List of mock interview questions for Machine Learning (ML) and Artificial Intelligence (AI) freshers:

1. **Foundational Concepts :**

- **What is the difference between supervised and unsupervised learning?**

|  |  |  |
| --- | --- | --- |
|  | SUPERVISED LEARNING | UNSUPERVISED LEARNING |
| Input Data | Uses Known and Labeled Data as input | Uses Unknown Data as input |
| Computational Complexity | Less Computational Complexity | More Computational Complex |
| Real Time | Uses off-line analysis | Uses Real Time Analysis of Data |
| Number of Classes | Number of Classes are known | Number of Classes are not known |
| Accuracy of Results | Accurate and Reliable Results | Moderate Accurate and Reliable Results |
| Output data | Desired output is given. | Desired output is not given. |
| Model | In supervised learning it is not possible to learn larger and more complex models than with supervised learning | In unsupervised learning it is possible to learn larger and more complex models than  with unsupervised learning |
| Training data | In supervised learning training data is used to infer model | In unsupervised learning training data is not used. |
| Another name | Supervised learning is also called classification. | Unsupervised learning is also called clustering. |
| Test of model | We can test our model. | We can not test our model. |
| Example | Optical Character Recognition | Find a face in an image. |

**- Can you explain the bias-variance tradeoff?**

if algorithms fit too complex (hypothesis with high degree equation) then it may be on high variance and low bias. In the latter condition, the new entries will not perform well. Well, there is something between both of these conditions, known as a Trade-off or Bias Variance Trade-off.

**- What is the difference between a parametric and a non-parametric algorithm?**

| **Parametric Methods** | **Non-Parametric Methods** |
| --- | --- |
| Parametric Methods uses a fixed number of parameters to build the model. | Non-Parametric Methods use the flexible number of parameters to build the model. |
| Parametric analysis is to test group means. | A non-parametric analysis is to test medians. |
| It is applicable only for variables. | It is applicable for both – Variable and Attribute. |
| It always considers strong assumptions about data. | It generally fewer assumptions about data. |
| Parametric Methods require lesser data than Non-Parametric Methods. | Non-Parametric Methods requires much more data than Parametric Methods. |
| Parametric methods assumed to be a normal distribution. | There is no assumed distribution in non-parametric methods. |
| Parametric data handles – Intervals data or ratio data. | But non-parametric methods handle original data. |
| Here when we use parametric methods then the result or outputs generated can be easily affected by outliers. | When we use non-parametric methods then the result or outputs generated cannot be seriously affected by outliers. |
| Parametric Methods can perform well in many situations but its performance is at peak (top) when the spread of each group is different. | Similarly, Non-Parametric Methods can perform well in many situations but its performance is at peak (top) when the spread of each group is the same. |
| Parametric methods have more statistical power than Non-Parametric methods. | Non-parametric methods have less statistical power than Parametric methods. |
| As far as the computation is considered these methods are computationally faster than the Non-Parametric methods. | As far as the computation is considered these methods are computationally slower than the Parametric methods. |
| Examples: Logistic Regression, Naïve Bayes Model, etc. | Examples: KNN, Decision Tree Model, etc. |

1. **Deep Learning :**

**- What is the difference between a neural network and a deep neural network?**

Deep learning, also a subset of machine learning, uses algorithms to recognize patterns in complex data and predict outputs. Unlike machine learning algorithms, which require labeled data sets, deep learning networks can be trained using unsupervised learning (which doesn’t require labeled data sets) to perform feature extraction with less reliance on human input.

It’s called deep learning because of the number of hidden layers used in the deep learning model. While a basic neural network comprises an input, output, and hidden layer, a deep neural network has multiple hidden layers of processing.

A deep neural network is a more complicated form of neural network. Where neural networks give a single result such as a word, solution, or action, deep ones create a global solution based on all the input data given.

Because of their multiple layers, a deep neural network takes longer to train than a neural network, but they offer higher performance, efficiency, and accuracy.

A neural network includes components such as neurons, connections, propagation functions, learning rate, and weight. In contrast, a deep learning network typically comprises a motherboard, processors (CPU or GPU), large quantities of RAM, and a large power supply unit (PSU) for processing complex deep learning functions and massive data sets.

**- How does backpropagation work?**

Backpropagation is a process involved in training a neural network. It involves taking the error rate of a forward propagation and feeding this loss backward through the neural network layers to fine-tune the weights.

**- What is the role of activation functions in a neural network?**

The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

**Explanation:**We know, the neural network has neurons that work in correspondence with *weight, bias,* and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as [*back-propagation*](https://www.geeksforgeeks.org/backpropagation-in-data-mining/). Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

**3. Algorithms and Techniques :**

1. **Explain the working of the k-means clustering algorithm.**

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

**The working of the K-Means algorithm is explained in the below steps:**

**Step-1:** Select the number K to decide the number of clusters.

**Step-2:** Select random K points or centroids. (It can be other from the input dataset).

**Step-3:** Assign each data point to their closest centroid, which will form the predefined K clusters.

**Step-4:** Calculate the variance and place a new centroid of each cluster.

**Step-5:** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

**Step-6:** If any reassignment occurs, then go to step-4 else go to FINISH.

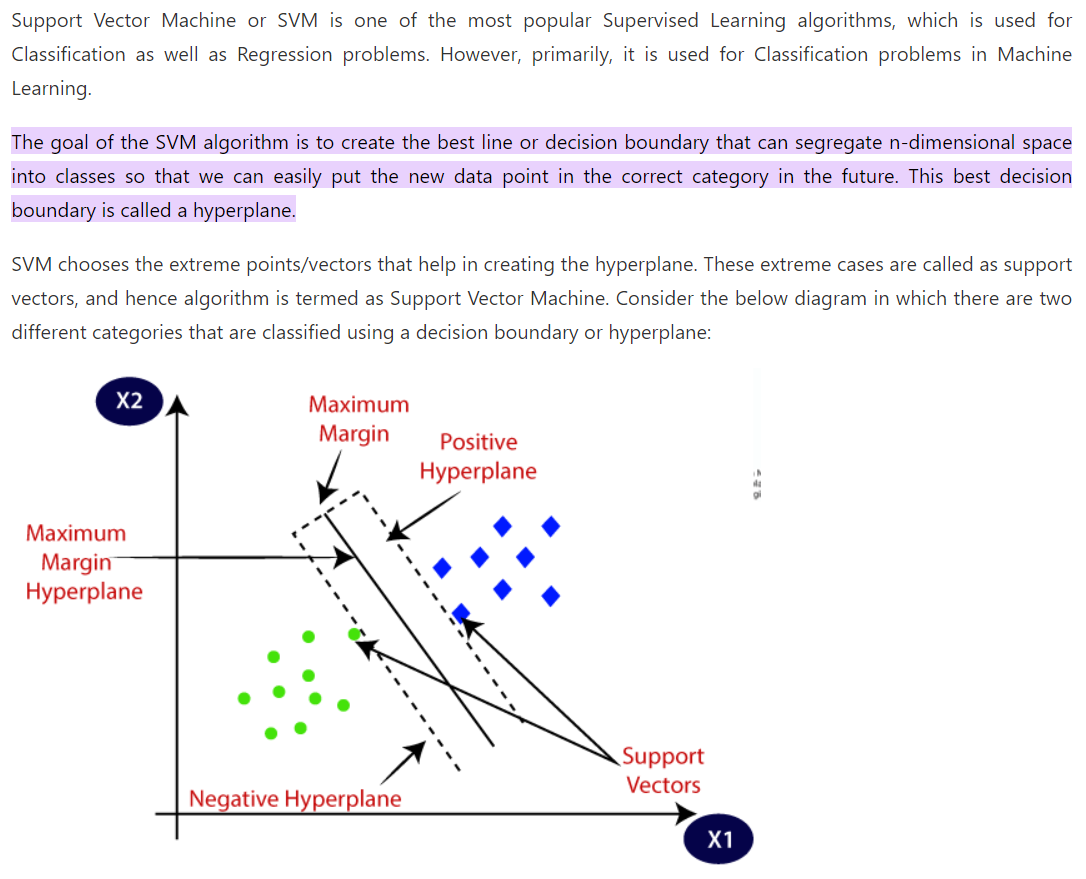
**Step-7**: The model is ready.

https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning#:~:text=The%20working%20of%20the%20K%2DMeans%20algorithm%20is%20explained%20in,form%20the%20predefined%20K%20clusters.

1. **How does a decision tree work?**

**A decision tree** algorithm is a machine learning algorithm that uses a decision tree to make predictions. It follows a tree-like model of decisions and their possible consequences. The algorithm works by recursively splitting the data into subsets based on the most significant feature at each node of the tree.

1. **Describe the principle behind the Support Vector Machine (SVM) algorithm.**

****

1. **What is the difference between random forests and gradient-boosted trees?**

There are two main differences between the gradient boosting trees and the random forests. We train the former sequentially, one tree at a time, each to correct the errors of the previous ones. In contrast, we construct the trees in a random forest independently. Because of this, we can train a forest in parallel but not the gradient-boosting trees.

The other principal difference is in how they output decisions. Since the trees in a random forest are independent, they can determine their outputs in any order. Then, we aggregate the individual predictions into a collective one: the majority class in classification problems or the average value in regression. On the other hand, the gradient boosting trees run in a fixed order, and that sequence cannot change. For that reason, they admit only sequential evaluation.

1. **Evaluation Metrics :**

**- What is the difference between precision and recall?**

1. **Precision:**
   * Precision is a metric that measures the accuracy of positive predictions made by a model. It answers the question: "Of all the positive predictions made by the model, how many were correct?"
   * Precision is calculated as the ratio of true positives (correctly predicted positive instances) to the sum of true positives and false positives (incorrectly predicted positive instances).
   * Precision focuses on the model's ability to avoid making false positive errors. A high precision indicates that the model is good at correctly identifying positive instances without misclassifying too many negative instances as positives.

Precision = True Positives / (True Positives + False Positives)

1. **Recall:**
   * Recall, also known as sensitivity or true positive rate, measures the model's ability to identify all relevant instances in the dataset. It answers the question: "Of all the actual positive instances, how many did the model correctly predict as positive?"
   * Recall is calculated as the ratio of true positives to the sum of true positives and false negatives (missed positive instances).
   * Recall focuses on the model's ability to avoid false negatives. A high recall indicates that the model is good at capturing most of the positive instances, but it may have a higher rate of false positives.

Recall = True Positives / (True Positives + False Negatives)

**- How would you evaluate the performance of a regression model versus a classification model?**

**Evaluating a Regression Model:**

In regression, the goal is to predict a continuous numeric value, such as a price, temperature, or a score. Here are some common evaluation metrics for regression models:

1. **Mean Absolute Error (MAE):**
   * MAE measures the average absolute difference between the predicted values and the actual values. It provides a straightforward understanding of the model's average prediction error.
   * Formula: MAE = (1/n) \* Σ|actual - predicted|
2. **Mean Squared Error (MSE):**
   * MSE calculates the average squared difference between the predicted values and the actual values. It amplifies the impact of larger errors compared to MAE.
   * Formula: MSE = (1/n) \* Σ(actual - predicted)^2
3. **Root Mean Squared Error (RMSE):**
   * RMSE is the square root of MSE and provides a measure of the typical error in the model's predictions in the same units as the target variable.
   * Formula: RMSE = sqrt(MSE)
4. **R-squared (R²) or Coefficient of Determination:**
   * R² measures the proportion of the variance in the target variable that is predictable from the independent variables in the model. It ranges from 0 to 1, where higher values indicate better model fit.
   * Interpretation: R² = 1 means a perfect fit, while R² = 0 means the model does not explain any variance.
5. **Adjusted R-squared (Adjusted R²):**
   * Adjusted R² takes into account the number of predictors in the model, penalizing excessive complexity. It is especially useful when dealing with multiple independent variables.

**Evaluating a Classification Model:**

In classification, the goal is to assign data points to predefined categories or classes. Here are some common evaluation metrics for classification models:

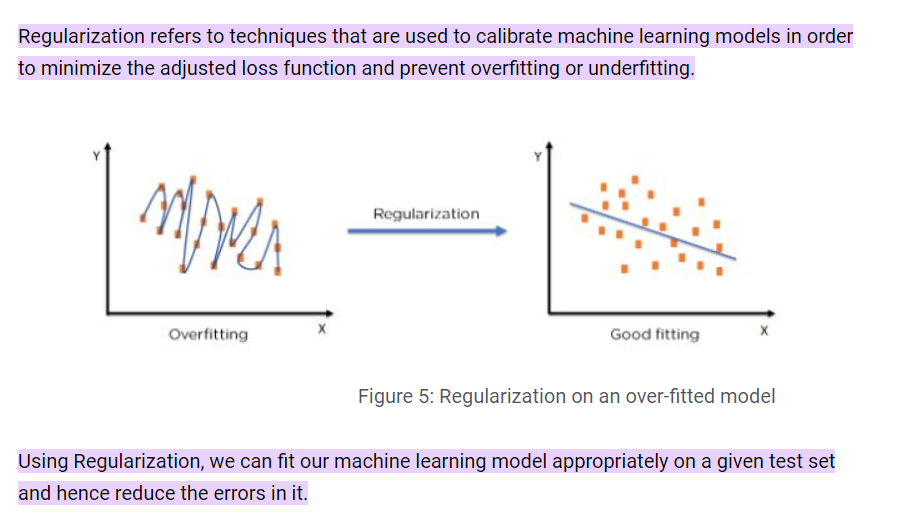
1. **Accuracy:**
   * Accuracy measures the proportion of correctly classified instances among all instances in the dataset. While it's a common metric, it can be misleading in imbalanced datasets.
   * Formula: Accuracy = (TP + TN) / (TP + TN + FP + FN)
2. **Precision:**
   * Precision measures the accuracy of positive predictions. It is the ratio of true positives to the sum of true positives and false positives.
   * Formula: Precision = TP / (TP + FP)
3. **Recall (Sensitivity or True Positive Rate):**
   * Recall measures the model's ability to correctly identify positive instances. It is the ratio of true positives to the sum of true positives and false negatives.
   * Formula: Recall = TP / (TP + FN)
4. **F1-Score:**
   * The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It's useful when you want to find a balance between precision and recall.
   * Formula: F1-Score = 2 \* (Precision \* Recall) / (Precision + Recall)
5. **ROC Curve and AUC (Area Under the ROC Curve):**
   * ROC curve plots the trade-off between true positive rate (recall) and false positive rate at different classification thresholds.
   * AUC measures the area under the ROC curve and indicates the model's ability to distinguish between classes.

**- What is the ROC curve and the AUC?**

* + ROC curve plots the trade-off between true positive rate (recall) and false positive rate at different classification thresholds.
  + AUC measures the area under the ROC curve and indicates the model's ability to distinguish between classes.

**5. Regularization and Optimization :**

1. **What is regularization and why is it important?**

****

1. **How does gradient descent work? What is the difference between batch, mini-batch, and stochastic gradient descent?**

Gradient Descent is an optimization algorithm used to minimize the cost or loss function in machine learning and deep learning models. It's a fundamental technique used for training various types of models, such as linear regression, neural networks, and support vector machines. Gradient Descent iteratively updates the model's parameters (weights and biases) to find the optimal values that minimize the cost function. Here's how it works:

1. **Initialization:** Initialize the model's parameters (weights and biases) with random values or zeros.
2. **Compute the Gradient:** Calculate the gradient of the cost function with respect to the model parameters. This gradient represents the direction and magnitude of the steepest increase in the cost.
3. **Update Parameters:** Adjust the model's parameters by subtracting a fraction of the gradient (learning rate times the gradient) from the current parameter values. This step aims to move the parameters in the direction that reduces the cost.
4. **Repeat:** Repeat steps 2 and 3 for a specified number of iterations (epochs) or until the cost converges to a minimum value or stops changing significantly.

Gradient Descent algorithms differ primarily in how they calculate and update gradients and how they process training data. The main variations are Batch Gradient Descent, Mini-Batch Gradient Descent, and Stochastic Gradient Descent (SGD):

1. **Batch Gradient Descent (BGD):**
   * In BGD, the entire training dataset is used to compute the gradient of the cost function at each iteration.
   * It provides a precise estimate of the gradient but can be slow for large datasets since it processes all data points in each iteration.
   * BGD is suitable for problems where the dataset can comfortably fit into memory.
2. **Mini-Batch Gradient Descent:**
   * Mini-Batch Gradient Descent strikes a balance between BGD and SGD. It divides the training dataset into small batches of fixed size (typically 32, 64, or 128 data points).
   * It calculates and updates the gradients using one mini-batch at a time in each iteration.
   * Mini-Batch GD is the most commonly used variant in practice because it combines some of the advantages of BGD (less noisy updates) and SGD (faster convergence).
3. **Stochastic Gradient Descent (SGD):**
   * In SGD, only one training data point is used to compute and update thegradient in each iteration.
   * It converges faster per iteration but has a noisy update process because each update is based on a single data point.
   * SGD can escape local minima more easily due to its stochastic nature and is often used when the dataset is large and doesn't fit into memory.

The choice between these variants depends on factors like the dataset size, computational resources, and convergence requirements. Mini-Batch Gradient Descent is a popular choice as it combines the advantages of both BGD and SGD and is often the default choice for training neural networks. The learning rate is an important hyperparameter in all these variants, as setting it too high can lead to divergence, while setting it too low can lead to slow convergence. Tuning the learning rate is typically part of the training process.

**6. Practical Applications :**

1. **How would you handle missing data in a dataset?**

Missing values can be handled by deleting the rows or columns having null values. If columns have more than half of the rows as null then the entire column can be dropped. The rows which are having one or more columns values as null can also be dropped.

1)drop

2)fillna

3)take average or median or mean

1. **What are some techniques to deal with imbalanced datasets?**

Dealing with imbalanced datasets is a common challenge in machine learning. Here are some techniques to address this issue:

1. **Resampling**:
   * **Oversampling**: Increase the number of instances in the minority class by duplicating samples or generating synthetic data points.
   * **Undersampling**: Reduce the number of instances in the majority class by randomly removing samples.
2. **Generate Synthetic Data**:
   * Use techniques like Synthetic Minority Over-sampling Technique (SMOTE) to create synthetic samples for the minority class, which helps balance the dataset.
3. **Data Augmentation**:
   * Augment the minority class data by applying random transformations or perturbations to existing samples, such as rotation, translation, or adding noise.
4. **Cost-sensitive Learning**:
   * Assign different misclassification costs to different classes to make the model more sensitive to the minority class. This can be done by adjusting class weights during training.
5. **Ensemble Methods**:
   * Use ensemble techniques like EasyEnsemble, BalancedRandomForest, or AdaBoost with resampled or augmented data to improve the model's performance on imbalanced data.
6. **Anomaly Detection**:
   * Treat the minority class as an anomaly detection problem and use techniques like One-Class SVM or Isolation Forest to identify anomalies.
7. **Algorithm Selection**:
   * Choose machine learning algorithms that are less sensitive to class imbalance, such as random forests, gradient boosting, or support vector machines.
8. **Threshold Adjustment**:
   * Adjust the classification threshold to favor the minority class, depending on the specific problem and the desired trade-off between precision and recall.
9. **Cost-sensitive Loss Functions**:
   * Design custom loss functions that penalize misclassifying the minority class more heavily than the majority class.
10. **Collect More Data**:
    * If possible, collect more data for the minority class to balance the dataset naturally.
11. **Use Anomaly Detection Models**:
    * For extremely imbalanced datasets, consider using anomaly detection models (e.g., autoencoders or isolation forests) instead of traditional classification algorithms.
12. **Evaluate with Appropriate Metrics**:
    * Use evaluation metrics like precision, recall, F1-score, ROC-AUC, or PR-AUC instead of accuracy to assess model performance on imbalanced datasets.
13. **Cross-Validation Strategies**:
    * Employ techniques like stratified k-fold cross-validation to ensure that each fold maintains the class distribution proportion.
14. **Combine Oversampling and Undersampling**:
    * Apply a combination of oversampling the minority class and undersampling the majority class to balance the dataset.
15. **Ensemble of Multiple Models**:
    * Combine the predictions of multiple models, each trained on different resampled versions of the dataset, to improve the overall classification performance.
16. **Use Transfer Learning**:
    * Leverage pre-trained models or embeddings, especially in natural language processing or computer vision tasks, to benefit from learned representations of data.

Selecting the most appropriate technique(s) for handling imbalanced datasets depends on the specific problem, available resources, and the characteristics of the data. Experimentation and careful evaluation of model performance are essential in finding the best approach.

Top of Form

1. **How would you approach a problem where the data is not linearly separable?**

When dealing with a problem where the data is not linearly separable, meaning that a linear classifier (e.g., logistic regression or a linear support vector machine) cannot effectively separate the classes, you need to explore more complex modeling techniques. Here's an approach to tackle such problems:

1. **Non-linear Models**:
   * Shift your focus from linear models to non-linear ones, such as decision trees, random forests, k-nearest neighbors, kernelized SVMs, neural networks, or gradient boosting machines (e.g., XGBoost, LightGBM).
2. **Feature Engineering**:
   * Transform or engineer features to make the data more amenable to linear classifiers or to uncover non-linear patterns. Techniques include polynomial features, interaction terms, and domain-specific feature transformations.
3. **Kernel Tricks**:
   * Use kernel functions in algorithms like Support Vector Machines (SVMs) to implicitly map data into higher-dimensional spaces where it might become linearly separable. Common kernels include the radial basis function (RBF) kernel.
4. **Ensemble Methods**:
   * Combine multiple models using ensemble methods like random forests, gradient boosting, or stacking. Ensembles can capture complex non-linear relationships by aggregating the predictions of several base models.
5. **Deep Learning**:
   * Consider deep neural networks, particularly when working with high-dimensional data, images, or sequences. Deep learning architectures like convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers are designed to handle complex non-linearities.
6. **Regularization**:
   * Apply regularization techniques to prevent overfitting in complex models. Common forms include L1 and L2 regularization for neural networks and SVMs.
7. **Dimensionality Reduction**:
   * Use dimensionality reduction techniques like Principal Component Analysis (PCA) or t-distributed Stochastic Neighbor Embedding (t-SNE) to project data into lower-dimensional spaces where linear separation might be possible.
8. **Local Patterns**:
   * Explore local patterns and relationships within the data using clustering algorithms (e.g., k-means) or density-based methods (e.g., DBSCAN). These can help identify regions of non-linearity.
9. **Feature Selection**:
   * Identify the most informative features using feature selection methods like Recursive Feature Elimination (RFE) or feature importance scores from tree-based models.
10. **Neighborhood-based Methods**:
    * Utilize instance-based learning methods like k-nearest neighbors (KNN) to make predictions based on the similarity of data points within a neighborhood.
11. **Transformations**:
    * Apply mathematical transformations to the data to make it linearly separable. For instance, you can use logarithmic, exponential, or Box-Cox transformations.
12. **Use Specialized Models**:
    * In some cases, specialized models designed for specific non-linear data patterns, such as Gaussian mixture models or radial basis function networks, may be appropriate.
13. **Ensemble of Diverse Models**:
    * Create an ensemble of diverse models, including linear and non-linear ones, to take advantage of the strengths of each model type.
14. **Cross-Validation and Hyperparameter Tuning**:
    * Perform thorough cross-validation and hyperparameter tuning to ensure your chosen non-linear models are well-optimized and generalize effectively.
15. **Evaluate Performance Metrics**:
    * Select appropriate performance metrics (e.g., F1-score, ROC-AUC, or precision-recall curves) to evaluate model performance on non-linear data, as accuracy can be misleading in such cases.

**7. Recent Trends and Research :**

1. **What are Generative Adversarial Networks (GANs) and how do they work?**

Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for [unsupervised learning](https://www.geeksforgeeks.org/supervised-unsupervised-learning/). It was developed and introduced by Ian J. Goodfellow in 2014. GANs are basically made up of a system of two competing neural network models which compete with each other and are able to analyze, capture and copy the variations within a dataset.

Generative Adversarial Networks (GANs) can be broken down into three parts:

* **Generative:** To learn a generative model, which describes how data is generated in terms of a probabilistic model.
* **Adversarial:** The training of a model is done in an adversarial setting.
* **Networks:** Use deep neural networks as artificial intelligence (AI) algorithms for training purposes.

**8. Ethics and Bias :**

1. **How can bias enter into ML models?**

Bias machine learning can be applied when collecting the data to build the models. It can come with testing the outputs of the models to verify their validity. It can even be applied when interpreting valid or invalid results from an approved data model**.**

1. **What are some ways to ensure fairness in AI models?**
2. **Can you give an example of an ethical dilemma in AI?**

**9. Tools and Libraries :**

1. **How do you handle large datasets in Python? Are you familiar with tools like Dask or Vaex?**

Handling large datasets efficiently in Python can be challenging, especially when the data doesn't fit into memory. Fortunately, there are tools and libraries available to help you manage and process large datasets. Two popular options are Dask and Vaex

**What is Dask?** Dask is a parallel computing library in Python that enables scalable and distributed data processing. It provides a high-level interface that allows you to work with larger-than-memory datasets by parallelizing operations across multiple cores or distributed clusters.

**Key Features:**

* **Dask DataFrame:** A pandas-like DataFrame that can handle datasets larger than memory by breaking them into smaller partitions.
* **Dask Array**: A numpy-like array that allows for parallel and out-of-core computation.
* **Dask Bag:** A collection type for handling semi-structured or unstructured data.
* **Dask ML**: An extension for machine learning tasks compatible with popular ML libraries.

**Benefits:**

* Dask provides a familiar API for users accustomed to libraries like pandas and numpy.
* It can scale from a single machine to distributed clusters, making it suitable for various data sizes and computational resources.
* Dask performs lazy evaluation, optimizing and parallelizing operations to minimize memory usage and maximize efficiency.

**Vaex:**

**What is Vaex?** Vaex is an alternative library for handling large datasets in Python. It is designed for fast, memory-efficient data manipulation and analysis, specifically for datasets that don't fit into memory.

**Key Features:**

* **Vaex DataFrame:** An in-memory DataFrame that uses an "out-of-core" approach to efficiently process large datasets.
* **Lazy Evaluation:** Vaex uses lazy evaluation, similar to Dask, to minimize memory consumption.
* **Fast Operations:** Vaex is optimized for speed and can perform operations like filtering, aggregations, and expressions quickly.

**Benefits:**

* Vaex is built for speed and can handle extremely large datasets with minimal memory overhead.
* It provides a simple and intuitive API that is similar to pandas, making it accessible to those familiar with pandas.

1. **What is the difference between TensorFlow and PyTorch?**

**PyTorch:**

It is an open-source library used in machine learning. It was developed by Facebook and was released to the public for the first time in 2016. It is imperative which means it runs immediately and the user can check if it is working or not before writing the full code. We can write a part of code and check it in real time, it is built-in python based implementation to provide compatibility as a deep learning platform. It rapidly gained users because of its user-friendly interface, which made the Tensorflow team acquire its popular features in Tensorflow 2.0.

**TensorFlow:**

Just like PyTorch, it is also an open-source library used in machine learning. It was developed by Google and was released in 2015. Its name itself expresses how you can perform and organize tasks on data. Production and research are the main uses of Tensorflow. Neural networks mostly use Tensorflow to develop machine learning applications.

1. **How would you deploy a machine learning model in a production environment?**

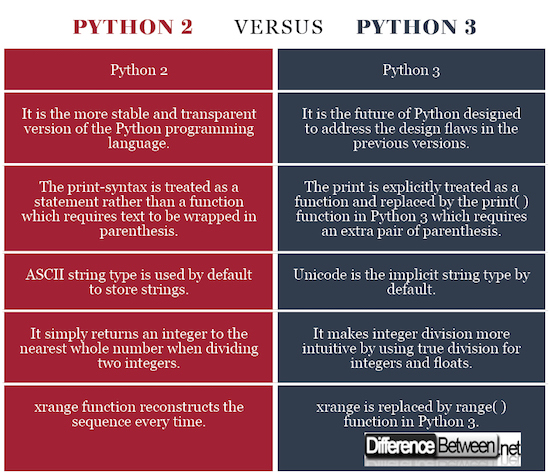
**10. Case Studies :**

1. **How would you design a recommendation system for an e-commerce website?**
2. **Imagine you're tasked with predicting the price of houses in a new city. How would you approach this problem?**
3. **How would you design a chatbot for customer support?**

**Python:**

**Basics:**

1. **What are the key differences between Python 2 and Python 3?**

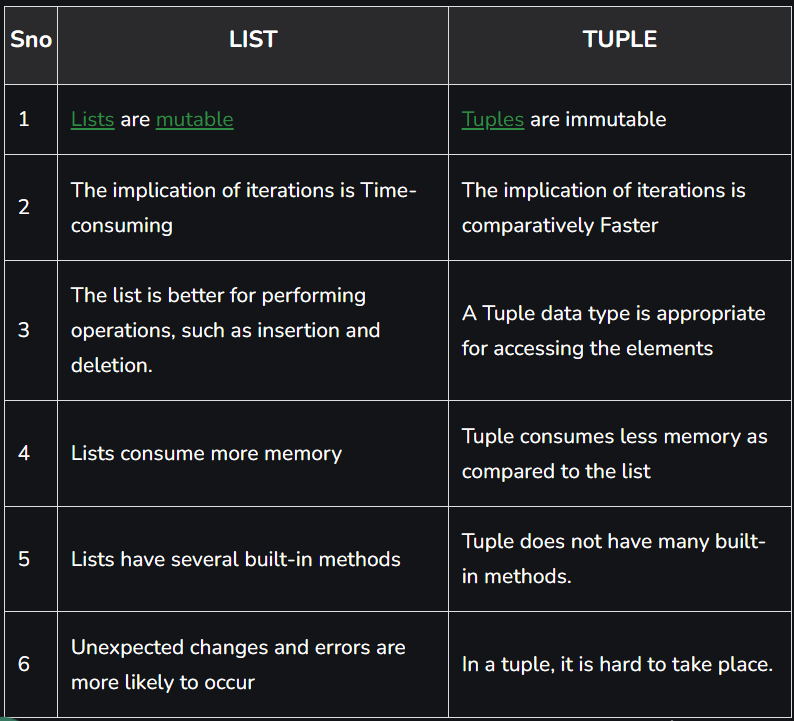


1. **How is memory managed in Python?**

Steps in managing memory in Python are -

* The Python memory is primarily managed by Python private heap space.
* All Python objects and data structures are located in a private heap.
* The programmer does not have access to this private heap and interpreter takes care of this Python private heap.
* The allocation of Python heap space for Python objects is done by Python memory manager.
* The core API gives access to some tools for the programmer to code.
* Python has an inbuilt garbage collector, that recycles all the unused memory and frees the memory and makes it available to the heap space.

1. **What are the differences between a list and a tuple?**

****

**OOP and Design Patterns:**

1. **Explain the concept of inheritance in object-oriented programming. How is it**

**implemented in Python?**

Inheritance allows us to define a class that inherits all the methods and properties from another class.

**Parent class** is the class being inherited from, also called base class.

**Child class** is the class that inherits from another class, also called derived class.

1. **What is a decorator in Python? Can you provide an example?**
2. **How would you implement a singleton pattern in Python?**

**Error Handling:**

1. **How does exception handling work in Python?**
2. **What is the difference between finally and else in a try-except block?**
3. **Advanced Concepts:**
4. **What are list comprehensions and how are they different from regular loops?**
5. **Explain the difference between deepcopy and shallowcopy.**
6. **How do you manage packages and environments in Python?**

**Pandas basics:**

1. **What is a DataFrame in pandas? How is it different from a Series?**

* A DataFrame in Pandas is **a two-dimensional** table with rows and columns, capable of holding data of different types.
* A Series in Pandas is a **one-dimensional** array-like structure representing a single column of data within a DataFrame.

1. **How would you read a CSV file using pandas?**

pd.read\_csv(“”)

1. **Describe the difference between iloc and loc.**

**Data Manipulation:**

1. **How would you handle missing data in a DataFrame?**
2. **Explain the difference between merge, join, and concat in pandas.**
3. **How can you apply a function to every element in a DataFrame or Series?**

**Advanced Operations:**

1. **How would you pivot a DataFrame?**
2. **Describe the groupby method and its uses.**
3. **What are the advantages of using the datetime functionality in pandas?**

**NumPy Basics:**

1. **What is an ndarray in NumPy? How is it different from a regular Python list?**
2. **How would you create an array of zeros or ones in NumPy?**
3. **Explain the concept of broadcasting in NumPy.**

**Array Operations:**

1. **How do you perform element-wise multiplication of two arrays?**
2. **What is the difference between dot and cross functions in NumPy?**
3. **How would you find the unique elements in a NumPy array?**

**Advanced Concepts:**

1. **How can you reshape a NumPy array?**
2. **Describe the use of slicing in NumPy arrays.**
3. **What are the main differences between numpy.array and numpy.matrix?**

**Python Data Structures:**

**Lists:**

1. **How do you add an element to the end of a list?**
2. **What is the difference between append() and extend() methods in a list?**
3. **How would you remove duplicates from a list?**

**Tuples:**

1. **Why are tuples considered immutable?**
2. **How would you convert a tuple to a list and vice versa?**
3. **When would you prefer using a tuple over a list?**

**Dictionaries:**

1. **How do you retrieve a value from a dictionary without causing an error if the key doesn't exist?**
2. **Explain the difference between dict.keys(), dict.values(), and dict.items().**
3. **How would you merge two dictionaries?**

**Sets:**

1. **What are the main use cases for sets in Python?**
2. **How do you perform set union and intersection operations?**
3. **What is the difference between a set and a frozenset?**

**Strings:**

1. **How would you reverse a string in Python?**
2. **Describe the difference between a string's find() and index() methods.**
3. **How would you check if a string is a palindrome?**
4. **Hands-on Coding Questions:**

**Lists:**

1. **Given a list of integers, write a function to move all zeros to the end without changing the order of non-zero elements.**
2. **Implement a function to find the second largest number in a list.**

**Tuples:**

1. **Write a function to compute the number of occurrences of each element in a tuple.**
2. **Dictionaries:**
3. **Implement a function that takes two lists as input: one list of keys and another list of values, and returns a dictionary.**
4. **Given a string, write a function to count the frequency of each character using a dictionary.**

**Sets:**

1. **Given two lists, write a function to find the common elements between them using sets.**
2. **Implement a function to check if a given set is a subset of another set.**
3. **Strings:**
4. **Write a function to determine if two strings are anagrams of each other.**
5. **Implement a function to find the first non-repeated character in a string.**

**Complex Data Structure Questions:**

**Lists:**

1. **Describe an algorithm to find the longest increasing subsequence in a list of numbers.**
2. **How would you implement an algorithm to rotate a list?**

**Dictionaries:**

1. **How would you design a least recently used (LRU) cache using dictionaries and linked lists?**
2. **Describe a method to find the first non-repeating character in a string using dictionaries.**

**Trees and Graphs:**

1. **Explain how you would implement a binary search tree. How would you ensure it remains balanced?**
2. **Describe an algorithm to detect a cycle in a directed graph.**

**Stacks and Queues:**

1. **How would you implement a function to sort a stack?**
2. **Describe the use of two queues to implement a stack.**

**Strings:**

1. **How would you check if a given string has all unique characters without using additional data structures?**
2. **Describe an algorithm to compress a string using counts of repeated characters. For instance, "aaabbcc" becomes "a3b2c2".**

**Hands-on Coding Questions with Sample Datasets:**

**Lists:**

1. **Given a list of stock prices (e.g., [100, 180, 260, 310, 40, 535, 695]), design an algorithm to find the maximum profit that can be earned by buying and selling the stock multiple times.**

**Dictionaries:**

1. **Given a dataset of student names and their grades (e.g., {'John': 85, 'Jane': 92, 'Doe': 78}), write a function to find the student with the highest average grade.**

**Trees:**

1. **Given a dataset representing a family tree (e.g., {'John': ['Jane', 'Doe'], 'Jane': ['Mary', 'Bob']} where the key is a parent and the value is a list of children), implement a function to find the ancestors of a given individual.**

**Stacks and Queues:**

1. **Given a sequence of opened and closed brackets (e.g., "{[()]}"), design an algorithm to check if they are balanced.**

**Strings:**

1. **Given a list of strings (e.g., ["apple", "boy", "cat", "dog", "egg", "fog"]), implement a function to group anagrams together.**

**Real-world Data Structure Questions:**

**Lists:**

1. **Imagine you're building a music streaming service. Users can add songs to a playlist. How would you design a feature to suggest the "next song" based on the current playlist's content?**

**Dictionaries:**

1. **You're tasked with building a voting system for a new feature request platform. Users can submit feature ideas and vote on them. How would you store and retrieve the number of votes for each feature efficiently?**

**Trees and Graphs:**

1. **Consider a company's organizational structure. Employees report to managers, managers report to VPs, and so on. How would you represent this hierarchy? How would you find the shortest path between two employees in terms of their reporting structure?**

**Stacks and Queues:**

1. **You're developing a print queue system for a large office. Print jobs have different priorities. How would you ensure that higher priority jobs are printed before lower priority ones, but also ensuring that low priority jobs aren't waiting forever?**

**Strings:**

1. **You're building a search feature for a large online bookstore. How would you design an algorithm to provide search suggestions as users type in the search box?**

**Real-world Hands-on Coding Scenarios:**

**Lists:**

**You have a list of transactions for an e-commerce website. Each transaction has a user ID and an amount. Write a function to find the top 5 users with the highest total transaction amounts.**

**Dictionaries:**

**Given a chat application, users send messages in different languages. You have a dictionary that maps each language to the number of messages sent in that language. Implement a function to display the top 3 most used languages in the chat.**

**Trees:**

**You're building a file system. Each folder can contain files or other folders. Design a function to display all the files under a given folder, including those nested in sub-folders.**

**Stacks and Queues:**

**Imagine a customer support ticketing system. Tickets come in at different times but have different priority levels. Write a function to retrieve the next ticket to be addressed based on its priority and time of arrival.**

**Strings:**

**You're working on a content management system. Users can tag their articles with keywords. Given a list of articles with their associated tags, design a function to find all articles that have a specific set of tags.  
  
Lists:**

**Given a list of flight itineraries represented as pairs of cities [from, to], reconstruct the itinerary in order. For example, given [['Mumbai', 'Paris'], ['Paris', 'Dubai'], ['Dubai', 'New York']], return ['Mumbai', 'Paris', 'Dubai', 'New York'].**

**Dictionaries:**

**You have a list of products and their respective prices. Write a function that takes a budget as input and returns all possible products that can be bought without exceeding the budget.**

**Trees:**

**Implement a function to serialize and deserialize a binary tree. Serialization is converting a tree to a string representation, and deserialization is converting the string back to the original tree structure.**

**Stacks and Queues:**

**Design a function that simulates a basic calculator to evaluate a simple expression string containing non-negative integers, +, -, \*, and / operators. Assume the expression is always valid.**

**Strings:**

**Implement a function that checks if a given word can be formed by combining two words from a dictionary. For example, given the word "newspaper" and a dictionary containing ["news", "paper", "new", "spa"], the function should return True as "newspaper" can be formed by "news" and "paper".**

**Graphs:**

**Given a list of tasks and a list of prerequisites (represented as pairs), determine a valid order in which tasks can be completed. If no valid order exists, return an empty list.**

**Arrays:**

**Write a function that rotates an NxN matrix 90 degrees.**

**Linked Lists:**

**Implement a function to detect a cycle in a linked list. If a cycle exists, determine the starting node of the cycle.**

**Dynamic Programming:**

**Given a list of coin denominations and a total amount, write a function to compute the fewest number of coins needed to make up that amount. If the amount cannot be made up by any combination of the coins, return -1.**

**Searching and Sorting:**

**Implement a function to search for a specific element in a rotated sorted array. For instance, in the array [4,5,6,7,0,1,2], the target 0 would return the index 4.**