set of data.

Create the following lists:

```
x_test = [2, 7, 9, 13, 16, 18]
y_test = [101, 104, 74, 67, 71, 43]
```

Call the model's predict() on x_test, printing the results.

We will now add a plot of the regression line to the scatter plot we displayed. We will start by using the **`predict()`** method to determine the end points of the line.

Create a list called **`x_line`** with two x-values: -2 and 22. Use the **`predict()`** method of the model object to find the corresponding y-values. Call the results **`y_line`**. Print the y-values.

We will now plot the scatter plot along with the regression line.

Recreate the scatter plot from above. Add a line to the scatter plot using the values in **`x_line`** and **`y_line`**. Set **`lw=2`** for the line. Select a color for the line that will stand out against the color used in the scatter plot.

Problem 2: Testing the ATM Class

In this problem, you will test your **`ATM`** class by using a loop to process a sequence of transactions.

Create an instance of the **`ATM`** class called **`my_account`**. Use your own first name and last name, and set an initial balance of $340.

Creating the following lists:

```
transaction = ['W', 'W', 'D', 'W', 'W', 'W', 'D', 'W', 'W']
amount = [150, 75, 100, 150, 120, 65, 50, 120, 37]
```

These lists record information about several ATM transactions. The list transaction records the type of the transaction. A value of **`W`** indicates a withdrawal and a value of **`D`** indicates a deposit. The list amount records the size of the transaction. The lists are paired so that **`transaction[i]`** corresponds to **`amount[i]`**.

Write a loop to process each of the transactions recorded in the two lists. Call the **`check_balance()`** method after each transaction.

Problem 3: Testing the Student Class

In this problem, you will test your **`student`** class by adding several terms of course information and calculating the GPA after each term.

A list of grades called gr and a list of credit hours called cr is provided below.

```
gr = ['B', 'C', 'B', 'A', 'B', 'B', 'B', 'D', 'B', 'A',
      'D', 'A', 'F', 'B', 'A', 'B', 'B', 'A', 'B', 'C']
cr = [3, 3, 3, 4, 4, 3, 5, 3, 3, 5, 5, 4, 3, 3, 3, 5, 4, 3, 3, 5]
```

Create an instance of the class **`Student`** named **`jd`**. This instance should be created for a student named John Doe and with a student ID of 123456.

Loop over the lists provided, and add the grades for the courses to **`jd`** one at a time using the **`add_grades()`** method. After each new grade is added, please call the **`gpa()`** method, printing the results.

Submission Instructions

When you are done, click **Kernel > Restart and Run All**. If any cell produces an error, then manually run every cell after that one, in order. Save your notebook, and then export the notebook as an HTML file. Upload the HTML file to Canvas and upload the IPYNB and PY files to CoCalc, placing them in the **Homework/HW 06** folder.