**iNeuron DS Assessment - 10th March 2024**

Github repo for this assessment - https://github.com/adi8837/10\_March\_FSDS/tree/main

Python

**Answer 1** -

Github link - <https://github.com/adi8837/10_March_FSDS/blob/main/Answer1.py>

Explanation -

1. Iterates through the nested dictionary.
2. For dictionaries inside, it recursively calls itself
3. It creates a list for each key, containing subsequent keys for nested dictionaries
4. It reverses the order of keys within nested dictionaries
5. The final dictionary has keys and value pairs

**Answer 2** -

Github link - <https://github.com/adi8837/10_March_FSDS/blob/main/Answer2.py>

Explanation - fxn function: This function checks if c horses can fit in a list (a) with at least b distance between them. It iterates through positions, keeping track of the furthest horse (d). If a position is far enough (a[i] - d >= b), it counts it as a valid horse spot. It will return True if enough valid spots are found (count >= c), False otherwise.

aggressive\_horse function: It finds the largest gap possible to place k horses in the list (a). It finds the highest position value (maxi). It tries different gaps (1 to maxi), checking with fxn if k horses fit with that gap. It returns the largest gap that works (ans), or -1 if none work.

Main code: It calls aggressive\_horse to find the maximum usable gap and print the result.

**Answer 3** -

Github link - <https://github.com/adi8837/10_March_FSDS/blob/main/Python_Asnwer3.py>

Explanation - door\_mat\_dim(N) - This function takes an odd integer N (number of rows).Then it creates a doormat pattern with a welcome message in the center.It returns a string representing the entire doormat pattern.

main() - This prompts the user for an odd number of rows (N) then calls door\_mat\_dim(N) to get the pattern string (but doesn't print it by default). Then it executes the script with python door\_mat\_creator.py and prints the output.

**Answer 4** -

Github link - https://github.com/adi8837/10\_March\_FSDS/blob/main/Answer4.py

Explanation - This function finds unique quadruplets in a list that sum to a target. It sorts the list for faster lookups and uses a dictionary to look at pairs and their index. It iterates through elements, calculating the complement needed for the target. If the complement exists in the dictionary it checks for unique quadruplet combinations and adds them to the results. Finally, it updates the dictionary with the current pair for future checks.

SQL

**Answer 1** - I’m getting empty result set in this SQL query due to NULL value in winner\_id column in races table. As NULL has a special meaning in SQL and doesn't "match" any other value, runners who haven't participated in any races (no entry in races) would also be excluded, leading to a empty result set.

We can use this alternate SQL statement -

SELECT \*

FROM runners r

LEFT JOIN races ra ON r.id = ra.winner\_id

WHERE ra.winner\_id IS NULL;

**Answer 2** -

SELECT \*

FROM test\_a

EXCEPT

SELECT \*

FROM test\_b;

Explanation - The “Except” operator will give us the result set of values which are present in test\_a but not prosent in test\_b

**Answer 3** -

SELECT u.user\_id, username, training\_id, training\_date

FROM users u

INNER JOIN training\_details td ON u.user\_id = td.user\_id

GROUP BY u.user\_id, username, training\_id

HAVING COUNT(training\_date) > 1

ORDER BY u.user\_id, training\_date DESC;

Explanation - The query uses an INNER JOIN to combine data from the users (u) and training\_details (td) tables on the user\_id column.Then the SELECT clause retrieves the provided columns. The GROUP BY clause groups the results by u.user\_id, username, and training\_id. The HAVING clause filters the grouped results. It keeps only groups where the COUNT(training\_date) is greater than 1. This identifies users who took the same training lesson more than once. The ORDER BY clause sorts the results in descending order by training\_date. This presents the most recent lesson dates first, followed by older dates, for each user and training lesson group.

**Answer 4** -

SELECT manager\_id, AVG(salary) AS average\_salary\_under\_manager

FROM employee

GROUP BY manager\_id;

Explanation - SELECT selects the manager\_id and calculates the average salary using AVG(salary) from the table.Group by groups the rows in the result based on the manager\_id.

Statistics

**Answer 1** - Six sigma is a statistical measure of process capability, signifying a very high level of quality. It indicates a process with minimal variation, aiming for near perfection. Here's how it translates:

* Concept**:** It reflects a process producing only 2.9 defects per million opportunities (DPMO).
* Quality Level**:** A six sigma process is exceptionally reliable and efficient.

**Eg:** Imagine an auto assembly line. Six sigma strives to ensure fewer than 2.9 defective cars per million produced.

**Answer 2** - Data that cannot be negative and skewed towards lower values typically won't follow a Gaussian or log-normal distribution. Here are some examples:

* Income: Income cannot be negative, and usually count of lower income people is high, so with a larger concentration towards lower income brackets, it’s usually a right-skewed distribution.
* Customer wait times: Wait times can't be negative, and they often have a peak at lower wait times (e.g., most customers are served quickly) with a longer queue for those who wait longer, leading to a right skew.
* Inventory levels: Inventory cannot be negative, and it often follows a right-skewed distribution with more frequent lower stock levels and occasional spikes for high-demand items.

**Answer 3** - The five-number summary is a concise way to describe the spread and center of a dataset using five key percentiles:

1. Minimum: The smallest value in the data set.
2. First Quartile (Q1): The value below which 25% of the data falls.
3. Median: The middle value, or the 50th percentile, when the data is ordered.
4. Third Quartile (Q3): The value below which 75% of the data falls.
5. Maximum: The largest value in the data set.

Example:

Consider a dataset of exam scores: {50, 72, 85, 90, 95, 100, 68, 88, 75}

* Minimum: 50
* Q1: 68 (25% of scores are below 68)
* Median: 85 (the middle score)
* Q3: 90 (75% of scores are below 90)
* Maximum: 100

**Answer 4** - <https://github.com/adi8837/10_March_FSDS/blob/main/Statistics_Answer4.ipynb>

Machine Learning

**Answer 1:**

Github link - <https://github.com/adi8837/10_March_FSDS/blob/main/MachineLearning_Answer1.ipynb>

**Answer 2:**

Github link - <https://github.com/adi8837/10_March_FSDS/blob/main/MachineLearning_Answer2.ipynb>

Machine Learning

**Answer 1:** Deep Learning (DL) Implementation in Real-World Applications

Deep Learning (DL) models are very helpful for real-world applications learn due to their ability to learn complex patterns from large datasets. Here's a breakdown of the key steps involved in implementing DL in real-world scenarios:

Problem Definition and Data Gathering:

Clearly define the problem you want to solve or the task you want to automate. This will guide your data collection and model selection.

Gathering a large, high-quality dataset relevant to your problem. Label and preprocess the data appropriately (e.g., handling missing values, normalization/standardization).

DL Model:

Choose a suitable DL architecture based on the nature of your data and problem:

Convolutional Neural Networks (CNNs) excel at image recognition and other grid-like data (e.g., medical scans, time series).

Recurrent Neural Networks (RNNs) are well-suited for sequential data like text (e.g., sentiment analysis, machine translation).

Transformers are powerful for natural language processing (NLP) tasks, especially when dealing with longer sequences.

Select an appropriate hyperparameter optimization strategy (e.g., grid search, random search) to find the best configuration for your model's architecture and training process.

Model Training and Evaluation:

Train your DL model using the prepared dataset, splitting it into training, validation, and testing sets. The training set feeds the model for learning, the validation set helps fine-tune hyperparameters to avoid overfitting etc. Use appropriate metrics to evaluate your model's effectiveness. Common metrics include mean squared error (MSE) or mean absolute error (MAE) etc

Monitor the training process (e.g., loss function, validation metrics) to identify potential issues like overfitting or underfitting. Make adjustments to the model architecture, hyperparameters, or training process as needed.

Consider employing regularization techniques (e.g., dropout, L1/L2 regularization) to improve model generalization and reduce overfitting.

Deployment and Monitoring:

Once we're satisfied with the model's performance, we have to deploy it into production. This may involve integrating it into an existing application, creating a standalone web service, or using cloud-based deployment options.

Continuously monitor the model's performance in production to ensure it maintains its effectiveness over time. Data distribution may shift, or new patterns may emerge. We have to retrain or adapt the model when necessary on a regular basis.

Activation Functions in Artificial Neural Networks (ANNs)

Activation functions introduce non-linearity into ANNs, enabling them to learn complex relationships between inputs and outputs. Without activation functions, ANNs would be limited to linear functions, only capable of representing straight lines. Here's what would happen if we didn't use them:

Limited Learning Capacity: With only linear functions, ANNs could only learn simple patterns like lines and hyperplanes. They wouldn't be able to model the non-linear relationships often observed in real-world data. This would significantly restrict their application in tasks like image recognition, natural language processing, and other domains where complex patterns exist.

Vanishing or Exploding Gradients: The lack of non-linearity could lead to vanishing or exploding gradients during the backpropagation training process. This could make it difficult for the model to learn effectively or even converge at all. Backpropagation relies on updating weights based on the gradient of the error function, and without activation functions, gradients might become very small (vanishing) or very large (exploding) over multiple layers, hindering gradient descent from finding the optimal weights.

Activation Function Examples:

Some common activation functions used in deep learning include:

Sigmoid: Maps input values between 0 and 1 but suffers from vanishing gradients in deeper networks.

Tanh: Similar to sigmoid but with a wider range (-1 to 1) and less prone to vanishing gradients.

ReLU (Rectified Linear Unit): Simple and computationally efficient, outputs the input value if it's positive, otherwise outputs 0. Can suffer from the "dying ReLU" problem where some neurons become inactive if negative inputs are too frequent.

Leaky ReLU: A variant of ReLU that allows a small non-zero gradient for negative inputs, addressing the dying ReLU issue.

Softmax: Used in the output layer of multi

**Answer 2:**

Github link - https://github.com/adi8837/10\_March\_FSDS/blob/main/DeepLearning\_Answer2.ipynb