

AI:- To create an application which can perform its own task without any human interaction.
Ex:- Netflix, Youtube, Amazon Prime they are all Recommended Systems.
Self Driving Car

ML:- It provides stats to analyze, visualize, predication and forecasting the data.

Data Science:- Data Science is an interdisciplinary field that combines statistical analysis, machine learning, data engineering, and domain expertise to extract meaningful insights and knowledge from structured and unstructured data. It involves collecting, processing, analyzing, and interpreting large volumes of data to solve complex problems, make data-driven decisions, and predict future trends.

Types Of Machine Learning:-

1. Supervised ML Technique:-

House Price Prediction

Dataset		
	Independent feature	Dependent or output feature
Size of House	# of Rooms	Price
5000	5	450K
6000	6	500K
-	-	-
-	-	-

Continuous

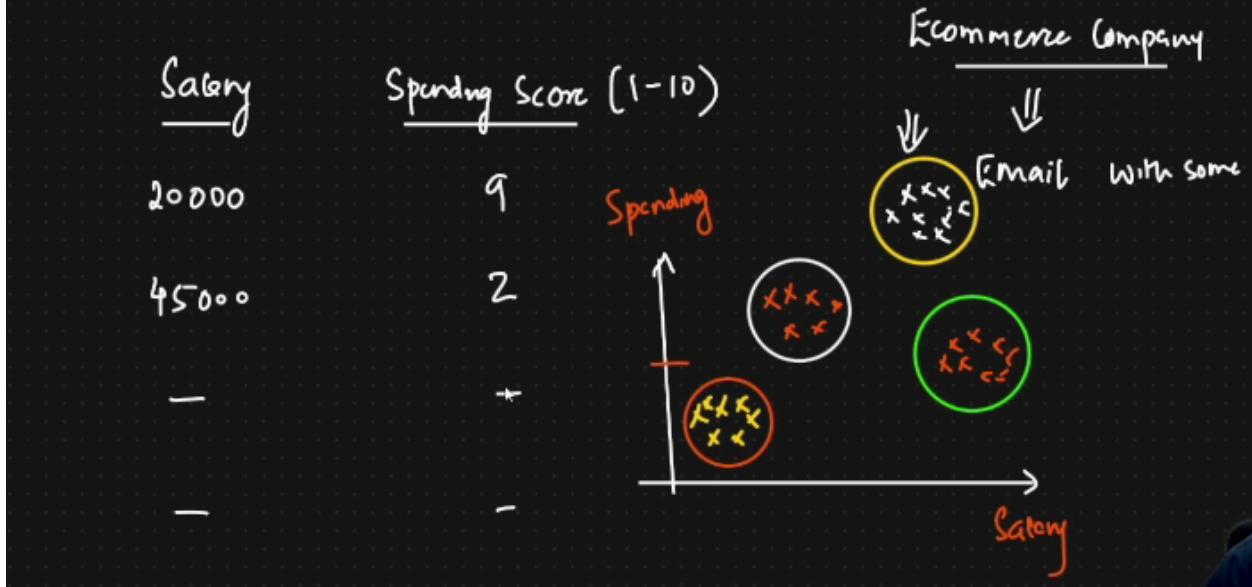
A. Classification Problem:

B. Regression Problem:

1. Dependent OR Output Feature
2. If Continuous >>> Regression
3. If Categorical >>> Classification

2. Unsupervised Machine Learning:-

② Unsupervised ML : Eg: Customer Segmentation \rightarrow Clusters



Supervised ML Algorithm :- These models are trained on labeled data, where the input data is paired with the correct output.

1. Linear Regression
2. Ridge & Lasso
3. ElasticNet
4. Logistic Regression (Classification)
5. Decision Tree
6. Random Forest
7. AdaBoost
8. Xgboost

Unsupervised ML Algorithm :-

These models are trained on unlabeled data, and the goal is to find hidden patterns or intrinsic structures in the input data.

1. K Means
2. Hierarchical Mean
3. DBScan Clustering

Reinforcement Learning Models:-

These models learn by interacting with an environment and receiving feedback in the form of rewards or penalties.

- Q-Learning
- Deep Q-Networks (DQN)
- Policy Gradient Methods

Supervised Learning Linear Models >>>

1. Linear Regression Algorithm:- A simple algorithm that models a linear relationship between inputs and a continuous numerical output variable.

Applications:-

Use Cases >>>

1. Stock price prediction
2. Predicting housing prices
3. Predicting Customer lifetime value

Advantages

1. Explainable method
2. Interpretable results by its output coefficients
3. Faster to train than other machine learning models

Disadvantages

1. Assumes linearity between inputs and output
2. Sensitive to outliers
3. Can underfit with small, high-dimensional data

2. Logistic Regression Algorithm:- A simple algorithm that models a linear relationship between inputs and a categorical output (1 or 0).

Applications:-

Use Cases >>>

1. Credit risk score prediction
2. Customer churn prediction

Advantages

1. Interpretable and explainable
2. Less prone to overfitting when using regularization
3. Applicable for multi-class predictions

Disadvantages

1. Assumes linearity between inputs and outputs
2. Can overfit with small, high-dimensional data

3. Ridge Regression Algorithm:- Part of the regression family — it penalizes features that have low predictive outcomes by shrinking their coefficients closer to zero. Can be used for classification or regression.

Applications:-

Use Cases >>>

1. Predictive maintenance for automobiles
2. Sales revenue prediction

Advantages

1. Less prone to overfitting
2. Best suited where data suffer from multicollinearity
3. Explainable & interpretable

Disadvantages

1. All the predictors are kept in the final model
2. Doesn't perform feature selection

4. Lasso Regression Algorithm:- Part of the regression family — it penalizes features that have low predictive outcomes by shrinking their coefficients to zero. Can be used for classification or regression.

Applications:-

Use Cases >>>

1. Predicting housing prices
2. Predicting clinical outcomes based on health data

Advantages

1. Less prone to overfitting
2. Can handle high-dimensional data
3. No need for feature selection

Disadvantages

1. Can lead to poor interpretability as it can keep highly correlated variables

Supervised Learning Tree-Based Models

1. Decision Tree Algorithm:- Decision Tree models make decision rules on the features to produce predictions. It can be used for classification or regression.

Applications:-

Use Cases >>>

1. Customer churn prediction
2. Credit score modeling
3. Disease prediction

Advantages

- 1.Explainable and interpretable
2. Can handle missing values

Disadvantages

- 1.Prone to overfitting
- 2.Sensitive to outliers

2. Random Forests Algorithm:- An ensemble learning method that combines the output of multiple decision trees.

Applications:-

Use Cases >>>

- 1.Credit score modeling
- 2.Predicting housing prices

Advantages

- 1.Reduces overfitting
- 2.Higher accuracy compared to other models

Disadvantages

- 1.Training complexity can be high
- 2.Not very interpretable

3.Gradient Boosting Regression Algorithm:- Gradient Boosting Regression employs boosting to make predictive models from an ensemble of weak predictive learners.

Applications:-

Use Cases >>>

- 1.Predicting car emissions
- 2.Predicting ride hailing fare amount

Advantages

1. Better accuracy compared to other regression models
2. It can handle multicollinearity
3. It can handle non-linear relationships

Disadvantages

- 1.Sensitive to outliers and can therefore cause overfitting
- 2.Computationally expensive and has high complexity

4.Gradient Boosting algorithm (XGBoost):- Gradient Boosting algorithm that is efficient & flexible. Can be used for both classification and regression tasks.

Applications:-

Use Cases >>>

- 1.Churn prediction
- 2.Claims processing in insurance

Advantages

- 1.Provides accurate results
- 2.Captures non linear relationships

Disadvantages

- 1.Hyperparameter tuning can be complex
- 2.Does not perform well on sparse datasets

5.LightGBM Regressor Algorithm:- A gradient boosting framework that is designed to be more efficient than other implementations.

Applications:-

Use Cases >>>

1. Predicting flight time for airlines
2. Predicting cholesterol levels based on health data

Advantages

- 1.Can handle large amounts of data
- 2.Computational efficient & fast training speed
- 3.Low memory usage

Disadvantages

- 1.Can overfit due to leaf-wise splitting and high sensitivity
- 2.Hyperparameter tuning can be complex

Unsupervised Learning Clustering Based Models:-

1.K-Means Clustering Algorithm:- K-Means is the most widely used clustering approach—it determines K clusters based on euclidean distances.

Applications:-

Use Cases >>>

- 1.Customer segmentation

2. Recommendation systems

Advantages

- 1. Scales to large datasets
- 2. Simple to implement and interpret
- 3. Results in tight clusters

Disadvantages

- 1. Requires the expected number of clusters from the beginning
- 2. Has troubles with varying cluster sizes and densities

2. Hierarchical Clustering Algorithm:- A bottom-up approach where each data point is treated as its own cluster—and then the closest two clusters are merged together iteratively.

Applications:-

- Use Cases >>>
- 1. Fraud detection
 - 2. Document clustering based on similarity

Advantages

- 1. There is no need to specify the number of clusters
- 2. The resulting dendrogram is informative

Disadvantages

- 1. Doesn't always result in the best clustering
- 2. Not suitable for large datasets due to high complexity

3. Gaussian Mixture Models Algorithm :- A probabilistic model for modeling normally distributed clusters within a dataset.

Applications:-

- Use Cases >>>
- 1. Customer segmentation
 - 2. Recommendation systems

Advantages

- 1. Computes a probability for an observation belonging to a cluster
- 2. Can identify overlapping clusters
- 3. More accurate results compared to K-means

Disadvantages

- 1. Requires complex tuning
- 2. Requires setting the number of expected mixture components or clusters

Unsupervised Learning Associations:-

Apriori algorithm:- Rule based approach that identifies the most frequent itemset in a given dataset where prior knowledge of frequent itemset properties is used.

Applications:-

Use Cases >>>

1. Product placements
2. Recommendation engines
3. Promotion optimization

Advantages

1. Results are intuitive and Interpretable
2. Exhaustive approach as it finds all rules based on the confidence and support

Disadvantages

1. Generates many uninteresting itemsets
2. Computationally and memory intensive
3. Results in many overlapping item sets

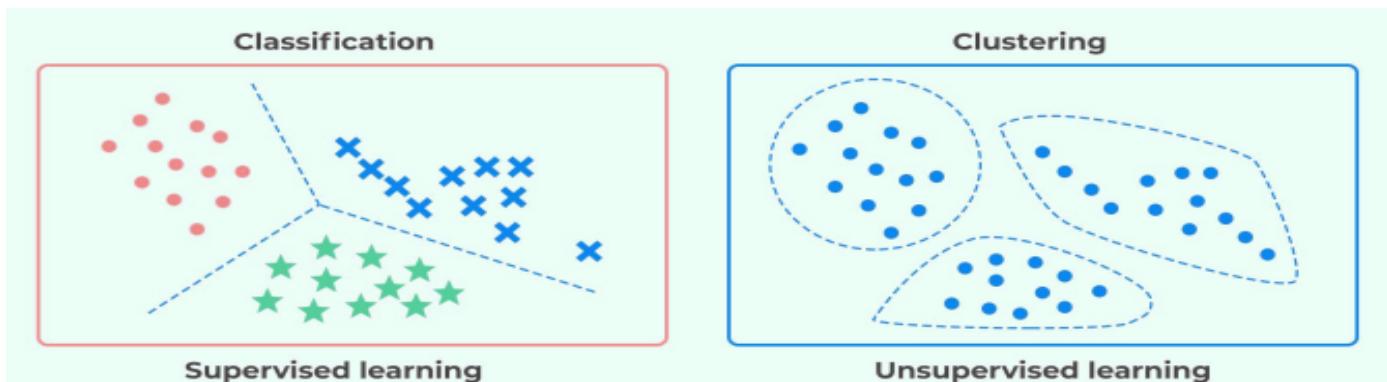
<<<< DATA SCIENTIST INTERVIEW QUESTIONS AND ANSWERS >>>>

1. What is the role of a data scientist in an organisation?

A data scientist is responsible for collecting, analysing, and interpreting complex data to help organisations make informed decisions.

2. Explain the difference between supervised and unsupervised learning.

Supervised learning uses labelled data for training, while unsupervised learning works with unlabeled data to find hidden patterns or relationships.



3. What is cross validation, and why is it important?

Cross validation is a technique used to assess how well a model generalises to an independent dataset. It is important for evaluating a model's performance and preventing overfitting.

4. Can you explain the steps involved in the data preprocessing process?

Data preprocessing includes data cleaning, handling missing values, data transformation, normalisation, and standardisation to prepare the data for analysis and modelling.

5. What are some common algorithms used in machine learning?

Common machine learning algorithms include linear regression, logistic regression, decision trees, random forests, support vector machines, and neural networks.

6. How do you handle missing data in a dataset?

Missing data can be handled by either removing the rows with missing values, imputing the missing values using statistical techniques, or using advanced imputation methods such as K-Nearest Neighbors.



7. What is the purpose of the K-Means clustering algorithm?

The K-Means algorithm is used for partitioning a dataset into K clusters, aiming to minimise the sum of squares within each cluster.

8. How do you assess the performance of a machine learning model?

Model performance can be assessed using metrics such as accuracy, precision, recall, F1 score, and the ROC curve for classification tasks, and metrics such as mean squared error for regression tasks

9. Explain the term 'bias' in the context of machine learning models.

Bias refers to the error introduced by approximating a real-world problem, often due to oversimplification of the model. High bias can result in underfitting.

10. What is the importance of feature scaling in machine learning?

Feature scaling ensures that the features are at a similar scale, preventing certain features from dominating the learning process and helping the algorithm converge faster.

11. Can you explain the concept of regularisation in machine learning?

Regularisation is a technique used to prevent overfitting by adding a penalty term to the loss function, discouraging complex models.

12. What is the difference between L1 and L2 regularisation?

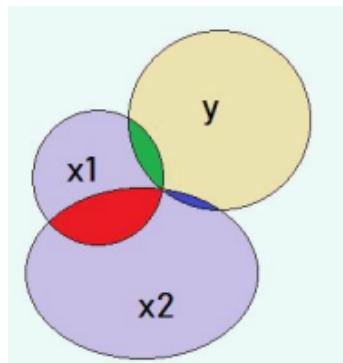
L1 regularisation adds the absolute value of the magnitude of coefficients as a penalty term, while L2 regularisation adds the square of the magnitude of coefficients as a penalty term.

13. What is the purpose of a confusion matrix in classification tasks?

A confusion matrix is used to visualise the performance of a classification model, showing the counts of true positive, true negative, false positive, and false negative predictions.

14. How do you handle multicollinearity in a dataset?

Multicollinearity can be handled by techniques such as removing one of the correlated features, using principal component analysis, or using regularisation techniques to reduce the impact of correlated features.

**15. Can you explain the difference between precision and recall?**

Precision refers to the ratio of correctly predicted positive observations to the total predicted positive observations, while recall refers to the ratio of correctly predicted positive observations to the total actual positive observations.

16. What is the purpose of the Naive Bayes algorithm in machine learning?

The Naive Bayes algorithm is used for classification tasks, based on the Bayes theorem with the assumption of independence between features.

17. How do you handle outliers in a dataset?

Outliers can be handled by either removing them if they are due to data entry errors, or by transforming them using techniques such as winsorization or log transformation.

18. Explain the concept of the Central Limit Theorem.

The Central Limit Theorem states that the sampling distribution of the sample means approaches a normal distribution as the sample size increases, regardless of the shape of the population distribution.

19. What is the purpose of a decision tree algorithm in machine learning?

Decision trees are used for both classification and regression tasks, creating a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

20. Can you explain the concept of ensemble learning?

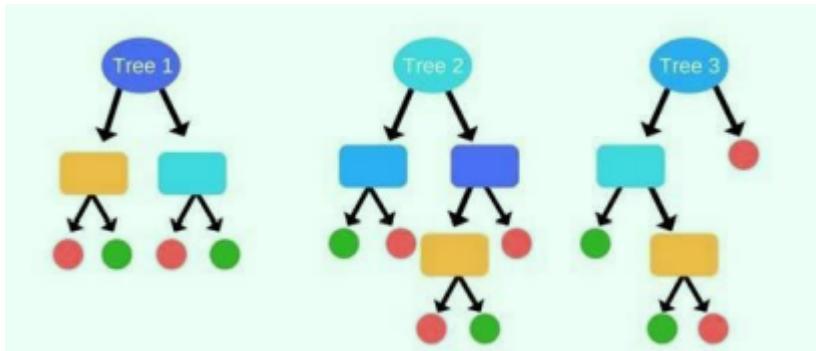
Ensemble learning involves combining multiple individual models to improve the overall performance and predictive power of the learning algorithm.

21. What is the difference between bagging and boosting?

Bagging involves training each model in the ensemble with a subset of the data, while boosting focuses on training each model sequentially, giving more weight to the misclassified data points.

22. Explain the purpose of the Random Forest algorithm in machine learning

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes or the mean prediction of the individual trees for classification and regression tasks, respectively.



23. How do you select the optimal number of clusters in a K-Means clustering algorithm?

The optimal number of clusters can be determined using techniques such as the elbow method, silhouette score, or the gap statistic.

24. What is the purpose of the Support Vector Machine (SVM) algorithm?

Support Vector Machines are used for classification and regression analysis, with the primary goal of finding the hyperplane that best separates the classes.

25. How do you handle a large volume of data that cannot fit into memory?

Large volumes of data can be handled using techniques such as data streaming, distributed computing frameworks like Hadoop or Spark, and data compression techniques.

26. Can you explain the purpose of a recommendation system?

Recommendation systems are used to predict and recommend items or products that a user may be interested in, based on their past preferences or behaviour.

27. What is the purpose of Principal Component Analysis (PCA) in machine learning?

Principal Component Analysis is used for dimensionality reduction, transforming a large set of variables into a smaller set of uncorrelated variables while retaining most of the information.

28. How do you handle a situation where the data is too imbalanced?

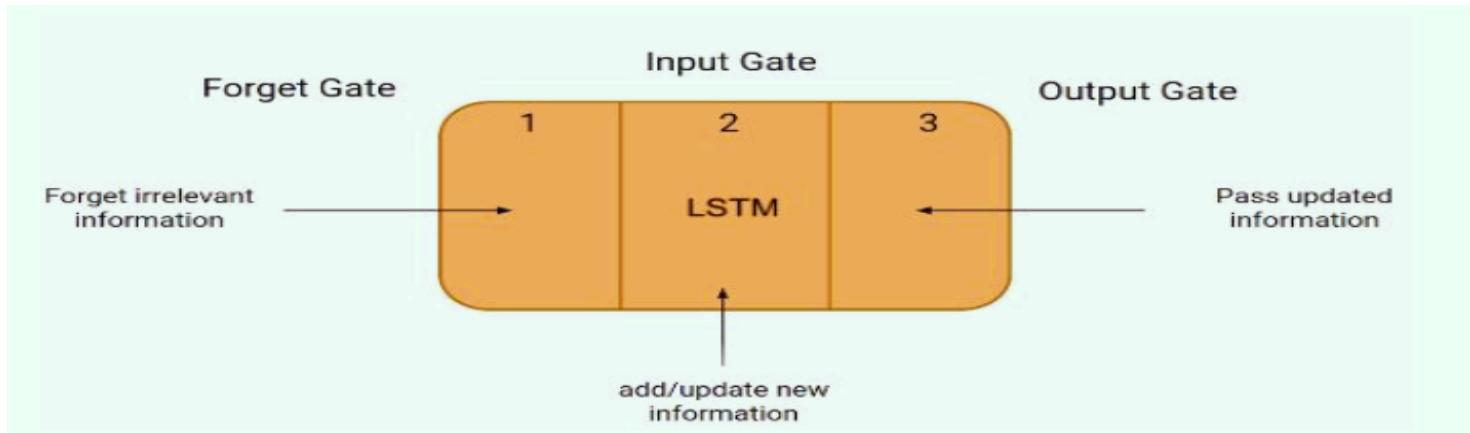
Imbalanced data can be handled using techniques such as oversampling the minority class, undersampling the majority class, or using algorithms specifically designed to handle imbalanced datasets.

29. What is the purpose of a Recurrent Neural Network (RNN) in deep learning?

Recurrent Neural Networks are used for sequence data, allowing information to persist over time, making them suitable for tasks such as natural language processing and time series analysis.

30. Explain the concept of a Long Short-Term Memory (LSTM) network.

LSTM networks are a type of RNN that addresses the vanishing gradient problem, making them more effective for learning and predicting sequences of data.



31. What is the purpose of the Word2Vec algorithm in natural language processing?

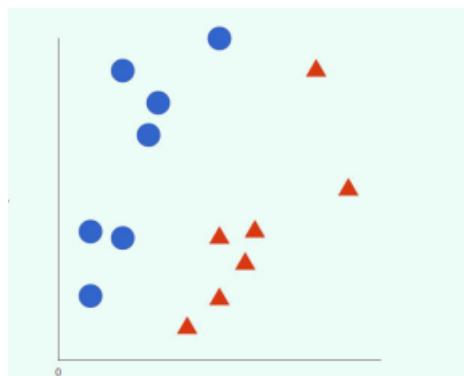
Word2Vec is used for learning word embeddings, representing words as vectors to capture semantic relationships between words in a text corpus.

32. How do you handle a situation where there are too many features compared to the number of observations?

The situation of having too many features compared to the number of observations can be handled by using feature selection techniques, such as Lasso regression, or by using dimensionality reduction techniques like PCA or t-SNE.

33. Explain the concept of a support vector in the context of a Support Vector Machine algorithm.

Support vectors are data points that lie closest to the decision boundary between the classes, influencing the position and orientation of the hyperplane in a Support Vector Machine.



34. What is the purpose of the Root Mean Square Error (RMSE) metric in regression tasks?

The Root Mean Square Error is a commonly used metric for evaluating the accuracy of a regression model by measuring the differences between the predicted values and the actual values.

35. Can you explain the purpose of the Apriori algorithm in association rule mining?

The Apriori algorithm is used for discovering frequent itemsets within a transactional database and is commonly employed in market basket analysis to identify patterns or relationships between different items.

36. How do you handle a situation where the data is highly skewed?

Highly skewed data can be handled by using transformations such as log transformations, square root transformations, or by using specialised models that can handle skewed data more effectively.

37. What is the purpose of the Mean Average Precision (MAP) metric in evaluating information retrieval systems?

Mean Average Precision is used to evaluate the performance of information retrieval systems, measuring the average precision at each relevant document retrieved across multiple queries.

38. Explain the purpose of the Euclidean distance metric in clustering tasks.

The Euclidean distance metric is used to measure the distance between two points in a multidimensional space and is commonly used in clustering algorithms such as K-Means.

39. How do you handle a situation where the data is not linearly separable?

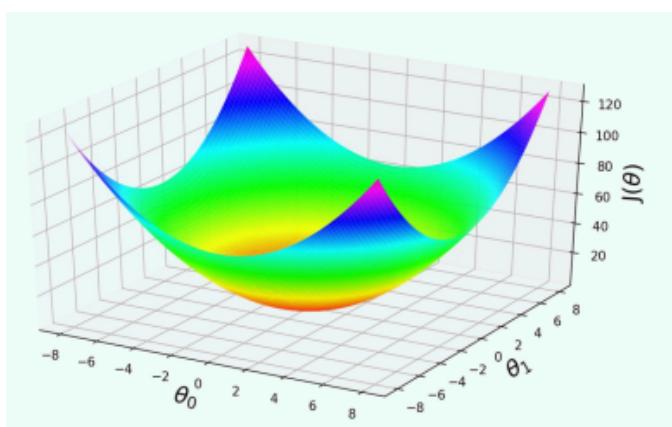
In cases where the data is not linearly separable, kernel functions can be used in algorithms like Support Vector Machines to map the data to a higher-dimensional space where it becomes linearly separable.

40. What is the purpose of the Chi-square test in feature selection?

The Chi-square test is used to determine the independence of two categorical variables, making it suitable for feature selection in classification tasks.

41. Can you explain the purpose of the Gradient Descent algorithm in machine learning?

Gradient Descent is an optimization algorithm used to minimize the cost function and find the optimal parameters of a model by iteratively updating the parameters in the direction of the steepest descent.



42. How do you handle a situation where the data is time-series data?

Time-series data can be handled using techniques such as autoregressive integrated moving average (ARIMA) models, exponential smoothing methods, or more advanced deep learning models like Long ShortTerm Memory (LSTM) networks.

43. What is the purpose of the K-Nearest Neighbors (KNN) algorithm in machine learning?

The K-Nearest Neighbors algorithm is used for classification and regression tasks, making predictions based on the majority vote of its k nearest neighbours.

44. Explain the purpose of the Log Loss metric in evaluating classification models.

Log Loss is used to evaluate the performance of a classification model that outputs probabilities, measuring the performance based on the likelihood of the predicted probabilities matching the actual labels.

45. How do you handle a situation where the data is high-dimensional?

High-dimensional data can be handled by using dimensionality reduction techniques such as Principal Component Analysis (PCA), t-Distributed Stochastic Neighbour Embedding (t-SNE), or by employing feature selection methods.

46. What is the purpose of the R-squared (R²) metric in evaluating regression models?

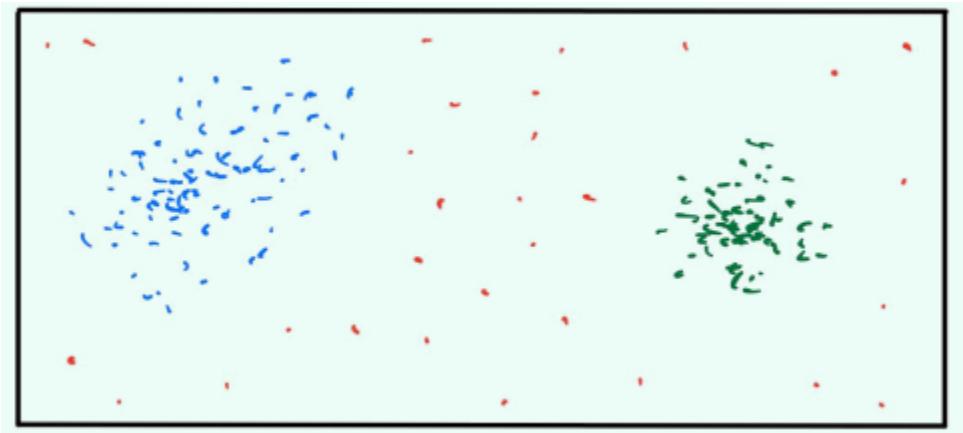
R-squared is a statistical measure that represents the proportion of the variance for a dependent variable that is explained by an independent variable in a regression model.

47. Can you explain the purpose of the Gini index in the context of a decision tree algorithm?

The Gini index is used to measure the impurity or the homogeneity of a node in a decision tree, helping to determine the best split for creating a more accurate decision tree.

48. How do you handle a situation where there is noise in the data?

Noise in the data can be handled by smoothing techniques such as moving averages, using robust statistics, or employing filtering methods to remove outliers and irrelevant data points.

**49. What is the purpose of the F1 score metric in evaluating classification models?**

The F1 score is the harmonic mean of precision and recall and is used to evaluate the balance between precision and recall in a classification model.

50. Can you explain the purpose of the LDA (Linear Discriminant Analysis) algorithm in machine learning?

Linear Discriminant Analysis is used for dimensionality reduction and classification tasks, aiming to find the linear combinations of features that best separate multiple classes in the data.

51. What is the difference between classification and regression in machine learning?

Classification is used to predict discrete categories, while regression is used to predict continuous quantities.

52. Can you explain the bias-variance trade-off in the context of model complexity?

The bias-variance trade-off highlights the trade-off between a model's ability to minimise errors due to bias and variance. Increasing model complexity reduces bias but increases variance and vice versa.

53. How do you handle imbalanced data sets when building a classification model?

Imbalanced datasets can be handled using techniques like oversampling, undersampling, or using algorithms designed for imbalanced data such as SMOTE (Synthetic Minority Over-sampling Technique).

54. Explain the purpose of the term 'regularisation' in machine learning models.

Regularisation is a technique used to prevent overfitting by adding a penalty term to the loss function, discouraging overly complex models.

55. What is the purpose of the term 'gradient descent' in the context of optimising a model?

Gradient descent is an iterative optimization algorithm used to minimize the cost function of a model by adjusting the model's parameters in the direction of steepest descent.

56. How do you assess the performance of a classification model apart from accuracy?

The performance of a classification model can be evaluated using metrics such as precision, recall, F1 score, and the area under the ROC curve.

57. Can you explain the concept of 'feature selection' and its importance in model building?

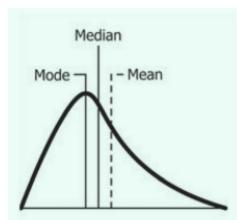
Feature selection involves selecting the most relevant features from a dataset. It is crucial for improving model performance, reducing overfitting, and enhancing interpretability.

58. What is the purpose of the term 'cross-validation' in model training and evaluation?

Cross-validation is used to assess how well a model generalises to an independent dataset, minimising the risk of overfitting and providing a more accurate estimate of the model's performance.

59. How do you handle missing data in a dataset while building a predictive model?

Missing data can be handled by techniques such as mean/median imputation, mode imputation, or using advanced methods like multiple imputation or K-Nearest Neighbors imputation.



60. Explain the purpose of the term 'ensemble learning' and its benefits in model building.

Ensemble learning involves combining multiple models to improve predictive performance and reduce overfitting, often resulting in better generalisation and more robust predictions.

61. What is the difference between unsupervised and supervised machine learning algorithms?

Supervised learning uses labelled data for training, while unsupervised learning works with unlabeled data to find patterns and relationships.

62. Can you explain the concept of 'clustering' and provide an example of when it is used?

Clustering is an unsupervised learning technique used to group similar data points together. An example is customer segmentation in marketing.

63. What is the purpose of 'dimensionality reduction' in data analysis, and how is it achieved?

Dimensionality reduction is used to reduce the number of features in a dataset. It is achieved through techniques like principal component analysis (PCA) and t-distributed stochastic neighbour embedding (tSNE).

64. How do you handle the problem of overfitting in machine learning models?

Overfitting can be mitigated by using techniques like cross-validation, regularisation, early stopping, and reducing model complexity.

65. Explain the purpose of the term 'Naive Bayes' in machine learning and its application.

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem with an assumption of independence between features. It is commonly used for text classification and spam filtering.

66. What is the purpose of the term 'decision trees' in machine learning, and how does it work?

Decision trees are predictive models that map features to conclusions about the target value. They work by splitting the dataset into smaller subsets based on the most significant differentiators in the data.

67. How do you handle the problem of multicollinearity in a dataset?

Multicollinearity can be addressed by techniques such as removing one of the correlated features, using principal component analysis (PCA), or using regularisation methods.

68. Can you explain the purpose of the term 'random forest' in machine learning and its advantages?

Random forests are an ensemble learning method that constructs multiple decision trees during training. They are effective for reducing overfitting and handling large datasets with high dimensionality.

69. What is the purpose of 'data preprocessing' in machine learning, and what are some common techniques used?

Data preprocessing involves preparing and cleaning data before it is fed into a machine learning model. Common techniques include data normalisation, standardisation, and handling missing values.



70. How do you handle the problem of underfitting in a machine learning model?

Underfitting can be addressed by using more complex models, adding more features, or reducing regularisation, allowing the model to capture more complex patterns in the data.

71. Explain the concept of 'hyperparameter tuning' in machine learning algorithms.

Hyperparameter tuning involves finding the best set of hyperparameters for a machine learning model to optimise its performance and generalisation.

72. What is the purpose of 'ANOVA' (Analysis of Variance) in statistical analysis, and when is it used?

ANOVA is used to analyse the differences among group means and is applied when comparing means of more than two groups to determine whether they are statistically significantly different.

73. How do you handle a situation where the data has outliers?

Outliers can be handled by removing them if they are due to data entry errors or by transforming them using techniques such as winsorization or log transformation.

74. Explain the concept of 'bias' in machine learning models.

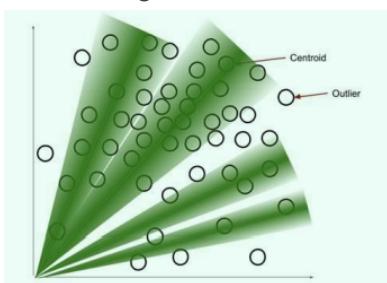
Bias refers to the error introduced by approximating a real-world problem, often due to oversimplification of the model. High bias can lead to underfitting.

75. What is the purpose of the 'mean squared error' metric in regression analysis?

Mean squared error is a commonly used metric for evaluating the performance of a regression model by measuring the average of the squares of the differences between predicted and actual values.

76. Can you explain the purpose of the term 'cosine similarity' in similarity measurements?

Cosine similarity is a metric used to measure the similarity between two non-zero vectors, often used in text mining and collaborative filtering.



77. How do you handle a situation where the data has a time component?

Data with a time component can be analysed using time series analysis techniques such as autoregressive integrated moving average (ARIMA) models, exponential smoothing, or Prophet forecasting models.

78. Explain the concept of 'precision' and 'recall' in the context of classification models.

Precision measures the proportion of true positive results among the predicted positive results, while recall measures the proportion of true positive results among the actual positive results.

79. What is the purpose of the 'Hadoop' framework in big data processing, and how is it used?

Hadoop is an open-source framework used for distributed storage and processing of large data sets across clusters of computers using simple programming models.



80. How do you handle a situation where the data has a lot of noise?

Noisy data can be managed through techniques such as data smoothing, filtering, or by using robust statistical measures that are less sensitive to outliers.

81. Explain the concept of 'correlation' in statistics and its different types.

Correlation measures the relationship between two variables and can be positive, negative, or zero, indicating the strength and direction of the relationship.

82. What is the purpose of the 'k-nearest neighbours' algorithm in machine learning, and how does it work?

The k-nearest neighbours algorithm is used for classification and regression tasks, making predictions based on the majority vote or averaging the values of the k nearest neighbours.

83. How do you handle a situation where the data has a lot of categorical variables?

Categorical variables can be handled through techniques such as one-hot encoding, label encoding, or using target encoding to convert them into a format suitable for machine learning models.

84. Explain the purpose of the 'SVM' (Support Vector Machine) algorithm in machine learning, and its advantages.

Support Vector Machines are supervised learning models used for classification and regression analysis. They are effective in high-dimensional spaces and work well with complex datasets.

85. What is the purpose of the 'LSTM' (Long Short-Term Memory) network in deep learning, and how is it used?

LSTM networks are a type of recurrent neural network (RNN) used for processing and making predictions based on sequential data, often used in natural language processing and time series analysis.

86. Can you explain the purpose of the term 'Principal Component Analysis' (PCA) in dimensionality reduction, and how is it used?

Principal Component Analysis is a technique used to reduce the dimensionality of a dataset while preserving as much variance as possible. It transforms the original variables into a new set of variables, the principal components, which are orthogonal and uncorrelated. This aids in simplifying the dataset and speeding up the subsequent learning algorithms while retaining most of the essential information.

87. Explain the concept of 'k-means clustering' and its application in unsupervised learning.

K-means clustering is a popular unsupervised learning algorithm used for partitioning a dataset into K clusters based on similarities in the data points.

88. What is the purpose of the 'R-squared' metric in regression analysis, and what does it indicate about the model's fit?

R-squared is a statistical measure that represents the proportion of the variance for a dependent variable explained by the independent variables in a regression model. It indicates the goodness of fit of the model.

89. What is the purpose of the term 't-Distributed Stochastic Neighbour Embedding' (t-SNE) in dimensionality reduction, and how is it used?

t-Distributed Stochastic Neighbour Embedding is a nonlinear dimensionality reduction technique used for visualising high-dimensional data in a low dimensional space. It is particularly useful for visualising complex datasets and identifying patterns or clusters within the data.

90. Explain the purpose of the 'F1 score' metric in evaluating classification models and its relationship with precision and recall.

The F1 score is the harmonic mean of precision and recall and is used to evaluate the balance between precision and recall in a classification model.

91. Can you explain the concept of 'backpropagation' in neural networks and its role in training the model?

Backpropagation is an algorithm used to train artificial neural networks by adjusting the weights of the connections in the network to minimize the difference between predicted and actual outputs.

92. What is the purpose of the 'chi-square test' in statistics, and when is it used?

The chi-square test is used to determine the independence of two categorical variables and is often used to test the significance of relationships between variables in a contingency table.

93. How do you handle a situation where the data is not normally distributed?

Non-normally distributed data can be transformed using techniques such as the Box-Cox transformation, Yeo-Johnson transformation, or log transformation to approximate a normal distribution.

94. Explain the concept of 'latent variables' in the context of factor analysis and its importance.

Latent variables are variables that are not directly observed but are inferred from observed variables. They are crucial for capturing underlying factors and reducing the dimensionality of the data.

95. What is the purpose of the 'Gini index' in decision trees, and how is it used in the context of building the tree?

The Gini index is a metric used to measure the impurity of a node in a decision tree. It is used to find the best split for creating a more accurate decision tree.

96. How do you handle a situation where the data has a lot of continuous variables?

Continuous variables can be handled through techniques such as scaling and normalisation to ensure that the variables are on a similar scale, preventing certain features from dominating the learning process.

97. Explain the purpose of 'association rules' in data mining, and provide an example of its application.

Association rules are used to discover interesting relationships between variables in large datasets. An example is market basket analysis used to identify products frequently purchased together.

98. What is the purpose of the 'logistic function' in logistic regression, and how is it used for binary classification?

The logistic function is used to model the probability of a binary outcome. It maps any real-valued number to a value between 0 and 1, making it suitable for binary classification tasks.

99. How do you handle a situation where the data has a lot of missing values?

Data with missing values can be managed through techniques such as imputation, using algorithms like K-Nearest Neighbours, decision trees, or employing advanced techniques like deep learning-based imputation.

100. Explain the concept of 'bagging' and 'boosting' in ensemble learning, and provide an example of when each technique is used.

Bagging involves training multiple models independently and combining their predictions, while boosting trains models sequentially, giving more weight to misclassified data points. Bagging is used for reducing variance, while boosting is used for reducing bias in ensemble models.

101. What is Data Science?

Data Science is an interdisciplinary field focused on extracting knowledge and insights from data using scientific methods, algorithms, and systems. It combines aspects of statistics, computer science, and domain expertise.

102. What are the differences between supervised and unsupervised learning?

Supervised learning involves training a model on labeled data, whereas unsupervised learning involves training a model on data without labels to find hidden patterns.

103. What is the difference between overfitting and underfitting?

Overfitting occurs when a model learns the noise in the training data, performing well on training data but poorly on new data. Underfitting occurs when a model is too simple to capture the underlying patterns in the data, performing poorly on both training and new data.

104. Explain the bias-variance tradeoff.

The bias-variance tradeoff is the balance between two sources of error that affect model performance. Bias is the error due to overly simplistic models, while variance is the error due to models being too complex. A good model should find the right balance between bias and variance.

105. What is the difference between parametric and non-parametric models?

Parametric models assume a specific form for the function that maps inputs to outputs and have a fixed number of parameters. Non-parametric models do not assume a specific form and can grow in complexity with the data.

106. What is cross-validation?

Cross-validation is a technique for assessing how a predictive model will generalize to an independent dataset. It involves partitioning the data into subsets, training the model on some subsets, and validating it on the remaining subsets.

107. What is a confusion matrix?

A confusion matrix is a table used to evaluate the performance of a classification model. It shows the counts of true positives, true negatives, false positives, and false negatives.

108. What is regularization, and why is it useful?

Regularization is a technique to prevent overfitting by adding a penalty to the model's complexity. Common types include L1 (Lasso) and L2 (Ridge) regularization.

109. What is the Central Limit Theorem?

The Central Limit Theorem states that the distribution of sample means approaches a normal distribution as the sample size becomes large, regardless of the original distribution of the data.

110. What are precision and recall?

Precision is the ratio of true positives to the sum of true and false positives, while recall is the ratio of true positives to the sum of true positives and false negatives.

111. Explain the ROC curve and AUC?

The ROC curve is a graphical representation of a classifier's performance, plotting the true positive rate against the false positive rate. AUC (Area Under the Curve) measures the entire two-dimensional area underneath the ROC curve.

112. What is a p-value?

A p-value measures the probability of obtaining test results at least as extreme as the observed results, assuming that the null hypothesis is true. It helps determine the statistical significance of the results.

113. What are the assumptions of linear regression?

Assumptions include linearity, independence, homoscedasticity (constant variance), normality of residuals, and no multicollinearity.

114. What is multicollinearity, and how can it be detected?

Multicollinearity occurs when independent variables in a regression model are highly correlated. It can be detected using Variance Inflation Factor (VIF) or correlation matrices.

115. : Explain the k-means clustering algorithm.

K-means is an unsupervised learning algorithm that partitions data into k clusters by minimizing the variance within each cluster. It iteratively assigns data points to the nearest centroid and updates centroids based on the mean of the points in each cluster.

116. What is a decision tree, and how does it work?

A decision tree is a flowchart-like structure used for classification and regression. It splits data into subsets based on the value of input features, creating branches until a decision is made at the leaf nodes.

117. How does the random forest algorithm work?

Random forest is an ensemble learning method that combines multiple decision trees to improve accuracy and control overfitting. It uses bootstrap sampling and random feature selection to build each tree.

118. What is gradient boosting?

Gradient boosting is an ensemble technique that builds models sequentially, with each new model attempting to correct the errors of the previous ones. It combines weak learners to form a strong learner.

119. Explain principal component analysis (PCA).

PCA is a dimensionality reduction technique that transforms data into a new coordinate system by projecting it onto principal components, which are orthogonal and capture the maximum variance in the data.

120. What is the curse of dimensionality?

The curse of dimensionality refers to the challenges and issues that arise when analyzing and organizing data in high-dimensional spaces. As the number of dimensions increases, the volume of the space increases exponentially, making data sparse and difficult to manage.

121. Explain the difference between bagging and boosting.

Bagging (Bootstrap Aggregating) is an ensemble method that trains multiple models independently using different subsets of the training data and averages their predictions. Boosting trains models sequentially, where each model focuses on correcting the errors of the previous ones.

122. What is the difference between L1 and L2 regularization?

L1 regularization (Lasso) adds the absolute value of the coefficients as a penalty term, promoting sparsity. L2 regularization (Ridge) adds the squared value of the coefficients as a penalty term, leading to smaller but nonzero coefficients.

123. What is the difference between a generative and discriminative model?

Generative models learn the joint probability distribution of input features and output labels and can generate new data points. Discriminative models learn the conditional probability of the output labels given the input features, focusing on the decision boundary.

124. Explain the backpropagation algorithm in neural networks.

Backpropagation is an algorithm used to train neural networks by calculating the gradient of the loss function with respect to each weight and updating the weights in the opposite direction of the gradient to minimize the loss.

125. What is the vanishing gradient problem?

The vanishing gradient problem occurs when the gradients used to update neural network weights become very small, causing slow or stalled training. This is common in deep networks with certain activation functions like sigmoid or tanh.

126. How do you handle imbalanced datasets?

Techniques include resampling (oversampling the minority class or undersampling the majority class), using different evaluation metrics (e.g., precision-recall curve), generating synthetic samples (e.g., SMOTE), and using algorithms designed for imbalanced data.

127. What is a convolutional neural network (CNN)?

CNN is a type of neural network designed for processing structured grid data like images. It uses convolutional layers to extract features and pooling layers to reduce dimensionality, followed by fully connected layers for classification.

128. : Explain recurrent neural networks (RNN) and their variants.

RNNs are neural networks designed for sequential data, where connections between nodes form directed cycles. Variants include Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), which address the vanishing gradient problem and capture long-term dependencies.

129. What is a support vector machine (SVM)?

SVM is a supervised learning algorithm used for classification and regression. It finds the hyperplane that best separates data points of different classes with the maximum margin, and can handle non-linear data using kernel functions.

130. Explain the Expectation-Maximization (EM) algorithm.

EM is an iterative algorithm used to find maximum likelihood estimates of parameters in probabilistic models with latent variables. It consists of two steps: Expectation (E-step) to estimate the expected value of the latent variables, and Maximization (M-step) to maximize the likelihood function with respect to the parameters.

131. Write a Python function to calculate the mean and variance of a list of numbers.

```
python Copy code
def mean_variance(data):
    mean = sum(data) / len(data)
    variance = sum((x - mean) ** 2 for x in data) / len(data)
    return mean, variance
```

132. Implement k-means clustering from scratch in Python.

```
python Copy code
import numpy as np

def kmeans(X, k, max_iters=100):
    centroids = X[np.random.choice(X.shape[0], k, replace=False)]
    for _ in range(max_iters):
        clusters = [np.argmin([np.linalg.norm(x - c) for c in centroids]) for x in X]
        new_centroids = [X[np.array(clusters) == i].mean(axis=0) for i in range(k)]
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids
    return centroids, clusters
```

133. Write a Python function to implement logistic regression using gradient descent.

```
python Copy code
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def logistic_regression(X, y, lr=0.01, epochs=1000):
    weights = np.zeros(X.shape[1])
    for _ in range(epochs):
        linear_model = np.dot(X, weights)
        predictions = sigmoid(linear_model)
        gradient = np.dot(X.T, (predictions - y)) / y.size
        weights -= lr * gradient
    return weights
```

134. Write a Python function to perform PCA on a given dataset.

```
python Copy code
import numpy as np

def pca(X, num_components):
    X_meaned = X - np.mean(X, axis=0)
    cov_matrix = np.cov(X_meaned, rowvar=False)
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
    sorted_index = np.argsort(eigenvalues)[::-1]
    sorted_eigenvectors = eigenvectors[:, sorted_index]
    eigenvector_subset = sorted_eigenvectors[:, :num_components]
    X_reduced = np.dot(eigenvector_subset.T, X_meaned.T).T
    return X_reduced
```

135. Implement a decision tree classifier from scratch in Python.

```
python Copy code
import numpy as np

class DecisionTree:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth

    def fit(self, X, y):
        self.n_classes = len(set(y))
        self.n_features = X.shape[1]
        self.tree = self._grow_tree(X, y)

    def _grow_tree(self, X, y, depth=0):
        num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes)]
        predicted_class = np.argmax(num_samples_per_class)
        node = Node(predicted_class=predicted_class)

        if depth < self.max_depth:
            idx, thr = self._best_split(X, y)
            if idx is not None:
                indices_left = X[:, idx] < thr
                X_left, y_left = X[indices_left], y[indices_left]
                X_right, y_right = X[~indices_left], y[~indices_left]
                node.feature_index = idx
                node.threshold = thr
                node.left = self._grow_tree(X_left, y_left, depth + 1)
                node.right = self._grow_tree(X_right, y_right, depth + 1)
        return node
```

```

def _best_split(self, X, y):
    m, n = X.shape
    if m <= 1:
        return None, None

    num_parent = [np.sum(y == c) for c in range(self.n_classes)]
    best_gini = 1.0 - sum((num / m) ** 2 for num in num_parent)
    best_idx, best_thr = None, None

    for idx in range(n):
        thresholds, classes = zip(*sorted(zip(X[:, idx], y)))
        num_left = [0] * self.n_classes
        num_right = num_parent.copy()
        for i in range(1, m):
            c = classes[i - 1]
            num_left[c] += 1
            num_right[c] -= 1
        gini_left = 1.0 - sum((num_left[x] / i) ** 2 for x in range(self.n_classes))
        gini_right = 1.0 - sum((num_right[x] / (m - i)) ** 2 for x in range(self.n_classes))
        gini = (i * gini_left + (m - i) * gini_right) / m
        if thresholds[i] == thresholds[i - 1]:
            continue
        if gini < best_gini:
            best_gini = gini
            best_idx = idx
            best_thr = (thresholds[i] + thresholds[i - 1]) / 2
    return best_idx, best_thr

```

```

def predict(self, X):
    return [self._predict(inputs) for inputs in X]

def _predict(self, inputs):
    node = self.tree
    while node.left:
        if inputs[node.feature_index] < node.threshold:
            node = node.left
        else:
            node = node.right
    return node.predicted_class

class Node:
    def __init__(self, predicted_class):
        self.predicted_class = predicted_class
        self.feature_index = None
        self.threshold = None
        self.left = None
        self.right = None

```

136. Write a Python function to calculate the F1 score.

python

 Copy code

```

def f1_score(y_true, y_pred):
    tp = sum((y_true == 1) & (y_pred == 1))
    fp = sum((y_true == 0) & (y_pred == 1))
    fn = sum((y_true == 1) & (y_pred == 0))
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    return 2 * (precision * recall) / (precision + recall)

```

138. Implement the k-nearest neighbors (k-NN) algorithm from scratch in Python.

```
python Copy code

import numpy as np
from collections import Counter

def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        return np.array([self._predict(x) for x in X])

    def _predict(self, x):
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]
```

141. Write a Python function to perform hierarchical clustering.

```
python Copy code

import numpy as np

def hierarchical_clustering(X, n_clusters):
    from scipy.cluster.hierarchy import linkage, fcluster
    Z = linkage(X, method='ward')
    clusters = fcluster(Z, t=n_clusters, criterion='maxclust')
    return clusters
```

142. Implement a Python function to calculate the silhouette score for clustering evaluation

python

 Copy code

```
from sklearn.metrics import pairwise_distances

def silhouette_score(X, labels):
    distances = pairwise_distances(X)
    unique_labels = np.unique(labels)
    silhouette_scores = []
    for label in unique_labels:
        intra_distances = np.mean([distances[i][j] for i in range(len(labels)) for j in range(len(labels)) if labels[i] == label and labels[j] == label])
        inter_distances = np.mean([distances[i][j] for i in range(len(labels)) for j in range(len(labels)) if labels[i] == label and labels[j] != label])
        silhouette_scores.append((inter_distances - intra_distances) /
                                max(inter_distances, intra_distances))
    return np.mean(silhouette_scores)
```

143. Implement a Python function to perform a grid search for hyperparameter tuning

python

 Copy code

```
from sklearn.model_selection import ParameterGrid

def grid_search(model, param_grid, X_train, y_train, scoring='accuracy'):
    best_score = -1
    best_params = None
    for params in ParameterGrid(param_grid):
        model.set_params(**params)
        model.fit(X_train, y_train)
        score = model.score(X_train, y_train)
        if score > best_score:
            best_score = score
            best_params = params
    return best_params, best_score
```

144. Write a Python function to implement the cross-entropy loss function.

python

 Copy code

```
import numpy as np

def cross_entropy(y_true, y_pred):
    n = y_true.shape[0]
    ce_loss = -np.sum(y_true * np.log(y_pred + 1e-9)) / n
    return ce_loss
```

145. Implement a Python function to calculate the Matthews correlation coefficient.

python

 Copy code

```
def matthews_corrcoef(y_true, y_pred):
    tp = sum((y_true == 1) & (y_pred == 1))
    tn = sum((y_true == 0) & (y_pred == 0))
    fp = sum((y_true == 0) & (y_pred == 1))
    fn = sum((y_true == 1) & (y_pred == 0))
    numerator = (tp * tn) - (fp * fn)
    denominator = ((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn)) ** 0.5
    return numerator / (denominator + 1e-9)
```

146. Write a Python function to implement the kmeans++ initialization.

python

 Copy code

```
import numpy as np

def kmeans_plus_plus(X, k):
    centroids = [X[np.random.choice(X.shape[0])]]
    for _ in range(1, k):
        distances = np.min([np.linalg.norm(x - centroid) for centroid in centroids]
                           for x in X)
        probabilities = distances / np.sum(distances)
        centroids.append(X[np.random.choice(X.shape[0], p=probabilities)])
    return np.array(centroids)
```

147. Implement a Python function to calculate the entropy of a dataset.

python

 Copy code

```
import numpy as np

def entropy(y):
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    return -np.sum(probabilities * np.log2(probabilities))
```

148. Implement the Markov Chain Monte Carlo (MCMC) method in Python.

python

 Copy code

```
import numpy as np

def metropolis_hastings(p, q, q_draw, n_samples, x_init):
    x = x_init
    samples = [x]
    for _ in range(n_samples - 1):
        x_new = q_draw(x)
        accept_prob = min(1, (p(x_new) * q(x, x_new)) / (p(x) * q(x_new, x)))
        if np.random.rand() < accept_prob:
            x = x_new
        samples.append(x)
    return np.array(samples)
```

<<<< Case-Based Questions:- >>>>

Case 1 : Customer Churn Prediction

Questions:- You are provided with customer data for a telecom company, including demographic information, service usage, and whether the customer has churned or not. How would you build a model to predict customer churn?

Answer:-

1. **Data Exploration:** Understand the data, check for missing values, and explore patterns.
2. **Feature Engineering:** Create relevant features like usage patterns, duration of service, and interaction with supports.
3. **Model Selection:** Use models like logistic regression, decision trees, or ensemble methods like random forests or XGBoost.
4. **Evaluation:** Use metrics like accuracy, precision, recall, and AUC-ROCs.
5. **Deployment:** Implement the model in a production environment and monitor performance.

Case 2 : A/B Testing

Questions:- An e-commerce company wants to test a new recommendation algorithm. How would you design an A/B test to measure its effectiveness?

Answer:-

1. **Hypothesis Definition:** Clearly state the null and alternative hypotheses.
2. **Sample Size Calculation:** Determine the required sample size to achieve statistical significance.
3. **Randomization:** Randomly assign users to the control (current algorithm) and treatment (new algorithm) groups.
4. **Metrics:** Define success metrics such as click-through rate, conversion rate, and average order value.
5. **Analysis:** Use statistical tests to compare the performance of both groups
6. **Conclusion:** Draw conclusions based on the results and make recommendations.

Case 3 : Fraud Detection

Questions:- You are tasked with detecting fraudulent transactions for a credit card company. How would you approach this problem?

Answer:-

1. **Data Understanding:** Analyze transaction data to identify patterns indicative of fraud.
2. **Feature Engineering:** Create features such as transaction amount, frequency, location, and time of day.
3. **Modeling:** Use supervised learning models like logistic regression, decision trees, and anomaly detection methods like isolation forests.
4. **Evaluation:** Evaluate using metrics like precision, recall, F1 score, and confusion matrix.

5. Monitoring: Continuously monitor model performance and update the model as fraud patterns evolve.

Case 4 : Sales Forecasting

Questions:- A retail company wants to forecast sales for the next quarter. How would you approach this task?

Answer:-

- 1. Data Collection:** Gather historical sales data, including seasonal trends and external factors like holidays.
 - 2. Exploratory Data Analysis (EDA):** Identify patterns, trends, and anomalies in the data.
 - 3. Feature Engineering:** Create features such as moving averages, lagged values, and external indicators.
 - 4. Model Selection:** Use time series models like ARIMA, exponential smoothing, or machine learning models like random forests and gradient boosting.
 - 5. Evaluation:** Validate model performance using metrics like RMSE, MAE, and MAPE.
- 6. Forecasting:** Generate forecasts and provide actionable insights.

Case 5 : Recommender Systems

Questions:- You need to build a recommendation system for an online streaming service. How would you approach it?

Answer:-

- 1. Data Understanding:** Analyze user behavior data, including watch history, ratings, and preferences.
- 2. Collaborative Filtering:** Implement user-based or item based collaborative filtering.
- 3. Content-Based Filtering:** Use metadata like genre, actors, and directors to recommend similar content.
- 4. Hybrid Approach:** Combine collaborative and content based filtering for better recommendations.
- 5. Evaluation:** Use metrics like precision, recall, and mean reciprocal rank (MRR) to evaluate the recommender system.
- 6. Personalization:** Continuously update the model based on user interactions to improve recommendations.

Case 6 : Sentiment Analysis

Questions:- A company wants to analyze customer reviews to understand their sentiments about its new product. How would you proceed?

Answer:-

- 1. Data Collection:** Gather customer reviews from various sources like social media, websites, and surveys.
- 2. Preprocessing:** Clean and preprocess the text data, including tokenization, stop-word removal, and stemming/lemmatization.
- 3. Feature Extraction:** Use techniques like TF-IDF, word embeddings, or BERT for feature extraction.
- 4. Modeling:** Use machine learning models like logistic regression, SVM, or deep learning models like LSTM and BERT.
- 5. Evaluation:** Evaluate model performance using metrics like accuracy, precision, recall, and F1 score.

6. Insights: Analyze the results to provide actionable insights to the company.

Case 7 : Anomaly Detection

Questions:-You are provided with server logs and need to detect anomalies in server performance. How would you approach this problem?

Answer:-

- 1. Data Understanding:** Analyze the server logs to identify normal and abnormal behavior patterns.
- 2. Feature Engineering:** Create features like CPU usage, memory usage, request count, and error rates.
- 3. Modeling:** Use unsupervised learning methods like clustering (e.g., DBSCAN), isolation forests, or autoencoders for anomaly detection.
- 4. Evaluation:** Validate the model using techniques like ROC curve and precision-recall curves.
- 5. Deployment:** Implement the model in a monitoring system to detect anomalies in real-time and alert the relevant teams.

Case 8 : Image Classification

Questions:- A healthcare company needs to classify X-ray images to detect pneumonia. How would you approach this problem?

Answer:-

- 1. Data Collection:** Gather a dataset of labeled X-ray images.
- 2. Preprocessing:** Preprocess the images by resizing, normalization, and augmentation to increase the dataset size.
- 3. Model Selection:** Use convolutional neural networks (CNN) architectures like ResNet, VGG, or transfer learning models.
- 4. Training:** Train the model using cross-validation to avoid overfitting.
- 5. Evaluation:** Use metrics like accuracy, precision, recall, F1 score, and AUC-ROC.
- 6. Deployment:** Implement the model in a clinical setting, ensuring it integrates with existing systems and provides explainable results.

Case 9 : Natural Language Processing (NLP)

Questions:- A customer support system needs to automatically categorize incoming support tickets. How would you approach this problem?

Answer:-

- 1. Data Collection:** Gather a dataset of historical support tickets and their categories.
- 2. Preprocessing:** Clean and preprocess the text data, including tokenization, stop-word removal, and stemming/lemmatization.
- 3. Feature Extraction:** Use techniques like TF-IDF, word embeddings, or BERT for feature extraction.
- 4. Modeling:** Use classification models like logistic regression, SVM, or deep learning models like LSTM and BERT.

5.Evaluation: Evaluate model performance using metrics like accuracy, precision, recall, and F1 score.

6.Deployment: Integrate the model into the support system to automatically categorize new tickets and continuously improve based on user feedback.

Case 10 : Market Basket Analysis

Questions:- A grocery store wants to analyze customer purchase patterns to increase sales. How would you approach this problem?

Answer:-

1.Data Collection: Gather transaction data, including items purchased and transaction timestamps.

2.Preprocessing: Clean the data, removing any inconsistencies or missing values.

3.Association Rule Mining: Use algorithms like Apriori or FP-Growth to find frequent itemsets and generate association rules.

4.Evaluation: Evaluate the rules using metrics like support, confidence, and lift.

5.Insights: Analyze the results to identify patterns and provide recommendations to increase cross-selling and up-selling.

6. Implementation: Implement changes in the store layout, promotions, and marketing strategies based on the insights.

[Statistical Analysis with SciPy]

Import SciPy stats module: from scipy import stats

Import NumPy for array operations: import numpy as np

Set random seed for reproducibility: np.random.seed(42)

Descriptive Statistics

Mean: np.mean(data)

Median: np.median(data)

Mode: stats.mode(data)

Variance: np.var(data)

Standard deviation: np.std(data)

Range: np.ptp(data)

Interquartile range: stats.iqr(data)

Skewness: stats.skew(data)

Kurtosis: stats.kurtosis(data)

Coefficient of variation: stats.variation(data)

Geometric mean: stats.gmean(data)

Harmonic mean: stats.hmean (data)

Trimmed mean: stats.trim_mean(data, 0.1)

Percentile: np.percentile(data, 75)
Quantile: np.quantile(data, [0.25, 0.5, 0.75])

Probability Distributions

Normal distribution PDF: stats.norm.pdf(x, loc=0, scale=1)
Normal distribution CDF: stats.norm.cdf(x, loc=0, scale=1)
Normal distribution inverse CDF: stats.norm.ppf(q, loc=0, scale=1)
Generate normal random numbers: stats.norm.rvs(loc=0, scale=1, size=1000)
Uniform distribution PDF: stats.uniform.pdf(x, loc=0, scale=1)
Uniform distribution CDF: stats.uniform.cdf(x, loc=0, scale=1)
Generate uniform random numbers: stats.uniform.rvs(loc=0, scale=1, size=1000)
Exponential distribution PDF: stats.expon.pdf(x, scale=1)
Exponential distribution CDF: stats.expon.cdf(x, scale=1)
Generate exponential random numbers: stats.expon.rvs(scale=1, size=1000)
Poisson distribution PMF: stats.poisson.pmf(k, mu=1)
Poisson distribution CDF: stats.poisson.cdf(k, mu=1)
Generate Poisson random numbers: stats.poisson.rvs(mu=1, size=1000)
Binomial distribution PMF: stats.binom.pmf(k, n, p)
Binomial distribution CDF: stats.binom.cdf(k, n, p)
Generate binomial random numbers: stats.binom.rvs(n, p, size=1000)
Chi-square distribution PDF: stats.chi2.pdf(x, df)
Chi-square distribution CDF: stats.chi2.cdf(x, df)
Generate chi-square random numbers: stats.chi2.rvs(df, size=1000)
Student's t-distribution PDF: stats.t.pdf(x, df)
Student's t-distribution CDF: stats.t.cdf(x, df)
Generate Student's t random numbers: stats.t.rvs(df, size=1000)
F-distribution PDF: stats.f.pdf(x, dfn, dfd)
F-distribution CDF: stats.f.cdf(x, dfn, dfd)
Generate F random numbers: stats.f.rvs(dfn, dfd, size=1000)

Hypothesis Testing

One-sample t-test: stats.ttest_1samp(data, popmean)
Independent two-sample t-test: stats.ttest_ind(data1, data2)
Paired t-test: stats.ttest_rel(data1, data2)
One-way ANOVA: stats.f_oneway(data1, data2, data3)
Two-way ANOVA: stats.f_oneway(*(group for name, group in data.groupby(['factor1', 'factor2'])))
Chi-square goodness of fit test: stats.chisquare(observed, expected)
Chi-square test of independence: stats.chi2_contingency(contingency_table)
Shapiro-Wilk test for normality: stats.shapiro(data)
Anderson-Darling test for normality: stats.anderson(data)

Kolmogorov-Smirnov test: stats.kstest(data, 'norm')
Mann-Whitney U test: stats.mannwhitneyu(data1, data2)
Wilcoxon signed-rank test: stats.wilcoxon(data1, data2)
Kruskal-Wallis H-test: stats.kruskal(data1, data2, data3)
Friedman test: stats.friedmanchi-square(data1, data2, data3)
Levene's test for equality of variances: stats.levene(data1, data2)
Bartlett's test for equality of variances: stats.bartlett(data1, data2)
Fligner-Killeen test for equality of variances: stats.fligner(data1, data2)

Correlation and Regression

Pearson correlation coefficient: stats.pearsonr(x, y)
Spearman rank correlation: stats.spearmanr(x, y)
Kendall's tau: stats.kendalltau(x, y)
Simple linear regression: stats.linregress(x, y)
Multiple linear regression: stats.linregress(X, y)
Polynomial regression: np.polyfit(x, y, deg=2)
R-squared (coefficient of determination): 1 - (np.sum((y - y_pred)**2) / np.sum((y - np.mean(y))**2))
Adjusted R-squared: 1 - ((1 - r_squared) * (n - 1) / (n - k - 1))
F-statistic: ((r_squared / (k - 1)) / ((1 - r_squared) / (n - k)))
Durbin-Watson statistic: stats.durbin_watson(residuals)

Non-parametric Methods

Kernel density estimation: stats.gaussian_kde(data)
Bootstrap sample: stats.bootstrap((data,), np.mean, n_resamples=1000)
Jackknife resampling: stats.jackknife(data, np.mean)
Permutation test: stats.permutation_test((data1, data2), stats.ttest_ind)

Multivariate Analysis

Principal Component Analysis: from sklearn.decomposition import PCA; PCA().fit_transform(X)
Canonical correlation analysis: from sklearn.cross_decomposition import CCA; CCA().fit(X, Y).transform(X, Y)
MANOVA: from statsmodels.multivariate.manova import MANOVA; MANOVA.from_formula('y1 + y2 ~ group', data=data).mv_test()
Hotelling's T-squared test: stats.hotelling_t2(X1, X2)

Time Series Analysis

Autocorrelation: stats.autocorr(data)

Partial autocorrelation: from statsmodels.tsa.stattools import pacf; pacf(data)

Augmented Dickey-Fuller test: from statsmodels.tsa.stattools import adfuller; adfuller(data)

KPSS test: from statsmodels.tsa.stattools import kpss; kpss(data)

Granger causality test: from statsmodels.tsa.stattools import grangercausalitytests; grangercausalitytests(data, maxlag=5)

Bayesian Statistics

Bayes factor: stats.bayes_mvs(data)

Bayesian Information Criterion (BIC): stats.bic(residuals)

Akaike Information Criterion (AIC): stats.aic(residuals)

Sampling and Experimental Design

Simple random sample: np.random.choice(population, size=n, replace=False)

Stratified sample: from sklearn.model_selection import StratifiedShuffleSplit; StratifiedShuffleSplit(n_splits=1, test_size=0.3).split(X, y)

Cluster sample: from sklearn.cluster import KMeans; KMeans(n_clusters=k).fit_predict(X)

Systematic sample: population[::-k]

Latin square design: stats.latin_square(n)

Power Analysis

Power of t-test: stats.ttest_ind_solve_power(effect_size=0.5, nobs1=100, alpha=0.05, ratio=1.0, alternative='two-sided')

Power of ANOVA: stats.f_oneway_solve_power(dfnum=2, dfden=27, alpha=0.05, effect_size=0.25)

Sample size calculation for t-test: stats.ttest_ind_solve_power(effect_size=0.5, power=0.8, alpha=0.05, ratio=1.0, alternative='two-sided')

Reliability Analysis

Cronbach's alpha: from statsmodels.stats.inter_rater import fleiss_kappa; fleiss_kappa(data)

Intraclass correlation coefficient: stats.ttest_ind(group1, group2)

Effect Size Calculations

Cohen's d: `(np.mean(group1) - np.mean(group2)) / np.sqrt((np.std(group1, ddof=1)**2 + np.std(group2, ddof=1)**2) / 2)`

Eta-squared: `ss_effect / (ss_effect + ss_error)`

Odds ratio: `(a * d) / (b * c)`

Risk ratio: `(a / (a + b)) / (c / (c + d))`

Data Transformation

Z-score normalization: `stats.zscore(data)`

Min-max scaling: `(data - np.min(data)) / (np.max(data) - np.min(data))`

Box-Cox transformation: `stats.boxcox(data)`

Yeo-Johnson transformation: `stats.yeojohnson(data)`

Logarithmic transformation: `np.log1p(data)`

Outlier Detection

Z-score method: `np.abs(stats.zscore(data)) > 3`

Interquartile range (IQR) method: `(data < Q1 - 1.5 * IQR) | (data > Q3 + 1.5 * IQR)`

Modified Z-score method: `0.6745 * (data - np.median(data)) / stats.median_abs_deviation(data) > 3.5`

Grubbs' test: `stats.grubbs(data)`

Confidence Intervals

Normal distribution CI: `stats.norm.interval(alpha=0.95, loc=np.mean(data), scale=stats.sem(data))`

T-distribution CI: `stats.t.interval(alpha=0.95, df=len(data)-1, loc=np.mean(data), scale=stats.sem(data))`

Binomial proportion CI: `stats.binom.interval(n=len(data), p=np.mean(data), alpha=0.05)`

Poisson CI: `stats.poisson.interval(alpha=0.95, mu=np.mean(data))`

Survival Analysis

Kaplan-Meier estimator: `from lifelines import KaplanMeierFitter; KaplanMeierFitter().fit(durations, event_observed)`

Cox proportional hazards model: `from lifelines import CoxPHFitter; CoxPHFitter().fit(df, duration_col='T', event_col='E')`

Log-rank test: `from lifelines.statistics import logrank_test; logrank_test(durations_1, durations_2, event_observed_1, event_observed_2)`

Spatial Statistics

Moran's I: from pysal.explore import esda; esda.Moran(y, w).I

Geary's C: from pysal.explore import esda; esda.Geary(y, w).C

Getis-Ord G: from pysal.explore import esda; esda.G(y, w).G

Multivariate Normality Tests

Mardia's test: from statsmodels.stats.multivariate_normal import mardia; mardia(data)

Henze-Zirkler test: from statsmodels.stats.multivariate_normal import henze_zirkler; henze_zirkler(data)

Robust Statistics

Median absolute deviation: stats.median_abs_deviation(data)

Huber's M-estimator: from statsmodels.robust import scale; scale.huber(data)

Theil-Sen estimator: from scipy.stats import theilslopes; theilslopes(y, x)

Factor Analysis

Exploratory Factor Analysis: from factor_analyzer import FactorAnalyzer; FactorAnalyzer().fit(data)

Confirmatory Factor Analysis: from statsmodels.stats.factor import FactorAnalysis; FactorAnalysis().fit(data)

Cluster Analysis

K-means clustering: from sklearn.cluster import KMeans; KMeans(n_clusters=k).fit(X)

Hierarchical clustering: from scipy.cluster.hierarchy import linkage; linkage(X, method='ward')

DBSCAN clustering: from sklearn.cluster import DBSCAN; DBSCAN().fit(X)

Time Series Decomposition

Seasonal decomposition: from statsmodels.tsa.seasonal import seasonal_decompose;

seasonal_decompose(data, model='additive')

Statistical Process Control

Control chart (X-bar chart): from statsmodels.stats.stattools import control_chart; control_chart(data)

Meta-Analysis

Fixed effects meta-analysis: from statsmodels.stats.meta_analysis import CombineResults;
CombineResults.combine_effects(effects, variances)

Random effects meta-analysis: from statsmodels.stats.meta_analysis import CombineResults;
CombineResults.combine_effects(effects, variances, method='random')

Structural Equation Modeling

Path analysis: from statsmodels.stats.sem import SEM; SEM.from_formula('y ~ x1 + x2', data=data).fit()

Item Response Theory

1PL (Rasch) model: from psychometrics import irt; irt.twopl(difficulty, discrimination=1, ability)

2PL model: from psychometrics import irt; irt.twopl(difficulty, discrimination, ability)

Multilevel Modeling

Random intercept model: from statsmodels.regression.mixed_linear_model import MixedLM;
MixedLM.from_formula('y ~ x', groups='group', data=data).fit()

Statistical Quality Control

Capability analysis: from statsmodels.stats.stattools import cpk_index; cpk_index(data, lower=lsl, upper=usl)

Process capability index: (usl - lsl) / (6 * np.std(data, ddof=1))

Nonlinear Regression

Curve fitting: from scipy.optimize import curve_fit; curve_fit(lambda x, a, b: a * np.exp(b * x), x_data, y_data)

Statistical Tests for Circular Data

Rayleigh test: from scipy.stats import rayleigh; rayleigh.fit(data)

Watson's U2 test: from astropy.stats import watson_u2; watson_u2(data)

Extreme Value Analysis

Generalized extreme value distribution fit: from scipy.stats import genextreme; genextreme.fit(data)

Peak over threshold analysis: from scipy.stats import genpareto; genpareto.fit(data[data > threshold])

Functional Data Analysis

Functional principal component analysis: from skfda.decomposition import FPCA; FPCA().fit_transform(data)

Statistical Learning Theory

Support Vector Machine: from sklearn.svm import SVC; SVC().fit(X, y)

Cross-validation: from sklearn.model_selection import cross_val_score; cross_val_score(model, X, y, cv=5)

Copulas

Gaussian copula: from scipy.stats import multivariate_normal; multivariate_normal.cdf(data)

Clayton copula: from copulas.multivariate import GaussianMultivariate;

GaussianMultivariate().fit(data).probability_density(data)

Stochastic Processes

Brownian motion simulation: np.cumsum(np.random.normal(0, 1, size=1000))

Ornstein-Uhlenbeck process: from scipy.integrate import odeint; odeint(lambda y, t, theta, mu, sigma: theta * (mu - y), y0, t, args=(theta, mu, sigma))

Causal Inference

Propensity score matching: from sklearn.linear_model import LogisticRegression; LogisticRegression().fit(X, treatment).predict_proba(X)[:, 1]

Difference-in-differences estimation: np.mean(post_treatment - pre_treatment) - np.mean(post_control - pre_control)

Spatial Point Pattern Analysis

Ripley's K function: from astropy.stats import RipleysKEstimator; RipleysKEstimator(area=area).evaluate(data)

Statistical Network Analysis

Erdős-Rényi random graph model: from networkx.generators.random_graphs import erdos_renyi_graph; erdos_renyi_graph(n, p)

Population:- The entire set of items or individuals of interest in a study. Denoted by N.

Sample:- A subset selected from the larger population; Denoted by n.

Parameter:- A numerical value that describes a characteristic of the entire population. It is the opposite of statistics.

Statistic:- A numerical value that describes a characteristic of a sample and used to estimate a population parameter. It is the opposite of a parameter.

Random Sample:- A sample in which every member of the population has an equal chance of being selected.

Representative Sample:- A sample that accurately mirrors the characteristics of the larger population.

Variable:- A characteristic or attribute that can take on different values or categories. E.g. height, occupation, age etc.

Types of Data:- The classification of data based on its nature. There are two types of data - categorical and numerical.

Categorical Data:- Data that represents categories or labels without inherent numerical value.

Numerical Data:- Data that represents quantifiable amounts or values. Can be further classified into discrete and continuous.

Discrete Data:- Numerical data that can only take on specific, distinct values. Opposite of continuous.

Continuous Data:- Numerical data that is 'infinite' and impossible to count. Opposite of discrete.

Levels of Measurement:- A way to classify data. There are two levels of measurement - qualitative and quantitative.

Qualitative Data:- A subgroup of levels of measurement. There are two types of qualitative data - nominal and ordinal.

Quantitative Data:- A subgroup of levels of measurement. There are two types of quantitative data - ratio and interval.

Nominal Level of Measurement:- Nominal level of measurement refers to variables that describe different categories or names. These categories cannot be put in any specific order.

Ordinal Level of Measurement:- Ordinal level of measurement refers to variables that describe different categories, and they can be ordered.

Ratio Level of Measurement:- Ratio level of measurement represents a number that has a unique and unambiguous zero point, no matter if a whole number or a fraction. For example, the temperature in Kelvin is a ratio variable.

Interval Level of Measurement:- An interval variable represents a number or an interval. There isn't a unique and unambiguous zero point. For example, degrees in Celsius and Fahrenheit are interval variables.

Frequency Distribution Table

A table showing the frequency of each variable.

Frequency Distribution Table

	Frequency
A	124
B	98
C	113
Total	335

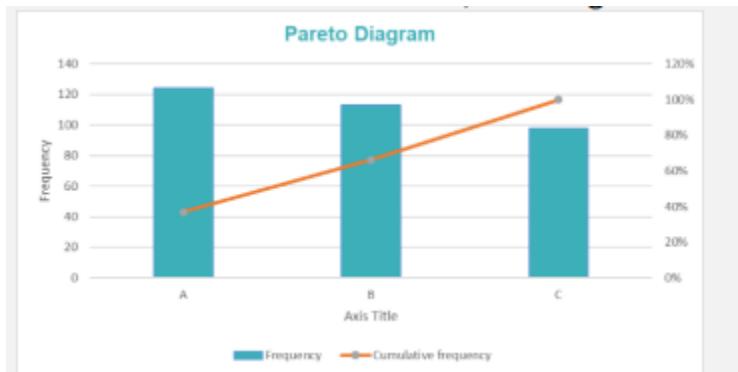
Frequency:- The number of times a particular value or category occurs in a dataset.

Absolute Frequency:- Measures the number of occurrences of a variable.

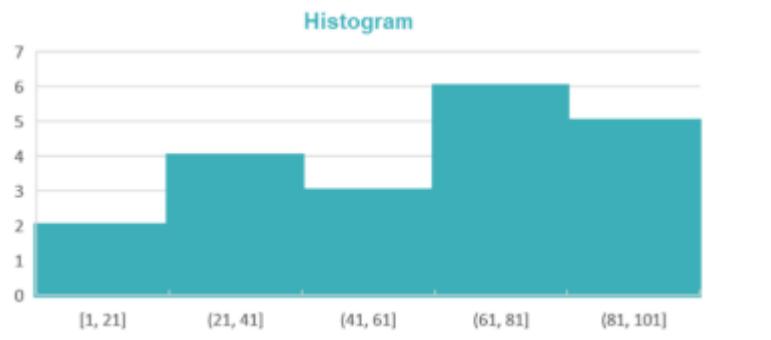
Relative Frequency:- Measures the relative number of occurrences of a variable. Usually expressed in percentages.

Cumulative Frequency:- The sum of the relative frequencies of all members in a dataset up to a certain point. The cumulative frequency of all members is 100% or 1.

Pareto Diagram:- A type of bar chart where frequencies are shown in descending order. There is an additional line on the chart, showing the cumulative frequency.



Histogram:- A type of bar chart that represents numerical data. It is divided into intervals (or bins) that are not overlapping and span from the first observation to the last. The intervals (bins) are adjacent - where one stops, the other starts.



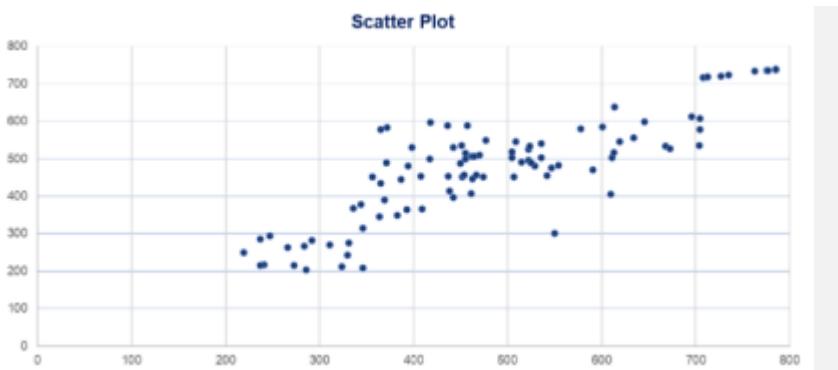
Cross Table (Contingency Table):- A table in a matrix format that displays the frequency distribution of the variables.

Cross table

Type of investment \ Investor	Investor A	Investor B	Investor C	Total
Stocks	96	185	39	320
Bonds	181	3	29	213
Real Estate	88	152	142	382
Total	365	340	210	915

Bins (Histogram):- The intervals that are represented in a histogram.

Scatter Plot:- A plot that represents numerical data. Graphically, each observation looks like a point on the scatter plot.



Measures of Central Tendency:- Measures of central tendency are statistical values that represent the center or typical value of a dataset. The most common are the mean, median and mode.

Mean:-

The arithmetic average of all data points in a dataset.

$$\mu = \frac{\sum_{i=1}^N X_i}{N}$$

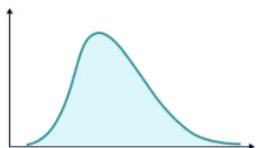
Median:- The middle number in a data set sorted in ascending or descending order.

----->>>

3,5,**7**,8,9

Mode:- The value that occurs most frequently in the dataset. A dataset can have one mode (unimodal), more than one mode (multimodal), or no mode at all.

Skewness:- A measure which indicates whether the observations in a dataset are concentrated on one side.



Sample Formula:- A formula that is calculated on a sample. The value obtained is a statistic.

Population Formula:- A formula that is calculated on a population. The value obtained is a parameter.

Measures of Variability:- Measures that describe the data through the level of dispersion (variability). The most common ones are variance and standard deviation.

Variance:- Measures the dispersion of the dataset around its mean. It is measured in units squared. Denoted σ^2 for a population and s^2 for a sample.

$$\sigma^2 = \sum_{i=1}^n \frac{(X_i - \mu)^2}{n}$$

$$s^2 = \sum_{i=1}^n \frac{(X_i - \bar{X})^2}{n-1}$$

Standard Deviation:- Measures the dispersion of the dataset around its mean. It is measured in original units. Denoted σ for a population and s for a sample.

$$\sigma = \sqrt{\sum_{i=1}^n \frac{(X_i - \mu)^2}{n}}$$

$$s = \sqrt{\sum_{i=1}^n \frac{(X_i - \bar{X})^2}{n-1}}$$

Coefficient of Variation:- Measures the dispersion of the dataset around its mean. The coefficient of variation is unitless. Therefore, it is useful when comparing the dispersion across different datasets that have different units of measurement.

$$CV = \frac{\sigma}{\mu}$$

$$CV = s/\bar{X}$$

Univariate Measure:- Univariate measure refers to the summary of a dataset that includes multiple categories of variables.

Multivariate Measure:- A measure which refers to multiple variables.

Covariance:- A statistical measure that quantifies the degree to which two random variables in a dataset change together. Usually, because of its scale of measurement, covariance is not directly interpretable.

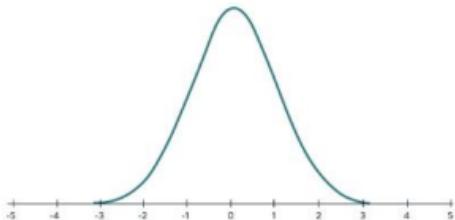
Linear Correlation Coefficient:- A measure of the strength and direction of a linear relationship between two variables. Very useful for direct interpretation as it takes on values from [-1, 1]. Denoted ρ_{xy} for a population and r_{xy} for a sample.

Correlation:- A statistical measure that describes the extent to which two variables change together. There are several ways to compute it, the most common being the linear correlation coefficient.

Distribution:- A function that shows the possible values for a variable and the probability of their occurrence.

Bell Curve:- A common name for the normal distribution.

Normal Distribution:- A continuous, symmetric probability distribution that is completely described by its mean and its variance. Also known as the Gaussian Distribution or Bell Curve.



Gaussian Distribution:- The original name of the normal distribution. Named after the famous mathematician Gauss, who was the first to explore it through his work on the Gaussian function.

Standard Normal Distribution:- A normal distribution with a mean of 0, and a standard deviation of 1.

Z-statistic:- The cumulative frequency of a data value in a frequency distribution.

Standardized Variable :- A variable which has been standardized using the z-score formula - by first subtracting the mean and then dividing by the standard deviation.

What does the Central Limit Theorem state?

The sampling distribution will approximate a normal distribution as the sample size increases. In general, a sample of at least 30 is often considered sufficient for the theorem to hold.

Sampling Distribution:- The probability distribution of a given statistic (like the mean or variance) based on all possible samples of a fixed size from a population.

Standard Error:- The standard deviation of the sampling distribution, which reflects the variability of sample means. It accounts for the sample size, with larger samples generally having smaller standard errors.

$$\text{Standard Error} = \frac{\sigma}{\sqrt{n}}$$

Estimator:- Estimations we make according to a function or rule.

Estimate:- The particular value that was estimated through an estimator.

Bias:- The difference between an estimator's expected value and the true population parameter. An unbiased estimator has an expected value equal to the population parameter.

Efficiency (in Estimators):- Refers to an estimator's variability. An efficient estimator has minimal variability compared to others.

Point Estimator:- A function or a rule, according to which we make estimations that will result in a single number.

Point Estimate:- The specific numerical value obtained from a point estimator.

Interval Estimator:- A function or a rule, according to which we make estimations that will result in an interval. In this course, we will only consider confidence intervals. Another instance that we don't discuss are also credible intervals (Bayesian statistics).

Interval Estimate:- The categorization of data into discrete groups based on their attributes.

Confidence Interval:- A confidence interval is the range within which you expect the population parameter to be. You have a certain probability of it being correct, equal to the significance level.

Reliability Factor:- A singular metric that captures the entire variance of a dataset.

Level of Confidence:- The probability that the population parameter lies within a given confidence interval. Denoted $1 - \alpha$.

Example: 95% confidence level means that in 95% of the cases, the population parameter will fall into the specified interval.

Critical Value:- A threshold value from a statistical table (z, t, F, etc.) associated with a chosen significance level.

z-table:- A table showing values of the Z statistic for various probabilities under the standard normal distribution.

z	.04	.05	.06	.07
1.8	.9671	.9678	.9686	.9693
1.9	.9738	.9744	.9750	.9756
2.0	.9793	.9798	.9803	.9808

t-statistic:- A statistic that is generally associated with the Student's T distribution, in the same way the z-statistic is associated with the normal distribution.

t-table:- A table showing t-statistic values for given probabilities and degrees of freedom.

df	.20	.10	.05	.025	.01	0.05
18	0.862	1.330	1.734	2.101	2.552	2.878
19	0.861	1.328	1.729	2.093	2.539	2.861
20	0.860	1.325	1.725	2.086	2.528	2.845

Degrees of Freedom:- The number of values in a statistical calculation that are free to vary without violating the data's constraints.

Margin of Error:- The range within which the true population parameter is likely to lie, given a specific confidence level. It quantifies the uncertainty associated with a sample estimate, often expressed as a percentage of the estimate itself.

Hypothesis:- A testable proposition or assumption about a population parameter.

Hypothesis Test:- A test that is conducted in order to verify if a hypothesis is true or false.

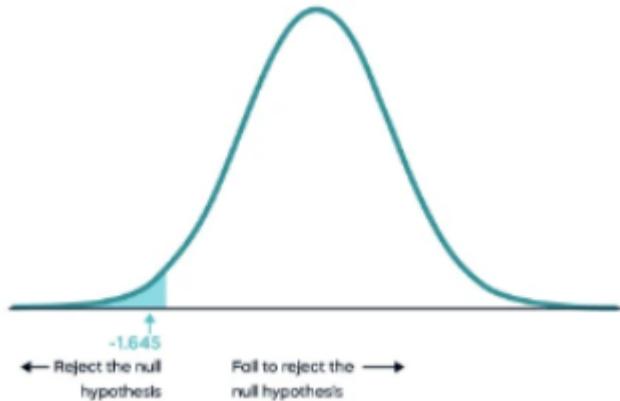
Null Hypothesis:- A default hypothesis for testing. Whenever we are conducting a test, we are trying to reject the null hypothesis.

Alternative Hypothesis:- The hypothesis that contradicts the null hypothesis. It represents the researcher's claim.

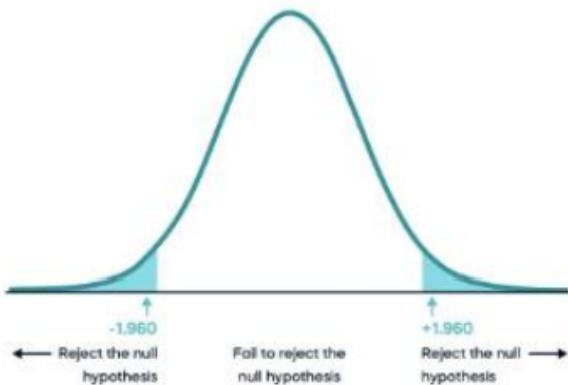
To Accept a Hypothesis:- The statistical evidence shows that the hypothesis is likely to be true.

To Reject a Hypothesis:- The statistical evidence shows that the hypothesis is likely to be false.

One-Tailed (One-Sided) Test:- A test that examines if a parameter is greater than or less than a specified value. In a one-tailed test, the alternative hypothesis focuses on a specific difference (higher than, lower than, or equal to).

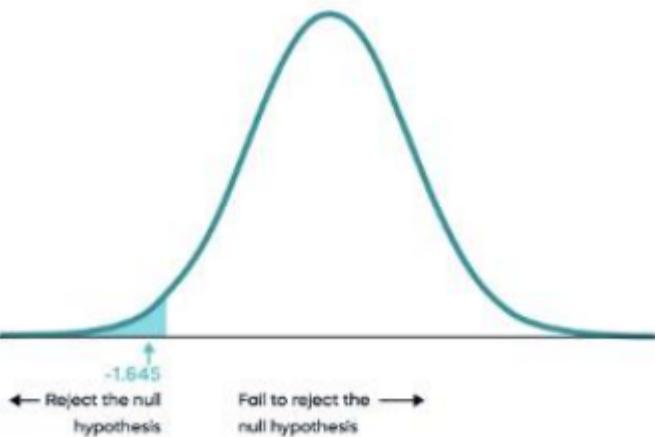


Two-Tailed (Two-Sided) Test :- A test that examines if a value is different (or equal) from a specified value. A two-tailed test considers the possibility of a difference in either direction from the null hypothesis.



Significance Level:- The probability of rejecting the null hypothesis when it is true. Denoted α . You choose the significance level. All else equal, the lower the level the better the test.

Rejection Region:- The part of the distribution, for which we would reject the null hypothesis.



Type I Error (False Positive):- Rejecting a null hypothesis that is true. The probability of committing α is the significance level.

Decision	H_0 True	H_0 False
Fail to reject H_0	Correct decision: Do not reject a true null hypothesis.	Type II error: Fail to reject a false null hypothesis. False negative.
Reject H_0	Type I error: Reject a true null hypothesis. False positive.	Correct decision: Reject a false null hypothesis.

Type II Error (False Negative):- Accepting a null hypothesis that is false. The probability of committing β is β .

Decision	H_0 True	H_0 False
Fail to reject H_0	Confidence level ($1-\alpha$)	β
Reject H_0	Level of significance α	Power of the test ($1-\beta$)

Power of the Test:- The probability of correctly rejecting a false null hypothesis (the researcher's goal). Denoted by $1 - \beta$.

Decision	H_0 True	H_0 False
Fail to reject H_0	Confidence level ($1-\alpha$)	β
Reject H_0	Level of significance α	Power of the test ($1-\beta$)

z-score:- A value indicating how many standard deviations an element is from the mean.

μ_0 :- The most frequent value occurring in a population dataset.

p-value:- The smallest significance level at which the null hypothesis can be rejected based on the observed data.

Data Science Cheatsheet 2.0

Last Updated June 19, 2021

Distributions

Discrete

Binomial - x successes in n events, each with p probability
 $\rightarrow \binom{n}{x} p^x q^{n-x}$, with $\mu = np$ and $\sigma^2 = npq$

- If $n = 1$, this is a Bernoulli distribution

Geometric - first success with p probability on the n^{th} trial
 $\rightarrow q^{n-1}p$, with $\mu = 1/p$ and $\sigma^2 = \frac{1-p}{p^2}$

Negative Binomial - number of failures before r successes

Hypergeometric - x successes in n draws, no replacement, from a size N population with X items of that feature

$$\rightarrow \frac{\binom{X}{x} \binom{N-X}{n-x}}{\binom{N}{n}}, \text{ with } \mu = \frac{nX}{N}$$

Poisson - number of successes x in a fixed time interval, where success occurs at an average rate $\lambda \rightarrow \frac{\lambda^x e^{-\lambda}}{x!}$, with $\mu = \sigma^2 = \lambda$

Continuous

Uniform - all values between a and b are equally likely

$$\rightarrow \frac{1}{b-a} \text{ with } \mu = \frac{a+b}{2} \text{ and } \sigma^2 = \frac{(b-a)^2}{12} \text{ or } \frac{n^2-1}{12} \text{ if discrete}$$

Normal/Gaussian $N(\mu, \sigma)$, Standard Normal $Z \sim N(0, 1)$

- Central Limit Theorem - sample mean of i.i.d. data approaches normal distribution
- Empirical Rule - 68%, 95%, and 99.7% of values lie within one, two, and three standard deviations of the mean
- Normal Approximation - discrete distributions such as Binomial and Poisson can be approximated using z-scores when np, nq , and λ are greater than 10

Exponential - memoryless time between independent events occurring at an average rate $\lambda \rightarrow \lambda e^{-\lambda x}$, with $\mu = \frac{1}{\lambda}$

Gamma - time until n independent events occurring at an average rate λ

Concepts

Prediction Error = Bias² + Variance + Irreducible Noise

Bias - wrong assumptions when training \rightarrow can't capture underlying patterns \rightarrow underfit

Variance - sensitive to fluctuations when training \rightarrow can't generalize on unseen data \rightarrow overfit

The bias-variance tradeoff attempts to minimize these two sources of error, through methods such as:

- Cross validation to generalize to unseen data
- Dimension reduction and feature selection

In all cases, as variance decreases, bias increases.

ML models can be divided into two types:

- Parametric - uses a fixed number of parameters with respect to sample size
- Non-Parametric - uses a flexible number of parameters and doesn't make particular assumptions on the data

Cross Validation - validates test error with a subset of training data, and selects parameters to maximize average performance

- k -fold - divide data into k groups, and use one to validate
- leave- p -out - use p samples to validate and the rest to train

Model Evaluation

Regression

Mean Squared Error (MSE) = $\frac{1}{n} \sum (y_i - \hat{y})^2$

Sum of Squared Error (SSE) = $\sum (y_i - \hat{y})^2$

Total Sum of Squares (SST) = $\sum (y_i - \bar{y})^2$

R² = $1 - \frac{SSE}{SST}$, the proportion of explained y -variability

Note, negative R^2 means the model is worse than just predicting the mean. R^2 is not valid for nonlinear models, as $SSE_{residual} + SSE_{error} \neq SST$.

Adjusted R² = $1 - (1 - R^2) \frac{N-1}{N-p-1}$, which changes only when predictors affect R^2 above what would be expected by chance

Classification

	Predict Yes	Predict No
Actual Yes	True Positive ($1 - \beta$)	False Negative (β)
Actual No	False Positive (α)	True Negative ($1 - \alpha$)

- Precision = $\frac{TP}{TP+FP}$, percent correct when predict positive
- Recall, Sensitivity = $\frac{TP}{TP+FN}$, percent of actual positives identified correctly (True Positive Rate)
- Specificity = $\frac{TN}{TN+FP}$, percent of actual negatives identified correctly, also $1 - FPR$ (True Negative Rate)
- $F_1 = 2 \frac{precision \cdot recall}{precision + recall}$, useful when classes are imbalanced

ROC Curve - plots TPR vs. FPR for every threshold α . Area Under the Curve measures how likely the model differentiates positives and negatives (perfect AUC = 1, baseline = 0.5).

Precision-Recall Curve - focuses on the correct prediction of the minority class, useful when data is imbalanced

Linear Regression

Models linear relationships between a continuous response and explanatory variables

Ordinary Least Squares - find $\hat{\beta}$ for $\hat{y} = \hat{\beta}_0 + \hat{\beta}X + \epsilon$ by solving $\hat{\beta} = (X^T X)^{-1} X^T Y$ which minimizes the SSE

Assumptions

- Linear relationship and independent observations
- Homoscedasticity - error terms have constant variance
- Errors are uncorrelated and normally distributed
- Low multicollinearity

Variance Inflation Factor - measures the severity of multicollinearity $\rightarrow \frac{1}{1-R_i^2}$, where R_i^2 is found by regressing X_i against all other variables (a common VIF cutoff is 10)

Regularization

Add a penalty λ for large coefficients to the cost function, which reduces overfitting. Requires normalized data.

Subset (L₀): $\lambda ||\beta||_0 = \lambda(\text{number of non-zero variables})$

- Computationally slow, need to fit 2^k models
- Alternatives: forward and backward stepwise selection

LASSO (L₁): $\lambda ||\hat{\beta}||_1 = \lambda \sum |\hat{\beta}|$

- Shrinks coefficients to zero, and is robust to outliers

Ridge (L₂): $\lambda ||\hat{\beta}||_2 = \lambda \sum (\hat{\beta})^2$

- Reduces effects of multicollinearity

Combining LASSO and Ridge gives Elastic Net

Logistic Regression

Predicts probability that y belongs to a binary class.

Estimates β through maximum likelihood estimation (MLE) by fitting a logistic (sigmoid) function to the data. This is equivalent to minimizing the cross entropy loss. Regularization can be added in the exponent.

$$P(Y=1) = \frac{1}{1 + e^{-(\beta_0 + \beta x)}}$$

The threshold a classifies predictions as either 1 or 0

Assumptions

- Linear relationship between X and log-odds of Y
- Independent observations
- Low multicollinearity

Odds - output probability can be transformed using

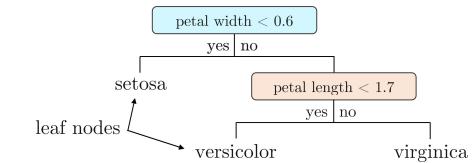
$$Odds(Y=1) = \frac{P(Y=1)}{1-P(Y=1)}$$

Coefficients are linearly related to odds, such that a one unit increase in x_1 affects odds by e^{β_1}

Decision Trees

Classification and Regression Tree

CART for regression minimizes SSE by splitting data into sub-regions and predicting the average value at leaf nodes. The complexity parameter cp only keeps splits that reduce loss by at least cp (small $cp \rightarrow$ deep tree)



CART for classification minimizes the sum of region impurity, where \hat{p}_i is the probability of a sample being in category i . Possible measures, each with a max impurity of 0.5.

- Gini Impurity = $1 - \sum (\hat{p}_i)^2$
- Cross Entropy = $-\sum (\hat{p}_i) \log_2(\hat{p}_i)$

At each leaf node, CART predicts the most frequent category, assuming false negative and false positive costs are the same. The splitting process handles multicollinearity and outliers. Trees are prone to high variance, so tune through CV.

Random Forest

Trains an ensemble of trees that vote for the final prediction

Bootstrapping - sampling with replacement (will contain duplicates), until the sample is as large as the training set

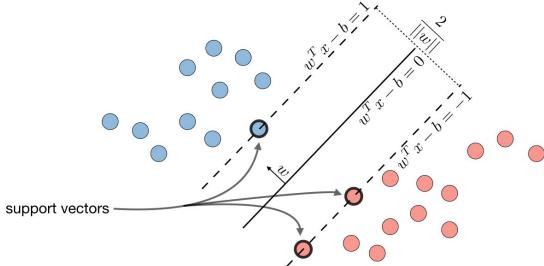
Bagging - training independent models on different subsets of the data, which reduces variance. Each tree is trained on ~63% of the data, so the out-of-bag 37% can estimate prediction error without resorting to CV.

Deep trees may overfit, but adding more trees does not cause overfitting. Model bias is always equal to one of its individual trees.

Variable Importance - ranks variables by their ability to minimize error when split upon, averaged across all trees

Support Vector Machines

Separates data between two classes by maximizing the margin between the hyperplane and the nearest data points of any class. Relies on the following:

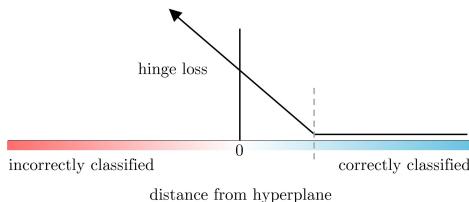


Support Vector Classifiers - account for outliers through the regularization parameter C , which penalizes misclassifications in the margin by a factor of $C > 0$

Kernel Functions - solve nonlinear problems by computing the similarity between points a, b and mapping the data to a higher dimension. Common functions:

- Polynomial $(ab + r)^d$
- Radial $e^{-\gamma(a-b)^2}$, where smaller $\gamma \rightarrow$ smoother boundaries

Hinge Loss - $\max(0, 1 - y_i(w^T x_i - b))$, where w is the margin width, b is the offset bias, and classes are labeled ± 1 . Acts as the cost function for SVM. Note, even a correct prediction inside the margin gives loss > 0 .



Multiclass Prediction

To classify data with $3+$ classes C , a common method is to binarize the problem through:

- One vs. Rest - train a classifier for each class c_i by setting c_i 's samples as 1 and all others as 0, and predict the class with the highest confidence score
- One vs. One - train $\frac{C(C-1)}{2}$ models for each pair of classes, and predict the class with the highest number of positive predictions

k-Nearest Neighbors

Non-parametric method that calculates \hat{y} using the average value or most common class of its k -nearest points. For high-dimensional data, information is lost through equidistant vectors, so dimension reduction is often applied prior to k-NN.

Minkowski Distance = $(\sum |a_i - b_i|^p)^{1/p}$

- $p = 1$ gives Manhattan distance $\sum |a_i - b_i|$
- $p = 2$ gives Euclidean distance $\sqrt{\sum (a_i - b_i)^2}$

Hamming Distance - count of the differences between two vectors, often used to compare categorical variables

Clustering

Unsupervised, non-parametric methods that group similar data points together based on distance

k-Means

Randomly place k centroids across normalized data, and assign observations to the nearest centroid. Recalculate centroids as the mean of assignments and repeat until convergence. Using the median or medoid (actual data point) may be more robust to noise and outliers. k -modes is used for categorical data.

k-means++ - improves selection of initial clusters

1. Pick the first center randomly
2. Compute distance between points and the nearest center
3. Choose new center using a weighted probability distribution proportional to distance
4. Repeat until k centers are chosen

Evaluating the number of clusters and performance:

Silhouette Value - measures how similar a data point is to its own cluster compared to other clusters, and ranges from 1 (best) to -1 (worst).

Davies-Bouldin Index - ratio of within cluster scatter to between cluster separation, where lower values are better

Hierarchical Clustering

Clusters data into groups using a predominant hierarchy

Agglomerative Approach

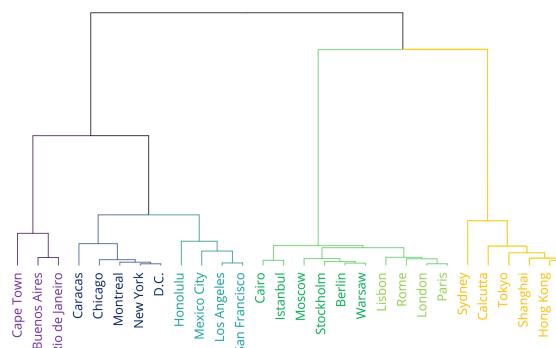
1. Each observation starts in its own cluster
2. Iteratively combine the most similar cluster pairs
3. Continue until all points are in the same cluster

Divisive Approach - all points start in one cluster and splits are performed recursively down the hierarchy

Linkage Metrics - measure dissimilarity between clusters and combines them using the minimum linkage value over all pairwise points in different clusters by comparing:

- Single - the distance between the closest pair of points
- Complete - the distance between the farthest pair of points
- Ward's - the increase in within-cluster SSE if two clusters were to be combined

Dendrogram - plots the full hierarchy of clusters, where the height of a node indicates the dissimilarity between its children



Aaron Wang

Dimension Reduction

High-dimensional data can lead to the *curse of dimensionality*, which increases the risk of overfitting and decreases the value added. The number of samples for each feature combination quickly becomes sparse, reducing model performance.

Principal Component Analysis

Projects data onto orthogonal vectors that maximize variance. Remember, given an $n \times n$ matrix A , a nonzero vector \vec{x} , and a scalar λ , if $A\vec{x} = \lambda\vec{x}$ then \vec{x} and λ are an eigenvector and eigenvalue of A . In PCA, the eigenvectors are uncorrelated and represent principal components.

1. Start with the covariance matrix of standardized data
2. Calculate eigenvalues and eigenvectors using SVD or eigendecomposition
3. Rank the principal components by their proportion of variance explained = $\frac{\lambda_i}{\sum \lambda}$

Data should be linearly related, and for a p -dimensional dataset, there will be p principal components.

Note, PCA explains the variance in X, not necessarily Y.

Sparse PCA - constrains the number of non-zero values in each component, reducing susceptibility to noise and improving interpretability

Linear Discriminant Analysis

Supervised method that maximizes separation between classes and minimizes variance within classes for a labeled dataset

1. Compute the mean and variance of each independent variable for every class C_i
2. Calculate the within-class (σ_w^2) and between-class (σ_b^2) variance
3. Find the matrix $W = (\sigma_w^2)^{-1}(\sigma_b^2)$ that maximizes Fisher's signal-to-noise ratio
4. Rank the discriminant components by their signal-to-noise ratio λ

Note, the number of components is at most $C_1 - 1$

Assumptions

- Independent variables are normally distributed
- Homoscedasticity - constant variance of error
- Low multicollinearity

Factor Analysis

Describes data using a linear combination of k latent factors. Given a normalized matrix X , it follows the form $X = Lf + \epsilon$, with factor loadings L and hidden factors f .

data	factor loadings	common factors
$\begin{bmatrix} \text{math scores} \\ \text{reading scores} \\ \text{science scores} \end{bmatrix}$	$= \begin{bmatrix} .13 & .95 \\ .78 & -.28 \\ -.87 & .05 \end{bmatrix}$	$\begin{bmatrix} -1.25 & 1.88 & \dots & -0.55 \\ 0.71 & -0.17 & \dots & -1.20 \end{bmatrix}$
$p \times n$	$p \times k$	$k \times n$

Scree Plot - graphs the eigenvalues of factors (or principal components) and is used to determine the number of factors to retain. The 'elbow' where values level off is often used as the cutoff.

Natural Language Processing

Transforms human language into machine-useable code
Processing Techniques

- Tokenization - splits text into individual words (tokens)
- Lemmatization - reduces words to its base form based on dictionary definition (*am, are, is* → *be*)
- Stemming - reduces words to its base form without context (*ended* → *end*)
- Stop words - removes common and irrelevant words (*the, is*)

Markov Chain - stochastic and memoryless process that predicts future events based only on the current state

n-gram - predicts the next term in a sequence of n terms based on Markov chains

Bag-of-words - represents text using word frequencies, without context or order

tf-idf - measures word importance for a document in a collection (corpus), by multiplying the term frequency (occurrences of a term in a document) with the inverse document frequency (penalizes common terms across a corpus)

Cosine Similarity - measures similarity between vectors, calculated as $\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$, which ranges from 0 to 1

Word Embedding

Maps words and phrases to numerical vectors

word2vec - trains iteratively over local word context windows, places similar words close together, and embeds sub-relationships directly into vectors, such that

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

Relyes on one of the following:

- Continuous bag-of-words (CBOW) - predicts the word given its context
- skip-gram - predicts the context given a word

GloVe - combines both global and local word co-occurrence data to learn word similarity

BERT - accounts for word order and trains on subwords, and unlike word2vec and GloVe, BERT outputs different vectors for different uses of words (*cell phone* vs. *blood cell*)

Sentiment Analysis

Extracts the attitudes and emotions from text

Polarity - measures positive, negative, or neutral opinions

- Valence shifters - capture amplifiers or negators such as '*really fun*' or '*hardly fun*'

Sentiment - measures emotional states such as happy or sad

Subject-Object Identification - classifies sentences as either subjective or objective

Topic Modelling

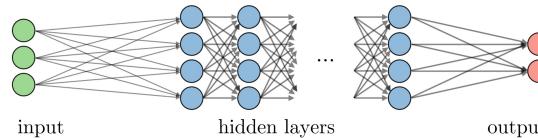
Captures the underlying themes that appear in documents

Latent Dirichlet Allocation (LDA) - generates k topics by first assigning each word to a random topic, then iteratively updating assignments based on parameters α , the mix of topics per document, and β , the distribution of words per topic

Latent Semantic Analysis (LSA) - identifies patterns using tf-idf scores and reduces data to k dimensions through SVD

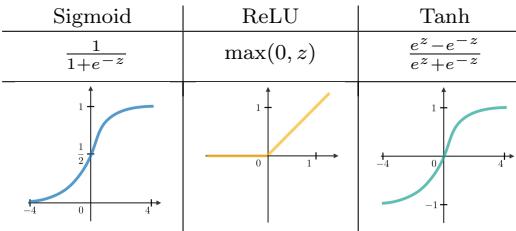
Neural Network

Feeds inputs through different hidden layers and relies on weights and nonlinear functions to reach an output



Perceptron - the foundation of a neural network that multiplies inputs by weights, adds bias, and feeds the result z to an activation function

Activation Function - defines a node's output



Softmax - given final layer outputs, provides class probabilities that sum to 1 → $\frac{e^{z_i}}{\sum e^{z_i}}$

If there is more than one 'correct' label, the sigmoid function provides probabilities for all, some, or none of the labels.

Loss Function - measures prediction error using functions such as MSE for regression and binary cross-entropy for probability-based classification

Gradient Descent - minimizes the average loss by moving iteratively in the direction of steepest descent, controlled by the learning rate γ (step size). Note, γ can be updated adaptively for better performance. For neural networks, finding the best set of weights involves:

1. Initialize weights W randomly with near-zero values
2. Loop until convergence:
 - Calculate the average network loss $J(W)$
 - **Backpropagation** - iterate backwards from the last layer, computing the gradient $\frac{\partial J(W)}{\partial W}$ and updating the weight $W \leftarrow W - \gamma \frac{\partial J(W)}{\partial W}$
3. Return the minimum loss weight matrix W

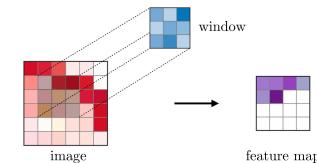
To prevent overfitting, regularization can be applied by:

- Stopping training when validation performance drops
- Dropout - randomly drop some nodes during training to prevent over-reliance on a single node
- Embedding weight penalties into the objective function
- Batch Normalization - stabilizes learning by normalizing inputs to a layer

Stochastic Gradient Descent - only uses a single point to compute gradients, leading to smoother convergence and faster compute speeds. Alternatively, mini-batch gradient descent trains on small subsets of the data, striking a balance between the approaches.

Convolutional Neural Network

Analyzes structural or visual data by extracting local features
Convolutional Layers - iterate over windows of the image, applying weights, bias, and an activation function to create feature maps. Different weights lead to different feature maps.



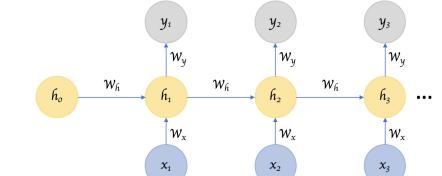
Pooling - downsamples convolution layers to reduce dimensionality and maintain spatial invariance, allowing detection of features even if they have shifted slightly. Common techniques return the max or average value in the pooling window.

The general CNN architecture is as follows:

1. Perform a series of convolution, ReLU, and pooling operations, extracting important features from the data
2. Feed output into a fully-connected layer for classification, object detection, or other structural analyses

Recurrent Neural Network

Predicts sequential data using a temporally connected system that captures both new inputs and previous outputs using hidden states



RNNs can model various input-output scenarios, such as many-to-one, one-to-many, and many-to-many. Relies on parameter (weight) sharing for efficiency. To avoid redundant calculations during backpropagation, downstream gradients are found by chaining previous gradients. However, repeatedly multiplying values greater than or less than 1 leads to:

- Exploding gradients - model instability and overflows
- Vanishing gradients - loss of learning ability

This can be solved using:

- Gradient clipping - cap the maximum value of gradients
- ReLU - its derivative prevents gradient shrinkage for $x > 0$
- Gated cells - regulate the flow of information

Long Short-Term Memory - learns long-term dependencies using gated cells and maintains a separate cell state from what is outputted. Gates in LSTM perform the following:

1. Forget and filter out irrelevant info from previous layers
2. Store relevant info from current input
3. Update the current cell state
4. Output the hidden state, a filtered version of the cell state

LSTMs can be stacked to improve performance.

Boosting

Sequentially fits many simple models that account for the previous model's errors. As opposed to bagging, boosting trains on all the data and combines models using the learning rate α .

AdaBoost - uses sample weighting and decision 'stumps' (one-level decision trees) to classify samples

1. Build decision stumps for every feature, choosing the one with the best classification accuracy
2. Assign more weight to misclassified samples and reward trees that differentiate them, where $\alpha = \frac{1}{2} \ln \frac{1 - TotalError}{TotalError}$
3. Continue training and weighting decision stumps until convergence

Gradient Boost - trains sequential models by minimizing a given loss function using gradient descent at each step

1. Start by predicting the average value of the response
2. Build a tree on the errors, constrained by depth or the number of leaf nodes
3. Scale decision trees by a constant learning rate α
4. Continue training and weighting decision trees until convergence

XGBoost - fast gradient boosting method that utilizes regularization and parallelization

Recommender Systems

Suggests relevant items to users by predicting ratings and preferences, and is divided into two main types:

- Content Filtering - recommends similar items
- Collaborative Filtering - recommends what similar users like

The latter is more common, and includes methods such as:

Memory-based Approaches - finds neighborhoods by using rating data to compute user and item similarity, measured using correlation or cosine similarity

- User-User - similar users also liked...
 - Leads to more diverse recommendations, as opposed to just recommending popular items
 - Suffers from sparsity, as the number of users who rate items is often low
- Item-Item - similar users who liked this item also liked...
 - Efficient when there are more users than items, since the item neighborhoods update less frequently than users
 - Similarity between items is often more reliable than similarity between users

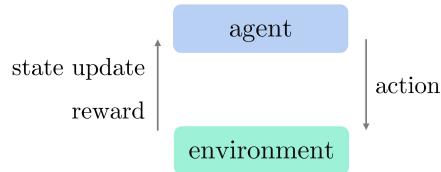
Model-based Approaches - predict ratings of unrated items, through methods such as Bayesian networks, SVD, and clustering. Handles sparse data better than memory-based approaches.

- Matrix Factorization - decomposes the user-item rating matrix into two lower-dimensional matrices representing the users and items, each with k latent factors

Recommender systems can also be combined through ensemble methods to improve performance.

Reinforcement Learning

Maximizes future rewards by learning through state-action pairs. That is, an *agent* performs *actions* in an *environment*, which updates the *state* and provides a *reward*.



Multi-armed Bandit Problem - a gambler plays slot machines with unknown probability distributions and must decide the best strategy to maximize reward. This exemplifies the exploration-exploitation tradeoff, as the best long-term strategy may involve short-term sacrifices.

RL is divided into two types, with the former being more common:

- Model-free - learn through trial and error in the environment
- Model-based - access to the underlying (approximate) state-reward distribution

Q-Value $Q(s, a)$ - captures the expected discounted total future reward given a state and action

Policy - chooses the best actions for an agent at various states $\pi(s) = \arg \max_a Q(s, a)$

Deep RL algorithms can further be divided into two main types, depending on their learning objective

Value Learning - aims to approximate $Q(s, a)$ for all actions the agent can take, but is restricted to discrete action spaces. Can use the ϵ -greedy method, where ϵ measures the probability of exploration. If chosen, the next action is selected uniformly at random.

- Q-Learning - simple value iteration model that maximizes the Q-value using a table on states and actions
- Deep Q Network - finds the best action to take by minimizing the Q-loss, the squared error between the target Q-value and the prediction

Policy Gradient Learning - directly optimize the policy $\pi(s)$ through a probability distribution of actions, without the need for a value function, allowing for continuous action spaces.

Actor-Critic Model - hybrid algorithm that relies on two neural networks, an actor $\pi(s, a, \theta)$ which controls agent behavior and a critic $Q(s, a, w)$ that measures how good an action is. Both run in parallel to find the optimal weights θ, w to maximize expected reward. At each step:

1. Pass the current state into the actor and critic
2. The critic evaluates the action's Q-value, and the actor updates its weight θ
3. The actor takes the next action leading to a new state, and the critic updates its weight w

Anomaly Detection

Identifies unusual patterns that differ from the majority of the data. Assumes that anomalies are:

- Rare - the minority class that occurs rarely in the data
- Different - have feature values that are very different from normal observations

Anomaly detection techniques spans a wide range, including methods based on:

Statistics - relies on various statistical methods to identify outliers, such as Z-tests, boxplots, interquartile ranges, and variance comparisons

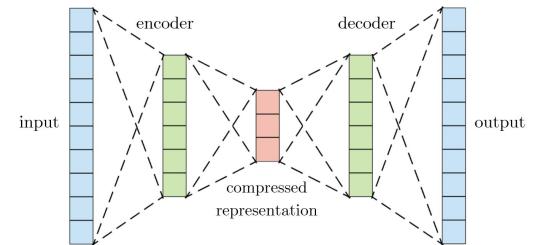
Density - useful when data is grouped around dense neighborhoods, measured by distance. Methods include k-nearest neighbors, local outlier factor, and isolation forest.

- Isolation Forest - tree-based model that labels outliers based on an anomaly score

1. Select a random feature and split value, dividing the dataset in two
 2. Continue splitting randomly until every point is isolated
 3. Calculate the anomaly score for each observation, based on how many iterations it took to isolate that point.
 4. If the anomaly score is greater than a threshold, mark it as an outlier
- Intuitively, outliers are easier to isolate and should have shorter path lengths in the tree

Clusters - data points outside of clusters could potentially be marked as anomalies

Autoencoders - unsupervised neural networks that compress data through an encoder and reconstruct it using a decoder. Autoencoders do not reconstruct the data perfectly, but rather focus on capturing important features in the data.



The decoder struggles to capture anomalous patterns, and the reconstruction error acts as a score to detect anomalies.

Autoencoders can also be used for image processing, dimension reduction, and information retrieval.

Hidden Markov Model - uses observed events O to model a set of n underlying states Q using $\lambda = (A, B, \pi)$

- A - $n \times n$ matrix of transition probabilities from state i to j
- B - sequence of likelihoods of emitting o_t in state i
- π - initial probability distribution over states

HMMs can calculate $P(O|\lambda)$, find the best hidden state sequence Q , or learn the parameters A and B . Anomalies are observations that are unlikely to occur across states.

HMMs can be applied to many problems such as signal processing and part of speech tagging.

Time Series

Extracts characteristics from time-sequenced data, which may exhibit the following characteristics:

- Stationarity - statistical properties such as mean, variance, and auto correlation are constant over time
- Trend - long-term rise or fall in values
- Seasonality - variations associated with specific calendar times, occurring at regular intervals less than a year
- Cyclical - variations without a fixed time length, occurring in periods of greater or less than one year
- Autocorrelation - degree of linear similarity between current and lagged values

CV must account for the time aspect, such as for each fold F_x :

- Sliding Window - train F_1 , test F_2 , then train F_2 , test F_3
- Forward Chain - train F_1 , test F_2 , then train F_1, F_2 , test F_3

Exponential Smoothing - uses an exponentially decreasing weight to observations over time, and takes a moving average. The time t output is $s_t = \alpha x_t + (1 - \alpha)s_{t-1}$, where $0 < \alpha < 1$.

Double Exponential Smoothing - applies a recursive exponential filter to capture trends within a time series

$$\begin{aligned}s_t &= \alpha x_t + (1 - \alpha)(s_{t-1} + b_{t-1}) \\ b_t &= \beta(s_t - s_{t-1}) + (1 - \beta)b_{t-1}\end{aligned}$$

Triple exponential smoothing adds a third variable γ that accounts for seasonality.

ARIMA - models time series using three parameters (p, d, q) :

- Autoregressive - the past p values affect the next value
- Integrated - values are replaced with the difference between current and previous values, using the difference degree d (0 for stationary data, and 1 for non-stationary)
- Moving Average - the number of lagged forecast errors and the size of the moving average window q

SARIMA - models seasonality through four additional seasonality-specific parameters: P , D , Q , and the seasonal length s

Prophet - additive model that uses non-linear trends to account for multiple seasonalities such as yearly, weekly, and daily. Robust to missing data and handles outliers well. Can be represented as: $y(t) = g(t) + s(t) + h(t) + \epsilon(t)$, with four distinct components for the growth over time, seasonality, holiday effects, and error. This specification is similar to a generalized additive model.

Generalized Additive Model - combine predictive methods while preserving additivity across variables, in a form such as $y = \beta_0 + f_1(x_1) + \dots + f_m(x_m)$, where functions can be non-linear. GAMs also provide regularized and interpretable solutions for regression and classification problems.

Naive Bayes

Classifies data using the label with the highest conditional probability, given data a and classes c . Naive because it assumes variables are independent.

$$\text{Bayes' Theorem } P(c_i|a) = \frac{P(a|c_i)P(c_i)}{P(a)}$$

Gaussian Naive Bayes - calculates conditional probability for continuous data by assuming a normal distribution

Statistics

p-value - probability that an effect could have occurred by chance. If less than the significance level α , or if the test statistic is greater than the critical value, then reject the null.

Type I Error (False Positive α) - rejecting a true null

Type II Error (False Negative β) - not rejecting a false null

Decreasing Type I Error causes an increase in Type II Error

Confidence Level $(1 - \alpha)$ - probability of finding an effect that did not occur by chance and avoiding a Type I error

Power $(1 - \beta)$ - probability of picking up on an effect that is present and avoiding a Type II Error

Confidence Interval - estimated interval that models the long-term frequency of capturing the true parameter value

z-test - tests whether normally distributed population means are different, used when n is large and variances are known

- z-score - the number of standard deviations between a data point x and the mean $\rightarrow \frac{x - \mu}{\sigma}$

t-test - used when population variances are unknown, and converges to the z-test when n is large

- t-score - uses the standard error as an estimate for population variance $\rightarrow \frac{x - \mu}{s/\sqrt{n}}$

Degrees of Freedom - the number of independent (free) dimensions needed before the parameter estimate can be determined

Chi-Square Tests - measure differences between categorical variables, using $\chi^2 = \sum \frac{\text{observed} - \text{expected}}{\text{expected}}$ to test:

- Goodness of fit - if samples of one categorical variable match the population category expectations
- Independence - if being in one category is independent of another, based off two categories
- Homogeneity - if different subgroups come from the same population, based off a single category

ANOVA - analysis of variance, used to compare 3+ samples

- F-score - compares the ratio of explained and unexplained variance $\rightarrow \frac{\text{between group variance}}{\text{within group variance}}$

Conditional Probability $P(A | B) = \frac{P(A \cap B)}{P(B)}$

If A and B are independent, then $P(A \cap B) = P(A)P(B)$.

Note, events that are independent of themselves must have probability either 1 or 0.

Union $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

Mutually Exclusive - events cannot happen simultaneously

Expected Value $E[X] = \sum x_i p_i$, with properties

- $E[X + Y] = E[X] + E[Y]$
- $E[XY] = E[X]E[Y]$ if X and Y are independent

Variance $\text{Var}(X) = E[X^2] - E[X]^2$, with properties

- $\text{Var}(X \pm Y) = \text{Var}(X) + \text{Var}(Y) \pm 2\text{Cov}(X, Y)$
- $\text{Var}(aX \pm b) = a^2\text{Var}(X)$

Covariance - measures the direction of the joint linear relationship of two variables $\rightarrow \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{n-1}$

Correlation - normalizes covariance to provide both strength and direction of linear relationships $\rightarrow r = \frac{\text{Cov}(x, y)}{\sigma_x \sigma_y}$

Independent variables are uncorrelated, though the inverse is not necessarily true

A/B Testing

Examines user experience through randomized tests with two variants. The typical steps are:

1. Determine the evaluation metric and experiment goals
2. Select a significance level α and power threshold $1 - \beta$
3. Calculate the required sample size per variation
4. Randomly assign users into control and treatment groups
5. Measure and analyze results using the appropriate test

The required sample size depends on α , β , and the MDE

Minimum Detectable Effect - the target relative minimum increase over the baseline that should be observed from a test

Overall Evaluation Criterion - quantitative measure of the test's objective, commonly used when short and long-term metrics have inverse relationships

Multivariate Testing - compares 3+ variants or combinations, but requires larger sample sizes

Bonferroni Correction - when conducting n tests, run each test at the $\frac{\alpha}{n}$ significance level, which lowers the false positive rate of finding effects by chance

Network Effects - changes that occur due to effect spillover from other groups. To detect group interference:

1. Split the population into distinct clusters
2. Randomly assign half the clusters to the control and treatment groups A_1 and B_1
3. Randomize the other half at the user-level and assign to control and treatment groups A_2 and B_2
4. Intuitively, if there are network effects, then the tests will have different results

To account for network effects, randomize users based on time, cluster, or location

Sequential Testing - allows for early experiment stopping by drawing statistical borders based on the Type I Error rate. If the effect reaches a border, the test can be stopped. Used to combat *peeking* (preliminarily checking results of a test), which can inflate *p*-values and lead to incorrect conclusions.

Cohort Analysis - examines specific groups of users based on behavior or time and can help identify whether novelty or primacy effects are present

Miscellaneous

Shapley Values - measures the marginal contribution of each variable in the output of a model, where the sum of all Shapley values equals the total value (prediction - mean prediction)

SHAP - interpretable Shapley method that utilizes both global and local importance to model variable explainability

Permutation - order matters $\rightarrow \frac{n!}{(n-k)!} = {}^n P_k$

Combination - order doesn't matter $\rightarrow \frac{n!}{k!(n-k)!} = {}^n C_k = \binom{n}{k}$

Left Skew - Mean < Median \leq Mode

Right Skew - Mean > Median \geq Mode

Probability vs Likelihood - given a situation θ and observed outcomes O , probability is calculated as $P(O|\theta)$. However, when true values for θ are unknown, O is used to estimate the θ that maximizes the likelihood function. That is, $L(\theta|O) = P(O|\theta)$.



Keywords

Keyword	Description	Code Examples
<code>False, True</code>	Boolean data type	<code>False == (1 > 2)</code> <code>True == (2 > 1)</code>
<code>and, or, not</code>	Logical operators → Both are true → Either is true → Flips Boolean	<code>True and True # True</code> <code>True or False # True</code> <code>not False # True</code>
<code>break</code>	Ends loop prematurely	<code>while True:</code> <code>break # finite loop</code>
<code>continue</code>	Finishes current loop iteration	<code>while True:</code> <code>continue</code> <code>print("42") # dead code</code>
<code>class</code>	Defines new class	<code>class Coffee:</code> <code># Define your class</code>
<code>def</code>	Defines a new function or class method.	<code>def say_hi():</code> <code>print('hi')</code>
<code>if, elif, else</code>	Conditional execution: - "if" condition == True? - "elif" condition == True? - Fallback: else branch	<code>x = int(input("ur val:"))</code> <code>if x > 3: print("Big")</code> <code>elif x == 3: print("3")</code> <code>else: print("Small")</code>
<code>for, while</code>	# For loop for i in [0,1,2]: print(i)	# While loop does same <code>j = 0</code> <code>while j < 3:</code> <code>print(j); j = j + 1</code>
<code>in</code>	Sequence membership	<code>42 in [2, 39, 42] # True</code>
<code>is</code>	Same object memory location	<code>y = x = 3</code> <code>x is y # True</code> <code>[3] is [3] # False</code>
<code>None</code>	Empty value constant	<code>print() is None # True</code>
<code>lambda</code>	Anonymous function	<code>(lambda x: x+3)(3) # 6</code>
<code>return</code>	Terminates function. Optional return value defines function result.	<code>def increment(x):</code> <code>return x + 1</code> <code>increment(4) # returns 5</code>

Basic Data Structures

Type	Description	Code Examples
<code>Boolean</code>	The Boolean data type is either <code>True</code> or <code>False</code> . Boolean operators are ordered by priority: <code>not</code> → <code>and</code> → <code>or</code>	<code>## Evaluates to True:</code> <code>1<2 and 0<1 and 3>2 and 2>=2 and 1==1 and 1!=0</code> <code>## Evaluates to False:</code> <code>bool(None or 0 or 0.0 or '' or [] or {} or set())</code> <code>Rule: None, 0, 0.0, empty strings, or empty container types evaluate to False</code>
<code>Integer, Float</code>	An <code>integer</code> is a positive or negative number without decimal point such as 3. A <code>float</code> is a positive or negative number with floating point precision such as 3.1415926. <code>Integer division rounds toward the smaller integer</code> (example: <code>3//2==1</code>).	<code>## Arithmetic Operations</code> <code>x, y = 3, 2</code> <code>print(x + y) # = 5</code> <code>print(x - y) # = 1</code> <code>print(x * y) # = 6</code> <code>print(x / y) # = 1.5</code> <code>print(x // y) # = 1</code> <code>print(x % y) # = 1</code> <code>print(-x) # = -3</code> <code>print(abs(-x)) # = 3</code> <code>print(int(3.9)) # = 3</code> <code>print(float(3)) # = 3.0</code> <code>print(x ** y) # = 9</code>
<code>String</code>	Python Strings are sequences of characters.	<code>## Indexing and Slicing</code> <code>s = "The youngest pope was 11 years"</code> <code>s[0] # 'T'</code> <code>s[1:3] # 'he'</code> <code>s[-3:-1] # 'ar'</code> <code>s[-3:] # 'ars'</code> <code>Slice [:2]</code> <code>1 2 3 4</code> <code>0 1 2 3</code> <code>x = s.split()</code> <code>x[-2] + " " + x[2] + "s" # '11 popes'</code> <code>## String Methods</code> <code>y = "Hello world\t\n "</code> <code>y.strip() # Remove Whitespace</code> <code>"HI".lower() # Lowercase: 'hi'</code> <code>"HI".upper() # Uppercase: 'HI'</code> <code>"hello".startswith("he") # True</code> <code>"hello".endswith("lo") # True</code> <code>"hello".find("ll") # Match at 2</code> <code>"cheat".replace("ch", "m") # 'meat'</code> <code>...join(["F", "B", "I"]) # 'FBI'</code> <code>len("hello world") # Length: 15</code> <code>"ear" in "earth" # True</code>

Complex Data Structures

Type	Description	Example
<code>List</code>	Stores a sequence of elements. Unlike strings, you can modify list objects (they're <i>mutable</i>).	<code>l = [1, 2, 2]</code> <code>print(len(l)) # 3</code>
<code>Adding elements</code>	Add elements to a list with (i) <code>append</code> , (ii) <code>insert</code> , or (iii) list concatenation.	<code>[1, 2].append(4) # [1, 2, 4]</code> <code>[1, 4].insert(1,9) # [1, 9, 4]</code> <code>[1, 2] + [4] # [1, 2, 4]</code>
<code>Removal</code>	Slow for lists	<code>[1, 2, 2, 4].remove(1) # [2, 2, 4]</code>
<code>Reversing</code>	Reverses list order	<code>[1, 2, 3].reverse() # [3, 2, 1]</code>
<code>Sorting</code>	Sorts list using fast Timsort	<code>[2, 4, 2].sort() # [2, 2, 4]</code>
<code>Indexing</code>	Finds the first occurrence of an element & returns index. Slow worst case for whole list traversal.	<code>[2, 2, 4].index(2)</code> # index of item 2 is 0 <code>[2, 2, 4].index(2,1)</code> # index of item 2 after pos 1 is 1
<code>Stack</code>	Use Python lists via the list operations <code>append()</code> and <code>pop()</code>	<code>stack = [3]</code> <code>stack.append(42) # [3, 42]</code> <code>stack.pop() # 42 (stack: [3])</code> <code>stack.pop() # 3 (stack: [])</code>
<code>Set</code>	An unordered collection of unique elements (<i>at-most-once</i>) → fast membership $O(1)$	<code>basket = {'apple', 'eggs', 'banana', 'orange'}</code> <code>same = set(['apple', 'eggs', 'banana', 'orange'])</code>

Type	Description	Example
<code>Dictionary</code>	Useful data structure for storing (key, value) pairs	<code>cal = {'apple': 52, 'banana': 89, 'choco': 546} # calories</code>
<code>Reading and writing elements</code>	Read and write elements by specifying the key within the brackets. Use the <code>keys()</code> and <code>values()</code> functions to access all keys and values of the dictionary	<code>print(cal['apple'] < cal['choco']) # True</code> <code>cal['cappu'] = 74</code> <code>print(cal['banana'] < cal['cappu']) # False</code> <code>print('apple' in cal.keys()) # True</code> <code>print(52 in cal.values()) # True</code>
<code>Dictionary Iteration</code>	You can access the (key, value) pairs of a dictionary with the <code>items()</code> method.	<code>for k, v in cal.items():</code> <code>print(k) if v > 500 else ''</code> <code># 'choco'</code>
<code>Membership operator</code>	Check with the <code>in</code> keyword if set, list, or dictionary contains an element. Set membership is faster than list membership.	<code>basket = {'apple', 'eggs', 'banana', 'orange'}</code> <code>print('eggs' in basket) # True</code> <code>print('mushroom' in basket) # False</code>
<code>List & set comprehension</code>	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a for clause. Close with zero or more for or if clauses. Set comprehension works similar to list comprehension.	<code>l = ['hi ' + x for x in ['Alice', 'Bob', 'Pete']]</code> # ['Hi Alice', 'Hi Bob', 'Hi Pete'] <code>12 = [x * y for x in range(3) for y in range(3) if x>y] # [0, 0, 2]</code> <code>squares = { x**2 for x in [0,2,4] if x < 4 } # {0, 4}</code>

Subscribe to the **11x FREE** Python Cheat Sheet Course:
<https://blog.finxter.com/python-cheat-sheets/>



Python Cheat Sheet: Keywords

“A puzzle a day to learn, code, and play” → Visit finxter.com

Keyword	Description	Code example
<code>False, True</code>	Data values from the data type Boolean	<code>False == (1 > 2), True == (2 > 1)</code>
<code>and, or, not</code>	Logical operators: $(x \text{ and } y) \rightarrow$ both x and y must be True $(x \text{ or } y) \rightarrow$ either x or y must be True $(\text{not } x) \rightarrow$ x must be false	<code>x, y = True, False</code> <code>(x or y) == True # True</code> <code>(x and y) == False # True</code> <code>(not y) == True # True</code>
<code>break</code>	Ends loop prematurely	<code>while(True):</code> <code>break # no infinite loop</code> <code>print("hello world")</code>
<code>continue</code>	Finishes current loop iteration	<code>while(True):</code> <code>continue</code> <code>print("43") # dead code</code>
<code>class</code>	Defines a new class \rightarrow a real-world concept (object oriented programming)	<code>class Beer:</code> <code>def __init__(self):</code> <code>self.content = 1.0</code> <code>def drink(self):</code> <code>self.content = 0.0</code>
<code>def</code>	Defines a new function or class method. For latter, first parameter (“self”) points to the class object. When calling class method, first parameter is implicit.	<code>becks = Beer() # constructor - create class</code> <code>becks.drink() # beer empty: b.content == 0</code>
<code>if, elif, else</code>	Conditional program execution: program starts with “if” branch, tries the “elif” branches, and finishes with “else” branch (until one branch evaluates to True).	<code>x = int(input("your value: "))</code> <code>if x > 3: print("Big")</code> <code>elif x == 3: print("Medium")</code> <code>else: print("Small")</code>
<code>for, while</code>	<code># For loop declaration</code> <code>for i in [0,1,2]:</code> <code>print(i)</code>	<code># While loop - same semantics</code> <code>j = 0</code> <code>while j < 3:</code> <code>print(j)</code> <code>j = j + 1</code>
<code>in</code>	Checks whether element is in sequence	<code>42 in [2, 39, 42] # True</code>
<code>is</code>	Checks whether both elements point to the same object	<code>y = x = 3</code> <code>x is y # True</code> <code>[3] is [3] # False</code>
<code>None</code>	Empty value constant	<code>def f():</code> <code>x = 2</code> <code>f() is None # True</code>
<code>lambda</code>	Function with no name (anonymous function)	<code>(lambda x: x + 3)(3) # returns 6</code>
<code>return</code>	Terminates execution of the function and passes the flow of execution to the caller. An optional value after the return keyword specifies the function result.	<code>def incrementor(x):</code> <code>return x + 1</code> <code>incrementor(4) # returns 5</code>

Python Cheat Sheet: Basic Data Types

“A puzzle a day to learn, code, and play” → Visit finxter.com

	Description	Example
Boolean	<p>The Boolean data type is a truth value, either <code>True</code> or <code>False</code>.</p> <p>The Boolean operators ordered by priority: <code>not x</code> → “if x is False, then x, else y” <code>x and y</code> → “if x is False, then x, else y” <code>x or y</code> → “if x is False, then y, else x”</p> <p>These comparison operators evaluate to True: <code>1 < 2 and 0 <= 1 and 3 > 2 and 2 >= 2 and 1 == 1 and 1 != 0</code> # True</p>	<pre>## 1. Boolean Operations x, y = True, False print(x and not y) # True print(not x and y or x) # True ## 2. If condition evaluates to False if None or 0 or 0.0 or '' or [] or {} or set(): # None, 0, 0.0, empty strings, or empty # container types are evaluated to False print("Dead code") # Not reached</pre>
Integer, Float	<p>An integer is a positive or negative number without floating point (e.g. <code>3</code>). A float is a positive or negative number with floating point precision (e.g. <code>3.14159265359</code>).</p> <p>The <code>//</code> operator performs integer division. The result is an integer value that is rounded toward the smaller integer number (e.g. <code>3 // 2 == 1</code>).</p>	<pre>## 3. Arithmetic Operations x, y = 3, 2 print(x + y) # = 5 print(x - y) # = 1 print(x * y) # = 6 print(x / y) # = 1.5 print(x // y) # = 1 print(x % y) # = 1s print(-x) # = -3 print(abs(-x)) # = 3 print(int(3.9)) # = 3 print(float(3)) # = 3.0 print(x ** y) # = 9</pre>
String	<p>Python Strings are sequences of characters.</p> <p>The four main ways to create strings are the following.</p> <ol style="list-style-type: none">1. Single quotes <code>'Yes'</code>2. Double quotes <code>"Yes"</code>3. Triple quotes (multi-line) <code>"""Yes We Can"""</code>4. String method <code>str(5) == '5' # True</code>5. Concatenation <code>"Ma" + "hatma" # 'Mahatma'</code> <p>These are whitespace characters in strings.</p> <ul style="list-style-type: none">• Newline <code>\n</code>• Space <code>\s</code>• Tab <code>\t</code>	<pre>## 4. Indexing and Slicing s = "The youngest pope was 11 years old" print(s[0]) # 'T' print(s[1:3]) # 'he' print(s[-3:-1]) # 'ol' print(s[-3:]) # 'old' x = s.split() # creates string array of words print(x[-3] + " " + x[-1] + " " + x[2] + "s") # '11 old popes' ## 5. Most Important String Methods y = " This is lazy\t\n " print(y.strip()) # Remove Whitespace: 'This is lazy' print("DrDre".lower()) # Lowercase: 'drdre' print("attention".upper()) # Uppercase: 'ATTENTION' print("smartphone".startswith("smart")) # True print("smartphone".endswith("phone")) # True print("another".find("other")) # Match index: 2 print("cheat".replace("ch", "m")) # 'meat' print(','.join(["F", "B", "I"])) # 'F,B,I' print(len("Rumpelstiltskin")) # String length: 15 print("ear" in "earth") # Contains: True</pre>

Python Cheat Sheet: Complex Data Types

“A puzzle a day to learn, code, and play” → Visit finxter.com

	Description	Example
List	A container data type that stores a sequence of elements. Unlike strings, lists are mutable: modification possible.	<pre>l = [1, 2, 2] print(len(l)) # 3</pre>
Adding elements	Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation. The append operation is very fast.	<pre>[1, 2, 2].append(4) # [1, 2, 2, 4] [1, 2, 4].insert(2,2) # [1, 2, 2, 4] [1, 2, 2] + [4] # [1, 2, 2, 4]</pre>
Removal	Removing an element can be slower.	<pre>[1, 2, 2, 4].remove(1) # [2, 2, 4]</pre>
Reversing	This reverses the order of list elements.	<pre>[1, 2, 3].reverse() # [3, 2, 1]</pre>
Sorting	Sorts a list. The computational complexity of sorting is linear in the no. list elements.	<pre>[2, 4, 2].sort() # [2, 2, 4]</pre>
Indexing	Finds the first occurrence of an element in the list & returns its index. Can be slow as the whole list is traversed.	<pre>[2, 2, 4].index(2) # index of element 4 is "0" [2, 2, 4].index(2,1) # index of element 2 after pos 1 is "1"</pre>
Stack	Python lists can be used intuitively as stacks via the two list operations <code>append()</code> and <code>pop()</code> .	<pre>stack = [3] stack.append(42) # [3, 42] stack.pop() # 42 (stack: [3]) stack.pop() # 3 (stack: [])</pre>
Set	A set is an unordered collection of unique elements (“at-most-once”).	<pre>basket = {'apple', 'eggs', 'banana', 'orange'} same = set(['apple', 'eggs', 'banana', 'orange'])</pre>
Dictionary	The dictionary is a useful data structure for storing (key, value) pairs.	<pre>calories = {'apple' : 52, 'banana' : 89, 'choco' : 546}</pre>
Reading and writing elements	Read and write elements by specifying the key within the brackets. Use the <code>keys()</code> and <code>values()</code> functions to access all keys and values of the dictionary.	<pre>print(calories['apple'] < calories['choco']) # True calories['cappu'] = 74 print(calories['banana'] < calories['cappu']) # False print('apple' in calories.keys()) # True print(52 in calories.values()) # True</pre>
Dictionary Looping	You can access the (key, value) pairs of a dictionary with the <code>items()</code> method.	<pre>for k, v in calories.items(): print(k) if v > 500 else None # 'chocolate'</pre>
Membership operator	Check with the ‘in’ keyword whether the set, list, or dictionary contains an element. Set containment is faster than list containment.	<pre>basket = {'apple', 'eggs', 'banana', 'orange'} print('eggs' in basket) # True print('mushroom' in basket) # False</pre>
List and Set Comprehension	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a for clause. Close with zero or more for or if clauses. Set comprehension is similar to list comprehension.	<pre># List comprehension l = [('Hi ' + x) for x in ['Alice', 'Bob', 'Pete']] print(l) # ['Hi Alice', 'Hi Bob', 'Hi Pete'] l2 = [x * y for x in range(3) for y in range(3) if x>y] print(l2) # [0, 0, 2] # Set comprehension squares = { x**2 for x in [0,2,4] if x < 4 } # {0, 4}</pre>

Python Cheat Sheet: Classes

“A puzzle a day to learn, code, and play” → Visit finxter.com

	Description	Example
Classes	<p>A class encapsulates data and functionality: data as attributes, and functionality as methods. It is a blueprint for creating concrete instances in memory.</p> <p>Class Instances</p> <p>name = "Alice" state = "sleeping" color = "grey"</p> <p>name = "Bello" state = "wag tail" color = "black"</p>	<pre>class Dog: """ Blueprint of a dog """ # class variable shared by all instances species = ["canis lupus"] def __init__(self, name, color): self.name = name self.state = "sleeping" self.color = color def command(self, x): if x == self.name: self.bark(2) elif x == "sit": self.state = "sit" else: self.state = "wag tail" def bark(self, freq): for i in range(freq): print("[" + self.name + "]: Woof!") bello = Dog("bello", "black") alice = Dog("alice", "white") print(bello.color) # black print(alice.color) # white bello.bark(1) # [bello]: Woof! alice.command("sit") print("[alice]: " + alice.state) # [alice]: sit bello.command("no") print("[bello]: " + bello.state) # [bello]: wag tail alice.command("alice") # [alice]: Woof! # [alice]: Woof! bello.species += ["wulf"] print(len(bello.species)) == len(alice.species)) # True (!)</pre>
Instance	<p>You are an instance of the class <code>human</code>. An instance is a concrete implementation of a class: all attributes of an instance have a fixed value. Your hair is blond, brown, or black--but never unspecified.</p> <p>Each instance has its own attributes independent of other instances. Yet, class variables are different. These are data values associated with the class, not the instances. Hence, all instance share the same class variable <code>species</code> in the example.</p>	
Self	<p>The first argument when defining any method is always the <code>self</code> argument. This argument specifies the instance on which you call the method.</p> <p><code>self</code> gives the Python interpreter the information about the concrete instance. To <i>define</i> a method, you use <code>self</code> to modify the instance attributes. But to <i>call</i> an instance method, you do not need to specify <code>self</code>.</p>	
Creation	<p>You can create classes “on the fly” and use them as logical units to store complex data types.</p> <pre>class Employee(): pass employee = Employee() employee.salary = 122000 employee.firstname = "alice" employee.lastname = "wonderland" print(employee.firstname + " " + employee.lastname + " " + str(employee.salary) + "\$") # alice wonderland 122000\$</pre>	

Python Cheat Sheet: Functions and Tricks

“A puzzle a day to learn, code, and play” → Visit finxter.com

		Description	Example	Result
ADVANCED FUNCTIONS	map(func, iter)	Executes the function on all elements of the iterable	<code>list(map(lambda x: x[0], ['red', 'green', 'blue']))</code>	['r', 'g', 'b']
	map(func, i1, ..., ik)	Executes the function on all k elements of the k iterables	<code>list(map(lambda x, y: str(x) + ' ' + y + 's', [0, 2, 2], ['apple', 'orange', 'banana']))</code>	['0 apples', '2 oranges', '2 bananas']
	string.join(iter)	Concatenates iterable elements separated by <code>string</code>	<code>' marries '.join(list(['Alice', 'Bob']))</code>	'Alice marries Bob'
	filter(func, iterable)	Filters out elements in iterable for which function returns <code>False</code> (or 0)	<code>list(filter(lambda x: True if x>17 else False, [1, 15, 17, 18]))</code>	[18]
	string.strip()	Removes leading and trailing whitespaces of string	<code>print("\n\t 42 \t".strip())</code>	42
	sorted(iter)	Sorts iterable in ascending order	<code>sorted([8, 3, 2, 42, 5])</code>	[2, 3, 5, 8, 42]
	sorted(iter, key=key)	Sorts according to the key function in ascending order	<code>sorted([8, 3, 2, 42, 5], key=lambda x: 0 if x==42 else x)</code>	[42, 2, 3, 5, 8]
	help(func)	Returns documentation of <code>func</code>	<code>help(str.upper())</code>	'... to uppercase.'
	zip(i1, i2, ...)	Groups the i-th elements of iterators <code>i1</code> , <code>i2</code> , ... together	<code>list(zip(['Alice', 'Anna'], ['Bob', 'Jon', 'Frank']))</code>	[('Alice', 'Bob'), ('Anna', 'Jon')]
TRICKS	Unzip	Equal to: 1) unpack the zipped list, 2) zip the result	<code>list(zip(*[('Alice', 'Bob'), ('Anna', 'Jon')]))</code>	[('Alice', 'Anna'), ('Bob', 'Jon')]
	enumerate(iter)	Assigns a counter value to each element of the iterable	<code>list(enumerate(['Alice', 'Bob', 'Jon']))</code>	[(0, 'Alice'), (1, 'Bob'), (2, 'Jon')]
	python -m http.server <P>	Want to share files between PC and phone? Run this command in PC's shell. <P> is any port number 0–65535. Type <IP address of PC>:<P> in the phone's browser. You can now browse the files in the PC directory.		
	Read comic	<code>import antigravity</code>	Open the comic series xkcd in your web browser	
	Zen of Python	<code>import this</code>	'...Beautiful is better than ugly. Explicit is ...'	
	Swapping numbers	Swapping variables is a breeze in Python. No offense, Java!	<code>a, b = 'Jane', 'Alice' a, b = b, a</code>	<code>a = 'Alice' b = 'Jane'</code>
	Unpacking arguments	Use a sequence as function arguments via asterisk operator *. Use a dictionary (key, value) via double asterisk operator **	<code>def f(x, y, z): return x + y * z f(*[1, 3, 4]) f(**{'z' : 4, 'x' : 1, 'y' : 3})</code>	13 13
Extended Unpacking	Extended Unpacking	Use unpacking for multiple assignment feature in Python	<code>a, *b = [1, 2, 3, 4, 5]</code>	<code>a = 1 b = [2, 3, 4, 5]</code>
	Merge two dictionaries	Use unpacking to merge two dictionaries into a single one	<code>x={'Alice' : 18} y={'Bob' : 27, 'Ann' : 22} z = {**x, **y}</code>	<code>z = {'Alice': 18, 'Bob': 27, 'Ann': 22}</code>

Python Cheat Sheet: 14 Interview Questions

"A puzzle a day to learn, code, and play" → Visit finxter.com

Question	Code	Question	Code
Check if list contains integer x	<pre>l = [3, 3, 4, 5, 2, 111, 5] print(111 in l) # True</pre>	Get missing number in [1...100]	<pre>def get_missing_number(lst): return set(range(lst[0], lst[-1])) - set(lst) l = list(range(1, 100)) l.remove(50) print(get_missing_number(l)) # 50</pre>
Find duplicate number in integer list	<pre>def find_duplicates(elements): duplicates, seen = set(), set() for element in elements: if element in seen: duplicates.add(element) seen.add(element) return list(duplicates)</pre>	Compute the intersection of two lists	<pre>def intersect(lst1, lst2): res, lst2_copy = [], lst2[:] for el in lst1: if el in lst2_copy: res.append(el) lst2_copy.remove(el) return res</pre>
Check if two strings are anagrams	<pre>def is_anagram(s1, s2): return set(s1) == set(s2) print(is_anagram("elvis", "lives")) # True</pre>	Find max and min in unsorted list	<pre>l = [4, 3, 6, 3, 4, 888, 1, -11, 22, 3] print(max(l)) # 888 print(min(l)) # -11</pre>
Remove all duplicates from list	<pre>lst = list(range(10)) + list(range(10)) lst = list(set(lst)) print(lst) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]</pre>	Reverse string using recursion	<pre>def reverse(string): if len(string)<=1: return string return reverse(string[1:])+string[0] print(reverse("hello")) # olleh</pre>
Find pairs of integers in list so that their sum is equal to integer x	<pre>def find_pairs(l, x): pairs = [] for (i, el_1) in enumerate(l): for (j, el_2) in enumerate(l[i+1:]): if el_1 + el_2 == x: pairs.append((el_1, el_2)) return pairs</pre>	Compute the first n Fibonacci numbers	<pre>a, b = 0, 1 n = 10 for i in range(n): print(b) a, b = b, a+b # 1, 1, 2, 3, 5, 8, ...</pre>
Check if a string is a palindrome	<pre>def is_palindrome(phrase): return phrase == phrase[::-1] print(is_palindrome("anna")) # True</pre>	Sort list with Quicksort algorithm	<pre>def qsort(L): if L == []: return [] return qsort([x for x in L[1:] if x < L[0]]) + L[0:1] + qsort([x for x in L[1:] if x >= L[0]]) lst = [44, 33, 22, 5, 77, 55, 999] print(qsort(lst)) # [5, 22, 33, 44, 55, 77, 999]</pre>
Use list as stack, array, and queue	<pre># as a list ... l = [3, 4] l += [5, 6] # l = [3, 4, 5, 6] # ... as a stack ... l.append(10) # l = [3, 4, 5, 6, 10] l.pop() # l = [3, 4, 5] # ... and as a queue l.insert(0, 5) # l = [5, 3, 4, 5] l.pop() # l = [5, 3, 4]</pre>	Find all permutations of string	<pre>def get_permutations(w): if len(w)<=1: return set(w) smaller = get_permutations(w[1:]) perms = set() for x in smaller: for pos in range(0, len(x)+1): perm = x[:pos] + w[0] + x[pos:] perms.add(perm) return perms print(get_permutations("nan")) # {'nna', 'ann', 'nan'}</pre>

Python Cheat Sheet: NumPy

“A puzzle a day to learn, code, and play” → Visit finxter.com

Name	Description	Example
a.shape	The shape attribute of NumPy array a keeps a tuple of integers. Each integer describes the number of elements of the axis.	<pre>a = np.array([[1,2],[1,1],[0,0]]) print(np.shape(a)) # (3, 2)</pre>
a.ndim	The ndim attribute is equal to the length of the shape tuple.	<pre>print(np.ndim(a)) # 2</pre>
*	The asterisk (star) operator performs the Hadamard product, i.e., multiplies two matrices with equal shape element-wise.	<pre>a = np.array([[2, 0], [0, 2]]) b = np.array([[1, 1], [1, 1]]) print(a*b) # [[2 0] [0 2]]</pre>
np.matmul(a,b), a@b	The standard matrix multiplication operator. Equivalent to the @ operator.	<pre>print(np.matmul(a,b)) # [[2 2] [2 2]]</pre>
np.arange([start,]stop, [step,])	Creates a new 1D numpy array with evenly spaced values	<pre>print(np.arange(0,10,2)) # [0 2 4 6 8]</pre>
np.linspace(start, stop, num=50)	Creates a new 1D numpy array with evenly spread elements within the given interval	<pre>print(np.linspace(0,10,3)) # [0. 5. 10.]</pre>
np.average(a)	Averages over all the values in the numpy array	<pre>a = np.array([[2, 0], [0, 2]]) print(np.average(a)) # 1.0</pre>
<slice> = <val>	Replace the <slice> as selected by the slicing operator with the value <val>.	<pre>a = np.array([0, 1, 0, 0, 0]) a[::2] = 2 print(a) # [2 1 2 0 2]</pre>
np.var(a)	Calculates the variance of a numpy array.	<pre>a = np.array([2, 6]) print(np.var(a)) # 4.0</pre>
np.std(a)	Calculates the standard deviation of a numpy array	<pre>print(np.std(a)) # 2.0</pre>
np.diff(a)	Calculates the difference between subsequent values in NumPy array a	<pre>fibs = np.array([0, 1, 1, 2, 3, 5]) print(np.diff(fibs, n=1)) # [1 0 1 1 2]</pre>
np.cumsum(a)	Calculates the cumulative sum of the elements in NumPy array a.	<pre>print(np.cumsum(np.arange(5))) # [0 1 3 6 10]</pre>
np.sort(a)	Creates a new NumPy array with the values from a (ascending).	<pre>a = np.array([10,3,7,1,0]) print(np.sort(a)) # [0 1 3 7 10]</pre>
np.argsort(a)	Returns the indices of a NumPy array so that the indexed values would be sorted.	<pre>a = np.array([10,3,7,1,0]) print(np.argsort(a)) # [4 3 1 2 0]</pre>
np.max(a)	Returns the maximal value of NumPy array a.	<pre>a = np.array([10,3,7,1,0]) print(np.max(a)) # 10</pre>
np.argmax(a)	Returns the index of the element with maximal value in the NumPy array a.	<pre>a = np.array([10,3,7,1,0]) print(np.argmax(a)) # 0</pre>
np.nonzero(a)	Returns the indices of the nonzero elements in NumPy array a.	<pre>a = np.array([10,3,7,1,0]) print(np.nonzero(a)) # [0 1 2 3]</pre>

💡 These are some excellent Azure Data Engineer interview questions to prepare! Here's how you can tackle them effectively:

Key Answers:

1 Parameters and Variables in ADF:

- **Parameters**: Used to pass values at runtime to pipelines. They are defined at the pipeline level and cannot be changed during execution.
- **Variables**: Used to store values within the pipeline and can change during execution. Variables are updated using **Set Variable** or **Append Variable** activities.

2 Time Travel in Your Project:

- Time travel is a **Delta Lake** feature that allows querying historical data (snapshots).
- Example: `SELECT * FROM table_name VERSION AS OF 5` or `TIMESTAMP AS OF '2023-01-15'`.
- Use Case: Debugging, auditing, or recreating datasets for ML models.

3 Resume Pipeline from Failed Activity:

- Enable **checkpointing** or **activity retry** in ADF. Use a **failure path** with logic to resume execution by using the **Get Metadata** activity to evaluate where the pipeline failed.

4 Pipelines You've Worked With:

- Example: ETL pipelines to ingest and transform raw data from Azure Data Lake using Data Flows and Spark jobs.
- Mention specifics like **copy data activities**, **data validation**, and **orchestration of transformations**.

5 Partition vs. Bucketing:

- **Partitioning**: Divides the data into directories based on keys (e.g., year, month).
- **Bucketing**: Hashes data into fixed-sized buckets, optimizing joins and aggregations.

[6] Medallion Architecture:

- A data architecture that separates data into three layers:
 - **Bronze**: Raw ingested data.
 - **Silver**: Cleaned and transformed data.
 - **Gold**: Business-level aggregates and insights.

[7] Azure Key Vault:

- Securely stores secrets, keys, and certificates.
- Use **Managed Identity** in ADF to access Key Vault without hardcoding credentials.

[8] Unity Catalog vs. Hive Metastore:

- **Unity Catalog**: Centralized data governance and access control for all your Databricks workspaces.
- **Hive Metastore**: Manages metadata for Hive and Spark tables, but lacks robust access control.

[9] Joins in PySpark:

- **Inner Join**: Matches rows from both datasets based on a condition.
- **Left/Right Join**: Keeps all rows from the left/right and matches with the right/left dataset.
- **Full Outer Join**: Includes all rows from both datasets.
- **Cross Join**: Cartesian product of both datasets.

Want further explanations, detailed examples, or help structuring your answers? 😊

💡 These are excellent questions to enhance your preparation for Azure Data Engineer interviews. Here's a breakdown of concise, yet impactful answers:

[10] How to Implement Parallel Processing in ADF?

- Use **For Each Activity** with the **Batch Count** property set for parallelism.

- Enable **concurrent execution** in pipeline settings.
- Use partitioned datasets for parallel reads/writes to optimize execution.

[1] [1] Difference Between Narrow and Wide Transformations:

- **Narrow:** Data is processed within the same partition (e.g., map, filter). Minimal shuffling.
- **Wide:** Data is shuffled across partitions (e.g., groupBy, join). Higher computational cost.

[1] [2] What is SCD? Explain SCD1, SCD2, SCD3:

- **SCD (Slowly Changing Dimensions)** handles historical changes in dimension data.
- **SCD1:** Overwrites old data with new data.
- **SCD2:** Maintains history by adding new rows for changes (e.g., adding an `Effective_Date`).
- **SCD3:** Adds new columns to store historical data for specific attributes.

[1] [3] Cluster Options in Databricks:

- **Standard Cluster:** For general-purpose workloads.
- **High-Concurrency Cluster:** Optimized for multiple concurrent users.
- **Single Node Cluster:** For lightweight testing and debugging.
- **Jobs Cluster:** Automatically created for specific jobs and deleted afterward.

[1] [4] Difference Between Managed and External Tables:

- **Managed Tables:** Databricks manages the data and metadata (stored in default storage).
- **External Tables:** Data is stored outside Databricks, and only metadata is managed in the metastore.

1 5 What is a Surrogate Key?

- A unique identifier for a record, not derived from application data.
- Example: **Auto-increment ID** in databases.

1 6 Spark Optimization Techniques:

- **Cache/persist** frequently used data.
- Use **broadcast joins** for smaller datasets.
- Partition data effectively.
- Enable **predicate pushdown** for filters.
- Avoid wide transformations where possible.

1 7 Why is Databricks Better Than Dataflow?

- **Flexibility:** Databricks supports more complex workloads (e.g., ML, streaming).
- **Notebook Interface:** Collaborative development environment.
- **Performance:** Databricks uses Apache Spark with optimizations like Delta Lake.
- **Dataflow** is simpler for straightforward ETL use cases.

1 8 Difference Between Data Lake and Delta Lake:

- **Data Lake:** Stores raw, unstructured data. No ACID compliance.
- **Delta Lake:** Built on top of a data lake with ACID transactions, time travel, and schema enforcement.

1 9 Explain Spark Architecture:

- **Driver:** Coordinates execution, maintains DAG, and schedules tasks.
- **Executors:** Run tasks assigned by the driver. Each executor has its memory and cache.
- **Cluster Manager:** (e.g., YARN, Kubernetes) Allocates resources to the driver and executors.

Need examples for any of these? Or a deeper dive into any topic? 

Here's a solid overview of answers to these questions, tailored to help you shine in interviews! 

2 | 0 Difference Between `groupByKey` and `reduceByKey`:

- **groupByKey:** Groups all key-value pairs by key and shuffles all data.
More memory-intensive.
- **reduceByKey:** Combines values at the mapper side before shuffling, reducing network traffic. Preferred for better performance.

2 | 1 Why is MapReduce Not Widely Used Now? Similarities Between Spark and MapReduce?

- **Why not MapReduce:**
 - High latency due to disk I/O for intermediate results.
 - Complex to code compared to Spark.
- **Similarities:**
 - Both process large-scale data using distributed computing.
 - Use key-value pairs for transformations.
- **Spark Advantages:**
 - In-memory computation, faster execution, rich APIs (Python, Scala).

2 | 2 What is Delta Lake? Key Features and Creating Delta Tables:

- **Delta Lake:** A storage layer on top of Data Lake offering ACID compliance and reliability.
- **Key Features:**
 - ACID transactions.
 - Schema enforcement and evolution.
 - Time travel and versioning.
- **Creating Delta Tables:**

```
CREATE TABLE delta_table USING DELTA LOCATION 'path_to_delta';
```

[2] [3] Difference Between Serverless Pool and Dedicated SQL Pool:

- **Serverless Pool:**

- Pay-per-query model.
- Used for ad-hoc queries on data lakes.

- **Dedicated SQL Pool:**

- Pre-provisioned resources with fixed cost.
- Designed for high-performance data warehousing.

[2] [4] Prerequisites Before Migration:

- Assess source and target environments.
- Ensure schema compatibility.
- Perform data profiling and cleansing.
- Set up network, storage, and permissions.
- Validate data transformation logic.

[2] [5] What is a Mount Point in Databricks?

- A **mount point** is a shortcut to a storage account, enabling easier access.
- Example: Mounting an Azure Data Lake Gen2 folder using a `dbutils.fs.mount` command.

[2] [6] How to Optimize Databricks Performance:

- Enable **Delta Lake optimizations** like Z-ordering and `OPTIMIZE`.
- Use **Auto-scaling** for clusters.
- Use **broadcast joins** for smaller datasets.
- Optimize shuffling with correct partitioning.
- Persist reusable datasets in memory with `cache()`.

[2] [7] Difference Between map and flatMap:

- **map**: Transforms each element into another element, 1-to-1 mapping.
- **flatMap**: Can produce 0 or more elements per input, 1-to-n mapping.

[2] [8] How to Fetch Details from Key Vault:

- Use **Azure Key Vault Linked Service** in ADF or Databricks.
- In Databricks:

```
secret_value = dbutils.secrets.get(scope="key_vault_scope",  
key="secret_name")
```

[2] [9] Applying Indexing on a Databricks Table:

- Use Delta Lake Z-order indexing:

```
OPTIMIZE delta_table_name ZORDER BY (column_name);
```

- Helps improve query performance for large datasets.

[3] [0] Transferring Data to Azure Synapse:

- Use **Azure Data Factory** for ETL pipelines.
- **COPY INTO** command in Synapse for fast ingestion from Data Lake.
- Databricks-to-Synapse via JDBC connector or PolyBase.

Need any of these elaborated further or some live coding examples? 

Here's a breakdown of these advanced Azure Data Engineering topics to keep your prep on point! 

[3] [1] What is Incremental Loading? How to Implement It?

- **Definition**: Loading only new or updated data to a target without reloading the entire dataset.

- **Implementation:**

- **Watermarking:** Use timestamps or surrogate keys to identify changes.
- **ADF:** Use Lookup + Filter activities.
- **Delta Lake:** Merge using UPSERT logic:
 - `MERGE INTO target_table AS target`
 - `USING source_table AS source`
 - `ON target.id = source.id`
 - `WHEN MATCHED THEN UPDATE SET target.col = source.col`

```
WHEN NOT MATCHED THEN INSERT (columns) VALUES (values);
```

3 | 2 How Does Z-Ordering Work?

- **Z-Ordering:** A data layout optimization in Delta Lake that reduces I/O by co-locating similar data on disk.
- **How:**
 - Applies a multi-dimensional sort algorithm.
 - Improves query performance on frequently filtered columns.

```
OPTIMIZE table_name ZORDER BY (column1, column2);
```

3 | 3 What is Dimension Modeling? Dimension and Fact Tables?

- **Dimension Modeling:** A design technique for data warehouses to optimize query performance using star or snowflake schemas.
- **Fact Tables:** Store numeric measures (e.g., sales amount).
- **Dimension Tables:** Describe the context of facts (e.g., customer, product).

3 | 4 Difference Between a Data Lake and a Data Warehouse:

- **Data Lake:**

- Stores raw, unstructured data.
- Scalable, cost-effective.
- Example: Azure Data Lake.

- **Data Warehouse:**

- Stores structured, processed data for analytics.
- Schema-on-write.
- Example: Azure Synapse.

[3] [5] Using Logic Apps in Your Project:

- Automates workflows between services like ADF, Synapse, and notifications.
- Example Use Case:
 - Trigger data pipelines based on events (e.g., file upload).
 - Send failure alerts via email or Teams.

[3] [6] What is Data Skewness?

- **Definition:** Uneven distribution of data across partitions, leading to performance bottlenecks.
- **Mitigation:**
 - Use **salting** techniques (adding random keys).
 - Optimize partitioning with balanced keys.

[3] [7] What is Fault Tolerance and Its Use in Real-Time Applications?

- **Definition:** The ability of a system to recover from failures.
- **Real-Time Use:**
 - Spark achieves fault tolerance by storing lineage and recomputing lost partitions.
 - In ADF, retry policies handle transient failures.

[3] [8] Converting RDD to DataFrame & Vice Versa:

- **RDD to DataFrame:**

- ```
from pyspark.sql import SparkSession
```

```
df = rdd.toDF(schema=["col1", "col2"])
```

- **DataFrame to RDD:**

```
rdd = df.rdd
```

## 3 9 Encryption Techniques:

- **At Rest:** Encrypt data in storage using Azure Storage Service Encryption (SSE).
- **In Transit:** Use TLS/SSL for secure data transfer.
- **Column-Level Encryption:** Secure sensitive data fields (e.g., PII).

## 4 0 How Does Auto Loader Work?

- A feature in Databricks for **incremental file processing** from cloud storage.
- **Working:**
  - Tracks metadata using checkpointing.
  - Processes new files automatically.
- Example:

- ```
df = spark.readStream.format("cloudFiles") \\\n    .option("cloudFiles.format", "json") \\\n    .load("path")
```

4 1 Explain Lazy Evaluation in PySpark:

- **Definition:** Transformations are not executed immediately but only when an action (e.g., `count`, `collect`) is triggered.
- **Benefits:**
 - Optimizes execution by combining transformations into a single stage.
 - Reduces unnecessary computations.

Want any topic expanded with examples or real-world scenarios? Let me know! 

Here's a detailed explanation of these additional Spark and PySpark-related questions:

4 2 What is DAG in Spark?

- **DAG (Directed Acyclic Graph):**
 - A sequence of computations where each node represents a transformation and edges represent dependencies.
 - Spark breaks the execution into stages using DAG, ensuring fault tolerance and optimized execution.
- **Significance:**
 - Tracks lineage for fault recovery.
 - Optimizes execution by combining transformations.

4 3 Significance of Catalyst Optimizer in PySpark?

- **What It Is:** A query optimization engine in Spark SQL.
- **Functions:**
 - Converts logical plans into optimized physical plans.
 - Pushes predicates (filter operations) early to minimize I/O.
- **Benefits:** Better performance with optimized execution plans.

4 4 Query to Find the 4th Highest Salary of an Employee:

```
SELECT DISTINCT salary
```

```
FROM employee  
ORDER BY salary DESC  
LIMIT 4 OFFSET 3;
```

- Alternatively, using **ROW_NUMBER**:

```
• SELECT salary  
• FROM (   
•     SELECT salary, ROW_NUMBER() OVER (ORDER BY salary DESC) AS rank  
•     FROM employee  
•   ) ranked
```

```
WHERE rank = 4;
```

4 5 PySpark Command to Read Data from a File into a DataFrame:

```
df = spark.read.csv("path/to/file.csv", header=True, inferSchema=True)
```

- Other Formats:

- JSON: `spark.read.json("path")`
- Parquet: `spark.read.parquet("path")`

4 6 Handling Nulls and Duplicates in PySpark:

- Drop Nulls:

```
df = df.dropna()
```

- Fill Nulls:

```
df = df.fillna({'col1': 'default_value', 'col2': 0})
```

- Remove Duplicates:

```
df = df.dropDuplicates(['col1', 'col2'])
```

4 7 Changing the Date Format for a Date Column:

```
from pyspark.sql.functions import date_format  
  
df = df.withColumn("new_date", date_format("date_column", "yyyy-MM-dd"))
```

4 8 What is the Explode Function in PySpark?

- **Explode:** Converts an array or map into multiple rows.
- **Example:**

```
from pyspark.sql.functions import explode
```

```
df = df.withColumn("exploded_col", explode("array_col"))
```

4 9 Code to Read a Parquet File:

```
df = spark.read.parquet("path/to/file.parquet")
```

5 0 Code to Add a Column to a Parquet File:

```
from pyspark.sql.functions import lit  
  
df = spark.read.parquet("path/to/file.parquet")  
  
df = df.withColumn("new_column", lit("value"))  
  
df.write.parquet("path/to/updated_file.parquet")
```

5 1 Different Approaches to Creating RDD in PySpark:

- **From a Collection:**

```
rdd = spark.sparkContext.parallelize([1, 2, 3, 4])
```

- **From a File:**

```
rdd = spark.sparkContext.textFile("path/to/file.txt")
```

5 | 2 | Different Approaches to Creating DataFrame in PySpark:

- **From RDD:**

- `from pyspark.sql import Row`
- `rdd = spark.sparkContext.parallelize([Row(name="Alice", age=25), Row(name="Bob", age=30)])`

```
df = rdd.toDF()
```

- **From a File:**

```
df = spark.read.csv("path/to/file.csv", header=True, inferSchema=True)
```

- **From a List/Dictionary:**

- `data = [("Alice", 25), ("Bob", 30)]`

```
df = spark.createDataFrame(data, schema=["name", "age"])
```

Let me know if you need code expansions or further clarifications! 

COMPREHENSIVE PYSPARK LESSONS, NOTES & INTERVIEW QUESTIONS & ANSWERS

APACHE SPARK INTERVIEW QUESTIONS

ABHINANDAN PATRA

DATA ENGINEER

1. What is Apache Spark?

Apache Spark is an open-source, distributed computing system that provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. It is designed to process large-scale data efficiently.

2. Why Apache Spark?

Apache Spark is used because it is faster than traditional big data tools like Hadoop MapReduce due to its in-memory processing capabilities, supports multiple languages (Scala, Python, R, Java), provides libraries for various tasks (SQL, machine learning, graph processing, etc.), and has robust fault tolerance.

3. What are the components of the Apache Spark Ecosystem?

The main components are:

- **Spark Core:** The foundational engine for large-scale parallel and distributed data processing.
- **Spark SQL:** For structured data processing.
- **Spark Streaming:** For real-time data processing.
- **Mlib:** A library for scalable machine learning.
- **GraphX:** For graph and graph-parallel computation.

4. What is Spark Core?

Spark Core is the general execution engine for the Spark platform, responsible for tasks such as scheduling, distributing, and monitoring applications.

5. Which languages does Apache Spark support?

Apache Spark supports:

- Scala
- Python
- Java
- R
- SQL

6. How is Apache Spark better than Hadoop?

Spark is better in several ways, including faster processing due to in-memory computation, ease of use with APIs for various programming languages, flexibility with built-in libraries for diverse tasks, and a rich set of APIs for transformations and actions.

7. What are the different methods to run Spark over Apache Hadoop?

Spark can run on Hadoop in the following modes:

- **Standalone:** Using its cluster manager.
- **YARN:** Hadoop's cluster manager.
- **Mesos:** Another cluster manager.

8. What is SparkContext in Apache Spark?

SparkContext is the entry point for any Spark application. It acts as a connection to the Spark cluster, allowing Spark jobs to be executed.

9. What is SparkSession in Apache Spark?

SparkSession is the unified entry point to work with DataFrames, Datasets, and SQL in Apache Spark. It replaces SQLContext and HiveContext.

10. SparkSession vs. SparkContext in Apache Spark

SparkSession is a combination of SQLContext, HiveContext, and SparkContext to provide a single point of entry to interact with Spark.

11. What are the abstractions of Apache Spark?

The primary abstractions are:

- **RDD (Resilient Distributed Dataset)**
- **DataFrames**
- **Datasets**

12. How can we create RDD in Apache Spark?

RDDs can be created in three ways:

- **Parallelizing a collection** in your program.
- **Referencing a dataset** in an external storage system (e.g., HDFS, S3, etc.).
- **Transforming** an existing RDD.

13. Why is Spark RDD immutable?

RDDs are immutable to provide fault tolerance and support functional programming principles, allowing Spark to rebuild lost data from the lineage information.

14. Explain the term paired RDD in Apache Spark.

Paired RDDs are RDDs where each element is a pair (key-value). They are used for operations like aggregation, grouping, and joins.

15. How is RDD in Spark different from Distributed Storage Management?

RDD is an in-memory data structure optimized for processing, while Distributed Storage (like HDFS) focuses on data storage and retrieval.

16. Explain transformation and action in RDD in Apache Spark.

- **Transformation:** Lazy operations that define a new RDD without executing until an action is called (e.g., map, filter).
- **Action:** Triggers the execution of transformations (e.g., count, collect).

17. What are the types of Apache Spark transformations?

Transformations can be narrow (e.g., map, filter) or wide (e.g., groupByKey, reduceByKey).

18. Explain the RDD properties.

RDD properties include:

- **Immutability:** Once created, RDDs cannot be changed.
- **Partitioned:** Distributed across various nodes in the cluster.
- **Lazy evaluation:** Operations are computed when an action is called.
- **Fault tolerance:** Recomputed using lineage information.

19. What is a lineage graph in Apache Spark?

A lineage graph tracks the sequence of transformations that created an RDD, used for recomputing lost data due to node failures.

20. Explain the terms Spark Partitions and Partitioners.

- **Partitions:** Logical division of data in RDDs, physically stored across nodes.
- **Partitioner:** Determines how data is distributed across partitions (e.g., HashPartitioner, RangePartitioner).

21. By default, how many partitions are created in RDD in Apache Spark?

By default, Spark creates partitions based on the number of cores available or the input file's HDFS block size.

22. What is Spark DataFrames?

DataFrames are distributed collections of data organized into named columns, similar to tables in a relational database.

23. What are the benefits of DataFrame in Spark?

Benefits include optimizations (Catalyst query optimizer), improved performance, and easier manipulation using SQL-like syntax.

24. What is Spark Dataset?

A Dataset is a distributed collection of data that provides type safety and object-oriented programming interfaces.

25. What are the advantages of datasets in Spark?

Advantages include compile-time type safety, optimizations through Tungsten, and the ability to leverage JVM object serialization.

26. What is Directed Acyclic Graph (DAG) in Apache Spark?

A DAG in Spark represents a sequence of computations performed on data, where each node is an RDD and edges represent transformations. It's used to optimize execution plans.

27. What is the need for Spark DAG?

The DAG allows Spark to optimize execution by scheduling tasks efficiently, minimizing data shuffling, and managing dependencies.

28. What is the difference between DAG and Lineage?

- **DAG**: Represents the entire execution plan of a Spark application.
- **Lineage**: Tracks transformations on a particular RDD, useful for fault recovery.

29. What is the difference between Caching and Persistence in Apache Spark?

- **Caching**: Default storage level is in-memory (`MEMORY_ONLY`).
- **Persistence**: Allows choosing different storage levels (disk, memory, etc.) for storing RDDs.

30. What are the limitations of Apache Spark?

Limitations include high memory consumption, limited built-in libraries compared to Hadoop, not suitable for small data or real-time streaming without specific tools.

31. Different Running Modes of Apache Spark

Spark can run in:

- **Local mode**: Single machine.
- **Standalone mode**: Using its cluster manager.
- **YARN mode**: On Hadoop's cluster manager.
- **Mesos mode**: On Mesos cluster manager.
- **Kubernetes mode**: On Kubernetes.

32. What are the different ways of representing data in Spark?

Data can be represented as:

- **RDDs (Resilient Distributed Datasets)**

- **DataFrames**
- **Datasets**

33. What is Write-Ahead Log (WAL) in Spark?

Write-Ahead Log is a fault-tolerance mechanism where every received data is first written to a log file (disk) before processing, ensuring no data loss.

34. Explain Catalyst Query Optimizer in Apache Spark.

Catalyst is Spark SQL's query optimizer that uses rule-based and cost-based optimization techniques to generate efficient execution plans.

35. What are shared variables in Apache Spark?

Shared variables are variables that can be used by tasks running on different nodes:

- **Broadcast variables:** Efficiently share read-only data across nodes.
- **Accumulators:** Used for aggregating information (e.g., sums) across tasks.

36. How does Apache Spark handle accumulated metadata?

Spark stores metadata like lineage information, partition data, and task details in the driver and worker nodes, managing it using its DAG scheduler.

37. What is Apache Spark's Machine Learning Library?

Mlib is Spark's scalable machine learning library, which provides algorithms and utilities for classification, regression, clustering, collaborative filtering, and more.

38. List commonly used Machine Learning Algorithms.

Common algorithms in Spark Mlib include:

- Linear Regression
- Logistic Regression
- Decision Trees
- Random Forests
- Gradient-Boosted Trees
- K-Means Clustering

39. What is the difference between DSM and RDD?

- **DSM (Distributed Storage Management):** Focuses on data storage across clusters.
- **RDD (Resilient Distributed Dataset):** Focuses on distributed data processing with fault tolerance.

40. List the advantage of Parquet file in Apache Spark.

Advantages of Parquet files:

- Columnar storage format, optimized for read-heavy workloads.
- Efficient compression and encoding schemes.
- Schema evolution support.

41. What is lazy evaluation in Spark?

Lazy evaluation defers execution until an action is performed, optimizing the execution plan by reducing redundant computations.

42. What are the benefits of Spark lazy evaluation?

Benefits include:

- Reducing the number of passes over data.
- Optimizing the computation process.
- Decreasing execution time.

43. How much faster is Apache Spark than Hadoop?

Apache Spark is generally up to 100x faster than Hadoop for in-memory processing and up to 10x faster for on-disk data.

44. What are the ways to launch Apache Spark over YARN?

Spark can be launched over YARN in:

- **Client mode:** Driver runs on the client machine.
- **Cluster mode:** Driver runs inside YARN cluster.

45. Explain various cluster managers in Apache Spark.

Spark supports:

- **Standalone Cluster Manager:** Default cluster manager.
- **Apache Mesos:** A general-purpose cluster manager.
- **Hadoop YARN:** A resource manager for Hadoop clusters.
- **Kubernetes:** For container orchestration.

46. What is Speculative Execution in Apache Spark?

Speculative execution is a mechanism to detect slow-running tasks and run duplicates on other nodes to speed up the process.

47. How can data transfer be minimized when working with Apache Spark?

Data transfer can be minimized by:

- Reducing shuffling and repartitioning.
- Using broadcast variables.
- Efficient data partitioning.

48. What are the cases where Apache Spark surpasses Hadoop?

Apache Spark outperforms Hadoop in scenarios involving iterative algorithms, in-memory computations, real-time analytics, and complex data processing workflows.

49. What is an action, and how does it process data in Apache Spark?

An action is an operation that triggers the execution of transformations (e.g., `count`, `collect`), performing computations and returning a result.

50. How is fault tolerance achieved in Apache Spark?

Fault tolerance is achieved through lineage information, allowing RDDs to be recomputed from scratch if a partition is lost.

51. What is the role of the Spark Driver in Spark applications?

The Spark Driver is responsible for converting the user's code into tasks, scheduling them on executors, and collecting the results.

52. What is a worker node in an Apache Spark cluster?

A worker node is a machine in a Spark cluster where the actual data processing tasks are executed.

53. Why is Transformation lazy in Spark?

Transformations are lazy to build an optimized execution plan (DAG) and to avoid unnecessary computation.

54. Can I run Apache Spark without Hadoop?

Yes, Spark can run independently using its built-in cluster manager or other managers like Mesos and Kubernetes.

55. Explain Accumulator in Spark.

An accumulator is a variable used for aggregating information across executors, like counters in MapReduce.

56. What is the role of the Driver program in a Spark Application?

The Driver program coordinates the execution of tasks, maintains the `SparkContext`, and communicates with the cluster manager.

57. How to identify that a given operation is a Transformation or Action in your program?

Transformations return RDDs (e.g., `map`, `filter`), while actions return non-RDD values (e.g., `collect`, `count`).

58. Name the two types of shared variables available in Apache Spark.

- Broadcast Variables
- Accumulators

59. What are the common faults of developers while using Apache Spark?

Common faults include:

- Inefficient data partitioning.
- Excessive shuffling and data movement.
- Inappropriate use of transformations and actions.
- Not leveraging caching and persistence properly.

60. By Default, how many partitions are created in RDD in Apache Spark?

The default number of partitions is based on the number of cores available in the cluster or the HDFS block size.

61. Why do we need compression, and what are the different compression formats supported?

Compression reduces the storage size of data and speeds up data transfer. Spark supports several compression formats:

- Snappy
- Gzip
- Bzip2
- LZ4
- Zstandard (Zstd)

62. Explain the filter transformation.

The `filter` transformation creates a new RDD by selecting only elements that satisfy a given predicate function.

63. How to start and stop Spark in the interactive shell?

To start Spark in the interactive shell:

- Use `spark-shell` for Scala or `pyspark` for Python. To stop Spark:
- Use `:quit` or `Ctrl + D` in the shell.

64. Explain the `sortByKey()` operation.

`sortByKey()` sorts an RDD of key-value pairs by the key in ascending or descending order.

65. Explain `distinct()`, `union()`, `intersection()`, and `subtract()` transformations in Spark.

- **`distinct()`:** Returns an RDD with duplicate elements removed.
- **`union()`:** Combines two RDDs into one.
- **`intersection()`:** Returns an RDD with elements common to both RDDs.
- **`subtract()`:** Returns an RDD with elements in one RDD but not in another.

66. Explain `foreach()` operation in Apache Spark.

`foreach()` applies a function to each element in the RDD, typically used for side effects like updating an external data store.

67. `groupByKey` VS `reduceByKey` in Apache Spark.

- **`groupByKey`:** Groups values by key and shuffles all data across the network, which can be less efficient.
- **`reduceByKey`:** Combines values for each key locally before shuffling, reducing network traffic.

68. Explain `mapPartitions()` and `mapPartitionsWithIndex()`.

- **`mapPartitions()`:** Applies a function to each partition of the RDD.
- **`mapPartitionsWithIndex()`:** Applies a function to each partition, providing the partition index.

69. What is `map` in Apache Spark?

`map` is a transformation that applies a function to each element in the RDD, resulting in a new RDD.

70. What is `flatMap` in Apache Spark?

`flatMap` is a transformation that applies a function to each element, resulting in multiple elements (a flat structure) for each input.

71. Explain `fold()` operation in Spark.

`fold()` aggregates the elements of an RDD using an associative function and a "zero value" (an initial value).

72. Explain `createOrReplaceTempView()` API.

`createOrReplaceTempView()` registers a DataFrame as a temporary table in Spark SQL, allowing it to be queried using SQL.

73. Explain `values()` operation in Apache Spark.

`values()` returns an RDD containing only the values of key-value pairs.

74. Explain `keys()` operation in Apache Spark.

`keys()` returns an RDD containing only the keys of key-value pairs.

75. Explain `textFile` vs `wholeTextFiles` in Spark.

- **`textFile()`:** Reads a text file and creates an RDD of strings, each representing a line.
- **`wholeTextFiles()`:** Reads entire files and creates an RDD of (filename, content) pairs.

76. Explain `cogroup()` operation in Spark.

`cogroup()` groups data from two or more RDDs sharing the same key.

77. Explain `pipe()` operation in Apache Spark.

`pipe()` passes each partition of an RDD to an external script or program and returns the output as an RDD.

78. Explain Spark `coalesce()` operation.

`coalesce()` reduces the number of partitions in an RDD, useful for minimizing shuffling when reducing the data size.

79. Explain the `repartition()` operation in Spark.

`repartition()` reshuffles data across partitions, increasing or decreasing the number of partitions, involving a full shuffle of data.

80. Explain `fullOuterJoin()` operation in Apache Spark.

`fullOuterJoin()` returns an RDD with all pairs of elements for matching keys and `null` for non-matching keys from both RDDs.

81. Explain Spark `leftOuterJoin()` and `rightOuterJoin()` operations.

- **`leftOuterJoin()`:** Returns all key-value pairs from the left RDD and matching pairs from the right, filling with `null` where no match is found.
- **`rightOuterJoin()`:** Returns all key-value pairs from the right RDD and matching pairs from the left, filling with `null` where no match is found.

82. Explain Spark `join()` operation.

`join()` returns an RDD with all pairs of elements with matching keys from both RDDs.

83. Explain `top()` and `takeOrdered()` operations.

- `top()`: Returns the top n elements from an RDD in descending order.
- `takeOrdered()`: Returns the top n elements from an RDD in ascending order.

84. Explain `first()` operation in Spark.

`first()` returns the first element of an RDD.

85. Explain `sum()`, `max()`, `min()` operations in Apache Spark.

These operations compute the sum, maximum, and minimum of elements in an RDD, respectively.

86. Explain `countByValue()` operation in Apache Spark RDD.

`countByValue()` returns a map of the counts of each unique value in the RDD.

87. Explain the `lookup()` operation in Spark.

`lookup()` returns the list of values associated with a given key in a paired RDD.

88. Explain Spark `countByKey()` operation.

`countByKey()` returns a map of the counts of each key in a paired RDD.

89. Explain Spark `saveAsTextFile()` operation.

`saveAsTextFile()` saves the RDD content as a text file or set of text files.

90. Explain `reduceByKey()` Spark operation.

`reduceByKey()` applies a reducing function to the elements with the same key, reducing them to a single element per key.

91. Explain the operation `reduce()` in Spark.

`reduce()` aggregates the elements of an RDD using an associative and commutative function.

92. Explain the action `count()` in Spark RDD.

`count()` returns the number of elements in an RDD.

93. Explain Spark `map()` transformation.

`map()` applies a function to each element of an RDD, creating a new RDD with the results.

94. Explain the `flatMap()` transformation in Apache Spark.

`flatMap()` applies a function that returns an iterable to each element and flattens the results into a single RDD.

95. What are the limitations of Apache Spark?

Limitations include high memory consumption, not ideal for OLTP (transactional processing), lack of a mature security framework, and dependency on cluster resources.

96. What is Spark SQL?

Spark SQL is a Spark module for structured data processing, providing a DataFrame API and allowing SQL queries to be executed.

97. Explain Spark SQL caching and uncaching.

- **Caching:** Storing DataFrames in memory for faster access.
- **Uncaching:** Removing cached DataFrames to free memory.

98. Explain Spark Streaming.

Spark Streaming is an extension of Spark for processing real-time data streams.

99. What is DStream in Apache Spark Streaming?

DStream (Discretized Stream) is a sequence of RDDs representing a continuous stream of data.

100. Explain different transformations in DStream in Apache Spark Streaming.

Transformations include:

- `map()`, `flatMap()`, `filter()`
- `reduceByKeyAndWindow()`
- `window()`, `countByWindow()`
- `updateStateByKey()`

101. What is the Starvation scenario in Spark Streaming?

Starvation occurs when all tasks are waiting for resources that are occupied by other long-running tasks, leading to delays or deadlocks.

102. Explain the level of parallelism in Spark Streaming.

Parallelism is controlled by the number of partitions in RDDs; increasing partitions increases the level of parallelism.

103. What are the different input sources for Spark Streaming?

Input sources include:

- Kafka
- Flume
- Kinesis
- Socket
- HDFS or S3

104. Explain Spark Streaming with Socket.

Spark Streaming can receive real-time data streams over a socket using `socketTextStream()`.

105. Define the roles of the file system in any framework.

The file system manages data storage, access, and security, ensuring data integrity and availability.

106. How do you parse data in XML? Which kind of class do you use with Java to parse data?

To parse XML data in Java, you can use classes from the `javax.xml.parsers` package, such as:

- **DocumentBuilder:** Used with the Document Object Model (DOM) for in-memory tree representation.
- **SAXParser:** Used with the Simple API for XML (SAX) for event-driven parsing.

107. What is PageRank in Spark?

PageRank is an algorithm used to rank web pages in search engine results, based on the number and quality of links to a page. In Spark, it can be implemented using RDDs or DataFrames to compute the rank of nodes in a graph.

108. What are the roles and responsibilities of worker nodes in the Apache Spark cluster? Is the Worker Node in Spark the same as the Slave Node?

- **Worker Nodes:** Execute tasks assigned by the Spark Driver, manage executors, and store data in memory or disk as required.
- **Slave Nodes:** Worker nodes in Spark are commonly referred to as slave nodes. Both terms are used interchangeably.

109. How to split a single HDFS block into partitions in an RDD?

When reading from HDFS, Spark splits a single block into multiple partitions based on the number of available cores or executors. You can also use the `repartition()` method to explicitly specify the number of partitions.

110. On what basis can you differentiate RDD, DataFrame, and DataSet?

- **RDD**: Low-level, unstructured data; provides functional programming APIs.
- **DataFrame**: Higher-level abstraction with schema; optimized for SQL queries and transformations.
- **Dataset**: Combines features of RDDs and DataFrames; offers type safety and object-oriented programming.

SPARK BASED TOPICS KEYWORDS:

Spark Intro:

1. Spark : In-memory processing engine
2. Why spark is fast: Due to less I/O disc reads and writes
3. RDD: It is a data structure to store data in spark
4. When RDD fails: Using lineage graph we track which RDD failed and reprocess it
5. Why RDD immutable : As it has to be recovered after its failure and to track which RDD failed
6. Operations in spark: Transformation and Action
7. Transformation: Change data from one form to another, are lazy.
8. Action: Operations which processes the tranformations, not lazy. creates DAG to remember sequence of steps.
9. Broadcast Variables: Data which is distributed to all the systems. Similar to map side join in hive
10. Accumulators: Shared copy in driver, executors can update but not read. Similar to counters in MR
11. MR before Yarn: Job tracker (scheduling &monitoring), task manager (manages tasks in its node)
12. Limitations of MR: Unable to add new clusters(scalable), resource under-utilization, only MR jobs handled
- 13.YARN: Resource manager(scheduling), application master(monitored & resource negotiation), node manager (manages tasks in its node)
- 14.Uberization: Tasks run on AM itself if they are very small

15. Spark components: Driver (gives location of executors) and executors(process data in memory)
16. Client Mode: Driver is at client side
17. Cluster Mode: Driver is inside AM in the cluster
18. Types of transformation: Narrow and Wide
19. Narrow: Data shuffling doesn't happen (map, flatMap, filter)
20. Wide: Data shuffling happens (reduceByKey, groupByKey)
21. reduceByKey() is a transformation and reduce() is an action
22. reduceByKey(): Data is processed at each partition, groupByKey() : Data is grouped at each partition and complete processing is done at reducer.
23. Repartition: used to increase/decrease partitions. Use it for INCREASE
Coalesce: used to decrease partitions and optimized as data shuffling is less

SPARK DATAFRAMES:

1. Cache() : It is used to cache the data only in memory.
Rdd.cache()
2. Persist() : it is used to cache the data in different storage levels (memory, disc, memory & disc, off heap).
Rdd.persist(StorageLevel.____)
3. Serialization: Process of converting data in object form into bytes, occupies less space

4. De-Serialization: Process of converting data in bytes back to objects, occupies more space.
5. DAG : Created when an action is called, represents tasks, stages of a job
6. Map : performs one-to-one mapping on each line of input
7. mapPartitions: performs map function only once on each partition
8. Driver: converts high level programming constructs to low level to be fed to executors (dataframe to rdd)
9. Executors: Present in memory to process the rdd
10. Spark context: creates entry point into spark cluster for spark appl
11. Spark session: creates unified entry point into spark cluster
12. Data frame: it is a dataset[row] where type error caught only at run time
13. Data set: it is a dataset[object] where type error caught at compile time
14. Modes of dealing with corrupted record: permissive, malformed, fail fast
15. Schema types: implicit, infer, explicit (case class, StructType, DDL string)

SPARK OPTIMIZATIONS

1. Spark optimization:
 - a. Cluster Configuration : To configure resources to the cluster so that spark jobs can process well.

- b. Code configuration: To apply optimization techniques at code level so that processing will be fast.
- 2. Thin executor: More no. of executors with less no. of resources. Multithreading not possible, too many broadcast variables required. Ex. 1 executor with each 2 cpu cores, 1 GB ram.
- 3. Fat executor: Less no. of executors with more amount of resources. System performance drops down, garbage collection takes time. Ex 1 executor 16 cpu cores, 32 GB ram.
- 4. Garbage collection: To remove unused objects from memory.
- 5. Off heap memory: Memory stored outside of executors/ jvm. It takes less time to clean objects than garbage collector, used for java overheads (extra memory which directly doesn't add to performance but required by system to carry out its operation)
- 6. Static allocation: Resources are fixed at first and will remain the same till the job ends.
- 7. Dynamic Allocation: Resources are allocated dynamically based on the job requirement and released during job stages if they are no longer required.
- 8. Edge node: It is also called as gateway node which is can be accessed by client to enter into hadoop cluster and access name node.
- 9. How to increase parallelism :
 - a. Salting : To increase no. of distinct keys so that work can be distributed across many tasks which in turn increase parallelism.
 - b. Increase no. of shuffle partitions
 - c. Increase the resources of the cluster (more cpu cores)
- 10. Execution memory : To perform computations like shuffle, sort, join
- 11. Storage memory : To store the cache
- 12. User memory : To store user's data structures, meta data etc.
- 13. Reserved memory : To run the executors

14. Kyro Serializer: Used to store the data in disk in serialized manner which occupies less space.
15. Broadcast join: Used to send the copies of data to all executors. Used when we have only 1 big table.
16. Optimization on using coalesce() rather than repartition while reducing no. of partitions
17. Join optimizations:
 - a. To avoid or minimize shuffling of data
 - b. To increase parallelism
 1. How to avoid/minimize shuffling?
 - a. Filter and aggregate data before shuffling
 - b. Use optimization methods which require less shuffling (coalesce())
 18. How to increase parallelism ?
 - a. **Min (total cpu cores, total shuffle partitions, total distinct keys)**
 - b. Use salting to increase no. of distinct keys
 - c. Increase default no. of shuffle partitions
 - d. Increase resources to inc total cpu cores
 19. Skew partitions : Partitions in which data is unevenly distributed. Bucketing, partitioning, salting can be used to handle it.
 20. Sort aggregate: Data is sorted based on keys and then aggregated. More processing time
 21. Hash aggregate: Hash table is created and similar keys are added to the same hash value. Less processing time.
 22. Stages of execution plan :
 - a. Parsed logical plan (unresolved logical plan) : To find out syntax errors

- b. Analytical logical plan (Resolved logical plan) : Checks for column and table names from the catalog.
- c. Optimized logical plan (Catalyst optimization) : Optimization done based on built in rules.
- d. Physical plan : Actual execution plan is selected based on cost effective model.
- e. Conversion into Rdd : Converted into rdd and sent to executors for processing.

****Note:**

1 hdfs block = 1 rdd partition = 128mb

1 hdfs block in local=1 rdd partition in local spark cluster= 32mb

1 rdd ~ can have n partitions in it

1 cluster = 1 machine

N cores = N blocks can run in parallel in each cluster/machine

N stages = N - 1 wide transformations

N tasks in each stage= N partitions in each stage for that rdd/data frame

Must Know Pyspark Coding Before Your Next Databricks Interview

Document by – Siddhartha Subudhi

[Visit my LinkedIn profile](#)

1. Find the second highest salary in a DataFrame using PySpark.

Scenario: You have a DataFrame of employee salaries and want to find the second highest salary.

```
from pyspark.sql import Window  
from pyspark.sql.functions import col, dense_rank  
  
windowSpec = Window.orderBy(col("salary").desc())  
  
df_with_rank = df.withColumn("rank", dense_rank().over(windowSpec))  
  
second_highest_salary = df_with_rank.filter(col("rank") == 2).select("salary")  
  
second_highest_salary.show()
```

2. Count the number of null values in each column of a PySpark DataFrame.

Scenario: Given a DataFrame, identify how many null values each column contains.

```
from pyspark.sql.functions import col, isnan, when, count
```

```
df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

3. Calculate the moving average over a window of 3 rows.

Scenario: For a stock price dataset, calculate a moving average over the last 3 days.

```
from pyspark.sql import Window  
from pyspark.sql.functions import avg  
  
windowSpec = Window.orderBy("date").rowsBetween(-2, 0)  
  
df_with_moving_avg = df.withColumn("moving_avg", avg("price").over(windowSpec))  
  
df_with_moving_avg.show()
```

4. Remove duplicate rows based on a subset of columns in a PySpark DataFrame.

Scenario: You need to remove duplicates from a DataFrame based on certain columns.

```
df = df.dropDuplicates(["column1", "column2"])  
  
df.show()
```



Siddhartha Subudhi

5. Split a single column with comma-separated values into multiple columns.

Scenario: Your DataFrame contains a column with comma-separated values. You want to split this into multiple columns.

```
from pyspark.sql.functions import split
```

```
df_split = df.withColumn("new_column1", split(df["column"], ",").getItem(0)) \  
    .withColumn("new_column2", split(df["column"], ",").getItem(1))  
df_split.show()
```

6. Group data by a specific column and calculate the sum of another column.

Scenario: Group sales data by "product" and calculate the total sales.

```
df.groupBy("product").sum("sales").show()
```

7. Join two DataFrames on a specific condition.

Scenario: You have two DataFrames: one for customer data and one for orders. Join these DataFrames on the customer ID.

```
df_joined = df_customers.join(df_orders, df_customers.customer_id == df_orders.customer_id, "inner")  
df_joined.show()
```

8. Create a new column based on conditions from existing columns.

Scenario: Add a new column "category" that assigns "high", "medium", or "low" based on the value of the "sales" column.

```
from pyspark.sql.functions import when  
  
df = df.withColumn("category", when(df.sales > 500, "high")  
    .when((df.sales <= 500) & (df.sales > 200), "medium")  
    .otherwise("low"))  
  
df.show()
```

Siddhartha Subudhi

Data Engineer

@siddhartha-subudhi

9. Calculate the percentage contribution of each value in a column to the total.

Scenario: For a sales dataset, calculate the percentage contribution of each product's sales to the total sales.

```
from pyspark.sql.functions import sum, col  
  
total_sales = df.agg(sum("sales").alias("total_sales")).collect()[0]["total_sales"]  
  
df = df.withColumn("percentage", (col("sales") / total_sales) * 100)  
  
df.show()
```

10. Find the top N records from a DataFrame based on a column.

Scenario: You need to find the top 5 highest-selling products.

```
df.orderBy(col("sales").desc()).limit(5).show()
```

11. Write PySpark code to pivot a DataFrame.

Scenario: You have sales data by "year" and "product", and you want to pivot the table to show "product" sales by year.

```
df_pivot = df.groupBy("product").pivot("year").sum("sales")  
  
df_pivot.show()
```

12. Add row numbers to a PySpark DataFrame based on a specific ordering.

Scenario: Add row numbers to a DataFrame ordered by "sales" in descending order.

```
from pyspark.sql.window import Window  
  
from pyspark.sql.functions import row_number  
  
windowSpec = Window.orderBy(col("sales").desc())  
  
df_with_row_number = df.withColumn("row_number", row_number().over(windowSpec))  
  
df_with_row_number.show()
```



Siddhartha Subudhi

13. Filter rows based on a condition.

Scenario: You want to filter only those customers who made purchases over ₹1000.

```
df_filtered = df.filter(df.purchase_amount > 1000)  
df_filtered.show()
```

14. Flatten a JSON column in PySpark.

Scenario: Your DataFrame contains a JSON column, and you want to extract specific fields from it.

```
from pyspark.sql.functions import from_json, col  
from pyspark.sql.types import StructType, StructField, StringType  
schema = StructType([  
    StructField("name", StringType(), True),  
    StructField("age", StringType(), True)  
)  
df = df.withColumn("json_data", from_json(col("json_column"), schema))  
df.select("json_data.name", "json_data.age").show()
```

15. Convert a PySpark DataFrame column to a list.

Scenario: Convert a column from your DataFrame into a list for further processing.

```
column_list = df.select("column_name").rdd.flatMap(lambda x: x).collect()
```

16. Handle NULL values by replacing them with a default value.

Scenario: Replace all NULL values in the "sales" column with 0.

```
df = df.na.fill({"sales": 0})  
df.show()
```

17. Perform a self-join on a PySpark DataFrame.

Scenario: You have a hierarchy of employees and want to find each employee's manager.

```
df_self_join = df.alias("e1").join(df.alias("e2"), col("e1.manager_id") == col("e2.employee_id"), "inner") \  
.select(col("e1.employee_name"), col("e2.employee_name").alias("manager_name"))  
df_self_join.show()
```



Siddhartha Subudhi

Data Engineer

@siddhartha-subudhi

18. Write PySpark code to unpivot a DataFrame.

Scenario: You have a DataFrame with "year" columns and want to convert them to rows.

```
from pyspark.sql.functions import expr  
df_unpivot = df.selectExpr("id", "stack(2, '2021', sales_2021, '2022', sales_2022) as (year, sales)")  
df_unpivot.show()
```

19. Write a PySpark code to group data based on multiple columns and calculate aggregate functions.

Scenario: Group data by "product" and "region" and calculate the average sales for each group.

```
df.groupBy("product", "region").agg({"sales": "avg"}).show()
```

20. Write PySpark code to remove rows with duplicate values in any column.

Scenario: You want to remove rows where any column has duplicate values.

```
df_cleaned = df.dropDuplicates()  
df_cleaned.show()
```

21. Write PySpark code to read a CSV file and infer its schema.

Scenario: You need to load a CSV file into a DataFrame, ensuring the schema is inferred.

```
df = spark.read.option("header", "true").option("inferSchema", "true").csv("path_to_csv")  
df.show()
```

22. Write PySpark code to merge multiple small files into a single file.

Scenario: You have multiple small files in HDFS, and you want to consolidate them into one large file.

```
df.coalesce(1).write.mode("overwrite").csv("output_path")
```



Siddhartha Subudhi

Data Engineer

@siddhartha-subudhi

23. Write PySpark code to calculate the cumulative sum of a column.

Scenario: You want to calculate a cumulative sum of sales in your DataFrame.

```
from pyspark.sql.window import Window  
from pyspark.sql.functions import sum  
  
windowSpec = Window.orderBy("date").rowsBetween(Window.unboundedPreceding, 0)  
df_with_cumsum = df.withColumn("cumulative_sum", sum("sales").over(windowSpec))  
df_with_cumsum.show()
```

24. Write PySpark code to find outliers in a dataset.

Scenario: Detect outliers in the "sales" column based on the $1.5 * \text{IQR}$ rule.

```
from pyspark.sql.functions import expr  
  
q1 = df.approxQuantile("sales", [0.25], 0.01)[0]  
q3 = df.approxQuantile("sales", [0.75], 0.01)[0]  
  
iqr = q3 - q1  
  
lower_bound = q1 - 1.5 * iqr  
upper_bound = q3 + 1.5 * iqr  
  
df_outliers = df.filter((col("sales") < lower_bound) | (col("sales") > upper_bound))  
df_outliers.show()
```

25. Write PySpark code to convert a DataFrame to a Pandas DataFrame.

Scenario: Convert your PySpark DataFrame into a Pandas DataFrame for local processing.

```
pandas_df = df.toPandas()
```



Siddhartha Subudhi

Data Engineer

@siddhartha-subudhi

Data Cleaning with Apache Spark

Spark Schema

```
.printSchema()

# Import schema

import pyspark.sql.types
from pyspark.sql.types import *

peopleSchema = StructType([
    # Define the name field
    StructField('name', StringType(), True), # Define the name field
    StructField('age', IntegerType(), True), # Add the age field
    StructField('city', StringType(), True)]) # Add the city field
# if false, means cannot be nullable.

people_df = spark.read.format('csv').load(name='rawdata.csv',
schema=peopleSchema)
```

Immutability and Lazy Processing

A component of functional programming

Defined once

Unable to be directly modified

Re-created if reassigned

Able to be shared efficiently

```
# Immutability Example
# Load the CSV file
aa_dfw_df = spark.read.format('csv').options(Header=True).load('AA_DFW_2018.csv')

# Add the airport column using the F.lower() method
aa_dfw_df = aa_dfw_df.withColumn('airport', F.lower(aa_dfw_df['Destination Airport']))

# Drop the Destination Airport column
aa_dfw_df = aa_dfw_df.drop(aa_dfw_df['Destination Airport'])

# Show the DataFrame
aa_dfw_df.show()
```

Understanding Parquet

Difficulties with CSV files

No defined schema (no data type, no column name beyond a header row)

Nested data requires special (content containing a comma requires escaping, using the escape character within content requires even further escaping)

handling Encoding format limited

for spark: slow to parse, cannot be shared during the import process; if no schema is defined, all data must be read before a schema can be inferred, forcing the code to read the file twice.

for spark: files cannot be filtered (no 'predicate pushdown', ordering tasks to do the least amount of work, filtering data prior to processing is one of the primary optimizations of predicate pushdown.)

for spark: any intermediate use requires redefining schema. Spark processes are often multi-step and may utilize an intermediate file representation. These representations allow data to be used later without regenerating the data from source.

The Parquet Format

columnar data format

supported in spark and other data processing frameworks

supports predicate pushdown

automatically stores schema information

binary file format

Parquet is a compressed columnar data format developed for use in any Hadoop based system. Include: Spark, Hadoop, Apache Impala... Perfect for intermediary or on-disk representation of processed data.

predicate pushdown This means Spark will only process the data necessary to complete the operations you define versus reading the entire dataset.

Working with Parquet

```
df = spark.read.format('parquet').load('filename.parquet')
df = spark.read.parquet('filename.parquet')

df.write.format('parquet').save('filename.parquet')
df.write.parquet('filename.parquet') #mode='overwrite'
```

Parquet with SQL

parquet - dataframe - table

```
flight_df = spark.read.parquet('flights.parquet')
flight_df.createOrReplaceTempView('flights')
short_flights_df = spark.sql('SELECT * FROM flights WHERE flightduration <100')

# Run a SQL query of the average flight duration
avg_duration = spark.sql('SELECT avg(flight_duration) from flights').collect()[0]
print('The average flight time is: %d' % avg_duration)
```

DataFrame column operations

DataFrames:

Made up of rows & columns

Immutable

Use various transformation operations to modify data

```
# Return rows where name starts with "M"
voter_df.filter(voter_df.name.like('M%'))
# Return name and position only
voters = voter_df.select('name', 'position')
```

Common DataFrame transformations

```
#Filter / Where interchangeable
voter_df.filter(voter_df.date > '1/1/2019') # or voter_df.where(...)

#Select
voter_df.select(voter_df.name)

#withColumn creates new column
voter_df.withColumn('year', voter_df.date.year)

#drop
voter_df.drop('unused_column')
```

Filtering data

Remove nulls

Remove odd entries

Split data from combined sources

Negate with ~

```
voter_df.filter(voter_df['name'].isNotNull())
voter_df.filter(voter_df.date.year > 1800)
voter_df.where(voter_df['_c0'].contains('VOTE'))
voter_df.where(~ voter_df._c1.isNull())

# Show the distinct VOTER_NAME entries
voter_df.select('VOTER_NAME').distinct().show(40, truncate=False)

# Filter voter_df where the VOTER_NAME is 1-20 characters in length
voter_df = voter_df.filter('length(VOTER_NAME) > 0 and length(VOTER_NAME) < 20')

# Filter out voter_df where the VOTER_NAME contains an underscore
voter_df = voter_df.filter(~ F.col('VOTER_NAME').contains('_'))

# Show the distinct VOTER_NAME entries again
voter_df.select('VOTER_NAME').distinct().show(40, truncate=False)
```

Column string transformations

```
#Contained in pyspark.sql.functions
import pyspark.sql.functions as F

#Applied per column as transformation
voter_df.withColumn('upper', F.upper('name'))

#Can create intermediary columns
voter_df.withColumn('splits', F.split('name', ' '))

#Can cast to other types
voter_df.withColumn('year', voter_df['c4'].cast(IntegerType()))
```

ArrayType() column functions

Various utility functions / transformations to interact with ArrayType() .

```
.size() - returns length of arrayType() column

.getItem() - used to retrieve a specific item at index of list column.

# Add a new column called splits separated on whitespace
voter_df = voter_df.withColumn('splits', F.split(voter_df.VOTER_NAME, '\s+'))

# Create a new column called first_name based on the first item in splits
voter_df = voter_df.withColumn('first_name', voter_df.splits.getItem(0))

# Get the last entry of the splits list and create a column called last_name
voter_df = voter_df.withColumn('last_name', voter_df.splits.getItem(F.size('splits') - 1))

# Drop the splits column
voter_df = voter_df.drop('splits')

# Show the voter_df DataFrame
voter_df.show()
```

Conditional DataFrame column operations

Conditional clauses

Conditional Clauses are:

Inline version of if / then / else

```
.when()  
.otherwise()  
  
df.select(df.Name, df.Age,  
.when(df.Age >= 18, "Adult")  
.when(df.Age < 18, "Minor"))  
  
df.select(df.Name, df.Age,  
.when(df.Age >= 18, "Adult")  
.otherwise("Minor"))  
  
# Add a column to voter_df for any voter with the title **Councilmember**  
voter_df = voter_df.withColumn('random_val',  
    when(voter_df.TITLE == 'Councilmember', F.rand()))  
  
# Add a column to voter_df for a voter based on their position  
voter_df = voter_df.withColumn('random_val',  
    when(voter_df.TITLE == 'Councilmember', F.rand())  
    .when(voter_df.TITLE == 'Mayor', 2)  
    .otherwise(0))
```

User defined functions

Python method

Wrapped via the `pyspark.sql.functions.udf` method

Stored as a variable

Called like a normal Spark function

```
def getFirstAndMiddle(names):
    # Return a space separated string of names
    return ' '.join(names[:-1])

# Define the method as a UDF
udfFirstAndMiddle = F.udf(getFirstAndMiddle, StringType())

# Create a new column using your UDF
voter_df = voter_df.withColumn('first_and_middle_name', udfFirstAndMiddle(voter_)

# Drop the unnecessary columns then show the DataFrame
voter_df = voter_df.drop('first_name')
voter_df = voter_df.drop('splits')
voter_df.show()
```

Reverse string UDF

```
#Define a Python method
def reverseString(mystr):
    return mystr[::-1]

#Wrap the function and store as a variable
udfReverseString = udf(reverseString, StringType())

#Use with Spark
user_df = user_df.withColumn('ReverseName', udfReverseString())
```

Argument-less example

```
def sortingCap():
    return random.choice(['G', 'H', 'R', 'S'])
udfSortingCap = udf(sortingCap, StringType())
user_df = user_df.withColumn('Class', udfSortingCap())
```

Partitioning and lazy processing

Partitioning

DataFrames are broken up into partitions

Partition size can vary

Each partition is handled independently

To check the number of partitions, use the method `.rdd.getNumPartitions()` on a DataFrame.

Lazy processing

Transformations are lazy

* `.withColumn(...)`

* `.select(...)`

* `.cache()`

Nothing is actually done until an action is performed

* `.count()`

* `.write(...)`

* `.show()`

Transformations can be re-ordered for best performance

Sometimes causes unexpected behavior

Adding IDs

Normal ID fields:

Common in relational databases

Most usually an integer increasing, sequential and unique

Not very parallel

Monotonically increasing IDs

`pyspark.sql.functions.monotonically_increasing_id()`

Integer (64-bit), increases in value, unique

Not necessarily sequential (gaps exist)

Completely parallel

```
# Select all the unique council voters
voter_df = df.select(df["VOTER NAME"]).distinct()

# Count the rows in voter_df
print("\nThere are %d rows in the voter_df DataFrame.\n" % voter_df.count())

# Add a ROW_ID
voter_df = voter_df.withColumn('ROW_ID', F.monotonically_increasing_id())

# Show the rows with 10 highest IDs in the set
voter_df.orderBy(voter_df.ROW_ID.desc()).show(10)
```

More ID Tricks

Depending on your needs, you may want to start your IDs at a certain value so there isn't overlap with previous runs of the Spark task. This behavior is similar to how IDs would behave in a relational database. Make sure that the IDs output from a monthly Spark task start at the highest value from the previous month.

Caching

Caching in Spark:

Stores DataFrames in memory or on disk

Improves speed on later transformations / actions

Reduces resource usage

Disadvantages of caching

Very large data sets may not fit in memory

Local disk based caching may not be a performance improvement

Cached objects may not be available

Caching tips When developing Spark tasks:

Cache only if you need it

Try caching DataFrames at various points and determine if your performance improves

Cache in memory and fast SSD / NVMe storage

Cache to slow local disk if needed

Use intermediate files!

Stop caching objects when finished

Eviction Policy Least Recently Used (LRU)

Caching is a lazy operation. It requires an action to trigger it. eg.

```
spark.sql("select count(*) from text").show()  
partitioned_df.count()
```

```
df.cache()      #df.persist() df.persist(storageLevel=pyspark.StorageLevel.MEMORY)  
df.unpersist()
```

```
#Determining whether a dataframe is cached  
df.is_cached
```

```
#storage level useDisk useMemory useOffHeap deserialized replication  
df.storageLevel
```

```
#Caching a table  
df.createOrReplaceTempView('df')  
spark.catalog.cacheTable('df')  
spark.catalog.isCached(tableName='df')  
spark.catalog.dropTempView('table1')  
# List the tables  
print("Tables:\n", spark.catalog.listTables())
```

```
# Uncaching a table  
spark.catalog.uncacheTable('df')  
spark.catalog.clearCache()
```

Improve import performance

Spark clusters

Spark Clusters are made of two types of processes

Driver process

Worker processes

Import performance

Important parameters:

Number of objects (Files, Network locations, etc)

More objects better than larger ones

Can import via wildcard

```
airport_df = spark.read.csv('airports-*txt.gz')
```

General size of objects

Spark performs better if objects are of similar size

It's safe to assume the more import objects available, the better the cluster can divvy up the job.

Schemas

A well-defined schema will drastically improve import performance

Avoids reading the data multiple times

Provides validation on import

How to split objects

Use OS utilities / scripts (split, cut, awk)

```
split -l 10000 -d largefile chunk-
```

每个文件100000行，字符，名字叫largefile，生成chunk0000开始

Use custom scripts

Write out to Parquet

```
df_csv = spark.read.csv('singlelargefile.csv')
```

```
df_csv.write.parquet('data.parquet')
df = spark.read.parquet('data.parquet')
```

Explaining the Spark execution plan

```
voter_df = df.select(df['VOTER NAME']).distinct()
voter_df.explain()
```

== Physical Plan ==

```
*(2) HashAggregate(keys=[VOTER NAME#15], functions=[])
+- Exchange
hashpartitioning(VOTER NAME#15, 200)

+- *(1) HashAggregate(keys=[VOTER NAME#15], functions=[])
+- *(1) FileScan csv [VOTER NAME#15] Batched: false, Format: CSV, Location:
InMemoryFileIndex[file:/DallasCouncilVotes.csv.gz],
PartitionFilters: [], PushedFilters: [],
ReadSchema: struct<VOTER NAME:string>
```

Shuffling

Shuffling refers to moving data around to various workers to complete a task

Hides complexity from the user

Can be slow to complete

Lowers overall throughput

Is often necessary, but try to minimize

How to limit shuffling

Limit use of .repartition(num_partitions)

Use .coalesce(num_partitions) instead

Use care when calling .join() Use .broadcast()

May not need to limit it

Broadcasting

Provides a copy of an object to each worker

Prevents undue / excess communication between nodes

Can drastically speed up `.join()` operations

A couple tips:

Broadcast the smaller DataFrame. The larger the DataFrame, the more time required to transfer to the worker nodes.

On small DataFrames, it may be better skip broadcasting and let Spark figure out any optimization on its own.

If you look at the query execution plan, a broadcastHashJoin indicates you've successfully configured broadcasting.

Use the `.broadcast(<DataFrame>)` method

```
from pyspark.sql.functions import broadcast
combined_df = df_1.join(broadcast(df_2))
```

Cluster Configurations

Configuration options

Spark contains many configuration settings

These can be modified to match needs

Reading configuration settings: `spark.conf.get(<configuration name>)`

Writing configuration settings `spark.conf.set(<configuration name>)`

Configure Spark to use 500 partitions

```
spark.conf.set('spark.sql.shuffle.partitions', 500)
```

```
# Name of the Spark application instance
app_name = spark.conf.get('spark.app.name')

# Driver TCP port
driver_tcp_port = spark.conf.get('spark.driver.port')

# Number of join partitions
num_partitions = spark.conf.get('spark.sql.shuffle.partitions')

# Store the number of partitions in variable
before = departures_df.rdd.getNumPartitions()

# Configure Spark to use 500 partitions
spark.conf.set('spark.sql.shuffle.partitions', 500)

# Recreate the DataFrame using the departures data file
departures_df = spark.read.csv('departures.txt.gz').distinct()
```

Cluster Types

Spark deployment options:

Single node (deploying all components on a single system, can be physical/VM/container)

Standalone (dedicated machines as the driver and workers)

Managed (cluster components are handled by a third party cluster manager)

YARN

Mesos

Kubernetes

Driver

Task assignment

Result consolidation

Shared data access

Tips:

Driver node should have double the memory of the worker

Fast local storage helpful

Worker

Runs actual tasks

Ideally has all code, data, and resources for a given task

Recommendations:

More worker nodes is often better than larger workers

Test to find the balance

Fast local storage extremely useful

Data Pipelines

Input(s)

CSV, JSON, web services, databases

Transformations

```
withColumn() ,  
.filter() ,  
.drop()
```

Output(s)

CSV, Parquet, database Validation

Analysis

Pipeline details

Not formally defined in Spark

Typically all normal Spark code required for task

```
schema = StructType([
    StructField('name', StringType(), False),
    StructField('age', StringType(), False)
])
df = spark.read.format('csv').load('datafile').schema(schema) df = df.withColur
...
df.write.parquet('outdata.parquet')
df.write.json('outdata.json')

# Import the data to a DataFrame
departures_df = spark.read.csv('2015-departures.csv.gz', header=True)

# Remove any duration of 0
departures_df = departures_df.filter(~departures_df[3].contains('0'))

# Add an ID column
departures_df = departures_df.withColumn('id', F.monotonically_increasing_id())

# Write the file out to JSON format
departures_df.write.json('output.json')
```

Data Handling

Parsing

Incorrect data

- * Empty rows
- * Commented lines
- * Headers

Nested structures

- * Multiple delimiters

Non-regular data

- * Differing numbers of columns per row

Removing blank lines, headers, and comments

Spark's CSV parser:

Automatically removes blank lines

Can remove comments using an optional argument

```
df1 = spark.read.csv('datafile.csv.gz', comment='#')
```

Handles header fields

Defined via argument

Ignored if a schema is defined

```
df1 = spark.read.csv('datafile.csv.gz', header='True')
```

Count the number of rows beginning with '#'

```
comment_count =  
annotations_df.where(col('_c0').startswith('#')).count()  
  
# Split _c0 on the tab character and store the list in a variable  
tmp_fields = F.split(annotations_df['_c0'], '\t')  
  
# Create the colcount column on the DataFrame  
annotations_df = annotations_df.withColumn('colcount', F.size(tmp_fields))  
  
# Remove any rows containing fewer than 5 fields  
annotations_df_filtered = annotations_df.filter(~ (annotations_df["colcount"] <  
  
# Count the number of rows  
final_count = annotations_df_filtered.count()  
print("Initial count: %d\nFinal count: %d" % (initial_count, final_count))
```

Automatic column creation

Spark will:

Automatically create columns in a DataFrame based on sep argument
df1 = spark.read.csv('datafile.csv.gz', sep=',')

Defaults to using ,

Can still successfully parse if sep is not in string

```
df1 = spark.read.csv('datafile.csv.gz', sep='*')
```

Stores data in column defaulting to _c0

Allows you to properly handle nested separators

```
# Split the content of _c0 on the tab character (aka, '\t')
split_cols = F.split(annotations_df['_c0'], '\t')

# Add the columns folder, filename, width, and height
split_df = annotations_df.withColumn('folder', split_cols.getItem(0))
split_df = split_df.withColumn('filename', split_cols.getItem(1))
split_df = split_df.withColumn('width', split_cols.getItem(2))
split_df = split_df.withColumn('height', split_cols.getItem(3))

def retriever(cols, colcount):
    # Return a list of dog data
    return cols[4:colcount]

# Define the method as a UDF
udfRetriever = F.udf(retriever, ArrayType(StringType()))

# Create a new column using your UDF
split_df = split_df.withColumn('dog_list', udfRetriever(split_df.split_cols, sp`))

# Remove the original column, split_cols, and the colcount
split_df = split_df.drop('_c0').drop('colcount').drop('split_cols')
```

Data Validation

Validation is:

Verifying that a dataset complies with the expected format

Number of rows / columns

Data types

Complex validation rules

Validating via joins

Compares data against known values

Easy to find data in a given set

Comparatively fast

```
parsed_df = spark.read.parquet('parsed_data.parquet')
company_df = spark.read.parquet('companies.parquet')
verified_df = parsed_df.join(company_df, parsed_df.company == company_df.compan)

# Rename the column in valid_folders_df
valid_folders_df = valid_folders_df.withColumnRenamed('_c0', 'folder')

# Count the number of rows in split_df
split_count = split_df.count()

# Join the DataFrames
joined_df = split_df.join(F.broadcast(valid_folders_df), "folder")

# Compare the number of rows remaining
joined_count = joined_df.count()
print("Before: %d\nAfter: %d" % (split_count, joined_count))
```

This automatically removes any rows with a company not in the valid_df !

Complex rule validation

Using Spark components to validate logic:

Calculations

Verifying against external source

Likely uses a UDF to modify / verify the DataFrame

```
# Determine the row counts for each DataFrame
split_count = split_df.count()
joined_count = joined_df.count()

# Create a DataFrame containing the invalid rows
invalid_df = split_df.join(F.broadcast(joined_df), 'folder', 'left_anti')

# Validate the count of the new DataFrame is as expected
invalid_count = invalid_df.count()
print(" split_df:\t%d\n joined_df:\t%d\n invalid_df: \t%d" % (split_count, joined_count, invalid_count))

# Determine the number of distinct folder columns removed
invalid_folder_count = invalid_df.select('folder').distinct().count()
print("%d distinct invalid folders found" % invalid_folder_count)
```

```
# Select the dog details and show 10 untruncated rows
print(joined_df.select('dog_list').show(truncate=False))

# Define a schema type for the details in the dog list
DogType = StructType([
    StructField("breed", StringType(), False),
    StructField("start_x", IntegerType(), False),
    StructField("start_y", IntegerType(), False),
    StructField("end_x", IntegerType(), False),
    StructField("end_y", IntegerType(), False)
])

# Create a function to return the number and type of dogs as a tuple
def dogParse(doglist):
    dogs = []
    for dog in doglist:
        (breed, start_x, start_y, end_x, end_y) = dog.split(',')
        dogs.append((breed, int(start_x), int(start_y), int(end_x), int(end_y)))
    return dogs

# Create a UDF
udfDogParse = F.udf(dogParse, ArrayType(DogType))

# Use the UDF to list of dogs and drop the old column
joined_df = joined_df.withColumn('dogs', udfDogParse('dog_list')).drop('dog_list')

# Show the number of dogs in the first 10 rows
joined_df.select(F.size('dogs')).show(10)
```

```
# Define a UDF to determine the number of pixels per image
def dogPixelCount(doglist):
    totalpixels = 0
    for dog in doglist:
        totalpixels += (dog[3] - dog[1]) * (dog[4] - dog[2])
    return totalpixels

# Define a UDF for the pixel count
udfDogPixelCount = F.udf(dogPixelCount, IntegerType())
joined_df = joined_df.withColumn('dog_pixels', udfDogPixelCount('dogs'))

# Create a column representing the percentage of pixels
joined_df = joined_df.withColumn('dog_percent', (joined_df.dog_pixels / (joined_df.dog_pixels + joined_df.non_dog_pixels)) * 100)

# Show the first 10 annotations with more than 60% dog
joined_df.where('dog_percent > 60').show(10)
```

Key Differences in Apache Spark Components and Concepts

Hadoop vs. Spark Architecture

Aspect	Hadoop	Spark
Storage	Uses HDFS for storage	Uses in-memory processing for speed
Processing	MapReduce is disk-based	In-memory processing improves performance
Integration	Runs independently or with Hadoop ecosystem	Can run on top of Hadoop; more flexible
Complexity	More complex setup and deployment	Simpler to deploy and configure
Performance	Slower for iterative tasks due to disk I/O	Better performance for iterative tasks

RDD vs. DataFrame vs. Dataset

Aspect	RDD	DataFrame	Dataset
API Level	Low-level, more control	High-level, optimized with Catalyst	High-level, type-safe
Schema	No schema, unstructured	Uses schema for structured data	Strongly typed, compile-time type safety
Optimization	No built-in optimization	Optimized using Catalyst	Optimized using Catalyst, with type safety
Type Safety	No type safety	No compile-time type safety	Provides compile-time type safety
Performance	Less optimized for performance	Better performance due to optimizations	Combines type safety with optimization

Follow me on LinkedIn – [Shivakiran kotur](#)



Follow on linkedin
@Shivakiran kotur

Action vs. Transformation

Aspect	Action	Transformation
Execution	Triggers execution of the Spark job	Builds up a logical plan of data operations
Return Type	Returns results or output	Returns a new RDD/DataFrame
Evaluation	Eager evaluation; executes immediately	Lazy evaluation; executed when an action is triggered
Computation	Involves actual computation (e.g., collect())	Defines data transformations (e.g., map())
Performance	Can cause data processing; affects performance	Does not affect performance until an action is called

Map vs. FlatMap

Aspect	Map	FlatMap
Output	Returns one output element per input element	Can return zero or more output elements per input
Flattening	Does not flatten output	Flattens the output into a single level
Use Case	Suitable for one-to-one transformations	Suitable for one-to-many transformations
Complexity	Simpler, straightforward	More complex due to variable number of outputs
Examples	<code>map(x => x * 2)</code>	<code>flatMap(x => x.split(" "))</code>

GroupByKey vs ReduceByKey

Aspect	GroupByKey	ReduceByKey
Operation	Groups all values by key	Aggregates values with the same key
Efficiency	Can lead to high shuffling	More efficient due to partial aggregation
Data Movement	Requires shuffling of all values	Minimizes data movement through local aggregation
Use Case	Useful for simple grouping	Preferred for aggregations and reductions
Performance	Less efficient with large datasets	Better performance for large datasets

Follow me on LinkedIn – [Shivakiran kotur](#)



Follow on linkedin
@Shivakiran kotur

Repartition Vs Coalesce

Aspect	Repartition	Coalesce
Partitioning	Can increase or decrease the number of partitions	Only decreases the number of partitions
Shuffling	Involves full shuffle	Avoids full shuffle, more efficient
Efficiency	More expensive due to shuffling	More efficient for reducing partitions
Use Case	Used for increasing partitions or balancing load	Used for reducing partitions, typically after filtering
Performance	Can be costly for large datasets	More cost-effective for reducing partitions

Cache Vs Persist

Aspect	Cache	Persist
Storage Level	Defaults to MEMORY_ONLY	Can use various storage levels (e.g., MEMORY_AND_DISK)
Flexibility	Simplified, with default storage level	Offers more options for storage levels
Use Case	Suitable for simple caching scenarios	Suitable for complex caching scenarios requiring different storage levels
Implementation	Easier to use, shorthand for MEMORY_ONLY	More flexible, allows custom storage options
Performance	Suitable when memory suffices	More efficient when dealing with larger datasets and limited memory

Narrow Vs Wide Transformation

Aspect	Narrow Transformation	Wide Transformation
Partitioning	Each parent partition is used by one child partition	Requires data from multiple partitions
Shuffling	No shuffling required	Involves shuffling of data
Performance	More efficient and less costly	Less efficient due to data movement
Examples	map(), filter()	groupByKey(), join()
Complexity	Simpler and faster	More complex and slower due to data movement

Collect vs Take

Aspect	Collect	Take
Output	Retrieves all data from the RDD/DataFrame	Retrieves a specified number of elements
Memory Usage	Can be expensive and use a lot of memory	More memory-efficient
Use Case	Used when you need the entire dataset	Useful for sampling or debugging
Performance	Can cause performance issues with large data	Faster and more controlled
Action Type	Triggers full data retrieval	Triggers partial data retrieval

Broadcast Variable vs Accumulator

Aspect	Broadcast Variable	Accumulator
Purpose	Efficiently shares read-only data across tasks	Tracks metrics and aggregates values
Data Type	Data that is shared and read-only	Counters and sums, often numerical
Use Case	Useful for large lookup tables or configurations	Useful for aggregating metrics like counts
Efficiency	Reduces data transfer by broadcasting data once	Efficient for aggregating values across tasks
Mutability	Immutable, read-only	Mutable, can be updated during computation

Follow me on LinkedIn – [Shivakiran kotur](#)



Spark SQL vs DataFrame API

Aspect	Spark SQL	DataFrame API
Interface	Executes SQL queries	Provides a programmatic interface
Syntax	Uses SQL-like syntax	Uses function-based syntax
Optimization	Optimized with Catalyst	Optimized with Catalyst
Use Case	Preferred for complex queries and legacy SQL code	Preferred for programmatic data manipulations
Integration	Can integrate with Hive and other SQL databases	Provides a unified interface for different data sources

Spark Streaming Vs Structured Streaming

Aspect	Spark Streaming	Structured Streaming
Processing	Micro-batch processing	Micro-batch and continuous processing
API	RDD-based API	SQL-based API with DataFrame/Dataset support
Complexity	More complex and lower-level	Simplified with high-level APIs
Consistency	Can be less consistent due to micro-batches	Provides stronger consistency guarantees
Performance	Can be slower for complex queries	Better performance with optimizations

Shuffle vs MapReduce

Aspect	Shuffle	MapReduce
Operation	Data reorganization across partitions	Data processing model for distributed computing
Efficiency	Can be costly due to data movement	Designed for batch processing with high I/O
Performance	Affects performance based on the amount of data movement	Optimized for large-scale data processing but less efficient for iterative tasks
Use Case	Used in Spark for data redistribution	Used in Hadoop for data processing tasks
Implementation	Integrated into Spark operations	Core component of the Hadoop ecosystem

Follow me on LinkedIn – [Shivakiran kotur](#)



Union vs Join

Aspect	Union	Join
Operation	Combines two DataFrames/RDDs into one	Combines rows from two DataFrames/RDDs based on a key
Data Requirements	Requires same schema for both DataFrames/RDDs	Requires a common key for joining
Performance	Generally faster as it does not require key matching	Can be slower due to key matching and shuffling
Output	Stacks data vertically	Merges data horizontally based on keys
Use Case	Appending data or combining datasets	Merging related data based on keys

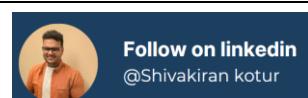
Executor vs Driver

Aspect	Executor	Driver
Role	Executes tasks and processes data	Coordinates and manages the Spark application
Memory	Memory allocated per executor for data processing	Memory used for managing application execution
Lifecycle	Exists throughout the application execution	Starts and stops the Spark application
Tasks	Runs the tasks assigned by the driver	Schedules and coordinates tasks and jobs
Parallelism	Multiple executors run in parallel	Single driver coordinates multiple executors

Checkpointing vs Caching

Aspect	Checkpointing	Caching
Purpose	Provides fault tolerance and reliability	Improves performance by storing intermediate data
Storage	Writes data to stable storage (e.g., HDFS)	Stores data in memory or on disk (depends on storage level)
Use Case	Used for recovery in case of failures	Used for optimizing repeated operations
Impact	Can be more costly and slow	Generally faster but not suitable for fault tolerance
Data	Data is written to external storage	Data is kept in memory or disk storage for quick access

Follow me on LinkedIn – [Shivakiran kotur](#)



ReduceByKey vs AggregateByKey

Aspect	ReduceByKey	AggregateByKey
Operation	Combines values with the same key using a function	Performs custom aggregation and combinatory operations
Efficiency	More efficient for simple aggregations	Flexible for complex aggregation scenarios
Shuffling	Involves shuffling but can be optimized	Can be more complex due to custom aggregation
Use Case	Suitable for straightforward aggregations	Ideal for advanced and custom aggregations
Performance	Generally faster for simple operations	Performance varies with complexity

SQL Context vs Hive Context vs Spark Session

Aspect	SQL Context	Hive Context	Spark Session
Purpose	Provides SQL query capabilities	Provides integration with Hive for SQL queries	Unified entry point for Spark functionality
Integration	Basic SQL capabilities	Integrates with Hive Metastore	Combines SQL, DataFrame, and Streaming APIs
Usage	Legacy, less functionality	Supports HiveQL and Hive UDFs	Supports all Spark functionalities including Hive
Configuration	Less flexible and older	Requires Hive setup and configuration	Modern and flexible, manages configurations
Capabilities	Limited to SQL queries	Extends SQL capabilities with Hive integration	Comprehensive access to all Spark features

Follow me on LinkedIn – [Shivakiran kotur](#)



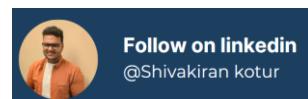
Broadcast Join Vs Shuffle Join

Aspect	Broadcast Join	Shuffle Join
Operation	Broadcasts a small dataset to all nodes	Shuffles data across nodes for joining
Data Size	Suitable for small datasets	Suitable for larger datasets
Efficiency	More efficient for small tables	More suited for large datasets
Performance	Faster due to reduced shuffling	Can be slower due to extensive shuffling
Use Case	Use when one dataset is small relative to others	Use when both datasets are large

Spark Context vs Spark Session

Aspect	Spark Context	Spark Session
Purpose	Entry point for Spark functionality	Unified entry point for Spark functionalities
Lifecycle	Created before Spark jobs start	Manages the Spark application lifecycle
Functionality	Provides access to RDD and basic Spark functionality	Provides access to RDD, DataFrame, SQL, and Streaming APIs
Configuration	Configuration is less flexible	More flexible and easier to configure
Usage	Older, used for legacy applications	Modern and recommended for new applications

Follow me on LinkedIn – [Shivakiran kotur](#)



Structured Streaming vs Spark Streaming

Aspect	Structured Streaming	Spark Streaming
Processing	Micro-batch and continuous processing	Micro-batch processing
API	SQL-based API with DataFrame/Dataset support	RDD-based API
Complexity	Simplified and high-level	More complex and low-level
Consistency	Provides stronger consistency guarantees	Can be less consistent due to micro-batches
Performance	Better performance with built-in optimizations	Can be slower for complex queries

Partitioning vs Bucketing

Aspect	Partitioning	Bucketing
Purpose	Divides data into multiple partitions based on a key	Divides data into buckets based on a hash function
Usage	Used to optimize queries by reducing data scanned	Used to improve join performance and maintain sorted data
Shuffling	Reduces shuffling by placing related data together	Reduces shuffle during joins and aggregations
Data Layout	Data is physically separated based on partition key	Data is organized into fixed-size buckets
Performance	Improves performance for queries involving partition keys	Enhances performance for join operations

DELTA LAKE

WITH SPARK SQL

Delta Lake is an open source storage layer that brings ACID transactions to Apache Spark™ and big data workloads.

delta.io | [Documentation](#) | [GitHub](#) | [Delta Lake on Databricks](#)

CREATE AND QUERY DELTA TABLES

Create and use managed database

```
-- Managed database is saved in the Hive metastore.  
Default database is named "default".  
DROP DATABASE IF EXISTS dbName;  
CREATE DATABASE dbName;  
USE dbName -- This command avoids having to specify  
dbName.tableName every time instead of just tableName.
```

Query Delta Lake table by table name (preferred)

```
/* You can refer to Delta Tables by table name, or by  
path. Table name is the preferred way, since named tables  
are managed in the Hive Metastore (i.e., when you DROP a  
named table, the data is dropped also – not the case for  
path-based tables.) */  
SELECT * FROM [dbName.] tableName
```

Query Delta Lake table by path

```
SELECT * FROM delta.`path/to/delta_table` -- note backticks
```

Convert Parquet table to Delta Lake format in place

```
-- by table name  
CONVERT TO DELTA [dbName.]tableName  
[PARTITIONED BY (col_name1 col_type1, col_name2  
col_type2)]  
  
-- path-based tables  
CONVERT TO DELTA parquet.`/path/to/table` -- note backticks  
[PARTITIONED BY (col_name1 col_type1, col_name2 col_type2)]
```

Create Delta Lake table as SELECT * with no upfront schema definition

```
CREATE TABLE [dbName.] tableName  
USING DELTA  
AS SELECT * FROM tableName | parquet.`path/to/data`  
[LOCATION `/path/to/table`]  
-- using location = unmanaged table
```

Create table, define schema explicitly with SQL DDL

```
CREATE TABLE [dbName.] tableName (  
    id INT [NOT NULL],  
    name STRING,  
    date DATE,  
    int_rate FLOAT)  
USING DELTA  
[PARTITIONED BY (time, date)] -- optional
```

Copy new data into Delta Lake table (with idempotent retries)

```
COPY INTO [dbName.] targetTable  
    FROM (SELECT * FROM `/path/to/table`)  
    FILEFORMAT = DELTA -- or CSV, Parquet, ORC, JSON, etc.
```

DELTA LAKE DDL/DML: UPDATE, DELETE, MERGE, ALTER TABLE

Update rows that match a predicate condition

```
UPDATE tableName SET event = 'click' WHERE event = 'clk'
```

Delete rows that match a predicate condition

```
DELETE FROM tableName WHERE "date < '2017-01-01"
```

Insert values directly into table

```
INSERT INTO TABLE tableName VALUES (  
    (8003, "Kim Jones", "2020-12-18", 3.875),  
    (8004, "Tim Jones", "2020-12-20", 3.750))  
;  
-- Insert using SELECT statement  
INSERT INTO tableName SELECT * FROM sourceTable  
-- Atomically replace all data in table with new values  
INSERT OVERWRITE loan_by_state_delta VALUES (...)
```

Upsert (update + insert) using MERGE

```
MERGE INTO target  
USING updates  
ON target.Id = updates.Id  
WHEN MATCHED AND target.delete_flag = "true" THEN  
    DELETE  
WHEN MATCHED THEN  
    UPDATE SET * -- star notation means all columns  
WHEN NOT MATCHED THEN  
    INSERT (date, Id, data) -- or, use INSERT *  
    VALUES (date, Id, data)
```

Insert with Deduplication using MERGE

```
MERGE INTO logs  
USING newDedupedLogs  
ON logs.uniqueId = newDedupedLogs.uniqueId  
WHEN NOT MATCHED  
    THEN INSERT *
```

Alter table schema – add columns

```
ALTER TABLE tableName ADD COLUMNS (  
    col_name data_type  
    [FIRST|AFTER colA_name])
```

Alter table – add constraint

```
-- Add "Not null" constraint:  
ALTER TABLE tableName CHANGE COLUMN col_name SET NOT NULL  
-- Add "Check" constraint:  
ALTER TABLE tableName  
ADD CONSTRAINT dateWithinRange CHECK date > "1900-01-01"  
-- Drop constraint:  
ALTER TABLE tableName DROP CONSTRAINT dateWithinRange
```

TIME TRAVEL

View transaction log (aka Delta Log)

```
DESCRIBE HISTORY tableName
```

Query historical versions of Delta Lake tables

```
SELECT * FROM tableName VERSION AS OF 0  
SELECT * FROM tableName@v0 -- equivalent to VERSION AS OF 0  
SELECT * FROM tableName TIMESTAMP AS OF "2020-12-18"
```

Find changes between 2 versions of table

```
SELECT * FROM tableName VERSION AS OF 12  
EXCEPT ALL SELECT * FROM tableName VERSION AS OF 11
```

TIME TRAVEL (CONTINUED)

Rollback a table to an earlier version

```
-- RESTORE requires Delta Lake version 0.7.0+ & DBR 7.4+.  
RESTORE tableName VERSION AS OF 0  
RESTORE tableName TIMESTAMP AS OF "2020-12-18"
```

UTILITY METHODS

View table details

```
DESCRIBE DETAIL tableName  
DESCRIBE FORMATTED tableName
```

Delete old files with Vacuum

```
VACUUM tableName [RETAIN num HOURS] [DRY RUN]
```

Clone a Delta Lake table

```
-- Deep clones copy data from source, shallow clones don't.  
CREATE TABLE [dbName.] targetName  
[SHALLOW | DEEP] CLONE sourceName [VERSION AS OF 0]  
[LOCATION "path/to/table"]  
-- specify location only for path-based tables
```

Interoperability with Python / DataFrames

```
-- Read name-based table from Hive metastore into DataFrame  
df = spark.table("tableName")  
-- Read path-based table into DataFrame  
df = spark.read.format("delta").load("/path/to/delta_table")
```

Run SQL queries from Python

```
spark.sql("SELECT * FROM tableName")  
spark.sql("SELECT * FROM delta.`path/to/delta_table`")
```

Modify data retention settings for Delta Lake table

```
-- logRetentionDuration -> how long transaction log history  
is kept, deletedFileRetentionDuration -> how long ago a file  
must have been deleted before being a candidate for VACUUM.  
ALTER TABLE tableName  
SET TBLPROPERTIES(  
    delta.logRetentionDuration = "interval 30 days",  
    delta.deletedFileRetentionDuration = "interval 7 days"  
);  
SHOW TBLPROPERTIES tableName;
```

PERFORMANCE OPTIMIZATIONS

Compact data files with Optimize and Z-Order

```
*Databricks Delta Lake feature  
OPTIMIZE tableName  
[ZORDER BY (colNameA, colNameB)]
```

Auto-optimize tables

```
*Databricks Delta Lake feature  
ALTER TABLE [table_name | delta.`path/to/delta_table`]  
SET TBLPROPERTIES (delta.autoOptimize.optimizeWrite = true)
```

Cache frequently queried data in Delta Cache

```
*Databricks Delta Lake feature  
CACHE SELECT * FROM tableName  
-- or:  
CACHE SELECT colA, colB FROM tableName WHERE colNameA > 0
```



Delta Lake is an open source storage layer that brings ACID transactions to Apache Spark™ and big data workloads.

delta.io | [Documentation](#) | [GitHub](#) | [API reference](#) | [Databricks](#)

READS AND WRITES WITH DELTA LAKE

Read data from pandas DataFrame

```
df = spark.createDataFrame(pdf)
# where pdf is a pandas DF
# then save DataFrame in Delta Lake format as shown below
```

Read data using Apache Spark™

```
# read by path
df = (spark.read.format("parquet"|"csv"|"json"|etc.)
      .load("/path/to/delta_table"))
# read by table name
df = spark.table("events")
```

Save DataFrame in Delta Lake format

```
(df.write.format("delta")
  .mode("append"|"overwrite")
  .partitionBy("date") # optional
  .option("mergeSchema", "true") # option - evolve schema
  .saveAsTable("events") | .save("/path/to/delta_table")
)
```

Streaming reads (Delta table as streaming source)

```
# by path or by table name
df = (spark.readStream
      .format("delta")
      .schema(schema)
      .table("events") | .load("/delta/events")
)
```

Streaming writes (Delta table as a sink)

```
streamingQuery = (
df.writeStream.format("delta")
  .outputMode("append"|"update"|"complete")
  .option("checkpointLocation", "/path/to/checkpoints")
  .trigger(once=True|processingTime="10 seconds")
  .table("events") | .start("/delta/events")
)
```

CONVERT PARQUET TO DELTA LAKE

Convert Parquet table to Delta Lake format in place

```
deltaTable = DeltaTable.convertToDelta(spark,
"parquet.`/path/to/parquet_table`")

partitionedDeltaTable = DeltaTable.convertToDelta(spark,
"parquet.`/path/to/parquet_table`", "part int")
```

WORKING WITH DELTA TABLES

```
# A DeltaTable is the entry point for interacting with
tables programmatically in Python - for example, to
perform updates or deletes.
from delta.tables import *

deltaTable = DeltaTable.forName(spark, tableName)
deltaTable = DeltaTable.forPath(spark,
delta.`path/to/table`)
```

DELTA LAKE DDL/DML: UPDATES, DELETES, INSERTS, MERGES

Delete rows that match a predicate condition

```
# predicate using SQL formatted string
deltaTable.delete("date < '2017-01-01'")
# predicate using Spark SQL functions
deltaTable.delete(col("date") < "2017-01-01")
```

Update rows that match a predicate condition

```
# predicate using SQL formatted string
deltaTable.update(condition = "eventType = 'clk'", 
  set = { "eventType": "'click'" } )
# predicate using Spark SQL functions
deltaTable.update(condition = col("eventType") == "clk",
  set = { "eventType": lit("click") } )
```

Upsert (update + insert) using MERGE

```
# Available options for merges [see documentation for
details]:
.whenMatchedUpdate(...) | .whenMatchedUpdateAll(...)
.whenNotMatchedInsert(...) | .whenMatchedDelete(...)
(deltaTable.alias("target")).merge(
  source = updatesDF.alias("updates"),
  condition = "target.eventId = updates.eventId")
.whenMatchedUpdateAll()
.whenNotMatchedInsert(
  values = {
    "date": "updates.date",
    "eventId": "updates.eventId",
    "data": "updates.data",
    "count": 1
  }
).execute()
```

Insert with Deduplication using MERGE

```
(deltaTable.alias("logs")).merge(
  newDedupedLogs.alias("newDedupedLogs"),
  "logs.uniqueId = newDedupedLogs.uniqueId")
.whenNotMatchedInsertAll()
.execute()
```

TIME TRAVEL

View transaction log (aka Delta Log)

```
fullHistoryDF = deltaTable.history()
```

Query historical versions of Delta Lake tables

```
# choose only one option: versionAsOf, or timestampAsOf
df = (spark.read.format("delta")
      .option("versionAsOf", 0)
      .option("timestampAsOf", "2020-12-18")
      .load("/path/to/delta_table"))
```

TIME TRAVEL (CONTINUED)

Find changes between 2 versions of a table

```
df1 = spark.read.format("delta").load(pathToTable)
df2 = spark.read.format("delta").option("versionAsOf",
2).load("/path/to/delta_table")
df1.exceptAll(df2).show()
```

Rollback a table by version or timestamp

```
deltaTable.restoreToVersion(0)
deltaTable.restoreToTimestamp('2020-12-01')
```

UTILITY METHODS

Run Spark SQL queries in Python

```
spark.sql("SELECT * FROM tableName")
spark.sql("SELECT * FROM delta.`/path/to/delta_table`")
spark.sql("DESCRIBE HISTORY tableName")
```

Compact old files with Vacuum

```
deltaTable.vacuum() # vacuum files older than default
retention period (7 days)
deltaTable.vacuum(100) # vacuum files not required by
versions more than 100 hours old
```

Clone a Delta Lake table

```
deltaTable.clone(target="/path/to/delta_table",
  isShallow=True, replace=True)
```

Get DataFrame representation of a Delta Lake table

```
df = deltaTable.toDF()
```

Run SQL queries on Delta Lake tables

```
spark.sql("SELECT * FROM tableName")
spark.sql("SELECT * FROM delta.`/path/to/delta_table`")
```

PERFORMANCE OPTIMIZATIONS

Compact data files with Optimize and Z-Order

```
*Databricks Delta Lake feature
spark.sql("OPTIMIZE tableName [ZORDER BY (colA, colB)]")
```

Auto-optimize tables

```
*Databricks Delta Lake feature. For existing tables:
spark.sql("ALTER TABLE [table_name |
delta.`path/to/delta_table`"]
SET TBLPROPERTIES (delta.autoOptimize.optimizeWrite = true)
To enable auto-optimize for all new Delta Lake tables:
spark.sql("SET spark.databricks.delta.properties.
defaults.autoOptimize.optimizeWrite = true")
```

Cache frequently queried data in Delta Cache

```
*Databricks Delta Lake feature
spark.sql("CACHE SELECT * FROM tableName")
-- or:
spark.sql("CACHE SELECT colA, colB FROM tableName
WHERE colNameA > 0")
```

What is PySpark ?

PySpark is nothing but the `Python API` for Apache Spark.

It offers `PySpark Shell` which connects the `Python API` to the spark core and in turn initializes the `Spark context`.

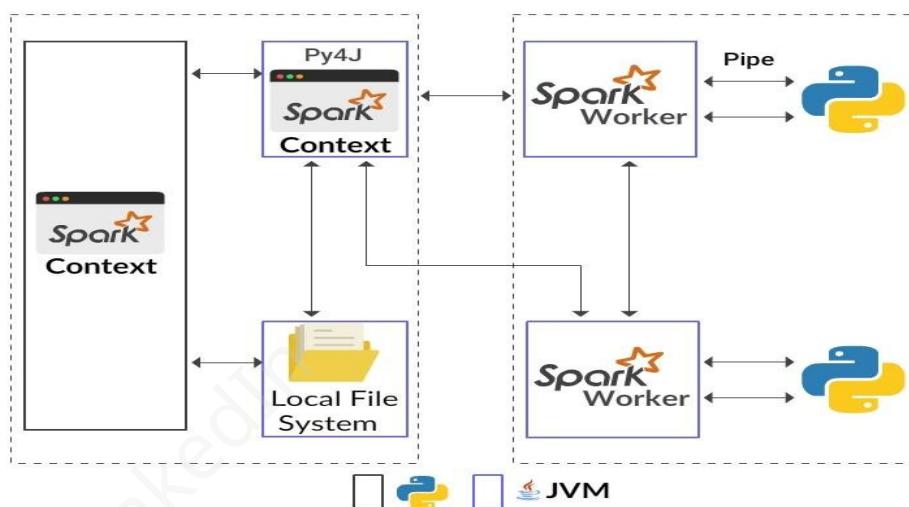
More on PySpark

- For any spark functionality, the entry point is `SparkContext`.
- `SparkContext` uses `Py4J` to launch a JVM and creates a `JavaSparkContext`.
- By default, PySpark has `SparkContext` available as `sc`, so creating a new `SparkContext` won't work.

Py4J

- PySpark is built on top of `Spark's Java API`.
- Data is processed in Python and `cached / shuffled` in the JVM.
- Py4J enables Python programs running in a `Python interpreter` to dynamically access `Java objects in a Java Virtual Machine`.
- Here methods are called as if the Java objects resided in the `Python interpreter and Java collections`. can be accessed through standard `Python collection methods`.

More on Py4J



- In the Python driver program, `SparkContext` uses `Py4J` to launch a `JVM` and create a `JavaSparkContext`.

- To establish local communication between the `Python` and `Java` `SparkContext` objects Py4J is used on the driver.

Installing and Configuring PySpark

- PySpark requires `Python 2.6 or higher`.
- PySpark applications are executed using a standard `CPython` interpreter in order to support `Python modules that use C extensions`.
- By default, PySpark requires python to be available on the `system PATH` and use it to run programs.
- Among PySpark's library dependencies all of them are bundled with `PySpark` including Py4J and they are `automatically imported`.

Getting Started

We can enter the `Spark's python environment` by running the given command in the shell.

```
./bin/pyspark
```

This will start your `PySpark shell`.

```
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.

Welcome to

      _      _ 
     / \_/\_ \_ \_ / /_ 
    _\ \V _ \V _ `/_/ ' /_
   /_ / ._/\_,/_/ /_/\_ \ version 2.2.0
  /_/
Using Python version 2.7.12 (default, Nov 20 2017 18:23:56)
SparkSession available as 'spark'.
<<<
```

Resilient Distributed Datasets (RDDs)

- Resilient distributed datasets (RDDs) are known as the main abstraction in Spark.
- It is a partitioned collection of objects spread across a cluster, and can be persisted in memory or on disk.
- Once RDDs are created they are immutable.

There are two ways to create RDDs:

1. Parallelizing a collection in driver program.
2. Referencing one dataset in an external storage system, like a shared filesystem, HBase, HDFS, or any data source providing a Hadoop InputFormat.

Features Of RDDs

- Resilient, i.e. tolerant to faults using RDD lineage graph and therefore ready to recompute damaged or missing partitions due to node failures.
- Dataset - A set of partitioned data with primitive values or values of values, For example, records or tuples.
- Distributed with data remaining on multiple nodes in a cluster.

Creating RDDs

Parallelizing a collection in driver program.

E.g., here is how to create a parallelized collection holding the numbers 1 to 5:

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

Here, distData is the new RDD created by calling SparkContext's parallelize method.

Creating RDDs

Referencing one dataset in an external storage system, like a shared filesystem, HBase, HDFS, or any data source providing a Hadoop InputFormat.

For example, text file RDDs can be created using the method SparkContext's textFile.

For the file (local path on the machine, hdfs://, s3n://, etc URI) the above method takes a URI and then reads it as a collection containing lines to produce the RDD.

```
distFile = sc.textFile("data.txt")
```

RDD Operations

RDDs support two types of operations: **transformations**, which create a new dataset from an existing one, and **actions**, which return a value to the driver program after running a computation on the dataset.

For example, **map** is a **transformation** that passes each dataset element through a function and returns a new RDD representing the results.

Similarly, **reduce** is an **action** which aggregates all RDD elements by using some functions and then returns the final result to driver program.

More On RDD Operations

As a recap to RDD basics, consider the simple program shown below:

```
lines = sc.textFile("data.txt")
lineLengths = lines.map(lambda s: len(s))
totalLength = lineLengths.reduce(lambda a, b: a + b)
```

The first line defines a **base RDD** from an **external file**.

The second line defines **lineLengths** as the result of a **map transformation**.

Finally, in the third line, we run **reduce**, which is an **action**.

Transformations

- Transformations are functions that use an RDD as the input and return one or more RDDs as the output.
- randomSplit, cogroup, join, reduceByKey, filter, and map are examples of few transformations.
- Transformations do not change the input RDD, but always create one or more new RDDs by utilizing the computations they represent.
- By using transformations, you incrementally create an RDD lineage with all the parent RDDs of the last RDD.
- Transformations are *lazy*, i.e. are not run immediately. Transformations are done on demand.
- Transformations are executed only after calling an action.

Examples Of Transformations

- **filter(func):** Returns a new dataset (RDD) that are created by choosing the elements of the source on which the function returns true.

- **map(func)**: Passes each element of the RDD via the supplied function.
- **union()**: New RDD contains elements from source argument and RDD.
- **intersection()**: New RDD includes only common elements from source argument and RDD.
- **cartesian()**: New RDD cross product of all elements from source argument and RDD.

Actions

- Actions return concluding results of RDD computations.
- Actions trigger execution utilising lineage graph to load the data into original RDD, and then execute all intermediate transformations and write final results out to file system or return it to Driver program.
- **Count, collect, reduce, take, and first** are few actions in spark.

Example of Actions

- **count()**: Get the number of data elements in the RDD.
- **collect()**: Get all the data elements in an RDD as an array.
- **reduce(func)**: Aggregate the data elements in an RDD using this function which takes two arguments and returns one.
- **take (n)**: Fetch first n data elements in an RDD computed by driver program.
- **foreach(func)**: Execute function for each data element in RDD. usually used to update an accumulator or interacting with external systems.
- **first()**: Retrieves the first data element in RDD. It is similar to take(1).
- **saveAsTextFile(path)**: Writes the content of RDD to a text file or a set of text files to local file system/HDFS.

What is Dataframe ?

In general **DataFrames** can be defined as a data structure, which is tabular in nature. It represents rows, each of them consists of a number of observations.

Rows can have a variety of data formats (**heterogeneous**), whereas a column can have data of the same data type (**homogeneous**).

They mainly contain some **metadata** in addition to data like column and row names.

Why DataFrames ?

- DataFrames are widely used for processing a large collection of structured or semi-structured data
- They are having the ability to handle petabytes of data
- In addition, it supports a wide range of data format for reading as well as writing

As a conclusion DataFrame is data organized into named columns

Features of DataFrame

- Distributed
- Lazy Evals
- Immutable

Features Explained

- DataFrames are **Distributed** in Nature, which makes it fault tolerant and highly available data structure.
- **Lazy Evaluation** is an evaluation strategy which will hold the evaluation of an expression until its value is needed.
- DataFrames are **Immutable** in nature which means that it is an object whose state cannot be modified after it is created.

DataFrame Sources

For constructing a DataFrame a wide range of sources are available such as:

- Structured data files
- Tables in Hive
- External Databases
- Existing RDDs

Spark SQL

Spark introduces a programming module for structured data processing called **Spark SQL**.

It provides a programming abstraction called *DataFrame* and can act as distributed **SQL query engine**.

Features of Spark SQL

The main capabilities of using structured and semi-structured data, by **Spark SQL**. Such as:

- Provides DataFrame abstraction in Scala, Java, and Python.
- Spark SQL can read and write data from Hive Tables, JSON, and Parquet in various structured formats.
- Data can be queried by using Spark SQL.

For more details about Spark SQL refer the fresco course [Spark SQL](#)

Important classes of Spark SQL and DataFrames

- `pyspark.sql.SparkSession` :Main entry point for Dataframe SparkSQL functionality
- `pyspark.sql.DataFrame` :A distributed collection of data grouped into named columns
- `pyspark.sql.Column` : A column expression in a DataFrame.
- `pyspark.sql.Row` : A row of data in a DataFrame.
- `pyspark.sql.GroupedData` :Aggregation methods, returned by DataFrame.groupBy().

More On Classes

- `pyspark.sql.DataFrameNaFunctions` : Methods for handling missing data (null values).
- `pyspark.sql.DataFrameStatFunctions` : Methods for statistics functionality.
- `pyspark.sql.functions` : List of built-in functions available for DataFrame.
- `pyspark.sql.types` : List of data types available.
- `pyspark.sql.Window` : For working with window functions.

Creating a DataFrame demo

The entry point into all functionality in Spark is the **SparkSession** class.

To create a basic `SparkSession`, just use `SparkSession.builder`:

```
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Data Frame Example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

More On Creation

Import the `sql module` from `pyspark`

```
from pyspark.sql import *
```

```

Student = Row("firstName", "lastName", "age", "telephone")
s1 = Student('David', 'Julian', 22, 100000)
s2 = Student('Mark', 'Webb', 23, 658545)
StudentData=[s1,s2]
df=spark.createDataFrame(StudentData)
df.show()

```

```

from pyspark.sql import SparkSession
from pyspark import *
spark = SparkSession \
    .builder \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
passenger = Row("Name", "age", "source", "destination")
s1 = passenger('David', 22, 'London', 'Paris')
s2 = passenger('Steve', 22, 'New York', 'Sydney')
x = [s1,s2]
df1=spark.createDataFrame(x)
df1.show()

```

Result of show()

Once `show()` is executed we can view the following result in the `pyspark shell`

firstName	lastName	age	telephone
David	Julian	22	100000
Mark	Webb	23	658545

Data Sources

- Spark SQL supports operating on a variety of `data sources` through the `DataFrame` interface.
- A `DataFrame` can be operated on using `relational transformations` and can also be used to create a temporary view.
- Registering a `DataFrame` as a temporary view allows you to run `SQL queries` over its data.

- This chapter describes the general methods for **loading and saving data** using the Spark Data Sources.

Generic Load/Save Functions

In most of the cases, the default data source will be used for all operations.

```
df = spark.read.load("file path")

# Spark load the data source from the defined file path

df.select("column name", "column name").write.save("file name")

# The DataFrame is saved in the defined format

# By default it is saved in the Spark Warehouse
```

File path can be from *local machine* as well as from *HDFS*.

Manually Specifying Options

You can also manually specify the **data source** that will be used along with any **extra options** that you would like to pass to the data source.

Data sources fully qualified name is used to specify them, but for built-in sources, you can also use their short names (**json, parquet, jdbc, orc, libsvm, csv, text**).

Specific File Formats

DataFrames which are loaded from any type of data can be converted to other types by using the syntax shown below.

A json file can be loaded:

```
df = spark.read.load("path of json file", format="json")
```

Apache Parquet

- Apache Parquet is a **columnar storage format** available to all projects in the **Hadoop ecosystem**, irrespective of the choice of the framework used for data processing, the model of data or programming language used.
- Spark SQL provides support for both **reading and writing Parquet files**.
- Automatic conversion to **nullable** occurs when one tries to write Parquet files, This is done due to **compatibility reasons**.

Reading A Parquet File

Here we are loading a `json` file into a dataframe.

```
df = spark.read.json("path of the file")
```

For saving the dataframe into `parquet` format.

```
df.write.parquet("parquet file name")
```

```
# Put your code here
from pyspark.sql import *
spark = SparkSession.builder.getOrCreate()
df = spark.read.json("emp.json")
df.show()
df.write.parquet("Employees")
df.createOrReplaceTempView("data")
res = spark.sql("select age,name,stream from data where stream='JAVA'")
res.show()
res.write.parquet("JavaEmployees")
```

Verifying The Result

We can verify the result by loading in `Parquet` format.

```
pf = spark.read.parquet("parquet file name")
```

Here we are reading in `Parquet` format.

To view the the `DataFrame` use `show()` method.

Why Parquet File Format ?

- Parquet stores nested data structures in a `flat columnar format`.
- On comparing with the traditional way instead of storing in `row-oriented way` in parquet is more `efficient`
- Parquet is the choice of `Big data` because it serves both needs, `efficient` and `performance` in both `storage` and `processing`.

Why Parquet File Format ?

- Parquet stores nested data structures in a `flat columnar format`.
- On comparing with the traditional way instead of storing in `row-oriented way` in parquet is more `efficient`

- Parquet is the choice of **Big data** because it serves both needs, **efficient** and **performance** in both **storage** and **processing**.

Advanced Concepts in Data Frame

In this chapter, you will learn how to perform some advanced operations on **DataFrames**.

Throughout the chapter, we will be focusing on **csv** files.

Reading Data From A CSV File

What is a CSV file?

- **CSV** is a file format which allows the user to store the data in tabular format.
- CSV stands for **comma-separated values**.
- Its data fields are most often **separated**, or **delimited**, by a **comma**.

CSV Loading

To load a **csv** data set user has to make use of **spark.read.csv** method to load it into a DataFrame.

Here we are loading a **football player dataset** using the spark **csvreader**.

```
df = spark.read.csv("path-of-file/fifa_players.csv", inferSchema = True, header = True)
```

CSV Loading

inferSchema (default false): From the data, it infers the input schema automatically.

header (default false): Using this it inherits the first line as column names.

To verify we can run **df.show(2)**.

The argument **2** will display the first two rows of the resulting DataFrame.

For every example from now onwards we will be using football player DataFrame

Schema of DataFrame

What is meant by **schema**?

It's just the **structure** of the DataFrame.

To check the schema one can make use of **printSchema** method.

It results in different `columns` in our DataFrame, along with the `datatype` and the `nullable` conditions.

How To Check The Schema

To check the `schema` of the loaded `csv` data.

```
df.printSchema()
```

Once executed we will get the following result.

```
root
| -- ID: integer (nullable = true)
| -- Name: string (nullable = true)
| -- Age: integer (nullable = true)
| -- Nationality: string (nullable = true)
| -- Overall: integer (nullable = true)
| -- Potential: integer (nullable = true)
| -- Club: string (nullable = true)
| -- Value: string (nullable = true)
| -- Wage: string (nullable = true)
| -- Special: integer (nullable = true)
```

Column Names and Count (Rows and Column)

For finding the column names, count of the number of rows and columns we can use the following methods.

For Column names

```
df.columns
['ID', 'Name', 'Age', 'Nationality', 'Overall', 'Potential', 'Club', 'Value',
'Wage', 'Special']
```

Row count

```
df.count()
17981
```

Column count

```
len(df.columns)  
10
```

Describing a Particular Column

To get the `summary` of any particular column make use of `describe` method.

This method gives us the `statistical summary` of the given `column`, if not specified, it provides the `statistical summary` of the `DataFrame`.

```
df.describe('Name').show()
```

The result we will be as shown below.

```
+-----+  
| summary |      Name |  
+-----+  
|   count |      17981 |  
|   mean  |      null |  
| stddev  |      null |  
|   min   |    A. Abbas |  
|   max   | Óscar Whalley |  
+-----+
```

Advanced Concepts in Data Frame

In this chapter, you will learn how to perform some advanced operations on `DataFrames`.

Throughout the chapter, we will be focusing on `csv` files.

Reading Data From A CSV File

What is a CSV file?

- **CSV** is a file format which allows the user to store the data in tabular format.
- CSV stands for `comma-separated values`.
- It's data fields are most often `separated`, or `delimited`, by a `comma`.

CSV Loading

To load a `csv` data set user has to make use of `spark.read.csv` method to load it into a `DataFrame`.

Here we are loading a **football player dataset** using the spark `csvreader`.

```
df = spark.read.csv("path-of-file/fifa_players.csv", inferSchema = True, header = True)
```

CSV Loading

inferSchema (default false): From the data, it infers the input schema automatically.

header (default false): Using this it inherits the first line as column names.

To verify we can run `df.show(2)`.

The argument `2` will display the first two rows of the resulting DataFrame.

For every example from now onwards we will be using football player DataFrame

Schema of DataFrame

What is meant by **schema**?

It's just the `structure` of the DataFrame.

To check the schema one can make use of `printSchema` method.

It results in different `columns` in our DataFrame, along with the `datatype` and the `nullable` conditions.

How To Check The Schema

To check the `schema` of the loaded `csv` data.

```
df.printSchema()
```

Once executed we will get the following result.

```
root
| -- ID: integer (nullable = true)
| -- Name: string (nullable = true)
| -- Age: integer (nullable = true)
| -- Nationality: string (nullable = true)
| -- Overall: integer (nullable = true)
| -- Potential: integer (nullable = true)
| -- Club: string (nullable = true)
| -- Value: string (nullable = true)
| -- Wage: string (nullable = true)
```

```
|-- Special: integer (nullable = true)
```

Column Names and Count (Rows and Column)

For finding the column names, count of the number of rows and columns we can use the following methods.

For Column names

```
df.columns  
['ID', 'Name', 'Age', 'Nationality', 'Overall', 'Potential', 'Club', 'Value',  
'Wage', 'Special']
```

Row count

```
df.count()  
17981
```

Column count

```
len(df.columns)  
10
```

Describing a Particular Column

To get the **summary** of any particular column make use of **describe** method.

This method gives us the **statistical summary** of the given **column**, if not specified, it provides the **statistical summary** of the **DataFrame**.

```
df.describe('Name').show()
```

The result we will be as shown below.

```
+-----+  
| summary |      Name |  
+-----+-----+  
|  count |      17981 |  
|  mean  |      null  |  
| stddev |      null  |  
|  min   |    A. Abbas |  
|  max   | Óscar Whalley |
```

```
+-----+-----+
```

Describing A Different Column

Now try it on some other column.

```
df.describe('Age').show()
```

Observe the various **Statistical Parameters**.

```
+-----+-----+
| summary |          Age |
+-----+-----+
|   count |      17981 |
|   mean  | 25.144541460430453 |
| stddev | 4.614272345005111 |
|   min   |      16 |
|   max   |      47 |
+-----+-----+
```

Selecting Multiple Columns

For selecting particular columns from the DataFrame, one can use the select method.

Syntax for performing selection operation is:

```
df.select('Column name 1','Column name 2',.....,'Column name n').show()
```

**show() is optional **

One can load the result into another DataFrame by simply equating.

ie

```
dfnew=df.select('Column name 1','Column name 2',.....,'Column name n')
```

Selection Operation

Selecting the column **ID** and **Name** and loading the result to a new **DataFrame**.

```
dfnew=df.select('ID','Name')
```

Verifying the result

```
dfnew.show(5)

+-----+-----+
|   ID|      Name|
+-----+-----+
| 20801|Cristiano Ronaldo|
|158023|      L. Messi|
|190871|        Neymar|
|176580|      L. Suárez|
|167495|       M. Neuer|
+-----+-----+
```

only showing top 5 rows

Filtering Data

For filtering the data `filter` command is used.

```
df.filter(df.Club=='FC Barcelona').show(3)
```

The result will be as follows:

```
+-----+-----+-----+-----+-----+-----+-----+
+-----+
|   ID|      Name|Age|Nationality|Overall|Potential|          Club| Value| Wage|
Special|
+-----+-----+-----+-----+-----+-----+-----+
+-----+
|158023|  L. Messi| 30| Argentina|     93|       93|FC Barcelona| €105M|€565K|
2154|
|176580| L. Suárez| 30| Uruguay|     92|       92|FC Barcelona| €97M|€510K|
2291|
|168651|I. Rakitić| 29| Croatia|     87|       87|FC Barcelona|€48.5M|€275K|
2129|
+-----+-----+-----+-----+-----+-----+-----+
+-----+
```

only showing top 3 rows since we had given 3 in the `show()` as the argument

Verify the same by your own

To Filter our data based on multiple conditions (AND or OR)

```
df.filter((df.Club=='FC Barcelona') & (df.Nationality=='Spain')).show(3)

+---+-----+---+-----+-----+-----+
+---+-----+
|   ID|      Name|Nationality|Overall|Potential|          Club| Value|
Wage|Special|
+---+-----+---+-----+-----+-----+
+---+-----+
|152729|    Piqué| 30|     Spain|     87| FC Barcelona|€37.5M|
€240K| 1974|
|  41|    Iniesta| 33|     Spain|     87| FC Barcelona|€29.5M|
€260K| 2073|
|189511|Sergio Busquets| 28|     Spain|     86| FC Barcelona| €36M|
€250K| 1998|
+---+-----+---+-----+-----+-----+
+---+-----+
```

only showing top 3 rows

In a similar way we can use other **logical operators**.

Sorting Data (OrderBy)

To sort the data use the **OrderBy** method.

In pyspark in default, it will sort in **ascending** order but we can change it into **descending** order as well.

```
df.filter((df.Club=='FC Barcelona') &
(df.Nationality=='Spain')).orderBy('ID',).show(5)
```

To sort in descending order:

```
df.filter((df.Club=='FC Barcelona') &
(df.Nationality=='Spain')).orderBy('ID', ascending=False).show(5)
```

Sorting

The result of the first order by operation results in the following output.

```
+---+-----+---+-----+-----+-----+-----+
+---+-----+
```

ID Wage	Special	Name	Age	Nationality	Overall	Potential	Club	Value
41 €260K	2073	Iniesta	33	Spain	87	87	FC Barcelona	€29.5M
152729 €240K	1974	Piqué	30	Spain	87	87	FC Barcelona	€37.5M
189332 €215K	2206	Jordi Alba	28	Spain	85	85	FC Barcelona	€30.5M
189511 €250K	1998	Sergio Busquets	28	Spain	86	86	FC Barcelona	€36M
199564 €140K	2071	Sergi Roberto	25	Spain	81	86	FC Barcelona	€19.5M

only showing top 5 rows

Random Data Generation

Random Data generation is useful when we want to test algorithms and to implement new ones.

In Spark under `sql.functions` we have methods to generate random data. e.g., uniform (`rand`), and standard normal (`randn`).

```
from pyspark.sql.functions import rand, randn
df = sqlContext.range(0, 7)
df.show()
```

The output will be as follows

---	+
id	

0	
1	
2	
3	

More on Random Data

By using uniform distribution and normal distribution generate two more columns.

```
df.select("id", rand(seed=10).alias("uniform"),
randn(seed=27).alias("normal")).show()

+---+-----+-----+
| id|      uniform|      normal|
+---+-----+-----+
| 0| 0.41371264720975787| 0.5888539012978773|
| 1| 0.1982919638208397| 0.06157382353970104|
| 2| 0.12030715258495939| 1.0854146699817222|
| 3| 0.44292918521277047| -0.4798519469521663|
+---+-----+-----+
```

Summary and Descriptive Statistics

The `first operation` to perform after importing data is to get some `sense` of what it looks like.

The function `describe` returns a DataFrame containing information such as number of non-null entries (count), `mean`, `standard deviation`, and `minimum` and `maximum` value for each numerical column.

Summary and Descriptive Statistics

```
df.describe('uniform', 'normal').show()
```

The result is as follows:

```
+-----+-----+-----+
| summary|      uniform|      normal|
+-----+-----+-----+
|  count|          10|          10|
|  mean| 0.3841685645682706|-0.15825812884638607|
| stddev| 0.31309395532409323| 0.963345903544872|
|  min| 0.03650707717266999|-2.1591956435415334|
|  max| 0.8898784253886249| 1.0854146699817222|
+-----+-----+-----+
```

Descriptive Statistics

For a quick review of a column `describe` works fine.

In the same way, we can also make use of some standard **statistical functions** also.

```
from pyspark.sql.functions import mean, min, max
df.select([mean('uniform'), min('uniform'), max('uniform')]).show()
+-----+-----+-----+
| avg(uniform) | min(uniform) | max(uniform) |
+-----+-----+-----+
| 0.3841685645682706 | 0.03650707717266999 | 0.8898784253886249 |
+-----+-----+-----+
```

Sample Co-Variance and Correlation

In statistics **Co-Variance** means how one **random variable** changes with respect to other.

Positive value indicates a trend in increase when the other increases.

Negative value indicates a trend in decrease when the other increases.

The sample co-variance of two columns of a DataFrame can be calculated as follows:

More On Co-Variance

```
from pyspark.sql.functions import rand
df = sqlContext.range(0, 10).withColumn('rand1',
rand(seed=10)).withColumn('rand2', rand(seed=27))
df.stat.cov('rand1', 'rand2')
0.031109767020625314
```

From the above we can infer that **co-variance** of two random columns is near to **zero**.

Correlation

Correlation provides the statistical dependence of two random variables.

```
df.stat.corr('rand1', 'rand2')
0.30842745432650953
```

Two randomly generated columns have **low correlation value**.

Cross Tabulation (Contingency Table)

Cross Tabulation provides a **frequency distribution table** for a given set of variables.

One of the powerful tool in statistics to observe the statistical **independence** of variables.

Consider an example

```
# Create a DataFrame with two columns (name, item)
names = ["Alice", "Bob", "Mike"]
items = ["milk", "bread", "butter", "apples", "oranges"]
df = sqlContext.createDataFrame([(names[i % 3], items[i % 5]) for i in range(100)], ["name", "item"])
```

Contingency Table)

For applying the **cross tabulation** we can make use of the crosstab method.

```
df.stat.crosstab("name", "item").show()
+-----+-----+-----+-----+
|name_item|apples|bread|butter|milk|oranges|
+-----+-----+-----+-----+
|      Bob|      6|     7|      7|     6|      7|
|    Mike|      7|     6|      7|     7|      6|
|   Alice|      7|     7|      6|     7|      7|
+-----+-----+-----+-----+
```

Cardinality of columns we run **crosstab** on cannot be too big.

```
# Put your code here
from pyspark.sql import *
from pyspark import SparkContext
from pyspark.sql.functions import rand, randn
from pyspark.sql import SQLContext
from pyspark.sql.types import FloatType
spark = SparkSession.builder.getOrCreate()
df1 = Row("1","2")
sqlContext = SQLContext(spark)
df = sqlContext.range(0, 10).withColumn('rand1', rand(seed=10)).withColumn('rand2', rand(seed=27))
a = df.stat.cov('rand1', 'rand2')
b = df.stat.corr('rand1', 'rand2')

s1 = df1("Co-variance",a)
s2 = df1("Correlation",b)
x=[s1,s2]
```

```
df2 = spark.createDataFrame(x)
df2.show()
df2.write.parquet("Result")
```

What is Spark SQL ?

Spark SQL brings native support for `SQL` to Spark.

Spark SQL blurs the lines between `RDD's` and `relational tables`.

By integrating these powerful features Spark makes it easy for developers to use SQL commands for querying external data with complex analytics, all within in a single application.

Performing SQL Queries

We can also pass `SQL queries` directly to any DataFrame.

For that, we need to create a table from the DataFrame using the `registerTempTable` method.

After that use `sqlContext.sql()` to pass the SQL queries.

Apache Hive



The `Apache Hive` data warehouse software allows `reading`, `writing`, and managing `large datasets` residing in distributed storage and queried using `SQL syntax`.

Features of Apache Hive

Apache Hive is built on top of Apache Hadoop.

The below mentioned are the features of Apache Hive.

- Apache Hive is having tools to allow easy and quick access to data using SQL, thus enables data warehousing tasks such like extract/transform/load (**ETL**), reporting, and data analysis.
- Mechanisms for imposing structure on a variety of **data formats**.
- Access to files stored either directly in **Apache HDFS** or in other data storage systems such as **Apache HBase**.

Features Of Apache Hive

- Query execution via **Apache Tez**, **Apache Spark**, or **MapReduce**.
- A procedural language with **HPL-SQL**.
- Sub-second query retrieval via **Hive LLAP**, **Apache YARN** and **Apache Slider**.

What Hive Provides ?

Apache Hive provides the standard SQL functionalities, which includes many of the later **SQL:2003** and **SQL:2011** features for analytics.

We can extend Hive's SQL with the user code by using user-defined functions (**UDFs**), user-defined aggregates (**UDAFs**), and user-defined table functions (**UDTFs**).

Hive comes with built-in connectors for **comma and tab-separated values (CSV/TSV) text files**, **Apache Parquet**, **Apache ORC**, and other formats.

Spark SQL supports **reading** and **writing** data stored in **Hive**. Connecting Hive From Spark

When working with Hive, one must instantiate **SparkSession** with **Hive support**, including connectivity to a persistent **Hive metastore**, support for **Hive serdes**, and **Hive user-defined functions**.

How To Enable Hive Support

```
from os.path import expanduser, join, abspath

from pyspark.sql import SparkSession
from pyspark.sql import Row
```

```
#warehouse_location points to the default location for managed databases and
#tables

warehouse_location = abspath('spark-warehouse')

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL Hive integration example") \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .enableHiveSupport() \
    .getOrCreate()
```

Creating Hive Table From Spark

We can easily create a table in `hive warehouse` programmatically from Spark.

The syntax for creating a table is as follows:

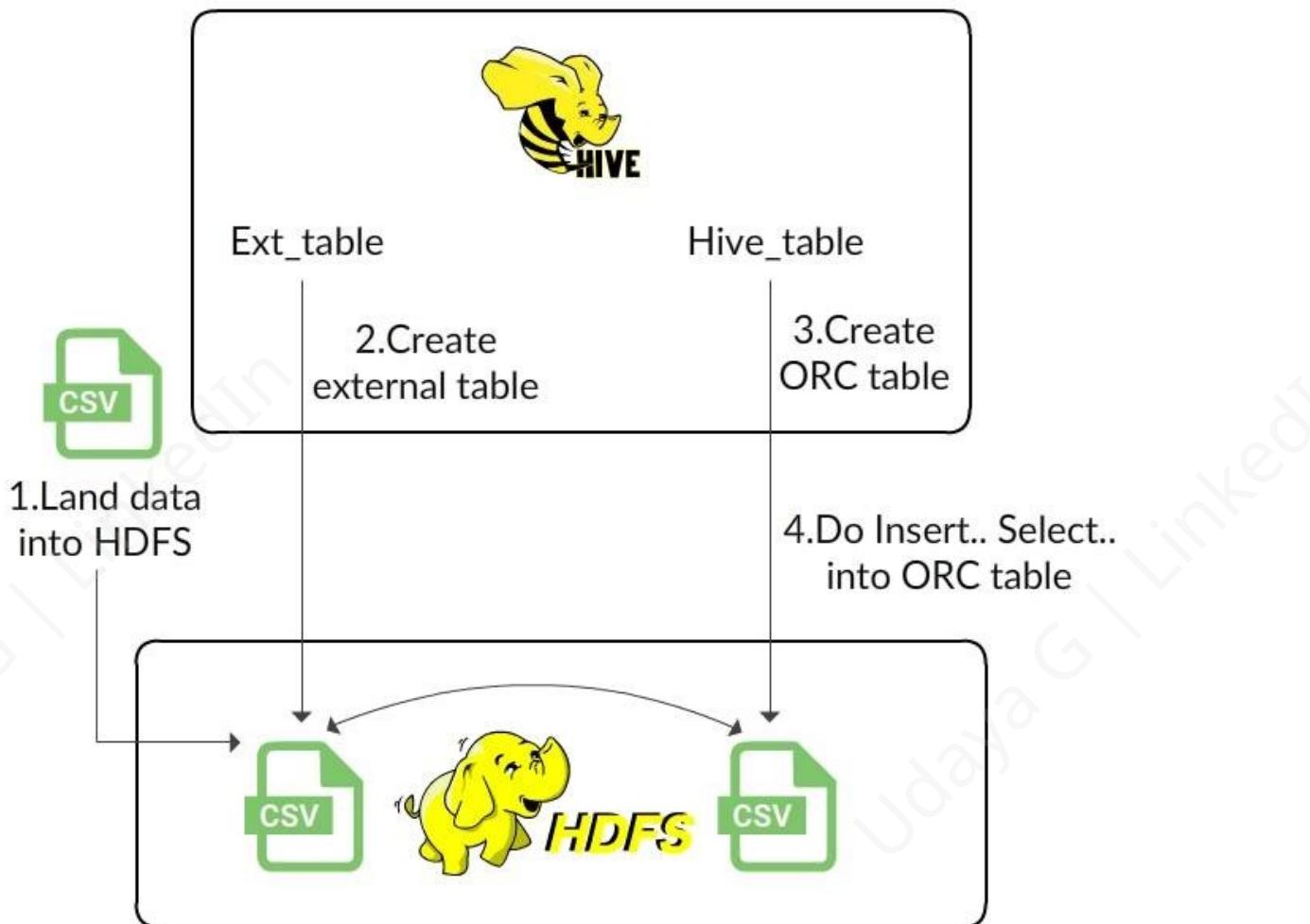
```
spark.sql("CREATE TABLE IF NOT EXISTS table_name(column_name_1
Data Type, column_name_2 Data Type,.....,column_name_n Data Type) USING hive")
```

To load a DataFrame into a table.

```
df.write.insertInto("table name", overwrite = True)
```

Now verify the result by using the select statement.

Hive External Table



- External tables are used to store data **outside** the hive.
- Data needs to remain in the **underlying location** even after the user drop the table.

Handling External Hive Tables From Apache Spark

First, create an **external table** in the hive by specifying the location.

One can create an external table in the hive by running the following query on **hive shell**.

```
hive> create external table table_name(column_name1 DataType,column_name2
(DataType,.....,column_name_n DataType) STORED AS Parquet location ' path of
external table';
```

- The table is created in **Parquet schema**.
- The table is saved in the **hdfs** directory.

Loading Data From Spark To The Hive Table

We can load data to `hive table` from the `DataFrame`.

For doing the same `schema` of both `hive table` and the `DataFrame` should be equal.

Let us take a sample CSV file.

We can read the `csv` the file by making use of `spark csv reader`.

```
df = spark.read.csv("path-of-file", inferSchema = True, header = True)
```

The `schema` of the DataFrame will be same as the schema of the CSV file itself.

Data Loading To External Table

For loading the data we have to save the dataframe in external hive table location.

```
df.write.mode('overwrite').format("format").save("location")
```

Since our hive external table is in `parquet format` in place of format we have to mention 'parquet'.

The location should be `same` as the hive external table location in hdfs directory.

If the `schema` is matching then data will load `automatically` to the hive table.

By querying the hive table we can verify it.

What is HBase ?



HBase is a `distributed` column-oriented data store built on top of `HDFS`.

HBase is an Apache open source project whose goal is to provide **storage** for the Hadoop Distributed Computing.

Data is logically organized into **tables**, **rows** and **columns**.

More On HBase

- HBase features compression, in-memory operation, and Bloom filters on a per-column basis as outlined in the original **Bigtable** paper.
- Tables in HBase can serve as the **input** and **output** for Map Reduce jobs run in Hadoop, and may be accessed through the **Java API** but also through **REST**, **Avro** or **Thrift gateway APIs**.
- It is a column-oriented key-value data store and has been idolized widely because of its **lineage** with **Hadoop** and **HDFS**.
- HBase runs on **top** of HDFS and is well-suited for **faster** read and write operations on large datasets with high **throughput** and low **input/output latency**.

How To Connect Spark and HBase

- To connect we require **hdfs**, **Spark** and **HBase** installed in the local machine.
- Make sure that your **versions** are **matching** with each other.
- Copy all the **HBase jar** files to the **Spark lib** folder.
- Once done set the **SPARK_CLASSPATH** in **spark-env.sh** with **lib**.

Building A Real-Time Data Pipeline

API Service

Real Time Pipeline using HDFS,Spark and HBase

Various Stages

It has **4 main stages** which includes:

- Transformation
- Cleaning
- Validation
- Writing of the data received from the various sources

Transformation And Cleaning

Data Transformation

- This is an entry point for the **streaming application**.
- Here the operations related to **normalization** of data are performed.
- Transformation of data can be performed by using built-in functions like **map**, **filter**, **foreachRDD** etc.

Data Cleaning

- During **preprocessing** cleaning is very important.
- In this stage, we can use **custom built libraries** for cleaning the data.

Validation And Writing

Data Validation

In this stage, we can **validate** the data with respect to some **standard validations** such as length, patterns and so on.

Writing

At last, data passed from the **previous three stages** is passed on to the **writing application** which simply writes this final set of data to **HBase** for further data analysis.

Spark In Real World

Uber – the online taxi company is an apt example for Spark. They are gathering terabytes of event data from its various users.

- Uses **Kafka, Spark Streaming, and HDFS**, to build a continuous **ETL pipeline**.
- Convert raw unstructured data into structured data as it is collected.
- Uses it further complex analytics and optimization of operations.

Spark In Real World

Pinterest – Uses a Spark ETL pipeline

- Leverages Spark Streaming to gain **immediate insight** into how users all over the world are **engaging** with **Pins** in real time.
- Can make more relevant recommendations as people navigate the site.
- Recommends related Pins.
- Determine which products to **buy**, or **destinations** to visit.

Spark In Real World

Conviva – 4 million video feeds per month.

- Conviva is using Spark for reducing the customer churn by **managing live video traffic** and **optimizing video streams**.
- They maintain a consistently smooth high-quality viewing experience.

Spark In Real World

Capital One – makes use of Spark and data science algorithms for a better understanding of its customers.

- Developing the next generation of **financial products and services**.
- Find attributes and patterns of increased **probability for fraud**.

Netflix – Movie recommendation engine from user data.

- User data is also used for **content creation**

```
# Put your code here
from pyspark.sql import *
spark = SparkSession.builder.getOrCreate()
df = Row("ID", "Name", "Age", "AreaofInterest")
s1 = df("1", "Jack", 22, "Data Science")
s2 = df("2", "Leo", 21, "Data Analytics")
s3 = df("3", "Luke", 24, "Micro Services")
s4 = df("4", "Mark", 21, "Data Analytics")
x = [s1,s2,s3,s4]
df1 = spark.createDataFrame(x)
df3 = df1.describe("Age")
df3.show()
df3.write.parquet("Age")
df1.createOrReplaceTempView("data")
df4 = spark.sql("select ID,Name,Age from data order by ID desc")
df4.show()
df4.write.parquet("NameSorted")
```

Data Science Interview Questions

Statistics:

1. What is the Central Limit Theorem and why is it important?

"Suppose that we are interested in estimating the average height among all people. Collecting data for every person in the world is impossible. While we can't obtain a height measurement from everyone in the population, we can still sample some people. The question now becomes, what can we say about the average height of the entire population given a single sample. The Central Limit Theorem addresses this question exactly." *Read more here.*

2. What is sampling? How many sampling methods do you know?

"Data sampling is a statistical analysis technique used to select, manipulate and analyze a representative subset of data points to identify patterns and trends in the larger data set being examined." *Read the full answer here.*

3. What is the difference between type I vs type II error?

"A type I error occurs when the null hypothesis is true, but is rejected. A type II error occurs when the null hypothesis is false, but erroneously fails to be rejected." *Read the full answer here.*

4. What is linear regression? What do the terms p-value, coefficient, and r-squared value mean? What is the significance of each of these components?

A linear regression is a good tool for quick predictive analysis: for example, the price of a house depends on a myriad of factors, such as its size or its location. In order to see the relationship between these variables, we need to build a linear regression, which predicts the line of best fit between them and can help conclude whether or not these two factors have a positive or negative relationship. *Read more here and here.*

5. What are the assumptions required for linear regression?

There are four major assumptions: 1. There is a linear relationship between the dependent variables and the regressors, meaning the model you are creating actually fits the data, 2. The errors or residuals of the data are normally distributed and independent from each other, 3. There is minimal multicollinearity between explanatory variables, and 4. Homoscedasticity. This means the variance around the regression line is the same for all values of the predictor variable.

6. What is a statistical interaction?

"Basically, an interaction is when the effect of one factor (input variable) on the dependent variable (output variable) differs among levels of another factor." *Read more here.*

7. What is selection bias?

"Selection (or 'sampling') bias occurs in an 'active,' sense when the sample data that is gathered and prepared for modeling has characteristics that are not representative of the true, future population of cases

the model will see. That is, active selection bias occurs when a subset of the data are systematically (i.e., non-randomly) excluded from analysis.” *Read more* here.

8. What is an example of a data set with a non-Gaussian distribution?

“The Gaussian distribution is part of the Exponential family of distributions, but there are a lot more of them, with the same sort of ease of use, in many cases, and if the person doing the machine learning has a solid grounding in statistics, they can be utilized where appropriate.” *Read more* here.

9. What is the Binomial Probability Formula?

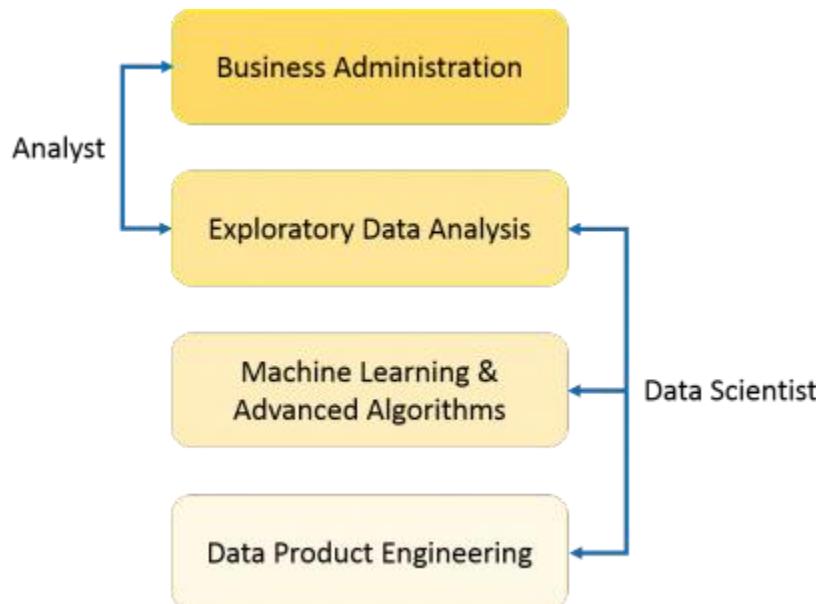
“The binomial distribution consists of the probabilities of each of the possible numbers of successes on N trials for independent events that each have a probability of π (the Greek letter pi) of occurring.” *Read more*

Data Science :

Q1. What is Data Science? List the differences between supervised and unsupervised learning.

Data Science is a blend of various tools, algorithms, and machine learning principles with the goal to discover hidden patterns from the raw data. How is this different from what statisticians have been doing for years?

The answer lies in the difference between explaining and predicting.



The differences between supervised and unsupervised learning are as follows;

Supervised Learning	Unsupervised Learning
Input data is labelled.	Input data is unlabelled.
Uses a training data set.	Uses the input data set.
Used for prediction.	Used for analysis.
Enables classification and regression.	Enables Classification, Density Estimation, & Dimension Reduction

Q2. What is Selection Bias?

Selection bias is a kind of error that occurs when the researcher decides who is going to be studied. It is usually associated with research where the selection of participants isn't random. It is sometimes referred to

as the selection effect. It is the distortion of statistical analysis, resulting from the method of collecting samples. If the selection bias is not taken into account, then some conclusions of the study may not be accurate.

The types of selection bias include:

1. **Sampling bias:** It is a systematic error due to a non-random sample of a population causing some members of the population to be less likely to be included than others resulting in a biased sample.
2. **Time interval:** A trial may be terminated early at an extreme value (often for ethical reasons), but the extreme value is likely to be reached by the variable with the largest variance, even if all variables have a similar mean.
3. **Data:** When specific subsets of data are chosen to support a conclusion or rejection of bad data on arbitrary grounds, instead of according to previously stated or generally agreed criteria.
4. **Attrition:** Attrition bias is a kind of selection bias caused by attrition (loss of participants) discounting trial subjects/tests that did not run to completion.

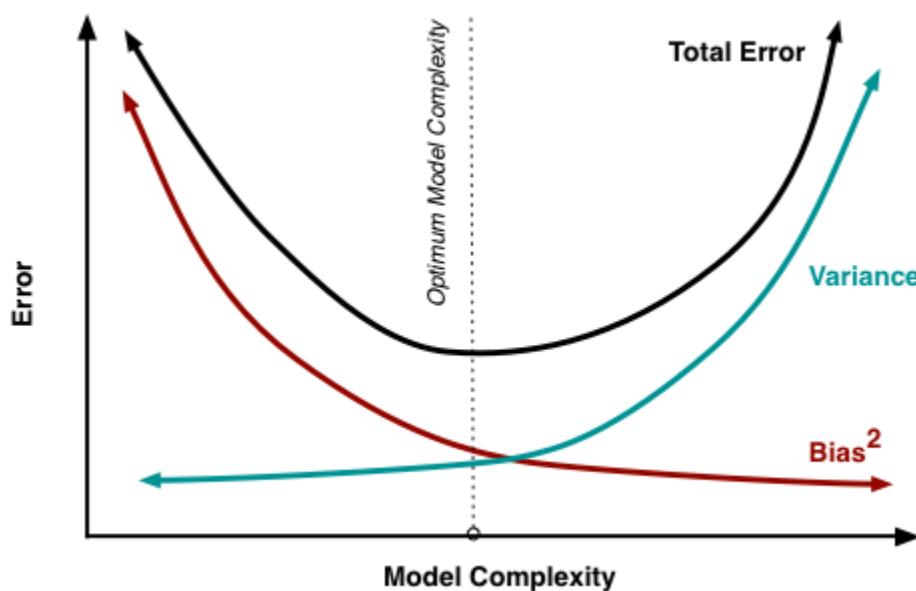
Q3. What is bias-variance trade-off?

Bias: Bias is an error introduced in your model due to oversimplification of the machine learning algorithm. It can lead to underfitting. When you train your model at that time model makes simplified assumptions to make the target function easier to understand.

Low bias machine learning algorithms — Decision Trees, k-NN and SVM High bias machine learning algorithms — Linear Regression, Logistic Regression

Variance: Variance is error introduced in your model due to complex machine learning algorithm, your model learns noise also from the training data set and performs badly on test data set. It can lead to high sensitivity and overfitting.

Normally, as you increase the complexity of your model, you will see a reduction in error due to lower bias in the model. However, this only happens until a particular point. As you continue to make your model more complex, you end up over-fitting your model and hence your model will start suffering from high variance.



Bias-Variance trade-off: The goal of any supervised machine learning algorithm is to have low bias and low variance to achieve good prediction performance.

1. The k-nearest neighbour algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbours that contribute to the prediction and in turn increases the bias of the model.
2. The support vector machine algorithm has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

There is no escaping the relationship between bias and variance in machine learning. Increasing the bias will decrease the variance. Increasing the variance will decrease bias.

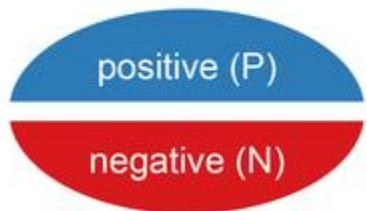
Q4. What is a confusion matrix?

The confusion matrix is a 2X2 table that contains 4 outputs provided by the **binary classifier**. Various measures, such as error-rate, accuracy, specificity, sensitivity, precision and recall are derived from it. *Confusion Matrix*

		Predicted class	
		P	N
Actual Class		P	True Positives (TP)
		N	False Negatives (FN)
P	N	P	False Positives (FP)
		N	True Negatives (TN)

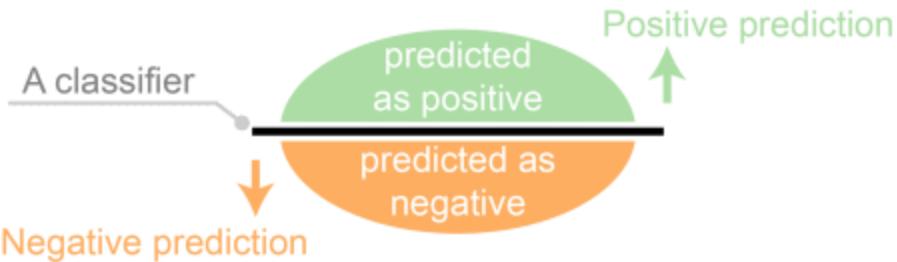
A data set used for performance evaluation is called a **test data set**. It should contain the correct labels and predicted labels.

Two actual classes or observed labels



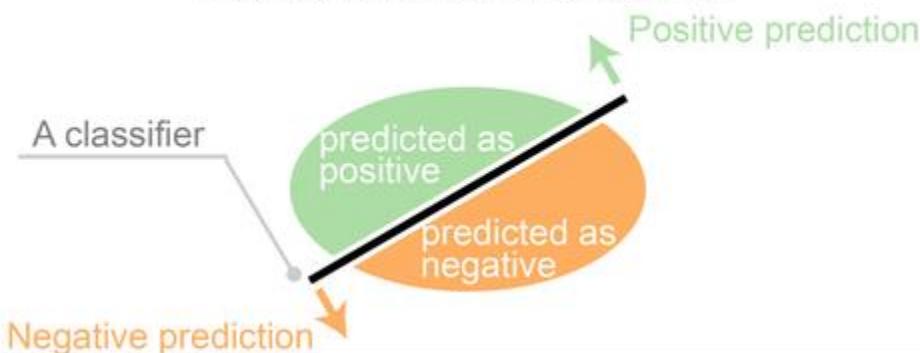
The predicted labels will exactly the same if the performance of a binary classifier is perfect.

Predicted classes of a perfect classifier



The predicted labels usually match with part of the observed labels in real-world scenarios.

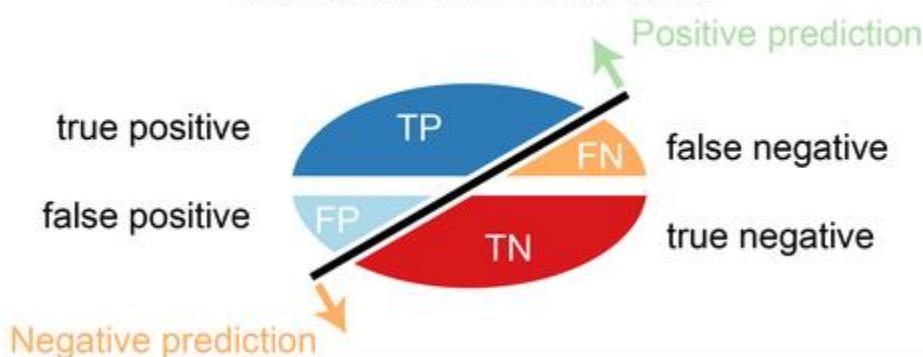
Predicted classes of a classifier



A binary classifier predicts all data instances of a test data set as either positive or negative. This produces four outcomes-

1. True-positive(TP) — Correct positive prediction
2. False-positive(FP) — Incorrect positive prediction
3. True-negative(TN) — Correct negative prediction
4. False-negative(FN) — Incorrect negative prediction

Four outcomes of a classifier



Basic measures derived from the confusion matrix

1. Error Rate = $(FP+FN)/(P+N)$
2. Accuracy = $(TP+TN)/(P+N)$
3. Sensitivity(Recall or True positive rate) = TP/P
4. Specificity(True negative rate) = TN/N
5. Precision(Positive predicted value) = $TP/(TP+FP)$
6. F-Score(Harmonic mean of precision and recall) = $(1+b)(PREC.REC)/(b^2PREC+REC)$ where b is commonly 0.5, 1, 2.

STATISTICS INTERVIEW QUESTIONS

Q5. What is the difference between “long” and “wide” format data?

In the **wide-format**, a subject's repeated responses will be in a single row, and each response is in a separate column. In the **long-format**, each row is a one-time point per subject. You can recognize data in wide format by the fact that columns generally represent groups.

Name	Height	Weight
John	160	67
Christopher	182	78

Name	Attribute	Value
John	Height	160
John	Weight	67
Christopher	Height	182
Christopher	Weight	78

Figure: Wide Format

Figure: Long Format

Q6. What do you understand by the term Normal Distribution?

Data is usually distributed in different ways with a bias to the left or to the right or it can all be jumbled up.

However, there are chances that data is distributed around a central value without any bias to the left or right and reaches normal distribution in the form of a bell-shaped curve.

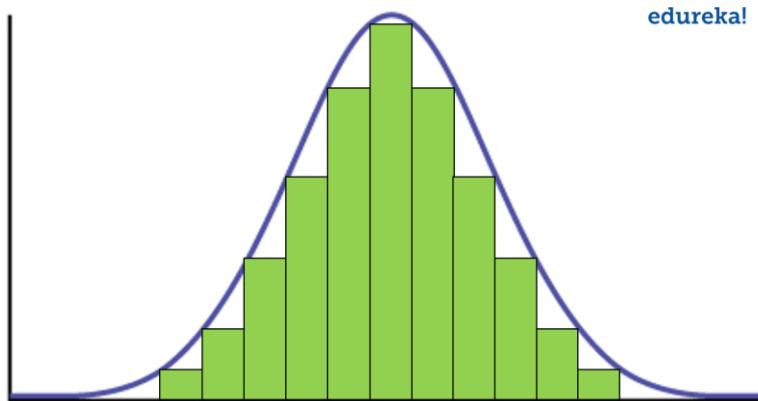


Figure: Normal distribution in a bell curve

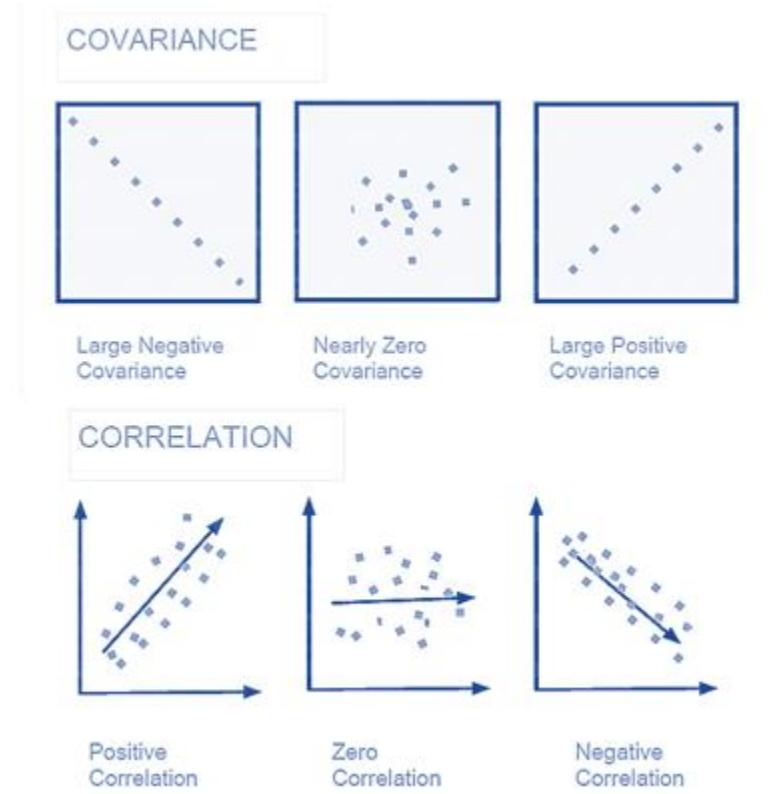
The random variables are distributed in the form of a symmetrical, bell-shaped curve.

Properties of Normal Distribution are as follows;

1. Unimodal -one mode
2. Symmetrical -left and right halves are mirror images
3. Bell-shaped -maximum height (mode) at the mean
4. Mean, Mode, and Median are all located in the center
5. Asymptotic

Q7. What is correlation and covariance in statistics?

Covariance and Correlation are two mathematical concepts; these two approaches are widely used in statistics. Both Correlation and Covariance establish the relationship and also measure the dependency between two random variables. Though the work is similar between these two in mathematical terms, they are different from each other.



Correlation:

Correlation is considered or described as the best technique for measuring and also for estimating the quantitative relationship between two variables. Correlation measures how strongly two variables are related.

Covariance: In covariance two items vary together and it's a measure that indicates the extent to which two random variables change in cycle. It is a statistical term; it explains the systematic relation between a pair of random variables, wherein changes in one variable reciprocal by a corresponding change in another variable.

Q8. What is the difference between Point Estimates and Confidence Interval?

Point Estimation gives us a particular value as an estimate of a population parameter. Method of Moments and Maximum Likelihood estimator methods are used to derive Point Estimators for population parameters.

A confidence interval gives us a range of values which is likely to contain the population parameter. The confidence interval is generally preferred, as it tells us how likely this interval is to contain the population parameter. This likeliness or probability is called Confidence Level or Confidence coefficient and represented by $1 - \alpha$, where α is the level of significance.

Q9. What is the goal of A/B Testing?

It is a hypothesis testing for a randomized experiment with two variables A and B.

The goal of A/B Testing is to identify any changes to the web page to maximize or increase the outcome of interest. A/B testing is a fantastic method for figuring out the best online promotional and marketing strategies for your business. It can be used to test everything from website copy to sales emails to search ads

An example of this could be identifying the click-through rate for a banner ad.

Q10. What is p-value?

When you perform a hypothesis test in statistics, a p-value can help you determine the strength of your results. p-value is a number between 0 and 1. Based on the value it will denote the strength of the results. The claim which is on trial is called the Null Hypothesis.

Low p-value (≤ 0.05) indicates strength against the null hypothesis which means we can reject the null Hypothesis. High p-value (≥ 0.05) indicates strength for the null hypothesis which means we can accept the null Hypothesis p-value of 0.05 indicates the Hypothesis could go either way. To put it in another way,

High P values: your data are likely with a true null. Low P values: your data are unlikely with a true null.

Q11. In any 15-minute interval, there is a 20% probability that you will see at least one shooting star. What is the probability that you see at least one shooting star in the period of an hour?

Probability of not seeing any shooting star in 15 minutes is

$$\begin{aligned} &= \frac{1}{P(\text{Seeing})} \\ &= 1 - 0.2 = 0.8 \end{aligned}$$

Probability of not seeing any shooting star in the period of one hour

$$= (0.8)^4 = 0.4096$$

Probability of seeing at least one shooting star in the one hour

$$\begin{aligned} &= \frac{1}{P(\text{Not seeing})} \\ &= 1 - 0.4096 = 0.5904 \end{aligned}$$

Q12. How can you generate a random number between 1 – 7 with only a die?

- Any die has six sides from 1-6. There is no way to get seven equal outcomes from a single rolling of a die. If we roll the die twice and consider the event of two rolls, we now have 36 different outcomes.
- To get our 7 equal outcomes we have to reduce this 36 to a number divisible by 7. We can thus consider only 35 outcomes and exclude the other one.
- A simple scenario can be to exclude the combination (6,6), i.e., to roll the die again if 6 appears twice.
- All the remaining combinations from (1,1) till (6,5) can be divided into 7 parts of 5 each. This way all the seven sets of outcomes are equally likely.

Q13. A certain couple tells you that they have two children, at least one of which is a girl. What is the probability that they have two girls?

In the case of two children, there are 4 equally likely possibilities

BB, BG, GB and GG;

where **B** = Boy and **G** = Girl and the first letter denotes the first child.

From the question, we can exclude the first case of BB. Thus from the remaining 3 possibilities of **BG, GB & BB**, we have to find the probability of the case with two girls.

Thus, $P(\text{Having two girls given one girl}) = 1 / 3$

Q14. A jar has 1000 coins, of which 999 are fair and 1 is double headed. Pick a coin at random, and toss it 10 times. Given that you see 10 heads, what is the probability that the next toss of that coin is also a head?

There are two ways of choosing the coin. One is to pick a fair coin and the other is to pick the one with two heads.

$$\begin{aligned} \text{Probability of selecting fair coin} &= 999/1000 = 0.999 \\ \text{Probability of selecting unfair coin} &= 1/1000 = 0.001 \end{aligned}$$

Selecting 10 heads in a row = Selecting fair coin * Getting 10 heads + Selecting an unfair coin

$$\begin{aligned} P(A) &= 0.999 * (1/2)^5 = 0.999 * (1/1024) = 0.000976 \\ P(B) &= 0.001 * 1 = 0.001 \\ P(A / A + B) &= 0.000976 / (0.000976 + 0.001) = 0.4939 \\ P(B / A + B) &= 0.001 / 0.001976 = 0.5061 \end{aligned}$$

$$\text{Probability of selecting another head} = P(A/A+B) * 0.5 + P(B/A+B) * 1 = 0.4939 * 0.5 + 0.5061 = 0.7531$$

Q15. What do you understand by statistical power of sensitivity and how do you calculate it?

Sensitivity is commonly used to validate the accuracy of a classifier (Logistic, SVM, Random Forest etc.).

Sensitivity is nothing but “Predicted True events/ Total events”. True events here are the events which were true and model also predicted them as true.

Calculation of seasonality is pretty straightforward.

$$\text{Seasonality} = (\text{True Positives}) / (\text{Positives in Actual Dependent Variable})$$

Q16. Why Is Re-sampling Done?

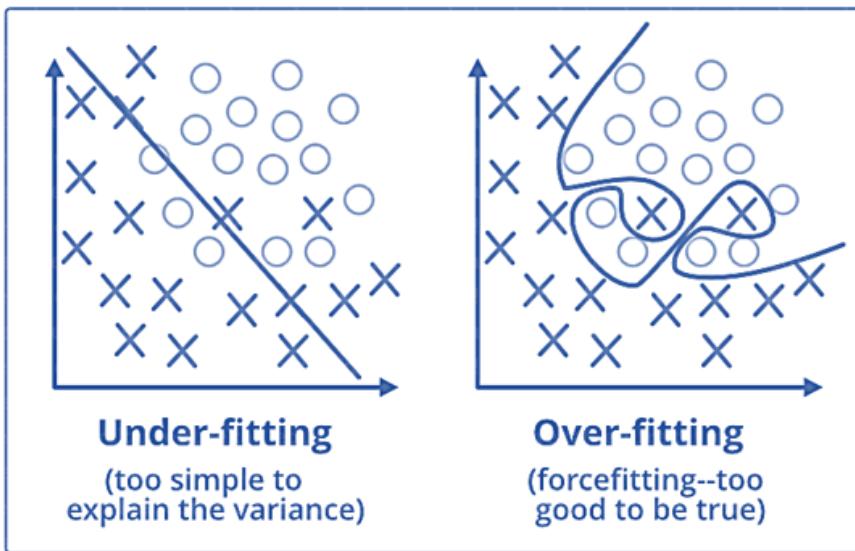
Resampling is done in any of these cases:

- Estimating the accuracy of sample statistics by using subsets of accessible data or drawing randomly with replacement from a set of data points
- Substituting labels on data points when performing significance tests
- Validating models by using random subsets (bootstrapping, cross-validation)

Q17. What are the differences between over-fitting and under-fitting?

In statistics and machine learning, one of the most common tasks is to fit a *model* to a set of training data, so as to be able to make reliable predictions on general untrained data.

Follow Steve Nouri for more AI and Data science posts: <https://lnkd.in/gZu463X>



In **overfitting**, a statistical model describes random error or noise instead of the underlying relationship. Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been overfitted, has poor predictive performance, as it overreacts to minor fluctuations in the training data.

Underfitting occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data. Underfitting would occur, for example, when fitting a linear model to non-linear data. Such a model too would have poor predictive performance.

Q18. How to combat Overfitting and Underfitting?

To combat overfitting and underfitting, you can resample the data to estimate the model accuracy (k-fold cross-validation) and by having a validation dataset to evaluate the model.

Q19. What is regularisation? Why is it useful?



Regularisation is the process of adding tuning parameter to a model to induce smoothness in order to prevent overfitting. This is most often done by adding a constant multiple to an existing weight vector. This constant is often the L1(Lasso) or L2(ridge). The model predictions should then minimize the loss function calculated on the regularized training set.

Q20. What Is the Law of Large Numbers?

It is a theorem that describes the result of performing the same experiment a large number of times. This theorem forms the basis of **frequency-style** thinking. It says that the sample means, the sample variance and the sample standard deviation converge to what they are trying to estimate.

Q21. What Are Confounding Variables?

In statistics, a confounder is a variable that influences both the dependent variable and independent variable.

For example, if you are researching whether a lack of exercise leads to weight gain,

lack of exercise = independent variable

weight gain = dependent variable.

A confounding variable here would be any other variable that affects both of these variables, such as the **age of the subject**.

Q22. What Are the Types of Biases That Can Occur During Sampling?

- Selection bias
- Under coverage bias
- Survivorship bias

Q23. What is Survivorship Bias?

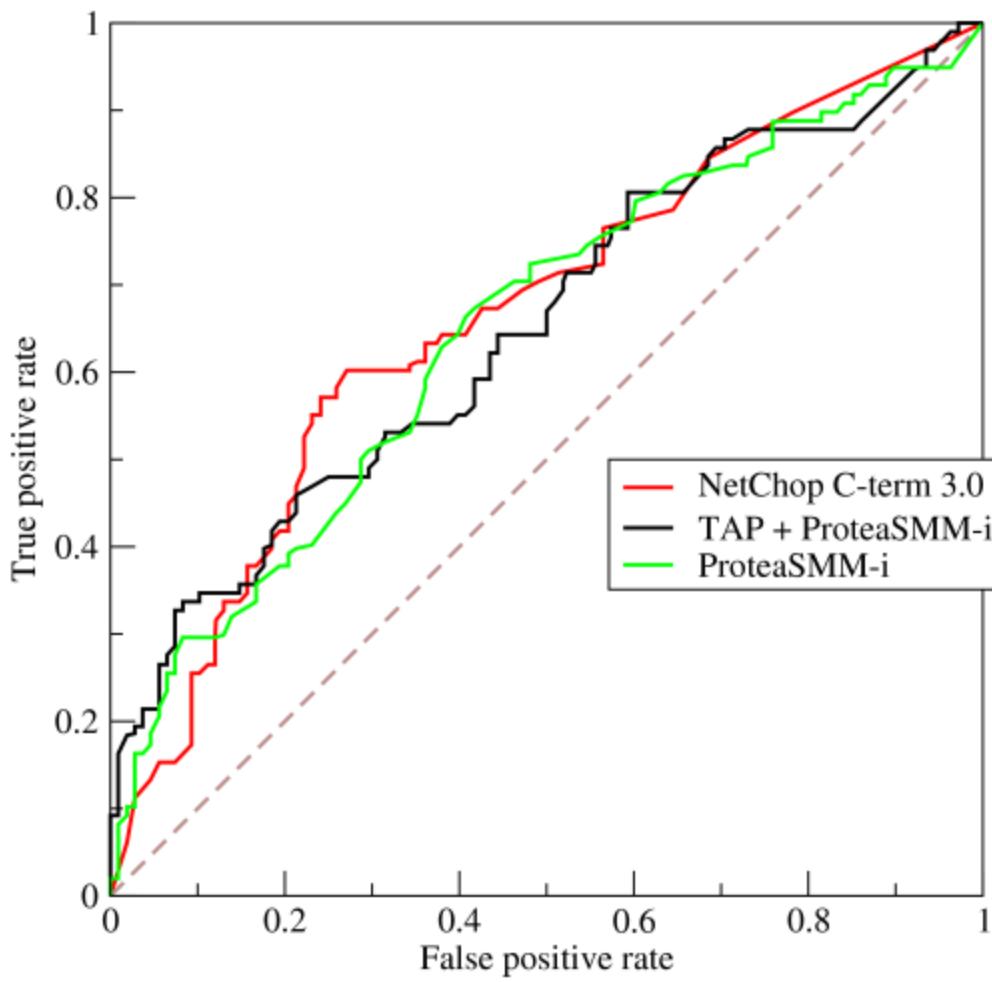
It is the logical error of focusing aspects that support surviving some process and casually overlooking those that did not work because of their lack of prominence. This can lead to wrong conclusions in numerous different means.

Q24. What is selection Bias?

Selection bias occurs when the sample obtained is not representative of the population intended to be analysed.

Q25. Explain how a ROC curve works?

The **ROC** curve is a graphical representation of the contrast between true positive rates and false-positive rates at various thresholds. It is often used as a proxy for the trade-off between the sensitivity(true positive rate) and false-positive rate.



Q26. What is TF/IDF vectorization?

TF-IDF is short for term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining.

The TF-IDF value increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

Q27. Why we generally use Softmax non-linearity function as last operation in-network?

It is because it takes in a vector of real numbers and returns a probability distribution. Its definition is as follows. Let x be a vector of real numbers (positive, negative, whatever, there are no constraints).

Then the i 'th component of $\text{Softmax}(x)$ is —

$$P(y=j | \theta^{(i)}) = \frac{e^{\theta_j^{(i)}}}{\sum_{j=0}^k e^{\theta_j^{(i)}}}$$

Softmax function

where $\theta = w_0x_0 + w_1x_1 + \dots + w_kx_k = \sum_{i=0}^k w_i x_i = w^T x$

It should be clear that the output is a probability distribution: each element is non-negative and the sum over all components is 1.

DATA ANALYSIS INTERVIEW QUESTIONS

Q28. Python or R – Which one would you prefer for text analytics?

We will prefer Python because of the following reasons:

- **Python** would be the best option because it has Pandas library that provides easy to use data structures and high-performance data analysis tools.
- **R** is more suitable for machine learning than just text analysis.
- Python performs faster for all types of text analytics.

Q29. How does data cleaning plays a vital role in the analysis?

Data cleaning can help in analysis because:

- Cleaning data from multiple sources helps to transform it into a format that data analysts or data scientists can work with.
- Data Cleaning helps to increase the accuracy of the model in machine learning.
- It is a cumbersome process because as the number of data sources increases, the time taken to clean the data increases exponentially due to the number of sources and the volume of data generated by these sources.
- It might take up to 80% of the time for just cleaning data making it a critical part of the analysis task.

Q30. Differentiate between univariate, bivariate and multivariate analysis.

Univariate analyses are descriptive statistical analysis techniques which can be differentiated based on the number of variables involved at a given point of time. For example, the pie charts of sales based on territory involve only one variable and can the analysis can be referred to as univariate analysis.

The **bivariate analysis** attempts to understand the difference between two variables at a time as in a scatterplot. For example, analyzing the volume of sale and spending can be considered as an example of bivariate analysis.

Multivariate analysis deals with the study of more than two variables to understand the effect of variables on the responses.

Q31. Explain Star Schema.

It is a traditional database schema with a central table. Satellite tables map IDs to physical names or descriptions and can be connected to the central fact table using the ID fields; these tables are known as lookup tables and are principally useful in real-time applications, as they save a lot of memory. Sometimes star schemas involve several layers of summarization to recover information faster.

Q32. What is Cluster Sampling?

Cluster sampling is a technique used when it becomes difficult to study the target population spread across a wide area and simple random sampling cannot be applied. Cluster Sample is a probability sample where each sampling unit is a collection or cluster of elements.

For eg., A researcher wants to survey the academic performance of high school students in Japan. He can divide the entire population of Japan into different clusters (cities). Then the researcher selects a number of clusters depending on his research through simple or systematic random sampling.

Let's continue our Data Science Interview Questions blog with some more statistics questions.

Q33. What is Systematic Sampling?

Systematic sampling is a statistical technique where elements are selected from an ordered sampling frame. In systematic sampling, the list is progressed in a circular manner so once you reach the end of the list, it is progressed from the top again. The best example of systematic sampling is equal probability method.

Q34. What are Eigenvectors and Eigenvalues?

Eigenvectors are used for understanding linear transformations. In data analysis, we usually calculate the eigenvectors for a correlation or covariance matrix. Eigenvectors are the directions along which a particular linear transformation acts by flipping, compressing or stretching.

Eigenvalue can be referred to as the strength of the transformation in the direction of eigenvector or the factor by which the compression occurs.

Q35. Can you cite some examples where a false positive is important than a false negative?

Let us first understand what false positives and false negatives are.

- **False Positives** are the cases where you wrongly classified a non-event as an event a.k.a Type I error.
- **False Negatives** are the cases where you wrongly classify events as non-events, a.k.a Type II error.

Example 1: In the medical field, assume you have to give chemotherapy to patients. Assume a patient comes to that hospital and he is tested positive for cancer, based on the lab prediction but he actually doesn't have cancer. This is a case of false positive. Here it is of utmost danger to start chemotherapy on this patient when he actually does not have cancer. In the absence of cancerous cell, chemotherapy will do certain damage to his normal healthy cells and might lead to severe diseases, even cancer.

Example 2: Let's say an e-commerce company decided to give \$1000 Gift voucher to the customers whom they assume to purchase at least \$10,000 worth of items. They send free voucher mail directly to 100 customers without any minimum purchase condition because they assume to make at least 20% profit on sold items above \$10,000. Now the issue is if we send the \$1000 gift vouchers to customers who have not actually purchased anything but are marked as having made \$10,000 worth of purchase.

Q36. Can you cite some examples where a false negative is more important than a false positive?

Example 1: Assume there is an airport 'A' which has received high-security threats and based on certain characteristics they identify whether a particular passenger can be a threat or not. Due to a shortage of staff, they decide to scan passengers being predicted as risk positives by their predictive model. What will happen if a true threat customer is being flagged as non-threat by airport model?

Example 2: What if Jury or judge decides to make a criminal go free?

Example 3: What if you rejected to marry a very good person based on your predictive model and you happen to meet him/her after a few years and realize that you had a false negative?

Q37. Can you cite some examples where both false positive and false negatives are equally important?

In the **Banking** industry giving loans is the primary source of making money but at the same time if your repayment rate is not good you will not make any profit, rather you will risk huge losses.

Banks don't want to lose good customers and at the same point in time, they don't want to acquire bad customers. In this scenario, both the false positives and false negatives become very important to measure.

Q38. Can you explain the difference between a Validation Set and a Test Set?

A **Validation set** can be considered as a part of the training set as it is used for parameter selection and to avoid overfitting of the model being built.

On the other hand, a **Test Set** is used for testing or evaluating the performance of a trained machine learning model.

In simple terms, the differences can be summarized as; training set is to fit the parameters i.e. weights and test set is to assess the performance of the model i.e. evaluating the predictive power and generalization.

Q39. Explain cross-validation.

Cross-validation is a model validation technique for evaluating how the outcomes of statistical analysis will **generalize** to an **independent dataset**. Mainly used in backgrounds where the objective is forecast and one wants to estimate how accurately a model will accomplish in practice.

The goal of cross-validation is to term a data set to test the model in the training phase (i.e. validation data set) in order to limit problems like overfitting and get an insight on how the model will generalize to an independent data set.

MACHINE LEARNING INTERVIEW QUESTIONS

Q40. What is Machine Learning?

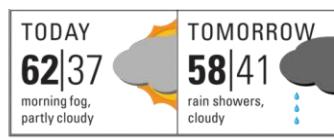
Machine Learning explores the study and construction of algorithms that can learn from and make predictions on data. Closely related to computational statistics. Used to devise complex models and algorithms that lend themselves to a prediction which in commercial use is known as predictive analytics. Given below, is an image representing the various domains Machine Learning lends itself to.

Speech
Recognition?

Face Recognition



Anti Virus



Weather Prediction

Q41. What is Supervised Learning?

Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples.

Algorithms: Support Vector Machines, Regression, Naive Bayes, Decision Trees, K-nearest Neighbor Algorithm and Neural Networks

E.g. If you built a fruit classifier, the labels will be “this is an orange, this is an apple and this is a banana”, based on showing the classifier examples of apples, oranges and bananas.

Q42. What is Unsupervised learning?

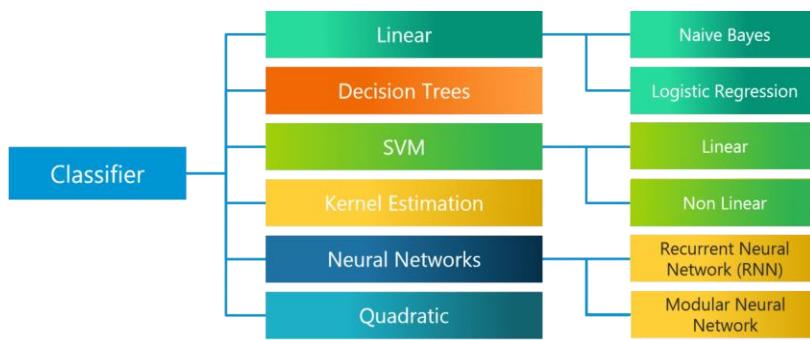
Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labelled responses.

Algorithms: Clustering, Anomaly Detection, Neural Networks and Latent Variable Models

E.g. In the same example, a fruit clustering will categorize as “fruits with soft skin and lots of dimples”, “fruits with shiny hard skin” and “elongated yellow fruits”.

Q43. What are the various classification algorithms?

The diagram lists the most important **classification algorithms**.



Q44. What is ‘Naive’ in a Naive Bayes?

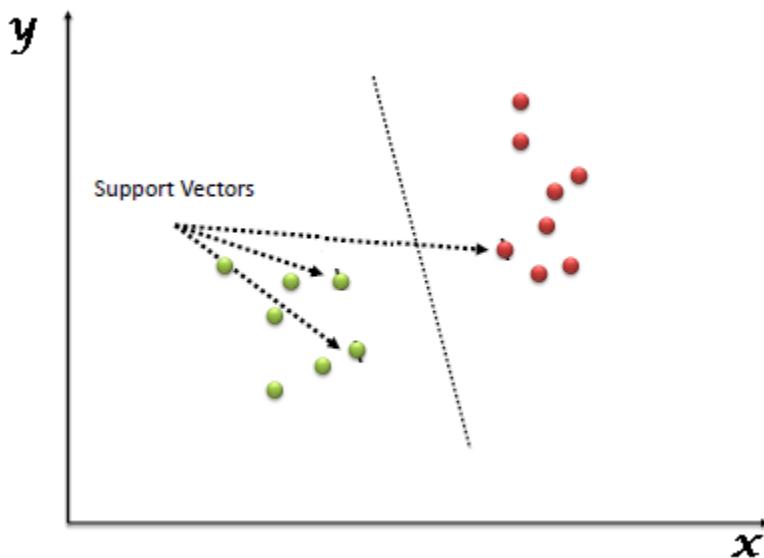
The **Naive Bayes Algorithm** is based on the Bayes Theorem. Bayes’ theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

The Algorithm is ‘naive’ because it makes assumptions that may or may not turn out to be correct.

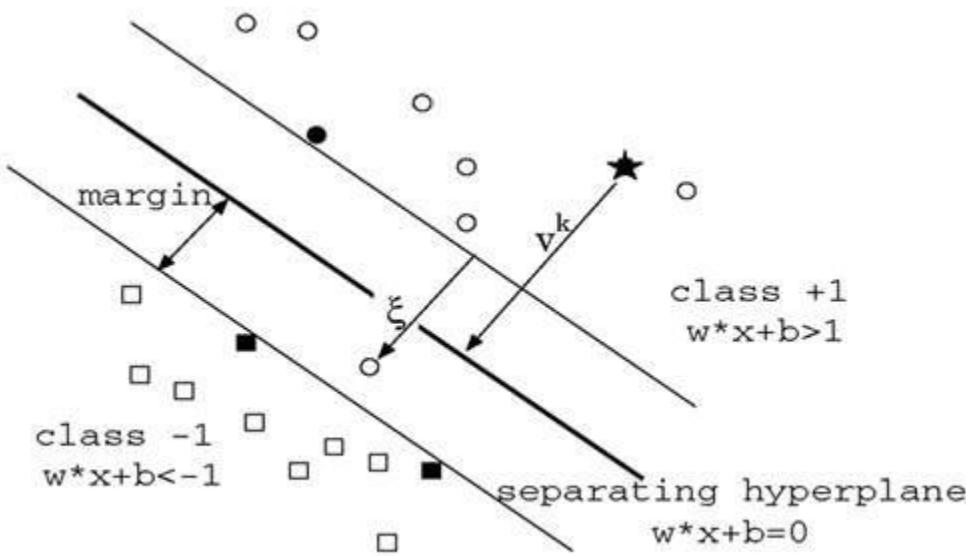
Q45. Explain SVM algorithm in detail.

SVM stands for support vector machine, it is a supervised machine learning algorithm which can be used for both **Regression and Classification**. If you have n features in your training data set, SVM tries to plot it

in n-dimensional space with the value of each feature being the value of a particular coordinate. SVM uses hyperplanes to separate out different classes based on the provided kernel function.



Q46. What are the support vectors in SVM?



In the diagram, we see that the thinner lines mark the distance from the classifier to the closest data points called the support vectors (darkened data points). The distance between the two thin lines is called the margin.

Q47. What are the different kernels in SVM?

There are four types of kernels in SVM.

1. Linear Kernel
2. Polynomial kernel
3. Radial basis kernel
4. Sigmoid kernel

Q48. Explain Decision Tree algorithm in detail.

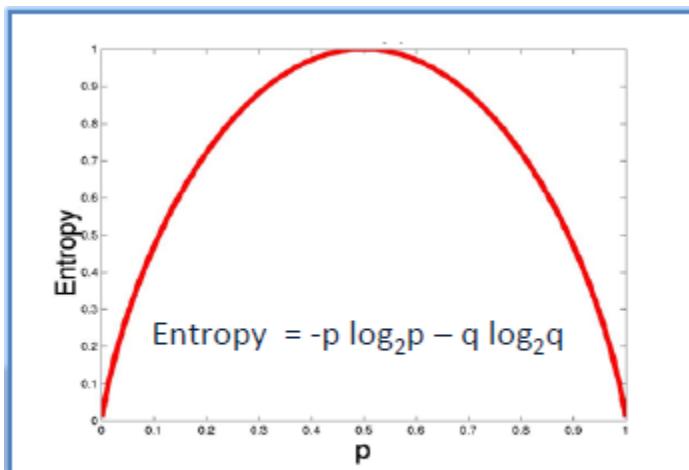
A **decision tree** is a supervised machine learning algorithm mainly used for **Regression and Classification**. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision tree can handle both categorical and numerical data.



Q49. What are Entropy and Information gain in Decision tree algorithm?

The core algorithm for building a decision tree is called **ID3**. ID3 uses **Entropy** and **Information Gain**.
Entropy

A decision tree is built top-down from a root node and involve partitioning of data into homogenous subsets. **ID3** uses entropy to check the homogeneity of a sample. If the sample is completely homogenous then entropy is zero and if the sample is an equally divided it has entropy of one.



$$\text{Entropy} = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

Information Gain

The **Information Gain** is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attributes that return the highest information gain.

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

$$G(\text{PlayGolf, Outlook}) = E(\text{PlayGolf}) - E(\text{PlayGolf, Outlook})$$

$$= 0.940 - 0.693 = 0.247$$

Q50. What is pruning in Decision Tree?

Pruning is a technique in machine learning and search algorithms that reduces the size of **decision trees** by removing sections of the **tree** that provide little power to classify instances. So, when we remove sub-nodes of a decision node, this process is called **pruning** or opposite process of splitting.

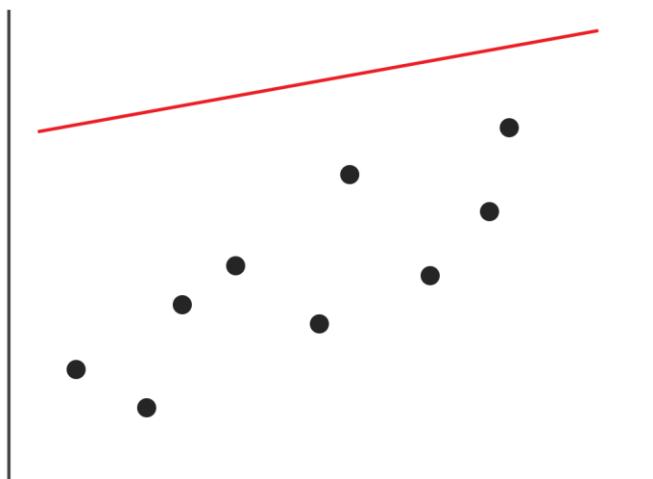
Q51. What is logistic regression? State an example when you have used logistic regression recently.

Logistic Regression often referred to as the logit model is a technique to predict the binary outcome from a linear combination of predictor variables.

For example, if you want to predict whether a particular political leader will win the election or not. In this case, the outcome of prediction is binary i.e. 0 or 1 (Win/Lose). The predictor variables here would be the amount of money spent for election campaigning of a particular candidate, the amount of time spent in campaigning, etc.

Q52. What is Linear Regression?

Linear regression is a statistical technique where the score of a variable Y is predicted from the score of a second variable X. X is referred to as the predictor variable and Y as the criterion variable.



D =

The regression line is the one with the least value of D

Follow me for more AI and Data science posts: <https://lnkd.in/gZu463X>

Q53. What Are the Drawbacks of the Linear Model?

Some drawbacks of the linear model are:

- The assumption of linearity of the errors.
- It can't be used for count outcomes or binary outcomes
- There are overfitting problems that it can't solve

Q54. What is the difference between Regression and classification ML techniques?

Both Regression and classification machine learning techniques come under **Supervised machine learning algorithms**. In Supervised machine learning algorithm, we have to train the model using labelled data set, While training we have to explicitly provide the correct labels and algorithm tries to learn the pattern from input to output. If our labels are discrete values then it will a classification problem, e.g A,B etc. but if our labels are continuous values then it will be a regression problem, e.g 1.23, 1.333 etc.

Q55. What are Recommender Systems?

Recommender Systems are a subclass of information filtering systems that are meant to predict the preferences or ratings that a user would give to a product. Recommender systems are widely used in movies, news, research articles, products, social tags, music, etc.

Examples include movie recommenders in IMDB, Netflix & BookMyShow, product recommenders in e-commerce sites like Amazon, eBay & Flipkart, YouTube video recommendations and game recommendations in Xbox.

Q56. What is Collaborative filtering?

The process of filtering used by most of the recommender systems to find patterns or information by collaborating viewpoints, various data sources and multiple agents.

Follow Steve Nouri for more AI and Data science posts: <https://lnkd.in/gZu463X>

Movie	Alice	Bob	Carol	Dave
Shutter Island	4	3	5	1
Fight Club	5	4	4	2
Dark Knight	5	3	4	?
21	4	3	?	5
Home Alone	4	4	5	5

Figure: Predicting the rating of Dave for Dark Knight and Carol for 21 using Collaborative Filtering

An example of collaborative filtering can be to predict the rating of a particular user based on his/her ratings for other movies and others' ratings for all movies. This concept is widely used in recommending movies in IMDB, Netflix & BookMyShow, product recommenders in e-commerce sites like Amazon, eBay & Flipkart, YouTube video recommendations and game recommendations in Xbox.

Q57. How can outlier values be treated?

Outlier values can be identified by using univariate or any other graphical analysis method. If the number of outlier values is few then they can be assessed individually but for a large number of outliers, the values can be substituted with either the 99th or the 1st percentile values.

All extreme values are not outlier values. The most common ways to treat outlier values

1. To change the value and bring it within a range.
2. To just remove the value.

Q58. What are the various steps involved in an analytics project?

The following are the various **steps involved in an analytics project**:

1. Understand the Business problem
2. Explore the data and become familiar with it.
3. Prepare the data for modelling by detecting outliers, treating missing values, transforming variables, etc.
4. After data preparation, start running the model, analyze the result and tweak the approach. This is an iterative step until the best possible outcome is achieved.
5. Validate the model using a new data set.
6. Start implementing the model and track the result to analyze the performance of the model over the period of time.

Q59. During analysis, how do you treat missing values?

The extent of the missing values is identified after identifying the variables with missing values. If any patterns are identified the analyst has to concentrate on them as it could lead to interesting and meaningful business insights.

If there are no patterns identified, then the missing values can be substituted with mean or median values (imputation) or they can simply be ignored. Assigning a default value which can be mean, minimum or maximum value. Getting into the data is important.

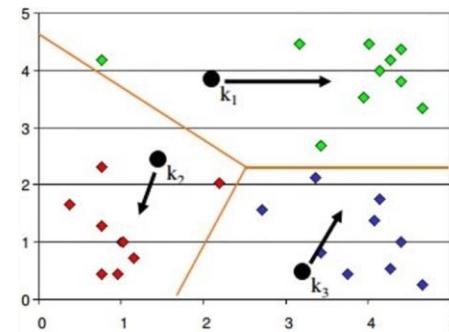
If it is a categorical variable, the default value is assigned. The missing value is assigned a default value. If you have a distribution of data coming, for normal distribution give the mean value.

If 80% of the values for a variable are missing then you can answer that you would be dropping the variable instead of treating the missing values.

Q60. How will you define the number of clusters in a clustering algorithm?

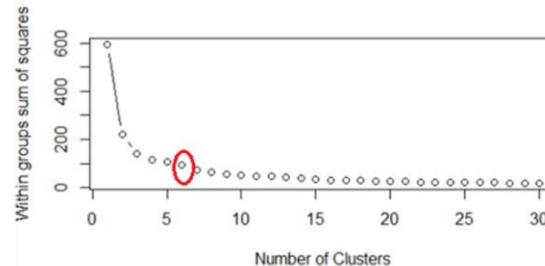
Though the Clustering Algorithm is not specified, this question is mostly in reference to **K-Means clustering** where “K” defines the number of clusters. The objective of clustering is to group similar entities in a way that the entities within a group are similar to each other but the groups are different from each other.

For example, the following image shows three different groups.



Within Sum of

squares is generally used to explain the homogeneity within a cluster. If you plot WSS for a range of number of clusters, you will get the plot shown below.



- The Graph is generally known as **Elbow Curve**.
- Red circled a point in above graph i.e. **Number of Cluster =6** is the point after which you don't see any decrement in WSS.
- This point is known as the **bending** point and taken as K in K – Means.

This is the widely used approach but few data scientists also use Hierarchical clustering first to create dendograms and identify the distinct groups from there.

Q61. What is Ensemble Learning?

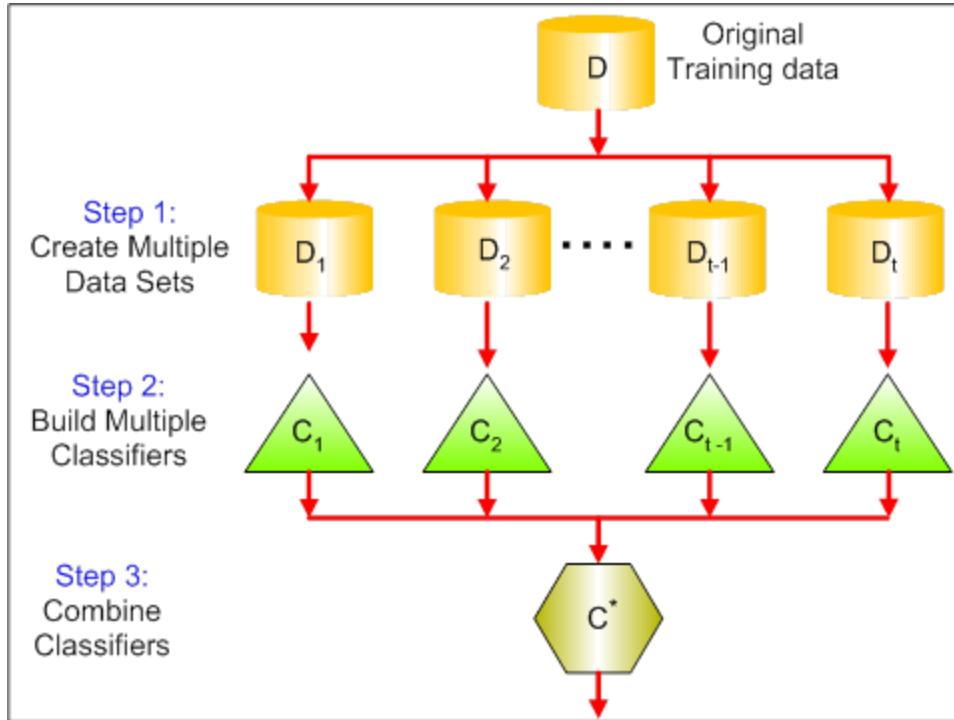
Ensemble Learning is basically combining a diverse set of learners(Individual models) together to improvise on the stability and predictive power of the model.

Q62. Describe in brief any type of Ensemble Learning?

Ensemble learning has many types but two more popular ensemble learning techniques are mentioned below.

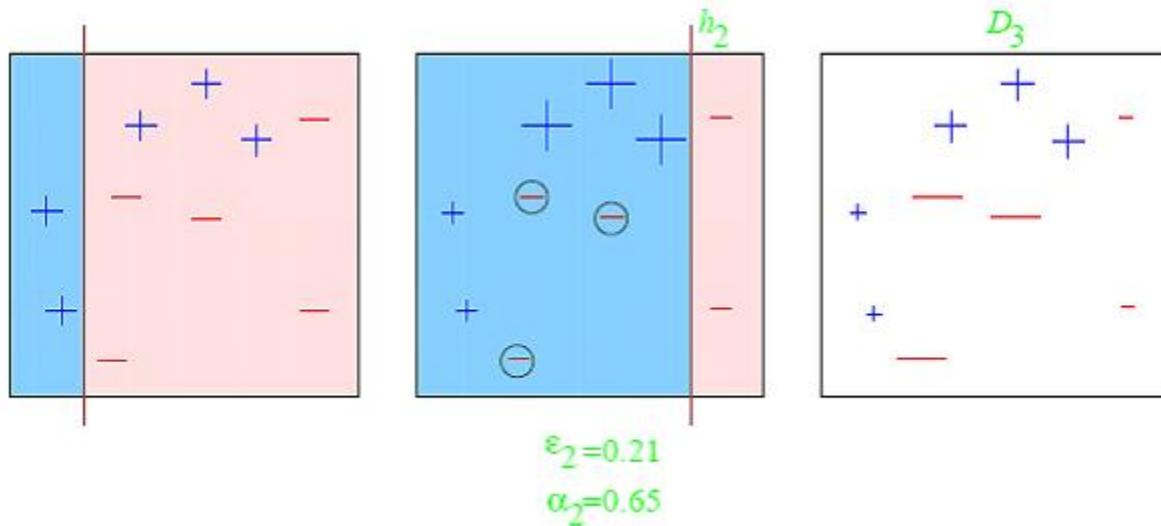
Bagging

Bagging tries to implement similar learners on small sample populations and then takes a mean of all the predictions. In generalised bagging, you can use different learners on different population. As you expect this helps us to reduce the variance error.



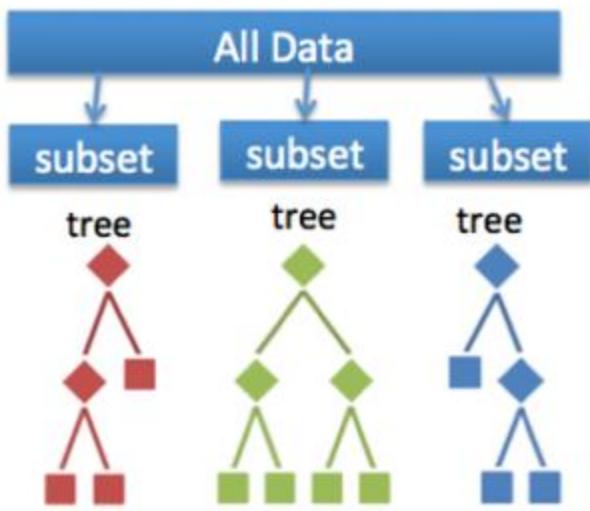
Boosting

Boosting is an iterative technique which adjusts the weight of an observation based on the last classification. If an observation was classified incorrectly, it tries to increase the weight of this observation and vice versa. Boosting in general decreases the bias error and builds strong predictive models. However, they may over fit on the training data.



Q63. What is a Random Forest? How does it work?

Random forest is a versatile machine learning method capable of performing both regression and classification tasks. It is also used for dimensionality reduction, treats missing values, outlier values. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model.



In Random Forest, we grow multiple trees as opposed to a single tree. To classify a new object based on attributes, each tree gives a classification. The forest chooses the classification having the most **votes**(Overall the trees in the forest) and in case of regression, it takes the average of outputs by different trees.

Q64. How Do You Work Towards a Random Forest?

The underlying principle of this technique is that several weak learners combined to provide a keen learner. The steps involved are

- Build several decision trees on bootstrapped training samples of data
- On each tree, each time a split is considered, a random sample of m predictors is chosen as split candidates, out of all p predictors
- Rule of thumb: At each split $m=p\sqrt{m}=p$
- Predictions: At the majority rule

Q65. What cross-validation technique would you use on a time series data set?

Instead of using k-fold cross-validation, you should be aware of the fact that a time series is not randomly distributed data — It is inherently ordered by chronological order.

In case of time series data, you should use techniques like forward-chaining — Where you will be model on past data then look at forward-facing data.

fold 1: training[1], test[2]

fold 1: training[1 2], test[3]

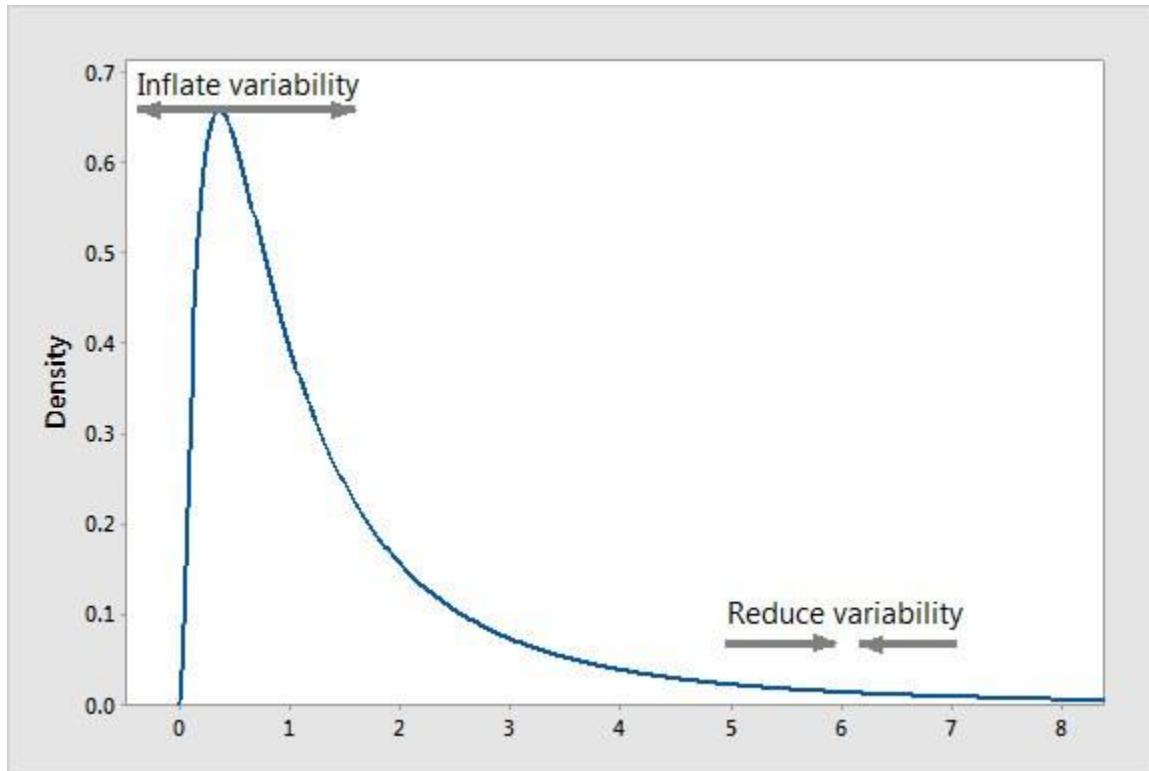
fold 1: training[1 2 3], test[4]

fold 1: training[1 2 3 4], test[5]

Q66. What is a Box-Cox Transformation?

The dependent variable for a regression analysis might not satisfy one or more assumptions of an ordinary least squares regression. The residuals could either curve as the prediction increases or follow the skewed distribution. In such scenarios, it is necessary to transform the response variable so that the data meets the required assumptions. A Box cox transformation is a statistical technique to transform non-normal dependent variables into a normal shape. If the given data is not normal then most of the statistical

techniques assume normality. Applying a box cox transformation means that you can run a broader number of tests.



A Box-Cox transformation is a way to transform non-normal dependent variables into a normal shape. Normality is an important assumption for many statistical techniques, if your data isn't normal, applying a Box-Cox means that you are able to run a broader number of tests. The Box-Cox transformation is named after statisticians **George Box** and **Sir David Roxbee Cox** who collaborated on a 1964 paper and developed the technique.

Q67. How Regularly Must an Algorithm be Updated?

You will want to update an algorithm when:

- You want the model to evolve as data streams through infrastructure
- The underlying data source is changing
- There is a case of non-stationarity
- The algorithm underperforms/ results lack accuracy

Q68. If you are having 4GB RAM in your machine and you want to train your model on 10GB data set. How would you go about this problem? Have you ever faced this kind of problem in your machine learning/data science experience so far?

First of all, you have to ask which ML model you want to train.

For Neural networks: Batch size with Numpy array will work.

Steps:

1. Load the whole data in the Numpy array. Numpy array has a property to create a mapping of the complete data set, it doesn't load complete data set in memory.
2. You can pass an index to Numpy array to get required data.
3. Use this data to pass to the Neural network.

4. Have a small batch size.

For SVM: Partial fit will work

Steps:

1. Divide one big data set in small size data sets.
2. Use a partial fit method of SVM, it requires a subset of the complete data set.
3. Repeat step 2 for other subsets.

However, you could actually face such an issue in reality. So, you could check out the [best laptop for Machine Learning](#) to prevent that. Having said that, let's move on to some questions on deep learning.

DEEP LEARNING INTERVIEW QUESTIONS

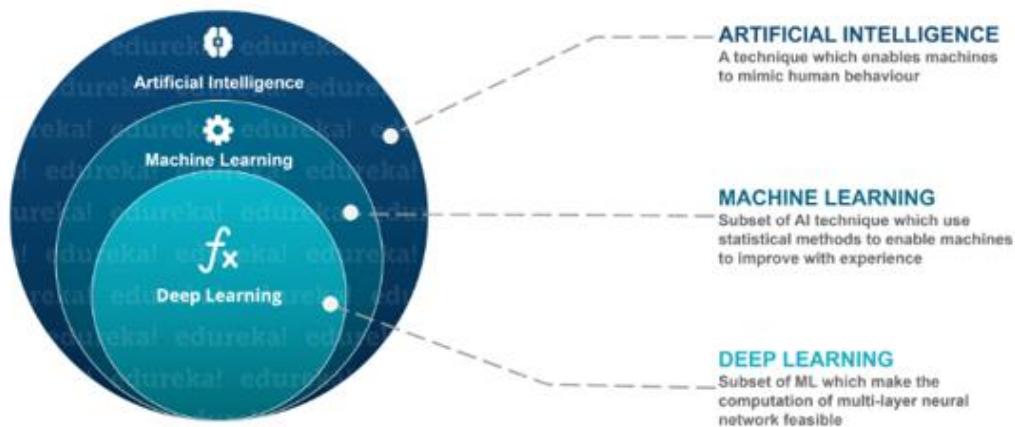
Q69. What do you mean by Deep Learning?

Deep Learning is nothing but a paradigm of machine learning which has shown incredible promise in recent years. This is because of the fact that Deep Learning shows a great analogy with the functioning of the human brain.

Q70. What is the difference between machine learning and deep learning?

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed. Machine learning can be categorised in the following three categories.

1. Supervised machine learning,
2. Unsupervised machine learning,
3. Reinforcement learning



Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks.

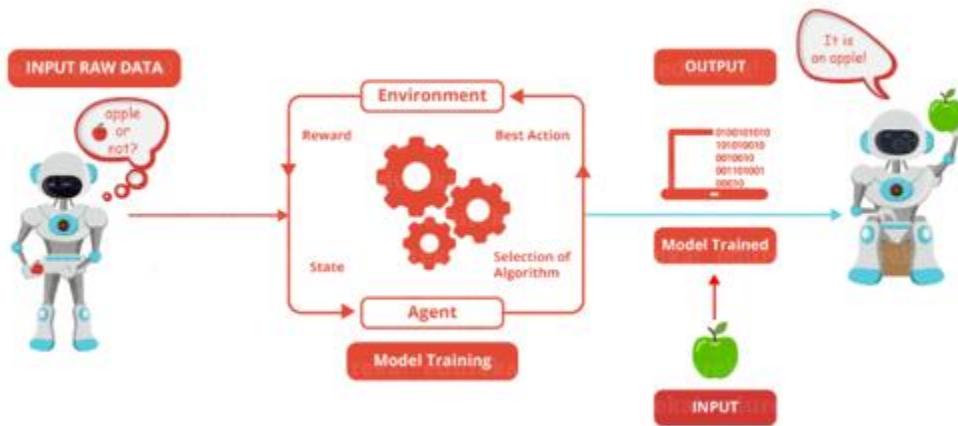
Q71. What, in your opinion, is the reason for the popularity of Deep Learning in recent times?

Now although Deep Learning has been around for many years, the major breakthroughs from these techniques came just in recent years. This is because of two main reasons:

- The increase in the amount of data generated through various sources
- The growth in hardware resources required to run these models

GPUs are multiple times faster and they help us build bigger and deeper deep learning models in comparatively less time than we required previously.

Q72. What is reinforcement learning?



Reinforcement Learning is

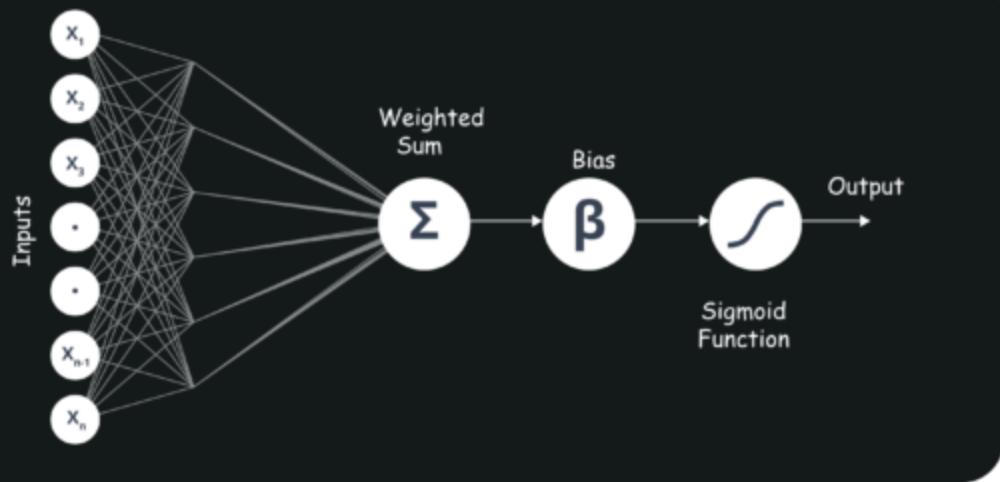
learning what to do and how to map situations to actions. The end result is to maximise the numerical reward signal. The learner is not told which action to take but instead must discover which action will yield the maximum reward. Reinforcement learning is inspired by the learning of human beings, it is based on the reward/penalty mechanism.

Q73. What are Artificial Neural Networks?

Artificial Neural networks are a specific set of algorithms that have revolutionized machine learning. They are inspired by biological neural networks. **Neural Networks** can adapt to changing the input so the network generates the best possible result without needing to redesign the output criteria.

Q74. Describe the structure of Artificial Neural Networks?

Artificial Neural Networks works on the same principle as a biological Neural Network. It consists of inputs which get processed with weighted sums and Bias, with the help of Activation Functions.



Q75. How Are Weights Initialized in a Network?

There are two methods here: we can either initialize the weights to zero or assign them randomly.

Initializing all weights to 0: This makes your model similar to a linear model. All the neurons and every layer perform the same operation, giving the same output and making the deep net useless.

Initializing all weights randomly: Here, the weights are assigned randomly by initializing them very close to 0. It gives better accuracy to the model since every neuron performs different computations. This is the most commonly used method.

Q76. What Is the Cost Function?

Also referred to as “loss” or “error,” cost function is a measure to evaluate how good your model’s performance is. It’s used to compute the error of the output layer during backpropagation. We push that error backwards through the neural network and use that during the different training functions.

Q77. What Are Hyperparameters?

With neural networks, you’re usually working with **hyperparameters** once the data is formatted correctly. A hyperparameter is a parameter whose value is set before the learning process begins. It determines how a network is trained and the structure of the network (such as the number of hidden units, the learning rate, epochs, etc.).

Q78. What Will Happen If the Learning Rate Is Set inaccurately (Too Low or Too High)?

When your learning rate is too low, training of the model will progress very slowly as we are making minimal updates to the weights. It will take many updates before reaching the minimum point.

If the learning rate is set too high, this causes undesirable divergent behaviour to the loss function due to drastic updates in weights. It may fail to converge (model can give a good output) or even diverge (data is too chaotic for the network to train).

Q79. What Is the Difference Between Epoch, Batch, and Iteration in Deep Learning?

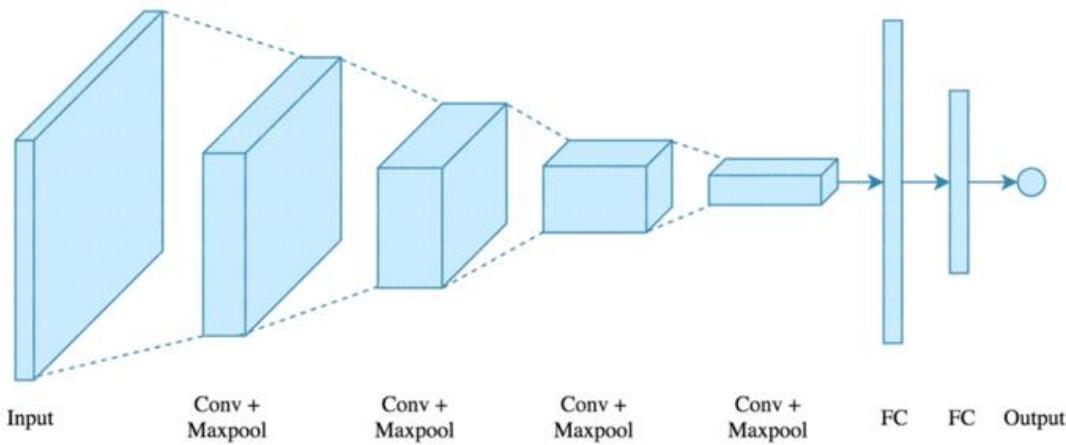
- Epoch – Represents one iteration over the entire dataset (everything put into the training model).
- Batch – Refers to when we cannot pass the entire dataset into the neural network at once, so we divide the dataset into several batches.

- Iteration – if we have 10,000 images as data and a batch size of 200. then an epoch should run 50 iterations (10,000 divided by 50).

Q80. What Are the Different Layers on CNN?

There are four layers in **CNN**:

1. Convolutional Layer – the layer that performs a convolutional operation, creating several smaller picture windows to go over the data.
2. ReLU Layer – it brings non-linearity to the network and converts all the negative pixels to zero. The output is a rectified feature map.
3. Pooling Layer – pooling is a down-sampling operation that reduces the dimensionality of the feature map.
4. Fully Connected Layer – this layer recognizes and classifies the objects in the image.



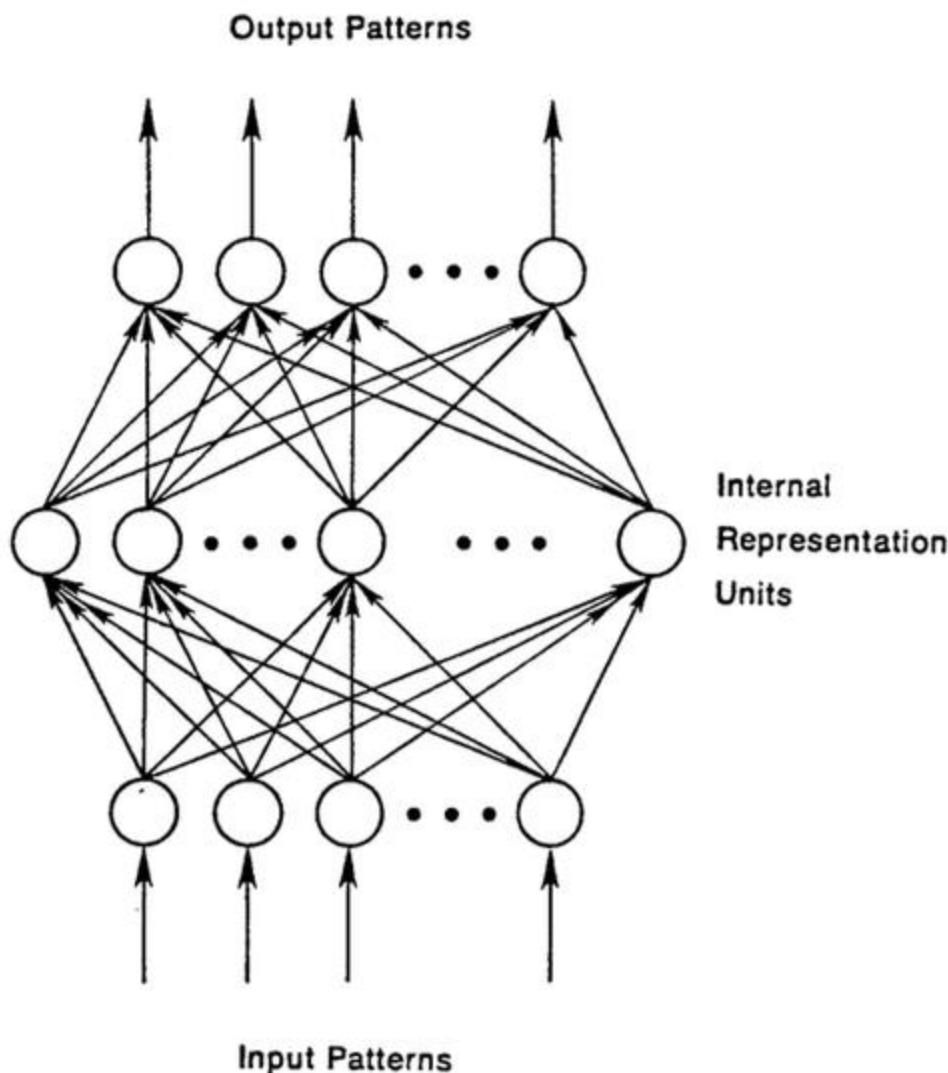
Q81. What Is Pooling on CNN, and How Does It Work?

Pooling is used to reduce the spatial dimensions of a CNN. It performs down-sampling operations to reduce the dimensionality and creates a pooled feature map by sliding a filter matrix over the input matrix.

Q82. What are Recurrent Neural Networks(RNNs)?

RNNs are a type of artificial neural networks designed to recognise the pattern from the sequence of data such as Time series, stock market and government agencies etc. To understand recurrent nets, first, you have to understand the basics of feedforward nets.

Both these networks RNN and feed-forward named after the way they channel information through a series of mathematical operations performed at the nodes of the network. One feeds information through straight(never touching the same node twice), while the other cycles it through a loop, and the latter are called recurrent.



Recurrent networks, on the other hand, take as their input, not just the current input example they see, but also the what they have perceived previously in time.

The decision a recurrent neural network reached at time $t-1$ affects the decision that it will reach one moment later at time t . So recurrent networks have two sources of input, the present and the recent past, which combine to determine how they respond to new data, much as we do in life.

The error they generate will return via backpropagation and be used to adjust their weights until error can't go any lower. Remember, the purpose of recurrent nets is to accurately classify sequential input. We rely on the backpropagation of error and gradient descent to do so.

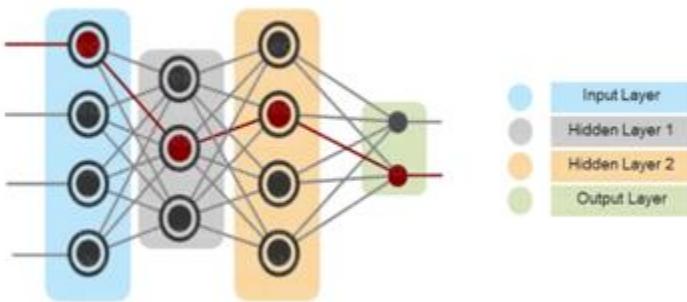
Q83. How Does an LSTM Network Work?

Long-Short-Term Memory (LSTM) is a special kind of recurrent neural network capable of learning long-term dependencies, remembering information for long periods as its default behaviour. There are three steps in an LSTM network:

- **Step 1:** The network decides what to forget and what to remember.
- **Step 2:** It selectively updates cell state values.
- **Step 3:** The network decides what part of the current state makes it to the output.

Q84. What Is a Multi-layer Perceptron(MLP)?

As in **Neural Networks**, **MLPs** have an input layer, a hidden layer, and an output layer. It has the same structure as a single layer **perceptron** with one or more hidden layers. A single layer perceptron can classify only linear separable classes with binary output (0,1), but MLP can classify nonlinear classes.



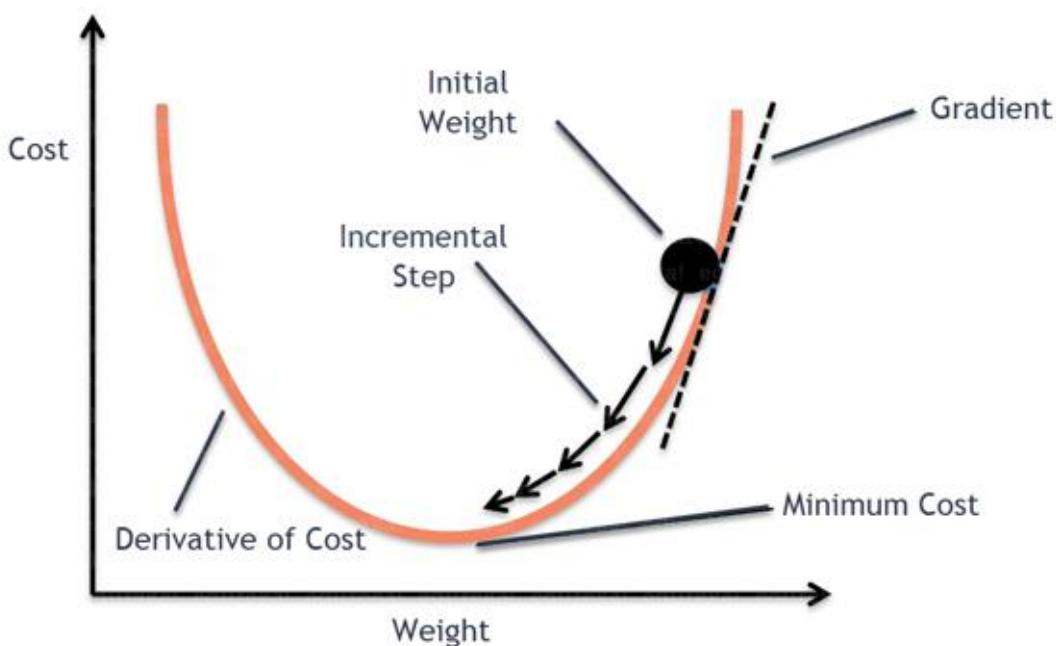
Except for the input layer, each node in the other layers uses a nonlinear activation function. This means the input layers, the data coming in, and the activation function is based upon all nodes and weights being added together, producing the output. MLP uses a supervised learning method called “backpropagation.” In **backpropagation**, the neural network calculates the error with the help of cost function. It propagates this error backward from where it came (adjusts the weights to train the model more accurately).

Q85. Explain Gradient Descent.

To Understand Gradient Descent, Let's understand what is a **Gradient** first.

A **gradient** measures how much the output of a function changes if you change the inputs a little bit. It simply measures the change in all weights with regard to the change in error. You can also think of a gradient as the slope of a function.

Gradient Descent can be thought of climbing down to the bottom of a valley, instead of climbing up a hill. This is because it is a minimization algorithm that minimizes a given function (**Activation Function**).



exploding gradients?

Q86. What is

While training an RNN, if you see **exponentially growing (very large) error gradients** which accumulate and result in very large updates to neural network model weights during training, they're known as exploding gradients. At an extreme, the values of weights can become so large as to overflow and result in NaN values.

This has the effect of your model is unstable and unable to learn from your training data.

Q87. What is vanishing gradients?

While training an RNN, your slope can become either too small; this makes the training difficult. When the slope is too small, the problem is known as a Vanishing Gradient. It leads to long training times, poor performance, and low accuracy.

Q89. What is Back Propagation and Explain it's Working.

Backpropagation is a training algorithm used for multilayer neural network. In this method, we move the error from an end of the network to all weights inside the network and thus allowing efficient computation of the gradient.

It has the following steps:



See Batch Details

- Forward Propagation of Training Data
- Derivatives are computed using output and target
- Back Propagate for computing derivative of error wrt output activation
- Using previously calculated derivatives for output
- Update the Weights

Q90. What are the variants of Back Propagation?

- **Stochastic Gradient Descent:** We use only a single training example for calculation of gradient and update parameters.
- **Batch Gradient Descent:** We calculate the gradient for the whole dataset and perform the update at each iteration.
- **Mini-batch Gradient Descent:** It's one of the most popular optimization algorithms. It's a variant of Stochastic Gradient Descent and here instead of single training example, mini-batch of samples is used.

Q91. What are the different **Deep Learning Frameworks**?

- **Pytorch**
- **TensorFlow**
- Microsoft Cognitive Toolkit

- Keras
- Caffe
- Chainer

Q92. What is the role of the Activation Function?

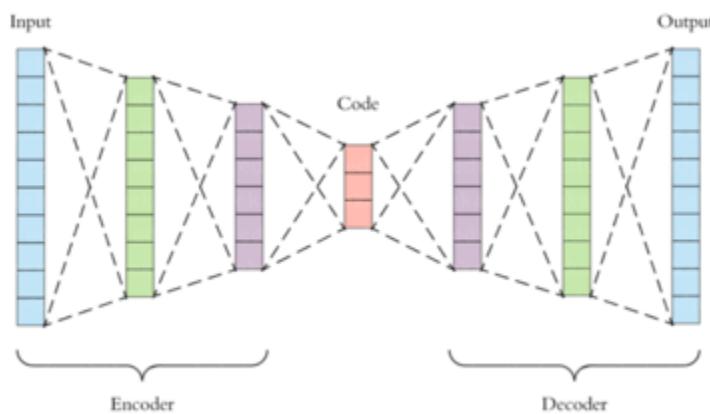
The **Activation function** is used to introduce non-linearity into the neural network helping it to learn more complex function. Without which the neural network would be only able to learn linear function which is a linear combination of its input data. An activation function is a function in an artificial neuron that delivers an output based on inputs.

Q93. Name a few **Machine Learning libraries** for various purposes.

Purpose	Libraries
Scientific Computation	Numpy
Tabular Data	Pandas
Data Modelling & Preprocessing	Scikit Learn
Time-Series Analysis	Statsmodels
Text processing	Regular Expressions, NLTK
Deep Learning	Tensorflow, Pytorch

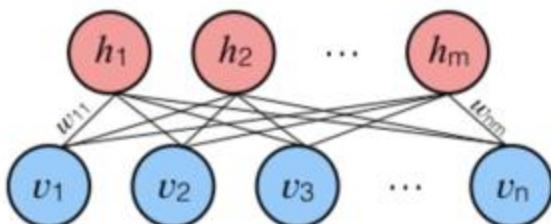
Q94. What is an Auto-Encoder?

Auto-encoders are simple learning networks that aim to transform inputs into outputs with the minimum possible error. This means that we want the output to be as close to input as possible. We add a couple of layers between the input and the output, and the sizes of these layers are smaller than the input layer. The auto-encoder receives unlabelled input which is then encoded to reconstruct the input.



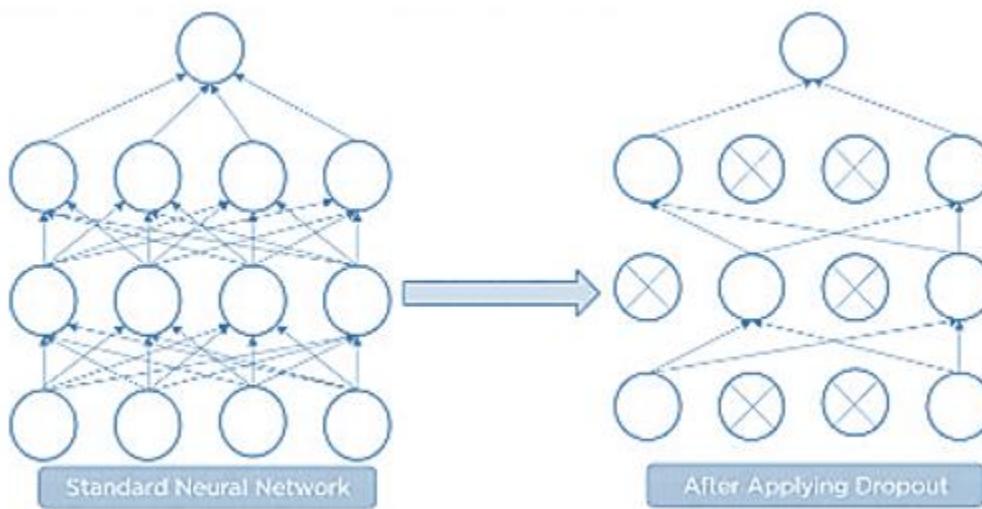
Q95. What is a Boltzmann Machine?

Boltzmann machines have a simple learning algorithm that allows them to discover interesting features that represent complex regularities in the training data. The Boltzmann machine is basically used to optimise the weights and the quantity for the given problem. The learning algorithm is very slow in networks with many layers of feature detectors. “**Restricted Boltzmann Machines**” algorithm has a single layer of feature detectors which makes it faster than the rest.



Q96. What Is Dropout and Batch Normalization?

Dropout is a technique of dropping out hidden and visible units of a network randomly to prevent overfitting of data (typically dropping 20 per cent of the nodes). It doubles the number of iterations needed to converge the network.



Batch normalization is the technique to improve the performance and stability of neural networks by normalizing the inputs in every layer so that they have mean output activation of zero and standard deviation of one.

Q97. What Is the Difference Between Batch Gradient Descent and Stochastic Gradient Descent?

Batch Gradient Descent	Stochastic Gradient Descent
The batch gradient computes the gradient using the entire dataset.	The stochastic gradient computes the gradient using a single sample.
It takes time to converge because the volume of data is huge, and weights update slowly.	It converges much faster than the batch gradient because it updates weight more frequently.

Q98. Why Is Tensorflow the Most Preferred Library in Deep Learning?

Tensorflow provides both C++ and Python APIs, making it easier to work on and has a faster compilation time compared to other Deep Learning libraries like Keras and Torch. Tensorflow supports both CPU and GPU computing devices.

Q99. What Do You Mean by Tensor in Tensorflow?

A tensor is a mathematical object represented as arrays of higher dimensions. These arrays of data with different dimensions and ranks fed as input to the neural network are called “**Tensors**.”

't'
'e'
'n'
's'
'o'
'r'

Tensor of dimension[1]

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4

Tensor of dimensions[2]

2	1	8	8	1
2	4	9	0	5
2	5	3	0	8
7	7	3	5	2
4	1	2	6	3

Tensor of dimensions[3]

Q100. What is the Computational Graph?

Everything in tensorflow is based on creating a computational graph. It has a network of nodes where each node operates, Nodes represent mathematical operations, and edges represent tensors. Since data flows in the form of a graph, it is also called a “DataFlow Graph.”

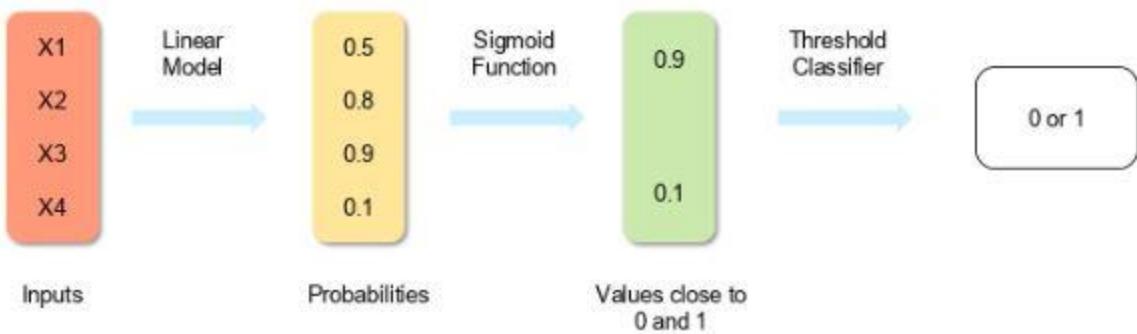
Q101. What are the differences between supervised and unsupervised learning?

Supervised Learning	Unsupervised Learning
<ul style="list-style-type: none"> • Uses known and labeled data as input • Supervised learning has a feedback mechanism • Most commonly used supervised learning algorithms are decision trees, logistic regression, and support vector machine 	<ul style="list-style-type: none"> • Uses unlabeled data as input • Unsupervised learning has no feedback mechanism • Most commonly used unsupervised learning algorithms are k-means clustering, hierarchical clustering, and apriori algorithm

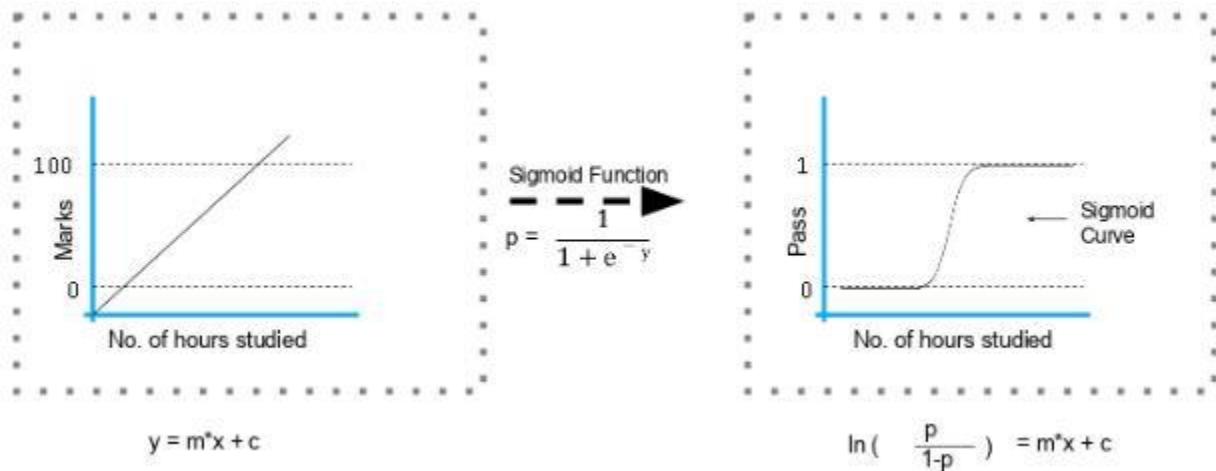
102. How is logistic regression done?

Logistic regression measures the relationship between the dependent variable (our label of what we want to predict) and one or more independent variables (our features) by estimating probability using its underlying logistic function (sigmoid).

The image shown below depicts how logistic regression works:



The formula and graph for the sigmoid function are as shown:



103. Explain the steps in making a decision tree.

1. Take the entire data set as input
2. Calculate entropy of the target variable, as well as the predictor attributes
3. Calculate your information gain of all attributes (we gain information on sorting different objects from each other)
4. Choose the attribute with the highest information gain as the root node
5. Repeat the same procedure on every branch until the decision node of each branch is finalized

For example, let's say you want to build a decision tree to decide whether you should accept or decline a job offer. The decision tree for this case is as shown:



It is clear from the decision tree that an offer is accepted if:

- Salary is greater than \$50,000
- The commute is less than an hour
- Incentives are offered

104. How do you build a random forest model?

A random forest is built up of a number of decision trees. If you split the data into different packages and make a decision tree in each of the different groups of data, the random forest brings all those trees together.

Steps to build a random forest model:

1. Randomly select 'k' features from a total of 'm' features where $k \ll m$
2. Among the 'k' features, calculate the node D using the best split point
3. Split the node into daughter nodes using the best split
4. Repeat steps two and three until leaf nodes are finalized
5. Build forest by repeating steps one to four for 'n' times to create 'n' number of trees

105. How can you avoid the overfitting your model?

Overfitting refers to a model that is only set for a very small amount of data and ignores the bigger picture. There are three main methods to avoid overfitting:

1. Keep the model simple—take fewer variables into account, thereby removing some of the noise in the training data
2. Use cross-validation techniques, such as k folds cross-validation
3. Use regularization techniques, such as LASSO, that penalize certain model parameters if they're likely to cause overfitting

106. Differentiate between univariate, bivariate, and multivariate analysis.

Univariate

Univariate data contains only one variable. The purpose of the univariate analysis is to describe the data and find patterns that exist within it.

Example: height of students

Height (in cm)
164
167.3
170
174.2
178
180

The patterns can be studied by drawing conclusions using mean, median, mode, dispersion or range, minimum, maximum, etc.

Bivariate

Bivariate data involves two different variables. The analysis of this type of data deals with causes and relationships and the analysis is done to determine the relationship between the two variables.

Example: temperature and ice cream sales in the summer season

Temperature (in Celcius)	Sales
20	2,000
25	2,100
26	2,300
28	2,400
30	2,600
36	3,100

Here, the relationship is visible from the table that temperature and sales are directly proportional to each other. The hotter the temperature, the better the sales.

Multivariate

Multivariate data involves three or more variables, it is categorized under multivariate. It is similar to a bivariate, but contains more than one dependent variable.

Example: data for house price prediction

No. of rooms	Floors	Area (sq ft)	Price
2	0	900	\$4000,00

3	2	1,100	\$600,000
3.5	5	1,500	\$900,000
4	3	2,100	\$1,200,000

The patterns can be studied by drawing conclusions using mean, median, and mode, dispersion or range, minimum, maximum, etc. You can start describing the data and using it to guess what the price of the house will be.

107. What are the feature selection methods used to select the right variables?

There are two main methods for feature selection:

Filter Methods

This involves:

- Linear discrimination analysis
- ANOVA
- Chi-Square

The best analogy for selecting features is "bad data in, bad answer out." When we're limiting or selecting the features, it's all about cleaning up the data coming in.

Wrapper Methods

This involves:

- Forward Selection: We test one feature at a time and keep adding them until we get a good fit
- Backward Selection: We test all the features and start removing them to see what works better
- Recursive Feature Elimination: Recursively looks through all the different features and how they pair together

Wrapper methods are very labor-intensive, and high-end computers are needed if a lot of data analysis is performed with the wrapper method.

108. In your choice of language, write a program that prints the numbers ranging from one to 50.

But for multiples of three, print "Fizz" instead of the number and for the multiples of five, print "Buzz." For numbers which are multiples of both three and five, print "FizzBuzz"

The code is shown below:

```
for fizzbuzz in range(51):
    if fizzbuzz % 3 == 0 and fizzbuzz % 5 == 0:
        print("FizzBuzz")
        continue
    elif fizzbuzz % 3 == 0:
        print("Fizz")
        continue
    elif fizzbuzz % 5 == 0:
        print("Buzz")
        continue
    print(fizzbuzz)
```

Note that the range mentioned is 51, which means zero to 50. However, the range asked in the question is one to 50. Therefore, in the above code, you can include the range as (1,51).

The output of the above code is as shown:

```
fizzbuzz
1
2
fizz
4
buzz
fizz
7
8
fizz
buzz
11
fizz
13
16
fizzbuzz
16
17
fizz
19
buzz
fizz
22
23
fizz
buzz
26
.
.
.
46
47
fizz
49
buzz
```

109. You are given a data set consisting of variables with more than 30 percent missing values. How will you deal with them?

The following are ways to handle missing data values:

If the data set is large, we can just simply remove the rows with missing data values. It is the quickest way; we use the rest of the data to predict the values.

For smaller data sets, we can substitute missing values with the mean or average of the rest of the data using pandas data frame in python. There are different ways to do so, such as df.mean(), df.fillna(mean).

110. For the given points, how will you calculate the Euclidean distance in Python?

```
plot1 = [1,3]
```

```
plot2 = [2,5]
```

The Euclidean distance can be calculated as follows:

```
euclidean_distance = sqrt( (plot1[0]-plot2[0])**2 + (plot1[1]-plot2[1])**2 )
```

111. What are dimensionality reduction and its benefits?

Dimensionality reduction refers to the process of converting a data set with vast dimensions into data with fewer dimensions (fields) to convey similar information concisely.

This reduction helps in compressing data and reducing storage space. It also reduces computation time as fewer dimensions lead to less computing. It removes redundant features; for example, there's no point in storing a value in two different units (meters and inches).

112. How will you calculate eigenvalues and eigenvectors of the following 3x3 matrix?

-2	-4	2
-2	1	2
4	2	5

The characteristic equation is as shown:

Expanding determinant:

$$(-2 - \lambda) [(1-\lambda)(5-\lambda)-2 \times 2] + 4[(-2) \times (5-\lambda) - 4 \times 2] + 2[(-2) \times 2 - 4(1-\lambda)] = 0$$

$$-\lambda^3 + 4\lambda^2 + 27\lambda - 90 = 0,$$

$$\lambda^3 - 4\lambda^2 - 27\lambda + 90 = 0$$

Here we have an algebraic equation built from the eigenvectors.

By hit and trial:

$$33 - 4 \times 32 - 27 \times 3 + 90 = 0$$

Hence, $(\lambda - 3)$ is a factor:

$$\lambda^3 - 4\lambda^2 - 27\lambda + 90 = (\lambda - 3)(\lambda^2 - \lambda - 30)$$

Eigenvalues are 3,-5,6:

$$(\lambda - 3)(\lambda^2 - \lambda - 30) = (\lambda - 3)(\lambda + 5)(\lambda - 6),$$

Calculate eigenvector for $\lambda = 3$

For $X = 1$,

$$-5 - 4Y + 2Z = 0,$$

$$-2 - 2Y + 2Z = 0$$

Subtracting the two equations:

$$3 + 2Y = 0,$$

Subtracting back into second equation:

$$Y = -(3/2)$$

$$Z = -(1/2)$$

Similarly, we can calculate the eigenvectors for -5 and 6.

113. How should you maintain a deployed model?

The steps to maintain a deployed model are:

Monitor

Constant monitoring of all models is needed to determine their performance accuracy. When you change something, you want to figure out how your changes are going to affect things. This needs to be monitored to ensure it's doing what it's supposed to do.

Evaluate

Evaluation metrics of the current model are calculated to determine if a new algorithm is needed.

Compare

The new models are compared to each other to determine which model performs the best.

Rebuild

The best performing model is re-built on the current state of data.

114. What are recommender systems?

A recommender system predicts what a user would rate a specific product based on their preferences. It can be split into two different areas:

Collaborative filtering

As an example, Last.fm recommends tracks that other users with similar interests play often. This is also commonly seen on Amazon after making a purchase; customers may notice the following message accompanied by product recommendations: "Users who bought this also bought..."

Content-based filtering

As an example: Pandora uses the properties of a song to recommend music with similar properties. Here, we look at content, instead of looking at who else is listening to music.

115. How do you find RMSE and MSE in a linear regression model?

RMSE and MSE are two of the most common measures of accuracy for a linear regression model.

RMSE indicates the Root Mean Square Error.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}}$$

MSE indicates the Mean Square Error.

$$MSE = \frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}$$

116. How can you select k for k-means?

We use the elbow method to select k for k-means clustering. The idea of the elbow method is to run k-means clustering on the data set where 'k' is the number of clusters.

Within the sum of squares (WSS), it is defined as the sum of the squared distance between each member of the cluster and its centroid.

117. What is the significance of p-value?

p-value typically ≤ 0.05

This indicates strong evidence against the null hypothesis; so you reject the null hypothesis.

p-value typically > 0.05

This indicates weak evidence against the null hypothesis, so you accept the null hypothesis.

p-value at cutoff 0.05

This is considered to be marginal, meaning it could go either way.

118. How can outlier values be treated?

You can drop outliers only if it is a garbage value.

Example: height of an adult = abc ft. This cannot be true, as the height cannot be a string value. In this case, outliers can be removed.

If the outliers have extreme values, they can be removed. For example, if all the data points are clustered between zero to 10, but one point lies at 100, then we can remove this point.

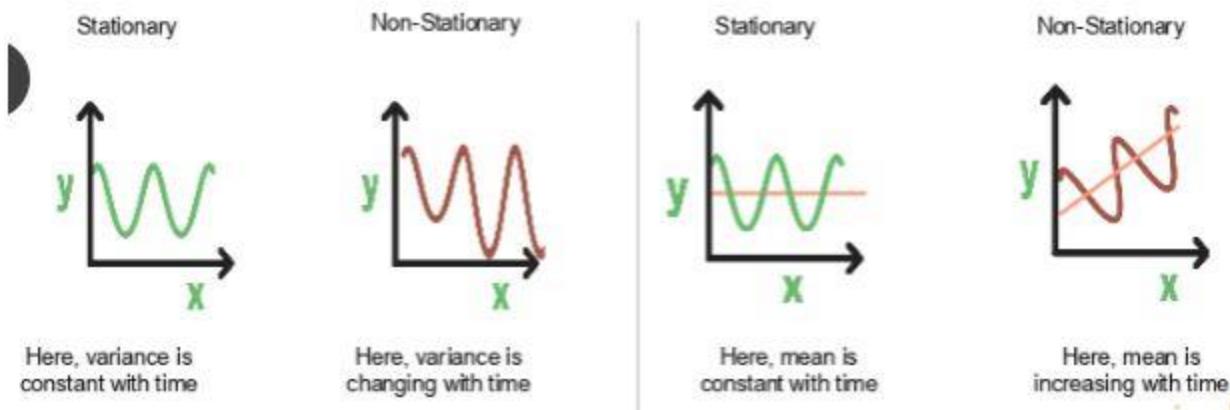
If you cannot drop outliers, you can try the following:

- Try a different model. Data detected as outliers by linear models can be fit by nonlinear models. Therefore, be sure you are choosing the correct model.
- Try normalizing the data. This way, the extreme data points are pulled to a similar range.
- You can use algorithms that are less affected by outliers; an example would be random forests.

119. How can a time-series data be declared as stationary?

It is stationary when the variance and mean of the series are constant with time.

Here is a visual example:



In the first graph, the variance is constant with time. Here, X is the time factor and Y is the variable. The value of Y goes through the same points all the time; in other words, it is stationary.

In the second graph, the waves get bigger, which means it is non-stationary and the variance is changing with time.

120. How can you calculate accuracy using a confusion matrix?

Consider this confusion matrix:

		actual		
		p	n	
predicted	P	262	15	False Positive
	N	26	347	True Negative
				True Positive
				False Negative

You can see the values for total data, actual values, and predicted values.

The formula for accuracy is:

$$\text{Accuracy} = (\text{True Positive} + \text{True Negative}) / \text{Total Observations}$$

$$= (262 + 347) / 650$$

$$= 609 / 650$$

$$= 0.93$$

As a result, we get an accuracy of 93 percent.

121. Write the equation and calculate the precision and recall rate.

Consider the same confusion matrix used in the previous question.

		actual		
		p	n	
predicted	P	262	15	False Positive
	N	26	347	True Negative
			True Positive	
			False Negative	

Precision = (True positive) / (True Positive + False Positive)

$$= 262 / 277$$

$$= 0.94$$

Recall Rate = (True Positive) / (Total Positive + False Negative)

$$= 262 / 288$$

$$= 0.90$$

122. 'People who bought this also bought...' recommendations seen on Amazon are a result of which algorithm?

The recommendation engine is accomplished with collaborative filtering. Collaborative filtering explains the behavior of other users and their purchase history in terms of ratings, selection, etc.

The engine makes predictions on what might interest a person based on the preferences of other users. In this algorithm, item features are unknown.



For example, a sales page shows that a certain number of people buy a new phone and also buy tempered glass at the same time. Next time, when a person buys a phone, he or she may see a recommendation to buy tempered glass as well.

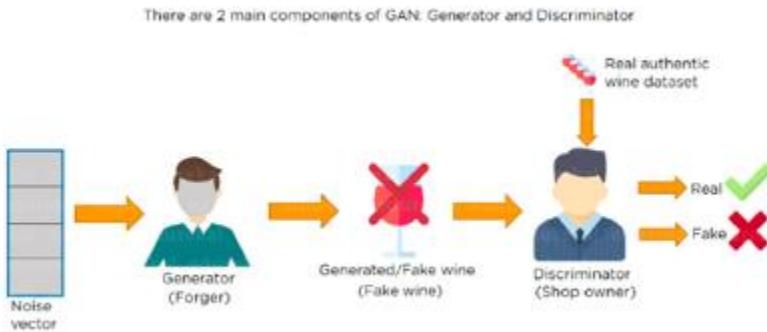
123. What is a Generative Adversarial Network?

Suppose there is a wine shop purchasing wine from dealers, which they resell later. But some dealers sell fake wine. In this case, the shop owner should be able to distinguish between fake and authentic wine.

The forger will try different techniques to sell fake wine and make sure specific techniques go past the shop owner's check. The shop owner would probably get some feedback from wine experts that some of the wine is not original. The owner would have to improve how he determines whether a wine is fake or authentic.

The forger's goal is to create wines that are indistinguishable from the authentic ones while the shop owner intends to tell if the wine is real or not accurately

Let us understand this example with the help of an image.



There is a noise vector coming into the forger who is generating fake wine.

Here the forger acts as a Generator.

The shop owner acts as a Discriminator.

The Discriminator gets two inputs; one is the fake wine, while the other is the real authentic wine. The shop owner has to figure out whether it is real or fake.

So, there are two primary components of Generative Adversarial Network (GAN) named:

1. Generator
2. Discriminator

The generator is a CNN that keeps keys producing images and is closer in appearance to the real images while the discriminator tries to determine the difference between real and fake images. The ultimate aim is to make the discriminator learn to identify real and fake images.

Apart from the very technical questions, your interviewer could even hit you up with a few simple ones to check your overall confidence, in the likes of the following.

124. You are given a dataset on cancer detection. You have built a classification model and achieved an accuracy of 96 percent. Why shouldn't you be happy with your model performance? What can you do about it?

Cancer detection results in imbalanced data. In an imbalanced dataset, accuracy should not be based as a measure of performance. It is important to focus on the remaining four percent, which represents the patients who were wrongly diagnosed. Early diagnosis is crucial when it comes to cancer detection, and can greatly improve a patient's prognosis.

Hence, to evaluate model performance, we should use Sensitivity (True Positive Rate), Specificity (True Negative Rate), F measure to determine the class wise performance of the classifier.

125. Which of the following machine learning algorithms can be used for inputting missing values of both categorical and continuous variables?

- K-means clustering
- Linear regression
- K-NN (k-nearest neighbor)
- Decision trees

The K nearest neighbor algorithm can be used because it can compute the nearest neighbor and if it doesn't have a value, it just computes the nearest neighbor based on all the other features.

When you're dealing with K-means clustering or linear regression, you need to do that in your pre-processing, otherwise, they'll crash. Decision trees also have the same problem, although there is some variance.

126. Below are the eight actual values of the target variable in the train file. What is the entropy of the target variable?

[0, 0, 0, 1, 1, 1, 1, 1]

Choose the correct answer.

1. $-(5/8 \log(5/8) + 3/8 \log(3/8))$
2. $5/8 \log(5/8) + 3/8 \log(3/8)$
3. $3/8 \log(5/8) + 5/8 \log(3/8)$
4. $5/8 \log(3/8) - 3/8 \log(5/8)$

The target variable, in this case, is 1.

The formula for calculating the entropy is:

Putting p=5 and n=8, we get

$$\text{Entropy} = A = -(5/8 \log(5/8) + 3/8 \log(3/8))$$

127. We want to predict the probability of death from heart disease based on three risk factors: age, gender, and blood cholesterol level. What is the most appropriate algorithm for this case?

Choose the correct option:

1. Logistic Regression
2. Linear Regression
3. K-means clustering
4. Apriori algorithm

The most appropriate algorithm for this case is A, logistic regression.

128. After studying the behavior of a population, you have identified four specific individual types that are valuable to your study. You would like to find all users who are most similar to each individual type. Which algorithm is most appropriate for this study?

Choose the correct option:

1. K-means clustering
2. Linear regression
3. Association rules
4. Decision trees

As we are looking for grouping people together specifically by four different similarities, it indicates the value of k. Therefore, K-means clustering (answer A) is the most appropriate algorithm for this study.

129. You have run the association rules algorithm on your dataset, and the two rules {banana, apple} => {grape} and {apple, orange} => {grape} have been found to be relevant. What else must be true?

Choose the right answer:

1. {banana, apple, grape, orange} must be a frequent itemset
2. {banana, apple} => {orange} must be a relevant rule
3. {grape} => {banana, apple} must be a relevant rule
4. {grape, apple} must be a frequent itemset

The answer is A: {grape, apple} must be a frequent itemset

130. Your organization has a website where visitors randomly receive one of two coupons. It is also possible that visitors to the website will not receive a coupon. You have been asked to determine if offering a coupon to website visitors has any impact on their purchase decisions. Which analysis method should you use?

1. One-way ANOVA
2. K-means clustering

3. Association rules

4. Student's t-test

The answer is A: One-way ANOVA

Additional Data Science Interview Questions on Basic Concepts

131. What are the feature vectors?

A feature vector is an n-dimensional vector of numerical features that represent an object. In machine learning, feature vectors are used to represent numeric or symbolic characteristics (called features) of an object in a mathematical way that's easy to analyze.

132. What are the steps in making a decision tree?

1. Take the entire data set as input.
2. Look for a split that maximizes the separation of the classes. A split is any test that divides the data into two sets.
3. Apply the split to the input data (divide step).
4. Re-apply steps one and two to the divided data.
5. Stop when you meet any stopping criteria.
6. This step is called pruning. Clean up the tree if you went too far doing splits.

133. What is root cause analysis?

Root cause analysis was initially developed to analyze industrial accidents but is now widely used in other areas. It is a problem-solving technique used for isolating the root causes of faults or problems. A factor is called a root cause if its deduction from the problem-fault-sequence averts the final undesirable event from recurring.

134. What is logistic regression?

Logistic regression is also known as the logit model. It is a technique used to forecast the binary outcome from a linear combination of predictor variables.

135. What are recommender systems?

Recommender systems are a subclass of information filtering systems that are meant to predict the preferences or ratings that a user would give to a product.

136. Explain cross-validation.

Cross-validation is a model validation technique for evaluating how the outcomes of a statistical analysis will generalize to an independent data set. It is mainly used in backgrounds where the objective is to forecast and one wants to estimate how accurately a model will accomplish in practice.

The goal of cross-validation is to term a data set to test the model in the training phase (i.e. validation data set) to limit problems like overfitting and gain insight into how the model will generalize to an independent data set.

137. What is collaborative filtering?

Most recommender systems use this filtering process to find patterns and information by collaborating perspectives, numerous data sources, and several agents.

138. Do gradient descent methods always converge to similar points?

They do not, because in some cases, they reach a local minima or a local optima point. You would not reach the global optima point. This is governed by the data and the starting conditions.

139. What is the goal of A/B Testing?

This is statistical hypothesis testing for randomized experiments with two variables, A and B. The objective of A/B testing is to detect any changes to a web page to maximize or increase the outcome of a strategy.

140. What are the drawbacks of the linear model?

- The assumption of linearity of the errors
- It can't be used for count outcomes or binary outcomes
- There are overfitting problems that it can't solve

141. What is the law of large numbers?

It is a theorem that describes the result of performing the same experiment very frequently. This theorem forms the basis of frequency-style thinking. It states that the sample mean, sample variance and sample standard deviation converge to what they are trying to estimate.

142. What are the confounding variables?

These are extraneous variables in a statistical model that correlates directly or inversely with both the dependent and the independent variable. The estimate fails to account for the confounding factor.

143. What is star schema?

It is a traditional database schema with a central table. Satellite tables map IDs to physical names or descriptions and can be connected to the central fact table using the ID fields; these tables are known as

lookup tables and are principally useful in real-time applications, as they save a lot of memory. Sometimes, star schemas involve several layers of summarization to recover information faster.

144. How regularly must an algorithm be updated?

You will want to update an algorithm when:

- You want the model to evolve as data streams through infrastructure
- The underlying data source is changing
- There is a case of non-stationarity

145. What are eigenvalue and eigenvector?

Eigenvalues are the directions along which a particular linear transformation acts by flipping, compressing, or stretching.

Eigenvectors are for understanding linear transformations. In data analysis, we usually calculate the eigenvectors for a correlation or covariance matrix.

146. Why is resampling done?

Resampling is done in any of these cases:

- Estimating the accuracy of sample statistics by using subsets of accessible data, or drawing randomly with replacement from a set of data points
- Substituting labels on data points when performing significance tests
- Validating models by using random subsets (bootstrapping, cross-validation)

147. What is selection bias?

Selection bias, in general, is a problematic situation in which error is introduced due to a non-random population sample.

148. What are the types of biases that can occur during sampling?

1. Selection bias
2. Undercoverage bias
3. Survivorship bias

149. What is survivorship bias?

Survivorship bias is the logical error of focusing on aspects that support surviving a process and casually overlooking those that did not because of their lack of prominence. This can lead to wrong conclusions in numerous ways.

150. How do you work towards a random forest?

The underlying principle of this technique is that several weak learners combine to provide a strong learner. The steps involved are:

1. Build several decision trees on bootstrapped training samples of data
2. On each tree, each time a split is considered, a random sample of m predictors is chosen as split candidates out of all p predictors
3. Rule of thumb: At each split $m=p\sqrt{m}=p$
4. Predictions: At the majority rule

151. What are the important skills to have in Python with regard to data analysis?

The following are some of the important skills to possess which will come handy when performing data analysis using Python.

- Good understanding of the built-in data types especially lists, dictionaries, tuples, and sets.
- Mastery of N-dimensional [NumPy Arrays](#).
- Mastery of [Pandas](#) dataframes.
- Ability to perform element-wise vector and matrix operations on NumPy arrays.
- Knowing that you should use the Anaconda distribution and the conda package manager.
- Familiarity with [Scikit-learn](#). [**Scikit-Learn Cheat Sheet**](#)
- Ability to write efficient list comprehensions instead of traditional for loops.
- Ability to write small, clean functions (important for any developer), preferably pure functions that don't alter objects.
- Knowing how to profile the performance of a Python script and how to optimize bottlenecks.

Credit: kdnuggets, Simplilearn, Edureka, Guru99, Hackeroon, Datacamp, Nitin Panwar, Michael Rundell