DSC x KPMG Coding Competition 2024 Questions

Question 0: Sample Question

Given a string, return the string in reverse order.

Note: You have been provided with the Solution file for this question. It is purely for you to familiarise yourself with the submission process.

For example:

```
Solution("Hello world") -> "dlrow olleH"
```

Question 1: Password Validation 1

Given a password, determine whether it is valid, based on the requirements: - It must contain at least 8 characters - It must contain at least 6 distinct characters

The uppercase and lowercase versions of a letter should be counted as two different characters.

For example:

```
Solution("Password1234") -> True
Solution("123456") -> False
Solution("Abababab") -> False
```

Question 2: Password Validation 2

Given a password, determine whether it is valid, based on the requirements: - It must contain at least 10 characters - It must contain all of the following: uppercase letters, lowercase letters, numbers

```
Solution("Password1234") -> True
Solution("NoNumbers") -> False
Solution("2Short") -> False
```

Question 3: Vectorized Numpy Operations

Packages: You will need to have numpy installed and usable for this question.

Given a numpy array (may not be one-dimensional) of floats representing temperatures in Celsius, convert the array into Fahrenheit. The output should also be a numpy array of the same shape.

The equation to convert Celsius to Fahrenheit is given by the following:

$$F = \left(C \times \frac{9}{5}\right) + 32$$

Bonus: A bonus point will be awarded if you do not loop over the array.

For example:

```
Solution(np.array([0, -32, 2.5, 5.6, -40, -1.25])) -> np.array([32, -25.6, 36.5, 42.08,
    -40, 29.75])
```

Question 4: Simple Pandas Operations

Packages: You will need to have pandas installed and usable for this question.

Given two pandas dataframes `students` and `test_results`, find the first name, last name, and test score for the three highest scoring students in descending score order with the columns renamed to `first_name`, `last_name`, `score`.

The output should be a pandas dataframe.

For example:

`students`:

id	fname	lname
1	Demarcus	Rabiot
2	Jeff	Chang
3	Jacob	Danlon
4	Joe	Flagstaff

`test_results`:

id	studentid	score
1	4	75
2	3	94
3	2	13
4	1	53

```
Solution(students, test_results) -> # the table below
```

first_name	last_name	score
Jacob	Danlon	94
Joe	Flagstaff	75
Demarcus	Rabiot	53

Question 5: Grid Arrangement

Determine the number of ways to arrange a given number of objects into different rows, with the same number of objects in each row.

For each arrangement, the number of rows cannot be larger than the number of objects in each rows.

For example:

```
Solution(24) -> 4 # one row of 24, two rows of 12, three rows of 8, four rows of 6
```

Question 6: Hours Passed

Given two strings, `start` and `end`, representing a time in 12 hour format, return the number of hours that have passed between them as an integer. If no time has passed, return `0`.

The time formats will be the hour value (1 to 12) followed by `:00`, a space, then either `AM` or `PM`.

The first time will always be the starting time and the second one will always be the ending time. In other words, the second one is after the first.

Note: Assume that the two times will be within 24 hours of each other and the times **will not** cross midnight (`start` will never be `PM` when `end` is `AM`).

```
Solution("3:00 AM", "9:00 AM") -> 6

Solution("2:00 PM", "4:00 PM") -> 2

Solution("1:00 AM", "3:00 PM") -> 14
```

Question 7: Factorial Integer

Given a positive integer, return `True` if the integer is a factorial of an integer. Otherwise, return `False`.

You will not need to handle the input, assume the input is a valid positive integer.

For example:

```
Solution(1) -> True # 1 = 1! = 0!

Solution(13) -> False

Solution(720) -> True # 720 = 1 * 2 * 3 * 4 * 5 * 6 = 6!
```

Question 8: Points in a Circle

Count the number of two-dimensional coordinates that are inside or on the edge of a given circle.

The formula for the distance between two 2d points p and q is given by the following:

$$d(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

The inputs are: - A list of 2-tuples representing a point's x and y coordinates respectively - The x value of the circle's center - The y value of the cicle's center - The radius of the circle

For example:

```
Solution(
        [(0, 0),
         (1, 1),
         (0, 5),
         (15, 0)],
        0, 0, 5
) -> 3
```

Question 9: Reverse Integer Digits

Given a signed integer `x`, return a signed integer with `x`'s digits reversed.

The input `x` must be within the range $[-2^{31}, 2^{31} - 1]$ - return 0 if it isn't. Keep the negative sign if it exists and remove any extra leading zeros.

Bonus: A bonus point will be awarded if you do not convert `x` into a different datatype.

For example:

```
Solution(123) -> 321
Solution(-4629) -> -9264
Solution(1000) -> 1
Solution(2 ** 31) -> 0
```

Question 10: Most Common Combination

Given a list of transactions, find the most common two items purchased together and return them in alphabetical order.

You can expect that there will be no ties.

For example:

```
Solution([["Bread","Eggs","Milk"], ["Cereal","Milk"], ["Bread","Milk"],
    ["Wine","Eggs"]]) -> ("Bread","Milk")
```

Question 11: Fibonacci Numbers

The Fibonacci numbers is defined recursively as follows:

$$F(0) = 0, F(1) = 1,$$

and

$$F(n) = F(n-1) + F(n-2)$$

for $n > 1$.

From this, starting from $F(0)$ we have the sequence $0, 1, 1, 2, 3, 5, 8, 13, 21$, and so on.

Given an integer `n`, calculate $F(n)$ if $n \geq 0$. If `n` is negative, return `-1`.

Hint: Try doing this question iteratively and not recursively, i.e. start from `F(0)` and `F(1)` and go up towards `F(n)`.

For example:

```
Solution(3) -> 2
Solution(17) -> 1597
Solution(-5) -> -1
```

Question 12: Pearson's Correlation Coefficient

Pearson's correlation coefficient measures the linear correlation between two sets of data. When applied on a sample, it can be expressed as the following:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

given paired data $(x_1, y_1), \ldots, (x_n, y_n)$ consisting of n pairs where

- r_{xy} is the sample Pearson's correlation coefficient for datasets x and y.
- x_i, y_i are the individual sample points indexed with i

- $$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

  (the sample mean); and analogously for $\bar{y}$.

Given two lists of floats, `x` and `y`, calculate the sample Pearson correlation coefficient. If `x` and `y` are not the same length, return `None`.

Bonus: A bonus point will be awarded if you do not usethe `numpy`, `pandas`, `statistics`, or `scipy` packages.

For example:

```
x = [-1.315958, -7.563353, -5.269449, -2.689798, 1.64862, -1.229754, 1.30937, -4.340184,
    0.4519081, -0.8255139]
y = [-2.788446, -11.68663, -2.574175, 9.553144, 10.28382, -4.568493, 3.014083, 1.387986,
    10.37869, 7.203864]
Solution(x, y) -> 0.6956863
```

Question 13: Simple Gradient Descent

Gradient descent is an essential concept in data science and machine learning. It is used to find a local minimum/maximum of a given mathematical function, for example minimising the loss function in a machine learning algorithm.

The basic concept of gradient descent is that at a certain point, x_n, the algorithm calculates the gradient $\nabla f(x_n)$ of the function f (same as the derivative for a unvariate function). This gradient is the same as the direction that the point needs to be moved for it to have the greatest increase. However, gradient descent scales the gradient by the learning rate η giving us $\eta\nabla f(x_n)$. We then subtract this from where we started (x_n) to give the new point

$$x_{n+1} = x_n - \eta\nabla f(x_n). \qquad (*)$$

This is one iteration of the algorithm.

The orange curve represents an f function (in machine learning, f is normally called the cost/loss function).

At the initial point ,x_0, represented by the black dot, the algorithm finds the derivative, $\nabla f(x_0)$, of the orange curve (you will use `gradient_func` given to you to calculate the derivative), and calculates the next x value by subtracting the gradient multiplied by the learning rate (given by the starred equation earlier). This movement is shown by the first of the arrows moving towards to minimum value.

Repeatedly doing these steps allows for gradient descent to approximate the local minimum of the function.

Your task is given the gradient function `gradient_func` of a univariate function (i.e. a function that takes one numeric input), a starting value `x_0`, a learning rate `lr`, and the number of iterations `n_iters`, find an approximation of the local minimum of the univariate function.

For example:

```
# The gradient function will be given to you
def gradient_func(x):
    return 2 * x - 4

Solution(
    gradient_func=gradient_func,
    x_0=5,
    lr=0.1,
    n_iters=100) -> # 2.0000000006111107
```

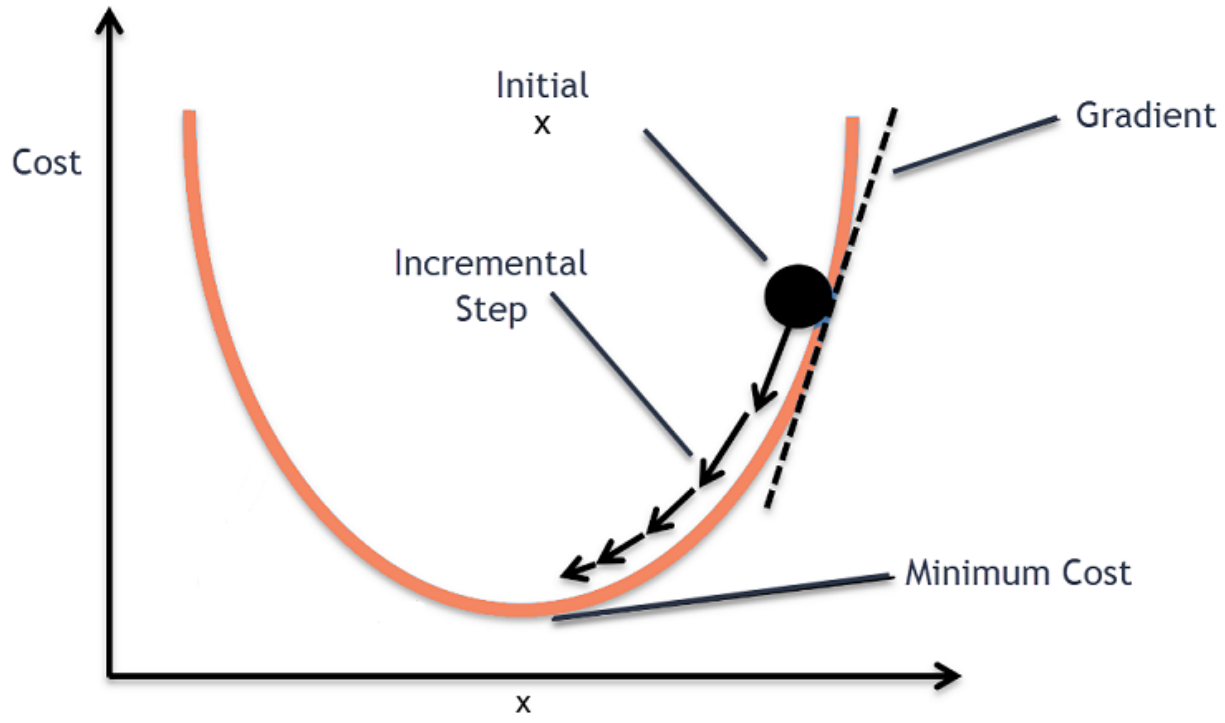

Figure 1: Gradient Descent Diagram

Question 14: Multiple Linear Regression Normal Equation

Task Background

Suppose some data consists of n observations $\{\mathbf{x}_i, y_i\}_{i=1}^{n}$. Each observation i includes a numerical response $y_i \in \mathbb{R}$ and a row vector $\mathbf{x}_i$ of p features (explanatory variables), i.e.,

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} & x_{i2} & \dots & x_{ip} \end{bmatrix}, \text{ where } x_{ij} \in \mathbb{R}.$$

In a linear regression model, the response variable, y_i, is given by a linear function of the features:

$$y_i = \beta_1 \, x_{i1} + \beta_2 \, x_{i2} + \cdots + \beta_p \, x_{ip} + \varepsilon_i,$$

or in vector form,

$$y_i = \mathbf{x}_i \boldsymbol{\beta} + \varepsilon_i = \begin{bmatrix} x_{i1} & x_{i2} & \dots & x_{ip} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix} + \varepsilon_i,$$

where $\mathbf{x}_i$ is a column vector of the i-th observation of all the explanatory variables; $\boldsymbol{\beta}$ is a $p \times 1$ vector of unknown parameters/coefficients; and the scalar $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ represents unobserved random variables (errors) of the i-th observation.

This model can also be written in matrix notation as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where $\mathbf{y}$ and $\boldsymbol{\varepsilon}$ are $n \times 1$ vectors of the response variables and the errors of the n observations, and $\mathbf{X}$ is an $n \times p$ matrix of regressors, also sometimes called the **design matrix**, whose row i is $\mathbf{x}_i$ and contains the i-th observations on all the explanatory variables.

Typically, a constant term is included in the set of regressors $\mathbf{X}$, by adding $x_{i0} = 1$ for all $i = 1, \ldots, n$. The coefficient β_0 corresponding to this regressor is called the **intercept**. Without the intercept, the fitted line is forced to cross the origin when $\mathbf{x}_i = \vec{0}$.

Note: From now on, we will be assuming that the **intercept is included** in the design matrix (i.e. $\mathbf{X}$) and $\boldsymbol{\beta}$. This means we have $p + 1$ parameters to estimate, $\beta_0\, \beta_1, \ldots, \beta_p$, and our result matrices look as follows:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}, \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$$

When we have more observations than parameters ($n > p + 1$) and $p + 1$ unknown coefficients, $\beta_0, \beta_1, \ldots, \beta_p$, there is no exact solution to the equation

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

instead, we estimate the coefficients such that the sum of squares residuals is minimised,

$$\text{Sum of squared residuals} = \sum_{i=1}^{n} \left(y_i - \left(\hat{\beta}_0 + \hat{\beta}_1\, x_{i1} + \hat{\beta}_2\, x_{i2} + \cdots + \hat{\beta}_p\, x_{ip} \right) \right)^2$$

This is achieved by solving what's called the normal equations $\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right) \hat{\boldsymbol{\beta}} = \mathbf{X}^{\mathrm{T}}\mathbf{y}$ for $\hat{\boldsymbol{\beta}}$:

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1} \mathbf{X}^{\mathrm{T}}\mathbf{y}.$$

Task

Your task is, given the following:

- `X` - a two-dimensional python list where each of the n rows is one observation `i` and each column are the values of the explanatory variables (i.e. $x_{i1}, ..., x_{ip}$ and not including the intercept x_{i0})
- `y` - a one-dimensional list of length n representing the response values y_i,

derive a one-dimensional list containing the values of $\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_p$ using the rearranged normal equation:

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1} \mathbf{X}^{\mathrm{T}}\mathbf{y}.$$

For example:

```
X = [
    # xi1  # xi2
    [5.1,   3.5], # x1
    [4.9,   3.0], # x2
    [4.7,   3.2], # x3
    [4.6,   3.1], # x4
```

```
    [5.0,   3.6]  # x5
]

y = [
    40.6676, # y1
    38.3504, # y2
    37.8056, # y3
    37.0268, # y4
    40.3952  # y5
]

Solution(X, y) -> [5.000,  # betahat0 - remember to add an intercept
                   5.256,  # betahat1 - coefficient for xi1
                   2.532]  # betahat2 - coefficient for xi2
```

Notes:

- The **input** `X` will have **at most two columns**. Use this fact when calculating the inverse of $X^{\mathrm{T}}X$ to your advantage.
- All values in the input lists will be numerical floats or integers. You do not need to handle text, ordinal, categorical, or other types of variables.
- A bonus point will be awarded if you do not use any external libraries.
- Remember to find the coefficient for the intercept. This means adding a column of ones to the beginning of `X`.

Bonus: A bonus point will be awarded if you do not use the `numpy` or `pandas` packages.

Hints: The equation for inverting a 2x2 matrix is given as follows:

$$\mathbf{M}^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{\det \mathbf{M}} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

The equation for inverting a 3x3 matrix is given as follows:

$$\mathbf{M}^{-1} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^{-1} = \frac{1}{\det(\mathbf{M})} \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix}^{\mathrm{T}} = \frac{1}{\det(\mathbf{M})} \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & I \end{bmatrix}$$

where the values of the intermediate matrix is given by

$$\begin{aligned} A &= \ \ (ei - fh), & D &= -(bi - ch), & G &= \ \ (bf - ce), \\ B &= -(di - fg), & E &= \ \ (ai - cg), & H &= -(af - cd), \\ C &= \ \ (dh - eg), & F &= -(ah - bg), & I &= \ \ (ae - bd). \end{aligned}$$

and the determinant $\det(\mathbf{M})$ can be calculated with the following:

$$\det(\mathbf{M}) = aA + bB + cC.$$