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

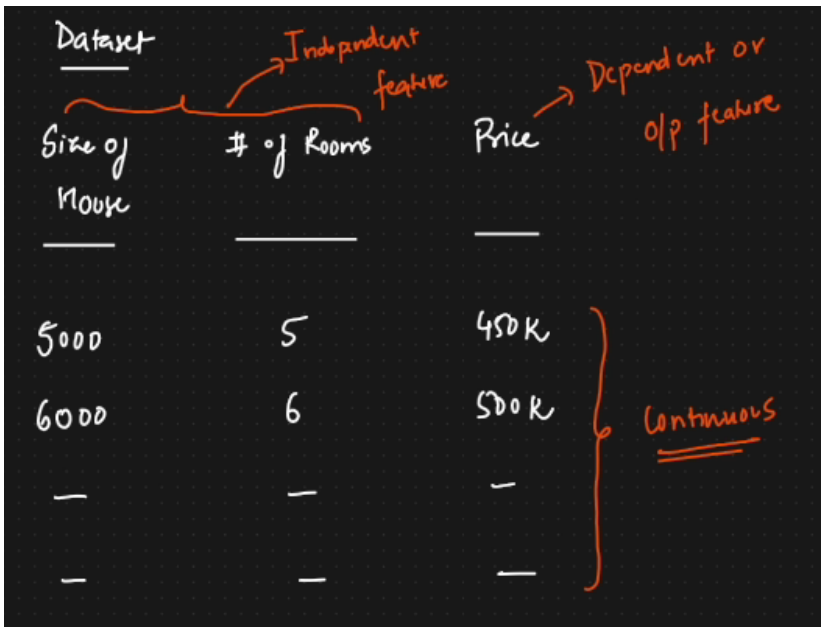
ML:- It provides stats to analyze, visualize, predication and forecasting the data.

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:-

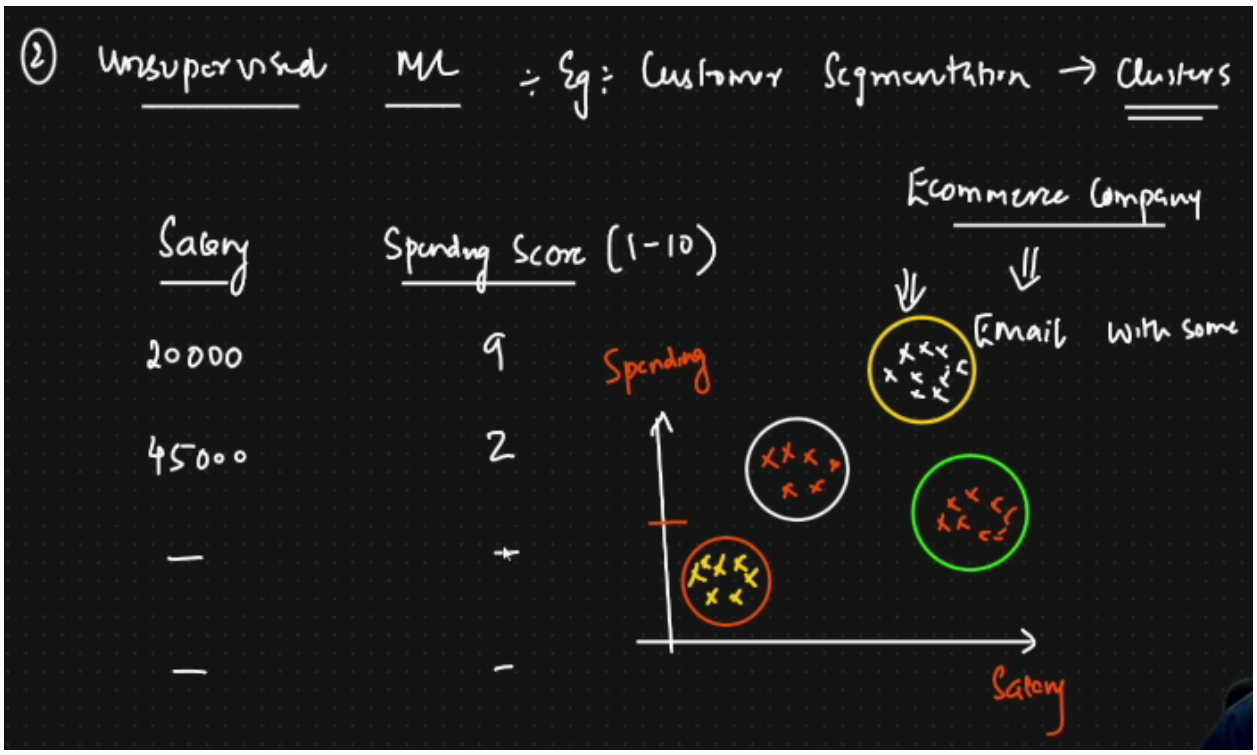
House Price Prediction



A. Classification Problem:
B. Regression Problem:

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

2. Unsupervised Machine Learning:-



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. Decsion Tree
6. RandomForest
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.Doesnt 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

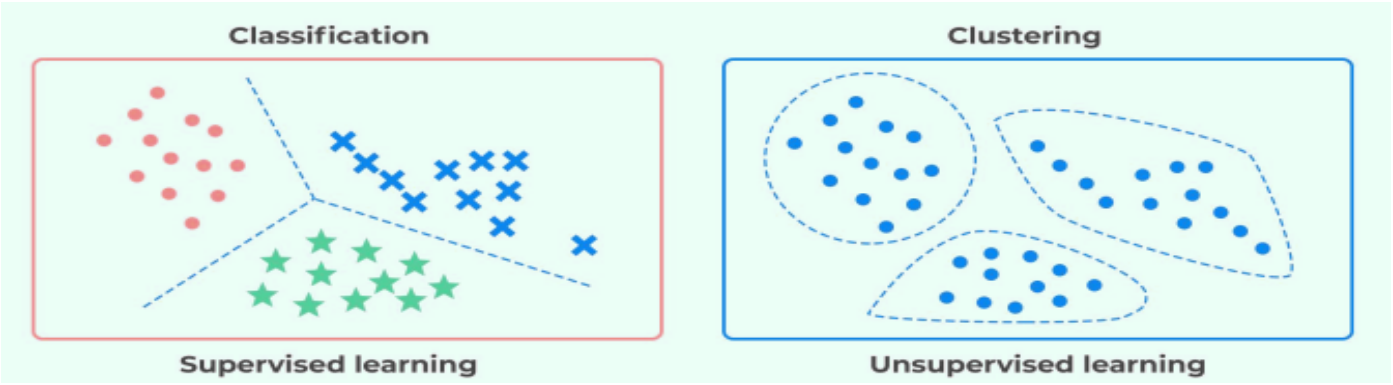
Advantages
1. Results are intuitive and Interpretable
2. Exhaustive approach as it finds all rules based on the confidence and support

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.

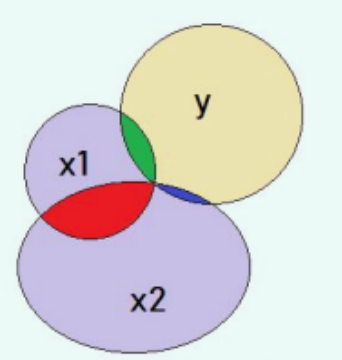
10. What is the importance of feature scaling in machine learning?
Feature scaling ensures that the features are at a similar scale, preventing certain features from dominating the learning process and helping the algorithm converge faster.

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.

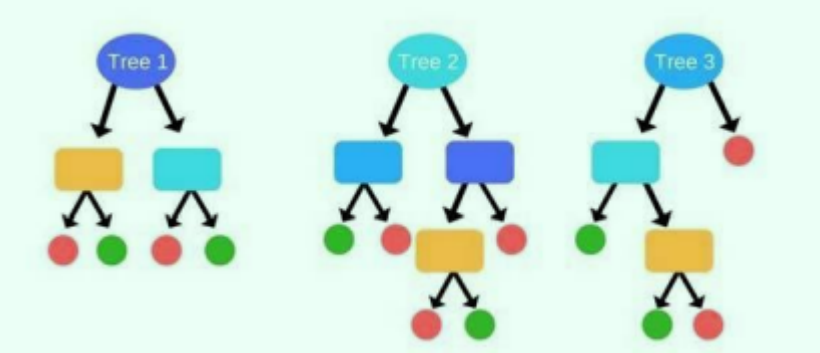
18. Explain the concept of the Central Limit Theorem.
The Central Limit Theorem states that the sampling distribution of the sample means approaches a normal distribution as the sample size increases, regardless of the shape of the population distribution.

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.

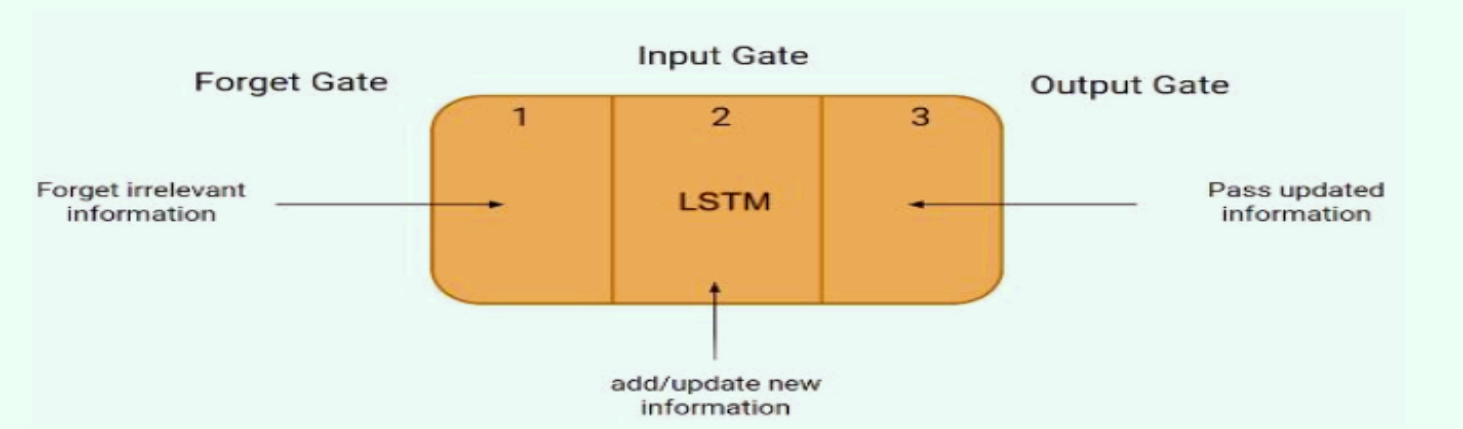
26. Can you explain the purpose of a recommendation system?
Recommendation systems are used to predict and recommend items or products that a user may be interested in, based on their past preferences or behaviour.

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.

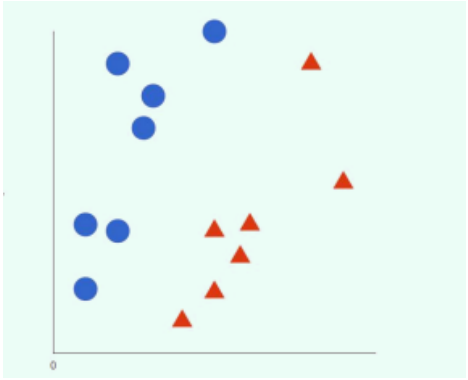
30. Explain the concept of a Long Short-Term Memory (LSTM) network.
LSTM networks are a type of RNN that addresses the vanishing gradient problem, making them more effective for learning and predicting sequences of data.



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.

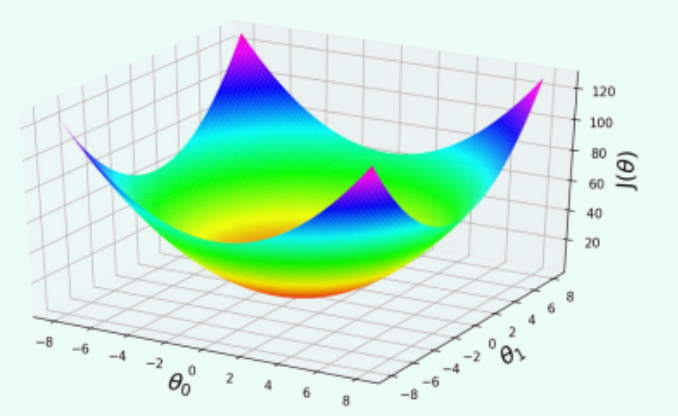
37. What is the purpose of the Mean Average Precision (MAP) metric in evaluating information retrieval systems?
Mean Average Precision is used to evaluate the performance of information retrieval systems, measuring the average precision at each relevant document retrieved across multiple queries.

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.

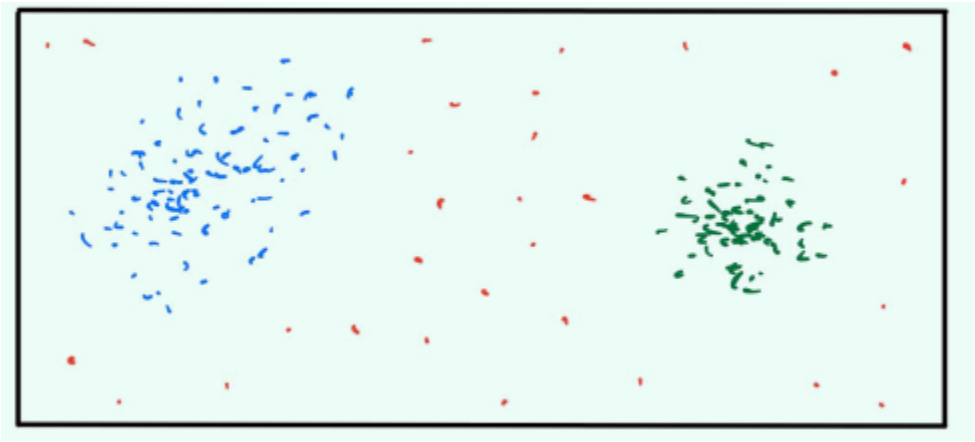
44. Explain the purpose of the Log Loss metric in evaluating classification models.
Log Loss is used to evaluate the performance of a classification model that outputs probabilities, measuring the performance based on the likelihood of the predicted probabilities matching the actual labels.

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.

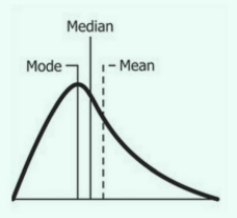
55. What is the purpose of the term 'gradient descent' in the context of optimising a model?
Gradient descent is an iterative optimization algorithm used to minimize the cost function of a model by adjusting the model's parameters in the direction of steepest descent.

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.

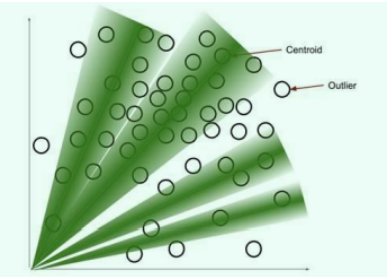
72. What is the purpose of 'ANOVA' (Analysis of Variance) in statistical analysis, and when is it used?
ANOVA is used to analyse the differences among group means and is applied when comparing means of more than two groups to determine whether they are statistically significantly different.

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
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
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
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
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
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
```

```python
    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
```

```python
    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

```python
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

```python
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

```python
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
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
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
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
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
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
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
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.friedmanchisquare(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.

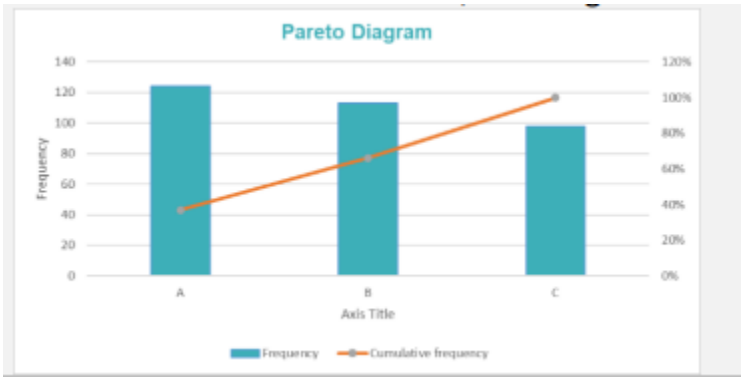
Frequency Distribution Table

A table showing the frequency of each variable.

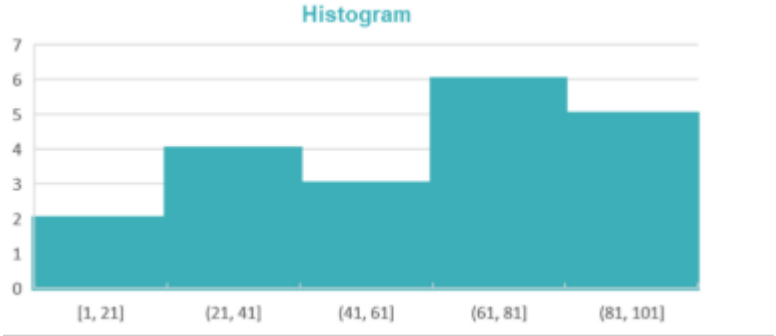
Frequency Distribution Table

	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.

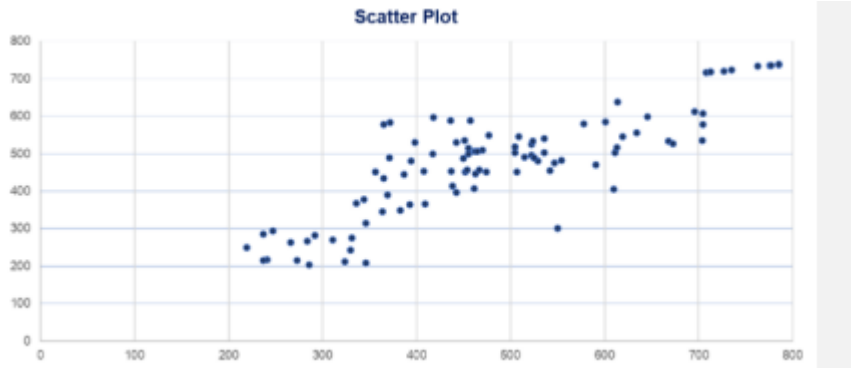


Cross Table (Contingency Table):- A table in a matrix format that displays the frequency distribution of the variables.

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.

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

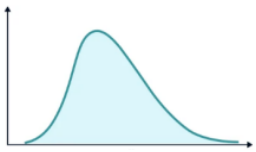
Median:- The middle number in a data set sorted in ascending or descending order.

—————————————>>>

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.

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

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

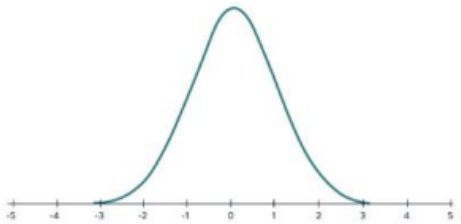
Coefficient of Variation:- Measures the dispersion of the dataset around its mean. The coefficient of variation is unitless. Therefore, it is useful when comparing the dispersion across different datasets that have different units of measurement.

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

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

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 ρxy for a population and rxy 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.

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

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 - α.
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.

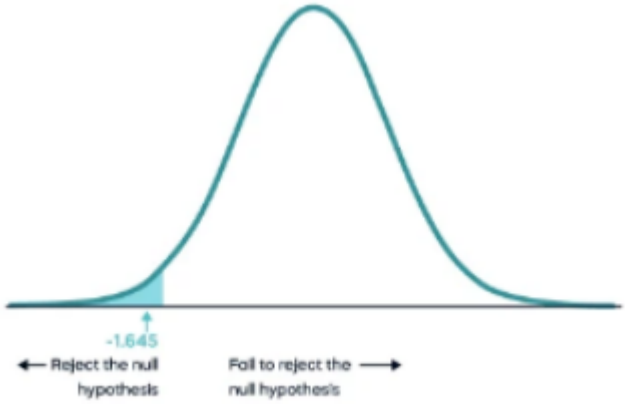
z	.04	.05	.06	.07
1.8	.9671	.9678	.9686	.9693
1.9	.9738	.9744	.9750	.9756
2.0	.9793	.9798	.9803	.9808

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.

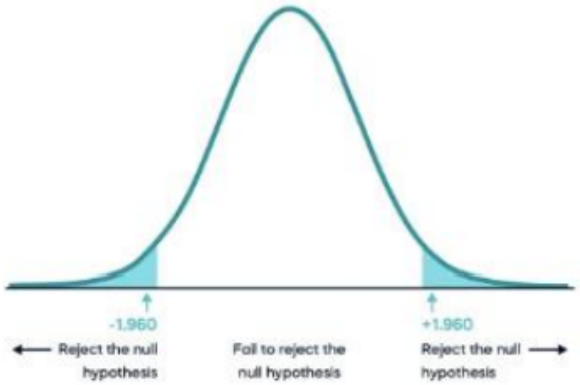
df	.20	.10	.05	.025	.01	0.05
18	0.862	1.330	1.734	2.101	2.552	2.878
19	0.861	1.328	1.729	2.093	2.539	2.861
20	0.860	1.325	1.725	2.086	2.528	2.845

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.

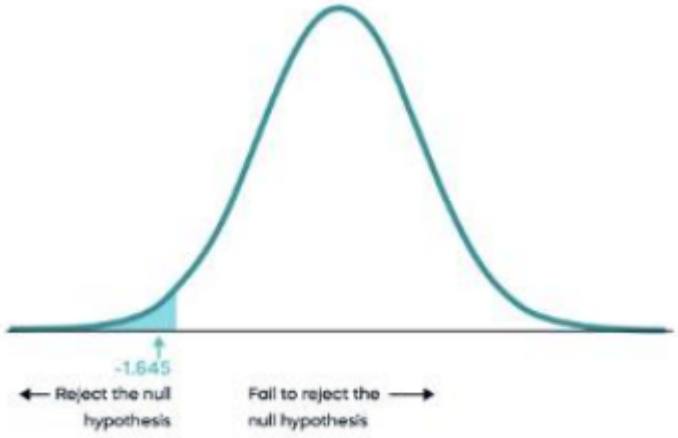
One-Tailed (One-Sided) Test:- A test that examines if a parameter is greater than or less than a specified value. In a one-tailed test, the alternative hypothesis focuses on a specific difference (higher than, lower than, or equal to).



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.

Decison	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.

Decison	H_0 True	H_0 False
Fail to reject H_0	Confidence level (1-α)	β
Reject H_0	Level of significance α	Power of the test (1-β)

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

Decison	H_0 True	H_0 False
Fail to reject H_0	Confidence level (1-α)	β
Reject H_0	Level of significance α	Power of the test (1-β)

z-score:- A value indicating how many standard deviations an element is from the mean.
μ0:- 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 $\rightarrow \frac{\binom{X}{x}\binom{N-X}{n-x}}{\binom{N}{n}}$, 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 $\rightarrow \frac{1}{b-a}$ with $\mu = \frac{a+b}{2}$ and $\sigma^2 = \frac{(b-a)^2}{12}$ or $\frac{n^2-1}{12}$ 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 = Bias^2 + 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) $= \sum(y_i - \bar{y})^2$

$\boldsymbol{R^2} = 1 - \frac{SSE}{SST}$, 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 $SS_{residual}{+}SS_{error} \neq SST$.

Adjusted $\boldsymbol{R^2}$ $= 1-(1-R^2)\frac{N-1}{N-p-1}$, 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 $\hat{\beta} = (X^TX)^{-1}X^TY$ 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 (L_0): $\lambda||\hat{\beta}||_0 = \lambda(number\ of\ non{-}zero\ variables)$

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

LASSO (L_1): $\lambda||\hat{\beta}||_1 = \lambda\sum|\hat{\beta}|$

- Shrinks coefficients to zero, and is robust to outliers

Ridge (L_2): $\lambda||\hat{\beta}||_2 = \lambda\sum(\hat{\beta})^2$

- 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

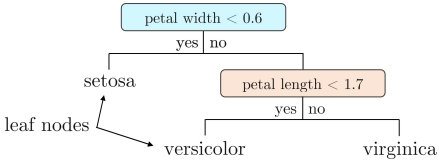
Odds - output probability can be transformed using $Odds(Y=1) = \frac{P(Y=1)}{1-P(Y=1)}$, where $P(\frac{1}{3})$ = 1:2 odds

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.

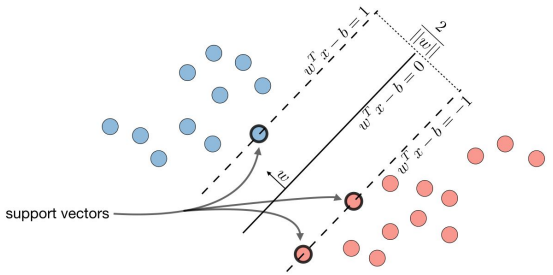
Deep trees may overfit, but adding more trees does not cause overfitting. Model bias is always equal to one of its individual trees.

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

Aaron Wang

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:

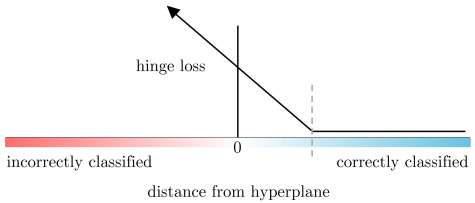


Support Vector Classifiers - account for outliers through the regularization parameter C, which penalizes misclassifications in the margin by a factor of $C > 0$

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 groups similar data points together based on distance

k-Means

Randomly place k centroids across normalized data, and assig 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

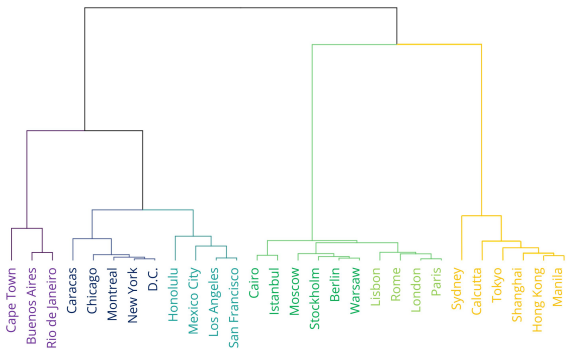
1. Each observation starts in its own cluster
2. Iteratively combine the most similar cluster pairs
3. Continue until all points are in the same cluster

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



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 scaler 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

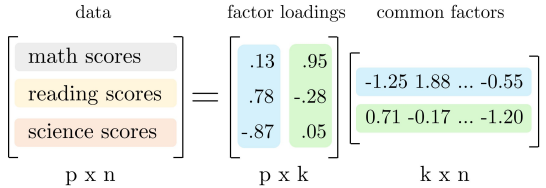
Note, the number of components is at most $C_1 - 1$

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.



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.

Aaron Wang

Natural Language Processing

Transforms human language into machine-usable code
Processing Techniques

- Tokenization - splits text into individual words (tokens)
- Lemmatization - reduces words to its base form based on dictionary definition (*am, are, is* $\rightarrow$ *be*)
- Stemming - reduces words to its base form without context (*ended* $\rightarrow$ *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 o 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 $king - man + woman \approx queen$
Relies 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-occurence 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'

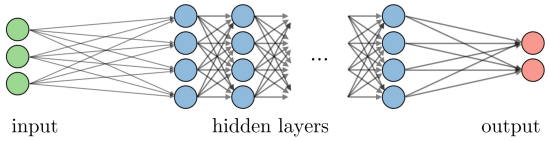
Sentiment - measures emotional states such as happy or sad
Subject-Object Identification - classifies sentences as either subjective or objective

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

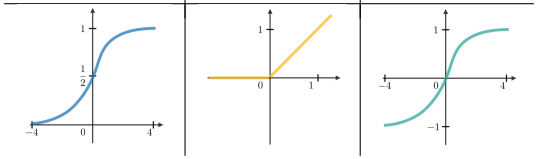
Neural Network

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

Sigmoid	ReLU	Tanh
$\frac{1}{1+e^{-z}}$	$\max(0, z)$	$\frac{e^z - e^{-z}}{e^z + e^{-z}}$



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

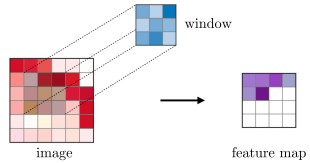
To prevent overfitting, regularization can be applied by:

- 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 features maps.



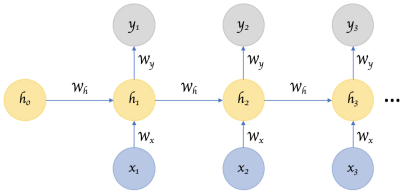
Pooling - downsamples convolution layers to reduce dimensionality and maintain spatial invariance, allowing detection of features even if they have shifted slightly. Common techniques return the max or average value in the pooling window.

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.

Aaron Wang

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

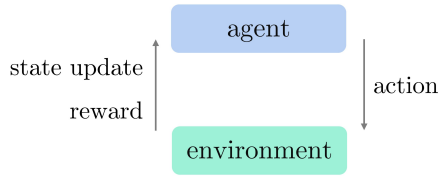
Model-based Approaches - predict ratings of unrated items, through methods such as Bayesian networks, SVD, and clustering. Handles sparse data better than memory-based approaches.

- 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 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

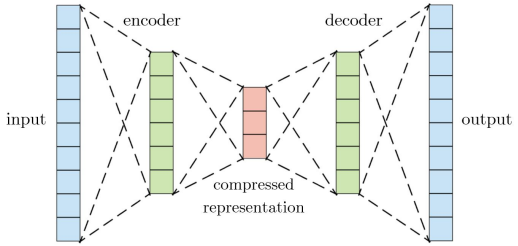
Anomaly detection techniques spans a wide range, including methods based on:
Statistics - relies on various statistical methods to identify outliers, such as Z-tests, boxplots, interquartile ranges, and variance comparisons
Density - useful when data is grouped around dense neighborhoods, measured by 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
- Cyclicality - 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

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

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 season 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 $y = \beta_0 + f_1(x_1) + \cdots + f_m(x_m)$, 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.

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 - α) - probability of finding an effect that did not occur by chance and avoiding a Type I error

Power (1 - β) - 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 $\chi^2 = \sum \frac{observed-expected}{expected}$ 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 $P(A \mid B) = \frac{P(A \cap B)}{P(B)}$

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 $\rightarrow r = \frac{Cov(x,y)}{\sigma_x \sigma_y}$

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 - β
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 $\rightarrow \frac{n!}{(n-k)!} = {}^nP_k$

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

Left Skew - Mean < Median ≤ Mode

Right Skew - Mean > Median ≥ 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, $L(\theta|O) = P(O|\theta)$.

finxter

The Ultimate Python Cheat Sheet

Keywords

Keyword	Description	Code Examples
`False`, `True`	Boolean data type	`False == (1 > 2)` `True == (2 > 1)`

`and`, `or`, `not`	Logical operators
	→ Both are true
	→ Either is true
	→ Flips Boolean
		`True and True  # True`
		`True or False  # True`
		`not False  # True`

`break`	Ends loop prematurely	`while True:` `  break # finite loop`

`continue`	Finishes current loop iteration	`while True:`
		`  continue`
		`  print("42") # dead code`

`class`	Defines new class	`class Coffee:` `  # Define your class`
`def`	Defines a new function or class method.	`def say_hi():` `  print('hi')`

`if`, `elif`, `else`	Conditional execution:
	- "if" condition == True?
	- "elif" condition == True?
	- Fallback: else branch
		`x = int(input("ur val:"))`
		`if  x > 3:  print("Big")`
		`elif x == 3:  print("3")`
		`else:  print("Small")`

`for`, `while`	`# For loop`
	`for i in [0,1,2]:`
	`  print(i)`
		`# While loop does same`
		`j = 0`
		`while j < 3:`
		`  print(j); j = j + 1`

`in`	Sequence membership	`42 in [2, 39, 42] # True`

`is`	Same object memory location	`y = x = 3`
		`x is y  # True`
		`[3] is [3]  # False`

`None`	Empty value constant	`print() is None # True`
`lambda`	Anonymous function	`(lambda x: x+3)(3) # 6`

`return`	Terminates function. Optional return value defines function result.	`def increment(x):`
		`  return x + 1`
		`increment(4) # returns 5`

Basic Data Structures

Type	Description	Code Examples

Boolean	The Boolean data type is either `True` or `False`. Boolean operators are ordered by priority: `not` → `and` → `or`
	{ } → 👎
	{1, 2, 3} → 👍
		`## Evaluates to True:`
		`1<2 and 0<=1 and 3>2 and 2>=2 and 1==1 and 1!=0`
		`## Evaluates to False:`
		`bool(None or 0 or 0.0 or '' or [] or {} or set())`
		Rule: *None, 0, 0.0, empty strings, or empty container types evaluate to False*

Integer, Float	An **integer** is a positive or negative number without decimal point such as `3`.
	A **float** is a positive or negative number with floating point precision such as `3.1415926`.
	Integer division rounds toward the smaller integer (example: `3//2==1`).
		`## 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)  # = 1`
		`print(-x)  # = -3`
		`print(abs(-x))  # = 3`
		`print(int(3.9))  # = 3`
		`print(float(3))  # = 3.0`
		`print(x ** y)  # = 9`

String	Python Strings are sequences of characters.
	String Creation Methods:
	1. Single quotes
	`>>> 'Yes'`
	2. Double quotes
	`>>> "Yes"`
	3. Triple quotes (multi-line)
	`>>> """Yes`
	`  We Can"""`
	4. String method
	`>>> str(5) == '5'`
	`True`
	5. Concatenation
	`>>> "Ma" + "hatma"`
	`'Mahatma'`
	Whitespace chars:
	Newline \n,
	Space \s,
	Tab \t
		`## Indexing and Slicing`
		`s = "The youngest pope was 11 years"`
		`s[0] # 'T'`
		`s[1:3] # 'he'`
		`s[-3:-1] # 'ar'`
		`s[-3:] # 'ars'`
		`x = s.split()`
		`x[-2] + " " + x[2] + "s" # '11 popes'`
		Slice [::2]: 1 2 3 4 / 0 1 2 3
		`## String Methods`
		`y = "  Hello world\t\n  "`
		`y.strip() # Remove Whitespace`
		`"HI".lower() # Lowercase: 'hi'`
		`"hi".upper() # Uppercase: 'HI'`
		`"hello".startswith("he") # True`
		`"hello".endswith("lo") # True`
		`"hello".find("ll") # Match at 2`
		`"cheat".replace("ch", "m") # 'meat'`
		`''.join(["F", "B", "I"]) # 'FBI'`
		`len("hello world") # Length: 15`
		`"ear" in "earth" # True`

Complex Data Structures

Type	Description	Example
List	Stores a sequence of elements. Unlike strings, you can modify list objects (they're *mutable*).	`l = [1, 2, 2]` `print(len(l)) # 3`

Adding elements	Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation.	`[1, 2].append(4) # [1, 2, 4]`
		`[1, 4].insert(1,9) # [1, 9, 4]`
		`[1, 2] + [4] # [1, 2, 4]`

Removal	Slow for lists	`[1, 2, 2, 4].remove(1) # [2, 2, 4]`
Reversing	Reverses list order	`[1, 2, 3].reverse() # [3, 2, 1]`
Sorting	Sorts list using fast Timsort	`[2, 4, 2].sort() # [2, 2, 4]`

Indexing	Finds the first occurrence of an element & returns index. Slow worst case for whole list traversal.	`[2, 2, 4].index(2)`
		`# index of item 2 is 0`
		`[2, 2, 4].index(2,1)`
		`# index of item 2 after pos 1 is 1`

Stack	Use Python lists via the list operations append() and pop()	`stack = [3]`
		`stack.append(42) # [3, 42]`
		`stack.pop() # 42 (stack: [3])`
		`stack.pop() # 3 (stack: [])`

Set	An unordered collection of unique elements (*at-most-once*) → fast membership *O(1)*	`basket = {'apple', 'eggs',`
		`  'banana', 'orange'}`
		`same = set(['apple', 'eggs',`
		`  'banana', 'orange'])`

Type	Description	Example
Dictionary	Useful data structure for storing (key, value) pairs	`cal = {'apple' : 52, 'banana' : 89,` `  'choco' : 546} # calories`

Reading and writing elements	Read and write elements by specifying the key within the brackets. Use the **`keys()`** and **`values()`** functions to access all keys and values of the dictionary	`print(cal['apple'] < cal['choco'])`
		`# True`
		`cal['cappu'] = 74`
		`print(cal['banana'] < cal['cappu'])`
		`# False`
		`print('apple' in cal.keys()) # True`
		`print(52 in cal.values())  # True`

Dictionary Iteration	You can access the (key, value) pairs of a dictionary with the **`items()`** method.	`for k, v in cal.items():`
		`  print(k) if v > 500 else ''`
		`# 'choco'`

Membership operator	Check with the **`in`** keyword if set, list, or dictionary contains an element. Set membership is faster than list membership.	`basket = {'apple', 'eggs',`
		`  'banana', 'orange'}`
		`print('eggs' in basket)  # True`
		`print('mushroom' in basket) # False`

List & 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 works similar to list comprehension.	`l = ['hi ' + x for x in ['Alice', 'Bob', 'Pete']]`
		`# ['Hi Alice', 'Hi Bob', 'Hi Pete']`
		`l2 = [x * y for x in range(3) for y in range(3) if x>y] # [0, 0, 2]`
		`squares = { x**2 for x in [0,2,4] if x < 4 } # {0, 4}`

Python Cheat Sheet: Keywords

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

Keyword	Description	Code example
`False, True`	Data values from the data type Boolean	`False == (1 > 2), True == (2 > 1)`

`and, or, not`	Logical operators:
	`(x and y)` → both x and y must be True
	`(x or y)` → either x or y must be True
	`(not x)` → x must be false
		`x, y = True, False`
		`(x or y) == True  # True`
		`(x and y) == False  # True`
		`(not y) == True  # True`

`break`	Ends loop prematurely	`while(True):`
		`  break # no infinite loop`
		`print("hello world")`

`continue`	Finishes current loop iteration	`while(True):`
		`  continue`
		`  print("43") # dead code`

`class`
 	
`def`
	Defines a new class → a real-world concept (object oriented programming)
	
	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.
		`class Beer:`
		`  def __init__(self):`
		`  self.content = 1.0`
		`  def drink(self):`
		`  self.content = 0.0`
		
		`becks = Beer() # constructor - create class`
		`becks.drink() # beer empty: b.content == 0`

`if, elif, else`	Conditional program execution: program starts with "if" branch, tries the "elif" branches, and finishes with "else" branch (until one branch evaluates to True).	`x = int(input("your value: "))`
		`if x > 3: print("Big")`
		`elif x == 3: print("Medium")`
		`else: print("Small")`

`for, while`	`# For loop declaration`
	`for i in [0,1,2]:`
	`  print(i)`
		`# While loop - same semantics`
		`j = 0`
		`while j < 3:`
		`  print(j)`
		`  j = j + 1`

`in`	Checks whether element is in sequence	`42 in [2, 39, 42] # True`

`is`	Checks whether both elements point to the same object	`y = x = 3`
		`x is y # True`
		`[3] is [3] # False`

`None`	Empty value constant	`def f():`
		`  x = 2`
		`f() is None # True`

`lambda`	Function with no name (anonymous function)	`(lambda x: x + 3)(3) # returns 6`

`return`	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.	`def incrementor(x):`
		`  return x + 1`
		`incrementor(4) # returns 5`

finxter

Python Cheat Sheet: Basic Data Types

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

	Description	Example

Boolean	The Boolean data type is a truth value, either `True` or `False`.
	
	The Boolean operators ordered by priority:
	`not x` → "if x is False, then x, else y"
	`x and y` → "if x is False, then x, else y"
	`x or y` → "if x is False, then y, else x"
	
	These comparison operators evaluate to `True`:
	`1 < 2 and 0 <= 1 and 3 > 2 and 2 >=2 and 1 == 1 and 1 != 0 # True`
		`## 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`

Integer, Float	An integer is a positive or negative number without floating point (e.g. `3`). A float is a positive or negative number with floating point precision (e.g. `3.14159265359`).
	
	The '`//`' operator performs integer division. The result is an integer value that is rounded toward the smaller integer number (e.g. `3 // 2 == 1`).
		`## 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`

String	Python Strings are sequences of characters.
	
	The four main ways to create strings are the following.
	
	1. Single quotes
	`'Yes'`
	2. Double quotes
	`"Yes"`
	3. Triple quotes (multi-line)
	`"""Yes`
	`We Can"""`
	4. String method
	`str(5) == '5' # True`
	5. Concatenation
	`"Ma" + "hatma" # 'Mahatma'`
	
	These are whitespace characters in strings.
	• Newline `\n`
	• Space `\s`
	• Tab `\t`
		`## 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`

finxter

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.	`l = [1, 2, 2]` `print(len(l)) # 3`

Adding elements	Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation.
	The append operation is very fast.
		`[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]`

Removal	Removing an element can be slower.	`[1, 2, 2, 4].remove(1) # [2, 2, 4]`
Reversing	This reverses the order of list elements.	`[1, 2, 3].reverse() # [3, 2, 1]`
Sorting	Sorts a list. The computational complexity of sorting is linear in the no. list elements.	`[2, 4, 2].sort() # [2, 2, 4]`
Indexing	Finds the first occurence of an element in the list & returns its index. Can be slow as the whole list is traversed.	`[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"`

Stack	Python lists can be used intuitively as stacks via the two list operations append() and pop().	`stack = [3]`
		`stack.append(42) # [3, 42]`
		`stack.pop() # 42 (stack: [3])`
		`stack.pop() # 3 (stack: [])`

Set	A set is an unordered collection of unique elements ("at-most-once").	`basket = {'apple', 'eggs', 'banana', 'orange'}` `same = set(['apple', 'eggs', 'banana', 'orange'])`
Dictionary	The dictionary is a useful data structure for storing (key, value) pairs.	`calories = {'apple' : 52, 'banana' : 89, 'choco' : 546}`

Reading and writing elements	Read and write elements by specifying the key within the brackets. Use the keys() and values() functions to access all keys and values of the dictionary.	`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`

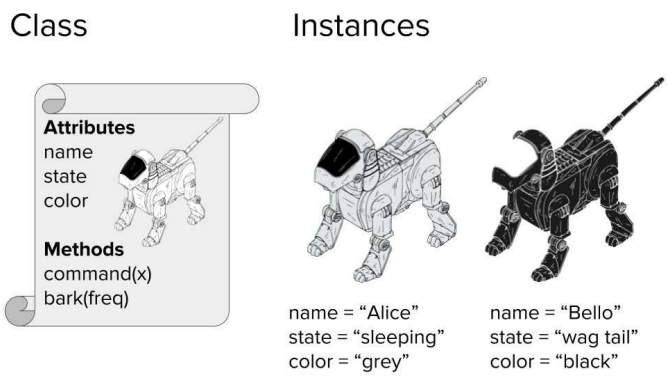
Dictionary Looping	You can access the (key, value) pairs of a dictionary with the `items()` method.	`for k, v in calories.items():` `  print(k) if v > 500 else None # 'chocolate'`

Membership operator	Check with the 'in' keyword whether the set, list, or dictionary contains an element. Set containment is faster than list containment.	`basket = {'apple', 'eggs', 'banana', 'orange'}`
		`print('eggs' in basket) # True`
		`print('mushroom' in basket) # False`

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.
		`# 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}`

Python Cheat Sheet: Classes

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

	Description	Example
Classes	A class encapsulates data and functionality: data as attributes, and functionality as methods. It is a blueprint for creating concrete instances in memory. 	
Instance	You are an instance of the class human. 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. 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 `species` in the example.	
Self	The first argument when defining any method is always the `self` argument. This argument specifies the instance on which you call the method. `self` gives the Python interpreter the information about the concrete instance. To *define* a method, you use `self` to modify the instance attributes. But to *call* an instance method, you do not need to specify `self`.	

Creation	You can create classes "on the fly" and use them as logical units to store complex data types.
	
	`class Employee():`
	`  pass`
	`employee = Employee()`
	`employee.salary = 122000`
	`employee.firstname = "alice"`
	`employee.lastname = "wonderland"`
	
	`print(employee.firstname + " "`
	`  + employee.lastname + " "`
	`  + str(employee.salary) + "$")`
	`# alice wonderland 122000$`	

```
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 (!)
```

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	`list(map(lambda x: x[0], ['red', 'green', 'blue']))`	`['r', 'g', 'b']`
	`map(func, i1, ..., ik)`	Executes the function on all k elements of the k iterables	`list(map(lambda x, y: str(x) + ' ' + y + 's' , [0, 2, 2], ['apple', 'orange', 'banana']))`	`['0 apples', '2 oranges', '2 bananas']`
	`string.join(iter)`	Concatenates iterable elements separated by `string`	`' marries '.join(list(['Alice', 'Bob']))`	`'Alice marries Bob'`
	`filter(func, iterable)`	Filters out elements in iterable for which function returns `False` (or 0)	`list(filter(lambda x: True if x>17 else False, [1, 15, 17, 18]))`	`[18]`
	`string.strip()`	Removes leading and trailing whitespaces of string	`print("\n  \t  42  \t ".strip())`	`42`
	`sorted(iter)`	Sorts iterable in ascending order	`sorted([8, 3, 2, 42, 5])`	`[2, 3, 5, 8, 42]`
	`sorted(iter, key=key)`	Sorts according to the key function in ascending order	`sorted([8, 3, 2, 42, 5], key=lambda x: 0 if x==42 else x)`	`[42, 2, 3, 5, 8]`
	`help(func)`	Returns documentation of `func`	`help(str.upper())`	`'... to uppercase.'`
	`zip(i1, i2, ...)`	Groups the i-th elements of iterators `i1, i2, ...` together	`list(zip(['Alice', 'Anna'], ['Bob', 'Jon', 'Frank']))`	`[('Alice', 'Bob'), ('Anna', 'Jon')]`
	Unzip	Equal to: 1) unpack the zipped list, 2) zip the result	`list(zip(*[('Alice', 'Bob'), ('Anna', 'Jon')]))`	`[('Alice', 'Anna'), ('Bob', 'Jon')]`
	`enumerate(iter)`	Assigns a counter value to each element of the iterable	`list(enumerate(['Alice', 'Bob', 'Jon']))`	`[(0, 'Alice'), (1, 'Bob'), (2, 'Jon')]`
TRICKS	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	`import antigravity`	Open the comic series xkcd in your web browser	
	Zen of Python	`import this`	`'...Beautiful is better than ugly. Explicit is ...'`	
	Swapping numbers	Swapping variables is a breeze in Python. No offense, Java!	`a, b = 'Jane', 'Alice'` `a, b = b, a`	`a = 'Alice'` `b = 'Jane'`

	Unpacking arguments	Use a sequence as function arguments via asterisk operator *. Use a dictionary (key, value) via double asterisk operator **	`def f(x, y, z): return x + y * z`
			`f(*[1, 3, 4])`
			`f(**{'z' : 4, 'x' : 1, 'y' : 3})`
				
				`13`
				`13`

	Extended Unpacking	Use unpacking for multiple assignment feature in Python	`a, *b = [1, 2, 3, 4, 5]`	`a = 1` `b = [2, 3, 4, 5]`

	Merge two dictionaries	Use unpacking to merge two dictionaries into a single one	`x={'Alice' : 18}`
			`y={'Bob' : 27, 'Ann' : 22}`
			`z = {**x,**y}`
				`z = {'Alice': 18, 'Bob': 27, 'Ann': 22}`

finxter

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	`l = [3, 3, 4, 5, 2, 111, 5]`
	`print(111 in l) # True`
		Get missing number in [1...100]	`def get_missing_number(lst):`
			`  return set(range(lst[len(lst)-1])[1:]) - set(l)`
			`l = list(range(1,100))`
			`l.remove(50)`
			`print(get_missing_number(l)) # 50`

Find duplicate number in integer list	`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)`
		Compute the intersection of two lists	`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`

Check if two strings are anagrams	`def is_anagram(s1, s2):`
	`  return set(s1) == set(s2)`
	`print(is_anagram("elvis", "lives")) # True`
		Find max and min in unsorted list	`l = [4, 3, 6, 3, 4, 888, 1, -11, 22, 3]`
			`print(max(l)) # 888`
			`print(min(l)) # -11`

Remove all duplicates from list	`lst = list(range(10)) + list(range(10))`
	`lst = list(set(lst))`
	`print(lst)`
	`# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
		Reverse string using recursion	`def reverse(string):`
			`  if len(string)<=1: return string`
			`  return reverse(string[1:])+string[0]`
			`print(reverse("hello")) # olleh`

Find pairs of integers in list so that their sum is equal to integer x	`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`
		Compute the first n Fibonacci numbers	`a, b = 0, 1`
			`n = 10`
			`for i in range(n):`
			`  print(b)`
			`  a, b = b, a+b`
			`# 1, 1, 2, 3, 5, 8, ...`

Check if a string is a palindrome	`def is_palindrome(phrase):`
	`  return phrase == phrase[::-1]`
	`print(is_palindrome("anna")) # True`
		Sort list with Quicksort algorithm	`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]`

Use list as stack, array, and queue	`# as a list ...`
	`l = [3, 4]`
	`l +=  [5, 6] # l = [3, 4, 5, 6]`
	
	`# ... as a stack ...`
	`l.append(10) # l = [4, 5, 6, 10]`
	`l.pop() # l = [4, 5, 6]`
	
	`# ... and as a queue`
	`l.insert(0, 5) # l = [5, 4, 5, 6]`
	`l.pop() # l = [5, 4, 5]`
		Find all permutations of string	`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'}`

finxter

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.	`a = np.array([[1,2],[1,1],[0,0]])` `print(np.shape(a))  # (3, 2)`
`a.ndim`	The ndim attribute is equal to the length of the shape tuple.	`print(np.ndim(a))  # 2`

`*`	The asterisk (star) operator performs the Hadamard product, i.e., multiplies two matrices with equal shape element-wise.	`a = np.array([[2, 0], [0, 2]])`
		`b = np.array([[1, 1], [1, 1]])`
		`print(a*b)  # [[2 0] [0 2]]`

`np.matmul(a,b), a@b`	The standard matrix multiplication operator. Equivalent to the @ operator.	`print(np.matmul(a,b))` `# [[2 2] [2 2]]`
`np.arange([start, ]stop, [step, ])`	Creates a new 1D numpy array with evenly spaced values	`print(np.arange(0,10,2))` `# [0 2 4 6 8]`
`np.linspace(start, stop, num=50)`	Creates a new 1D numpy array with evenly spread elements within the given interval	`print(np.linspace(0,10,3))` `# [ 0.  5. 10.]`
`np.average(a)`	Averages over all the values in the numpy array	`a = np.array([[2, 0], [0, 2]])` `print(np.average(a))  # 1.0`

`<slice> = <val>`	Replace the <slice> as selected by the slicing operator with the value <val>.	`a = np.array([0, 1, 0, 0, 0])`
		`a[::2] = 2`
		`print(a)  # [2 1 2 0 2]`

`np.var(a)`	Calculates the variance of a numpy array.	`a = np.array([2, 6])` `print(np.var(a))  # 4.0`
`np.std(a)`	Calculates the standard deviation of a numpy array	`print(np.std(a))  # 2.0`

`np.diff(a)`	Calculates the difference between subsequent values in NumPy array a	`fibs = np.array([0, 1, 1, 2, 3, 5])`
		`print(np.diff(fibs, n=1))`
		`# [1 0 1 1 2]`

`np.cumsum(a)`	Calculates the cumulative sum of the elements in NumPy array a.	`print(np.cumsum(np.arange(5)))` `# [ 0  1  3  6 10]`

`np.sort(a)`	Creates a new NumPy array with the values from a (ascending).	`a = np.array([10,3,7,1,0])`
		`print(np.sort(a))`
		`# [ 0  1  3  7 10]`

`np.argsort(a)`	Returns the indices of a NumPy array so that the indexed values would be sorted.	`a = np.array([10,3,7,1,0])`
		`print(np.argsort(a))`
		`# [4 3 1 2 0]`

`np.max(a)`	Returns the maximal value of NumPy array a.	`a = np.array([10,3,7,1,0])` `print(np.max(a))  # 10`
`np.argmax(a)`	Returns the index of the element with maximal value in the NumPy array a.	`a = np.array([10,3,7,1,0])` `print(np.argmax(a))  # 0`
`np.nonzero(a)`	Returns the indices of the nonzero elements in NumPy array a.	`a = np.array([10,3,7,1,0])` `print(np.nonzero(a))  # [0 1 2 3]`

finxter