1. ABSTRACT

This project aims to investigate various supervised and unsupervised algorithms for analyzing the Connect-4 game dataset. The dataset consists of legal game positions in Connect-4 where neither player has achieved victory yet, and the next move is not predetermined. The project commences with data visualization, including a bar plot representing the distribution of outcomes and a heatmap illustrating the game board positions. To pre-process the data, categorical features are encoded using one-hot encoding. Several supervised algorithms such as Logistic Regression, Decision Trees, Support Vector Machines, and Neural Networks are employed, alongside unsupervised learning techniques like K-means clustering. The performance of the supervised learning algorithms is evaluated based on accuracy, while the unsupervised clustering approach is evaluated using metrics such as the silhouette score and completeness score. This project aims to provide valuable insights into the efficacy of different techniques for analyzing the Connect-4 dataset.

2. INTRODUCTION

This project aims to explore and analyze the Connect-4 game dataset using a variety of supervised and unsupervised algorithms. Connect-4 is a popular two-player board game where the objective is to connect four discs of the same color in a row, column, or diagonal on a grid. The dataset used in this project comprises legal game positions in Connect-4, where neither player has achieved victory yet and the next move is not predetermined.

2.1. Goals:

The primary goal of this project is to examine the effectiveness of different techniques for analyzing the Connect-4 dataset. The project will employ both supervised learning algorithms, such as Logistic Regression, Decision Trees, Support Vector Machines, and Neural Networks, as well as unsupervised learning techniques, specifically K-means clustering.

2.2. Dataset Description

The dataset utilized in this project contains legal game positions in the Connect-4 game, focusing on situations where neither player has achieved victory yet and the next move is not forced. Each instance in the dataset represents a unique game state, characterized by various features such as the positions of the players' discs on the game board. The dataset provides the necessary information to explore strategies, patterns, and potential winning moves in Connect-4.

By employing a range of supervised and unsupervised algorithms, along with comprehensive data visualization techniques, this project seeks to uncover valuable insights into the Connect-4 dataset and shed light on the effectiveness of different analytical approaches.

3. THEORETICAL BACKGROUND

3.1. Logistic Regression

Logistic Regression is a popular and widely used statistical model for binary classification problems. It is a supervised learning algorithm that predicts the probability of an instance belonging to a particular class.

The theoretical foundation of Logistic Regression lies in the logistic function, also known as the sigmoid function. The sigmoid function maps any real-valued number to a value between 0 and 1, making it suitable for modelling probabilities. The logistic function is defined as:

$g(z) = 1 / (1 + e^{(-z)})$

In Logistic Regression, the goal is to find the optimal decision boundary that separates the two classes in the input feature space. The decision boundary is a hyperplane that divides the feature space into two regions, one for each class. The logistic function is employed to transform the linear combination of input features and corresponding weights into a probability value between 0 and 1.

The mathematical representation of Logistic Regression is as follows:

$p(y = 1 \mid x) = g(w^T x + b)$

where:

$p(y = 1 \mid x)$ represents the probability of the target variable y being 1 given the input feature vector x.

$g(z)$ is the logistic function.

w represents the weight vector associated with the input features.

x is the input feature vector.

b is the bias term.

3.2. Decision Trees

Decision trees are a popular machine learning algorithm for both classification and regression tasks. Decision trees are constructed by recursively splitting the data into subsets based on the values of a single attribute. At each node, the algorithm selects the attribute that best splits the data into the most homogeneous subsets (i.e., subsets with the lowest impurity or highest information gain). The process continues until the data is perfectly classified or until some stopping criterion is met.

3.2.1. Pruning

Pruning is a technique used to reduce the size of a decision tree by removing branches that do not significantly improve the accuracy of the model. Pruning can help reduce overfitting, which occurs when the model is too complex and captures noise in the training data.

3.2.2. Bagging

Bagging (Bootstrap Aggregating) is an ensemble method that uses bootstrap sampling to create multiple decision trees on different subsets of the training data. The predictions of these individual trees are then combined by taking their average (for regression) or majority vote (for classification). Bagging helps reduce variance and improve the accuracy of the model.

3.2.3. Random Forest

Random Forest is an extension of bagging that further reduces variance by introducing randomness into the model. Instead of using all features to split the data, only a random subset of features is considered at each node. This reduces the correlation between individual trees and improves the overall performance of the model.

3.2.4. Boosting

Boosting is another ensemble method that iteratively builds a series of weak models and combines them to form a strong model. At each iteration, the algorithm places more emphasis on the data that was misclassified by the previous model, thus improving its accuracy. Boosting can improve the performance of weak models and reduce bias.

3.2.5. Tuning Parameters

The performance of decision tree-based models can be sensitive to various tuning parameters, including the maximum depth of the tree, the minimum number of samples required to split a node, the number of trees in the ensemble, and the learning rate (for boosting). Careful tuning of these parameters is necessary to prevent overfitting and achieve optimal performance.

3.2.6. Run time

The run time of decision tree-based models can be relatively fast, especially for smaller datasets. However, for larger datasets or complex models, the run time can become prohibitive. Additionally, some ensemble methods, such as bagging and boosting, require multiple iterations, which can increase the overall run time.

3.3. Support Vector Machines (SVM)

They are a type of supervised learning algorithm used for classification tasks. The goal of an SVM is to find the best hyperplane that separates data points into different classes. The hyperplane is chosen to have the maximum margin, which is the distance between the hyperplane and the nearest data points from each class. SVMs are known to be effective for high-dimensional data sets and can handle both linear and non-linearly separable data. SVMs use kernels to transform data into a higher-dimensional space where it is more easily separable. The kernel function calculates the similarity between two data points and measures the distance between them.

The three types of kernels are:

3.3.1. Linear kernel

The linear kernel assumes that the data is linearly separable. It is the simplest and fastest kernel to compute, making it a good choice for large datasets with high dimensions.

3.3.2. Radial basis function (RBF) kernel

The RBF kernel is used when the data is not linearly separable. It maps the data into a high dimensional space using a Gaussian function and finds the hyperplane that separates the data points.

3.3.3. Polynomial kernel

The polynomial kernel is used to find the decision boundary between data points that are not linearly separable. It maps the data into a higher-dimensional space using a polynomial function.

3.3.4. Tuning Parameters

In addition to selecting the appropriate kernel, SVMs also require the selection of tuning parameters such as the regularization parameter C and the kernel parameter γ for the RBF kernel. The regularization parameter C controls the trade-off between achieving a low training error and a low testing error. The kernel parameter γ controls the width of the kernel and affects the smoothness of the decision boundary.

Overall, SVMs are a powerful tool for classification tasks and can provide accurate results when properly tuned and applied.

3.4. Neural Networks

Neural networks are a class of machine learning algorithms that draw inspiration from the structure and functioning of the human brain. They consist of interconnected nodes, known as neurons, organized into multiple layers. These networks process information and make predictions based on the patterns they learn from the data. In the context of bird species identification, neural networks can be trained to recognize and learn the unique patterns and features associated with each species based on their distinctive sounds.

Types of Neural Networks:

Neural networks encompass various types, each with its own architecture and characteristics.

3.4.1. Feedforward Neural Networks (FNNs)

Feedforward Neural Networks, also known as Multilayer Perceptron (MLPs), are the foundational type of neural network. They consist of layers of interconnected neurons, with information flowing strictly in one direction, from input to output. FNNs are widely used for tasks such as regression, classification, and pattern recognition.

3.4.2. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are specifically designed for processing grid-like data, such as images or spectrograms. They are widely used in computer vision tasks, including image classification, object detection, and image segmentation. CNNs leverage specialized layers such as convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply filters to extract local patterns and features from the input data, while pooling layers down sample the spatial dimensions. CNNs have demonstrated remarkable performance in analyzing and classifying complex visual data, making them applicable to bird species identification based on their unique sounds.

3.4.3. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks are designed to handle sequential data, where the order and temporal dependencies of the data points are important. RNNs leverage recurrent connections to retain memory of past inputs, enabling them to process sequences of varying lengths. RNNs are commonly used in tasks such as natural language processing, speech recognition, and time series analysis.

3.4.4. Long Short-Term Memory (LSTM) Networks

LSTM Networks are a specialized type of RNN that effectively address the vanishing gradient problem associated with traditional RNNs. LSTMs incorporate memory cells and gating mechanisms to selectively retain and update information over long sequences. They excel in capturing long-term dependencies and have been successfully applied in speech recognition, language modeling, and other tasks involving sequential data.

3.4.5. Compilation Techniques

In the context of neural networks, compilation refers to the configuration and optimization of the model before training. Several compilation techniques are employed to enhance the efficiency and performance of the neural network.

One crucial aspect of compilation is the choice of the loss function. The loss function quantifies the discrepancy between the predicted outputs of the network and the true labels. The selection of an appropriate loss function depends on the specific task at hand. For example, binary classification tasks often use binary cross-entropy loss, while multi-class classification tasks commonly employ categorical cross-entropy loss.

Another compilation technique is the choice of the optimizer, which determines the algorithm used to update the network's weights during training. Optimizers play a vital role in guiding the learning process and finding the optimal set of weights that minimize the loss function. Popular optimizers include stochastic gradient descent (SGD), Adam, and RMSprop, each with its own characteristics and performance trade-offs.

Furthermore, compilation involves specifying the evaluation metrics used to assess the performance of the trained model. These metrics can include accuracy, precision, recall, F1-score, and others, depending on the nature of the classification problem. They provide valuable insights into the model's ability to correctly classify bird species based on their sounds.

Additionally, compilation techniques may also involve adjusting other hyperparameters of the neural network, such as learning rate, batch size, and regularization techniques (e.g., dropout or L2 regularization). These hyperparameters significantly impact the training process and the generalization ability of the model.

By carefully selecting and fine-tuning the compilation techniques, the neural network can be optimized to achieve better accuracy, convergence speed, and overall performance in the task of bird species identification based on their sounds.

3.4.6. Activation function

Activation functions play a crucial role in neural networks by introducing non-linearity to the outputs of individual neurons. They enhance the network's ability to learn complex patterns and make accurate predictions. In this project, two commonly used activation functions are employed: the sigmoid function and the SoftMax function. The sigmoid function maps its input to a value between 0 and 1, which is useful for binary classification problems. The SoftMax function, on the other hand, transforms the inputs into a probability distribution across a set of output classes, making it well-suited for multi-class classification problems.

3.5 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique commonly used for feature extraction and data visualization. It aims to transform a high-dimensional dataset

into a lower-dimensional space while preserving the most important information and maximizing the variance in the data.
The key idea behind PCA is to find a new set of orthogonal variables, called principal components, that capture the most significant patterns or variations in the data. These principal components are linear combinations of the original features and are ordered in terms of the amount of variance they explain. The first principal component explains the highest amount of variance, and each subsequent component explains as much of the remaining variance as possible.
PCA can be applied to various domains, including image processing, genetics, finance, and natural language processing. It is often used for dimensionality reduction, noise reduction, feature selection, and visualization of high-dimensional data.
By using PCA, you can reduce the dimensionality of the data, gain insights into the most important features, and potentially improve the performance of machine learning algorithms by reducing noise and removing irrelevant or redundant features.

3.6. Singular Value Decomposition (SVD):

Singular Value Decomposition (SVD) is a mathematical technique used for decomposing a matrix into three constituent matrices, providing a compact representation of the original matrix's structure. It is widely used in various fields, including linear algebra, signal processing, image processing, recommender systems, and dimensionality reduction. SVD plays a fundamental role in many advanced data analysis techniques, such as Principal Component Analysis (PCA) and Latent Semantic Analysis (LSA).
The theoretical background of SVD involves understanding the properties and concepts associated with it:
Matrix Decomposition: SVD is a matrix decomposition technique that breaks down a matrix A into three matrices: U, Σ, and V^T (transpose of V). This decomposition is expressed as $A = U\Sigma V^T$. Here, U and V are orthogonal matrices, and Σ is a diagonal matrix.
Orthogonal Matrices: An orthogonal matrix is a square matrix where the columns are mutually orthogonal unit vectors. The product of an orthogonal matrix and its transpose yields the identity matrix. In SVD, both U and V are orthogonal matrices.
Diagonal Matrix: A diagonal matrix is a square matrix where the non-diagonal elements are all zero. The diagonal elements of Σ in SVD are called singular values and represent the importance or magnitude of each dimension or component.

Singular Values: The singular values in Σ are arranged in descending order along the diagonal. They indicate the amount of variation or information contained in each component. Larger singular values correspond to dimensions or components with more significance.

Rank: The rank of a matrix is the number of linearly independent columns or rows it contains. In SVD, the rank of the original matrix A is equal to the number of non-zero singular values in Σ. If a singular value is zero, it implies that the corresponding dimension or component does not contribute much to the structure of the original matrix.

Low-Rank Approximation: SVD allows us to approximate the original matrix A by retaining only the top k singular values and their corresponding columns in U and V. This approximation, called low-rank approximation, can capture the most important features or structure of the matrix while reducing its dimensionality.

The SVD decomposition provides several applications and benefits:

Dimensionality Reduction: SVD can be used for dimensionality reduction by selecting the top k singular values and their corresponding columns. This helps in reducing the complexity and storage requirements of high-dimensional datasets while preserving important information.

Data Compression: SVD can be utilized for compressing data by approximating the original matrix with fewer singular values and vectors. This compression technique finds applications in image and video compression, where it helps in reducing the size of the data without significant loss of visual quality.

Matrix Approximation: SVD enables us to find the best low-rank approximation of a matrix. This is useful in applications such as collaborative filtering in recommender systems, where the approximation can be used to make predictions or recommendations based on partial information.

PCA and LSA: SVD is the underlying mathematical technique used in Principal Component Analysis (PCA) and Latent Semantic Analysis (LSA). PCA utilizes SVD to find the principal components that capture the most significant variation in the data, while LSA applies SVD to analyze relationships between documents and terms in text analysis.

Overall, SVD provides a powerful tool for matrix decomposition and dimensionality reduction, allowing us to understand the underlying structure of data and extract meaningful features. Its applications extend across various domains, making it an essential technique in data analysis and machine learning.

3.7. Clustering

Clustering is an unsupervised learning technique used to identify groups or clusters within a dataset. The goal of clustering is to find natural groupings or patterns in the data, where similar objects are grouped together based on their intrinsic characteristics. It is often used for exploratory data analysis, pattern recognition, and data compression.

The main idea behind clustering algorithms is to partition the data points into clusters, where points within the same cluster are more similar to each other than to points in other clusters. There are various clustering algorithms available, and one popular algorithm is K-means clustering. K-means clustering aims to minimize the within-cluster sum of squares by iteratively assigning data points to clusters and updating cluster centroids.

Clustering can be useful for various applications, such as customer segmentation, image segmentation, anomaly detection, and recommendation systems.

3.8 Justifying my choice of supervised learning method:

Supervised learning methods are used in this project to predict the outcome variable (win, loss, or draw) based on the given features. The choice of supervised learning methods, such as Logistic Regression, Decision Trees, Support Vector Machines (SVM), and Neural Networks, is justified based on their suitability for classification tasks.

Logistic Regression is a widely used supervised learning algorithm for binary classification problems. In this project, it is employed to predict the outcome of the Connect-4 game. Logistic Regression estimates the probability of a binary outcome based on the input features. It is a linear model that applies a logistic function to the linear combination of the features. The model learns the coefficients for the features during the training process, which allows it to make predictions on new instances.

Decision Trees are non-linear models that recursively split the data based on features to make predictions. They are suitable for classification tasks as they partition the feature space into regions corresponding to different classes. Decision Trees are well-suited for this project as they can capture complex patterns and relationships between the game positions and the outcome variable.

Random Forest is an ensemble learning method that combines multiple decision trees and aggregates their predictions. It is known for its ability to handle high-dimensional data and reduce overfitting. Random Forests can capture non-linear relationships and interactions between features, making them suitable for this project.

Gradient Boosting is another ensemble learning technique used in this project. It combines multiple weak models into a strong predictive model. Boosting algorithms sequentially train models on the training data, focusing on the instances that were misclassified by the previous models. This iterative process helps improve the model's performance by gradually reducing the prediction errors.

Support Vector Machines (SVM) are powerful supervised learning algorithms that can perform both classification and regression. In this project, linear and radial SVMs are employed. SVMs aim to find the optimal hyperplane that separates the data points belonging to different classes. The linear SVM uses a linear kernel to separate classes, while the radial SVM uses a radial basis function (RBF) kernel to allow non-linear separation.

Neural Networks are a type of deep learning model that can learn complex patterns and relationships in the data. They consist of multiple layers of interconnected nodes (neurons) and can model highly non-linear relationships. Neural Networks have been successful in various domains, including image recognition and natural language processing. In this project, a neural network model with three dense layers is used to predict the outcome of the Connect-4 game.

3.9 Difference between Supervised and Unsupervised Learning:

The supervised learning methods differ from the unsupervised methods used in this project (such as K-means clustering) in terms of their objectives and approaches. Unsupervised learning aims to discover patterns, structures, or relationships in the data without the presence of labeled target variables. It can be used for tasks such as clustering or dimensionality reduction. On the other hand, supervised learning is concerned with predicting or classifying the target variable based on the input features.

In supervised learning, the models are trained using labeled data, where the target variable is known. The models learn to map the input features to the corresponding target variable. In contrast, unsupervised learning techniques do not require labeled data and aim to find patterns or groupings in the data without any prior knowledge of the target variable.

The supervised learning methods in this project are evaluated based on their accuracy in predicting the outcome variable. The accuracy represents the percentage of correctly classified instances. In contrast, the unsupervised clustering approach is evaluated using metrics such as the silhouette score and completeness score. The silhouette score measures the compactness and separation of the clusters, while the completeness score measures the extent to which all instances of a class belong to a single cluster.

4. METHODOLOGY

4.1. Data Processing/Cleaning:

The initial step in the methodology involves importing the necessary libraries and loading the Connect-4 dataset, which contains legal positions in the game of Connect-4. The dataset is read into a Pandas Data Frame, and appropriate column names are assigned to the dataset's columns. To ensure data quality, a check is performed to identify any missing values, and it is determined that the dataset does not contain any null values. Visualizations, such as bar plots and heatmaps, are created to gain insights into the distribution of the outcome variable (win, loss, or draw) and the positions on the Connect-4 board. Then the dataset is divided into two components: the features(X) and the target variable(y). The features represent the positions on the Connect-4 board, denoted as black, player 1(x), or player 2(o). The target variable indicates the class of the game outcome. To facilitate analysis, one-hot encoding is applied to categorical features. The class labels are mapped to numerical values. Finally, the dataset is split into training and testing sets using the train-test split technique, which will be utilized in subsequent steps of the methodology.

4.2. Logistic Regression:

Logistic regression is performed using the Logistic Regression class from scikit-learn, and the accuracy of the model is calculated.

4.3. Decision Trees:

Next, a decision tree classifier is fitted to the data. The accuracy of the decision tree on the test data is calculated, and feature importance's are determined. The feature importance's indicate the relative importance of each feature in predicting the outcome.\

4.4. Random Forest:

Random forest, an ensemble learning method, is employed in this step. The random forest classifier is trained on the training data, and predictions are made on the test data. The accuracy of the model is calculated, and the feature importance's are determined. The top five important features are visualized using a bar plot.

4.5. Boosting:

In this step, gradient boosting is utilized. A gradient boosting classifier is created and fitted to the training data. Predictions are made on the test data, and the accuracy of the model is calculated. Feature importance's are determined, indicating the most important features in the dataset.

4.6. Support Vector Machines (SVM):

Two types of SVMs are employed in this step: linear SVM and radial SVM. Feature selection is performed using SelectKBest with an F-score. The selected features are used to train the SVM models. Predictions are made on the test data, and the accuracies of both linear SVM and radial SVM are calculated.

4.7. Neural Networks:

The model architecture is defined using the Sequential model from TensorFlow. The model consists of three dense layers. The first dense layer has 64 units and uses the ReLU activation function. It accepts input with a shape corresponding to the number of features in the training data. The second dense layer has 32 units and also uses the ReLU activation function. The final dense layer has a number of units equal to the number of classes, and it employs the softmax activation function to produce class probabilities. After defining the model architecture, it is compiled using the Adam optimizer, categorical cross-entropy loss function, and accuracy as the evaluation metric. This step involves training a neural network model using the Keras library. The target variable is converted to categorical format, and the model architecture is defined. The model is trained on the training data and evaluated on the test data. The accuracy of the neural network model is calculated.

4.8. SVD:

SVD is then performed on the encoded data to decompose it into U, s, and Vt matrices, representing the singular vectors and singular values. The proportion of variance explained by each principal component is calculated, and a scree plot is generated to visualize the variance explained by each component.

In the next step, the data is projected onto the first two principal components, creating a lower-dimensional representation. This is done by multiplying the centered encoded data with the first two columns of the Vt matrix. The resulting matrix, X_pca, represents the data projected onto the first two principal components. A scatter plot is then created, where the x-axis corresponds to the first principal component and the y-axis corresponds to the second. Each data point is colored based on its class label (if available), providing insights into clustering and patterns in the data.

4.9. Clustering:

In the methodology, the K-means clustering algorithm is employed to group the data into distinct clusters. The algorithm is applied to the encoded data, with a specified number of clusters (in this case, 3). The resulting cluster labels are then assigned to the original data. The observations within each cluster are displayed to examine the characteristics of the data points belonging to the same cluster.

To evaluate the clustering results, a scatter plot is created using the first two principal components obtained from the dimensionality reduction step. The data points are colored based on their assigned cluster labels, providing a visual representation of the clustering outcomes. Additionally, two performance metrics are computed. The completeness score is calculated to measure the extent to which each true class is assigned to a single cluster. The silhouette score is used to evaluate the compactness and separation of the clusters, indicating the quality of the clustering results.

In summary, the methodology encompasses data processing, application of various machine learning algorithms such as logistic regression, decision trees, random forest, gradient boosting, SVMs, neural networks and clustering. It also includes feature selection, accuracy evaluation, and visualization of feature importances. These steps provide insights into the performance and predictive capabilities of different machine learning models for the Connect-4 dataset.

5. COMPUTATIONAL RESULTS

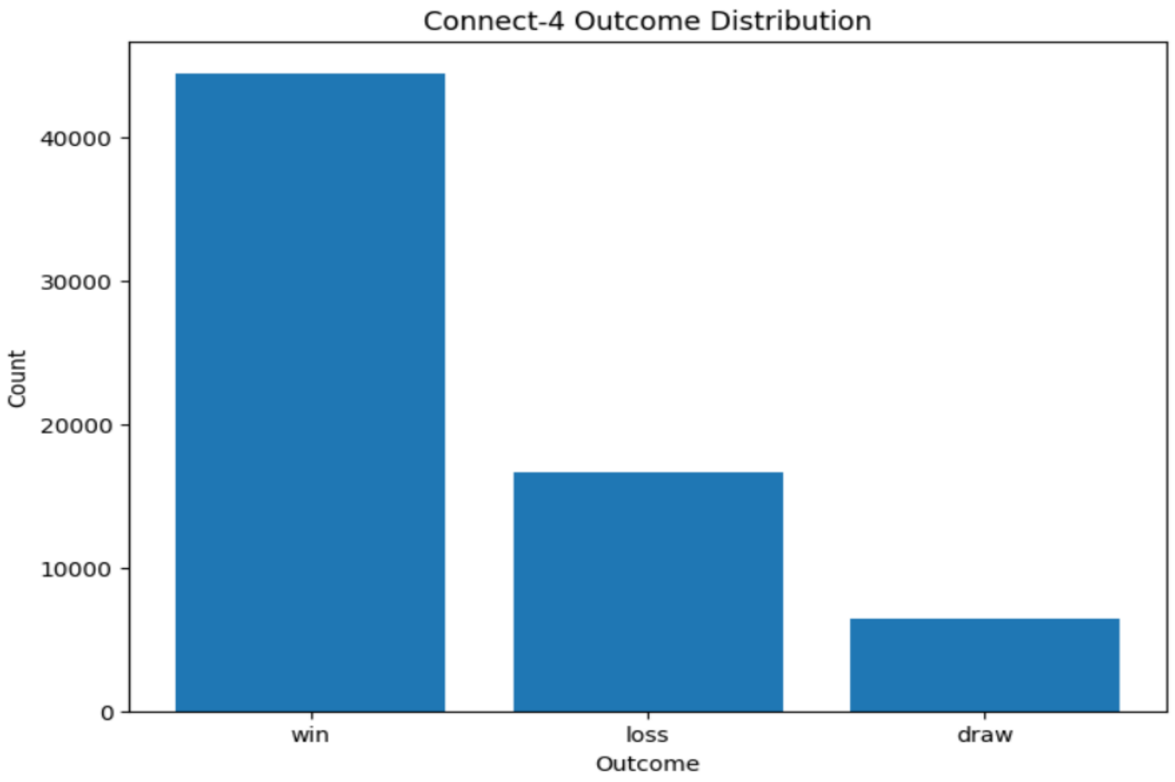
Models	Logistic Regression	Decision Trees	Random Forest	Boosting	SVM(Linear)	SVM(Binary)	Neural Network
Accuracy	65.75%	75.87%	81.94%	78.06%.	49.59%.	74.52%	79.55%

Clustering Completeness Score	Clustering Silhouette Score
0.0023	0.088

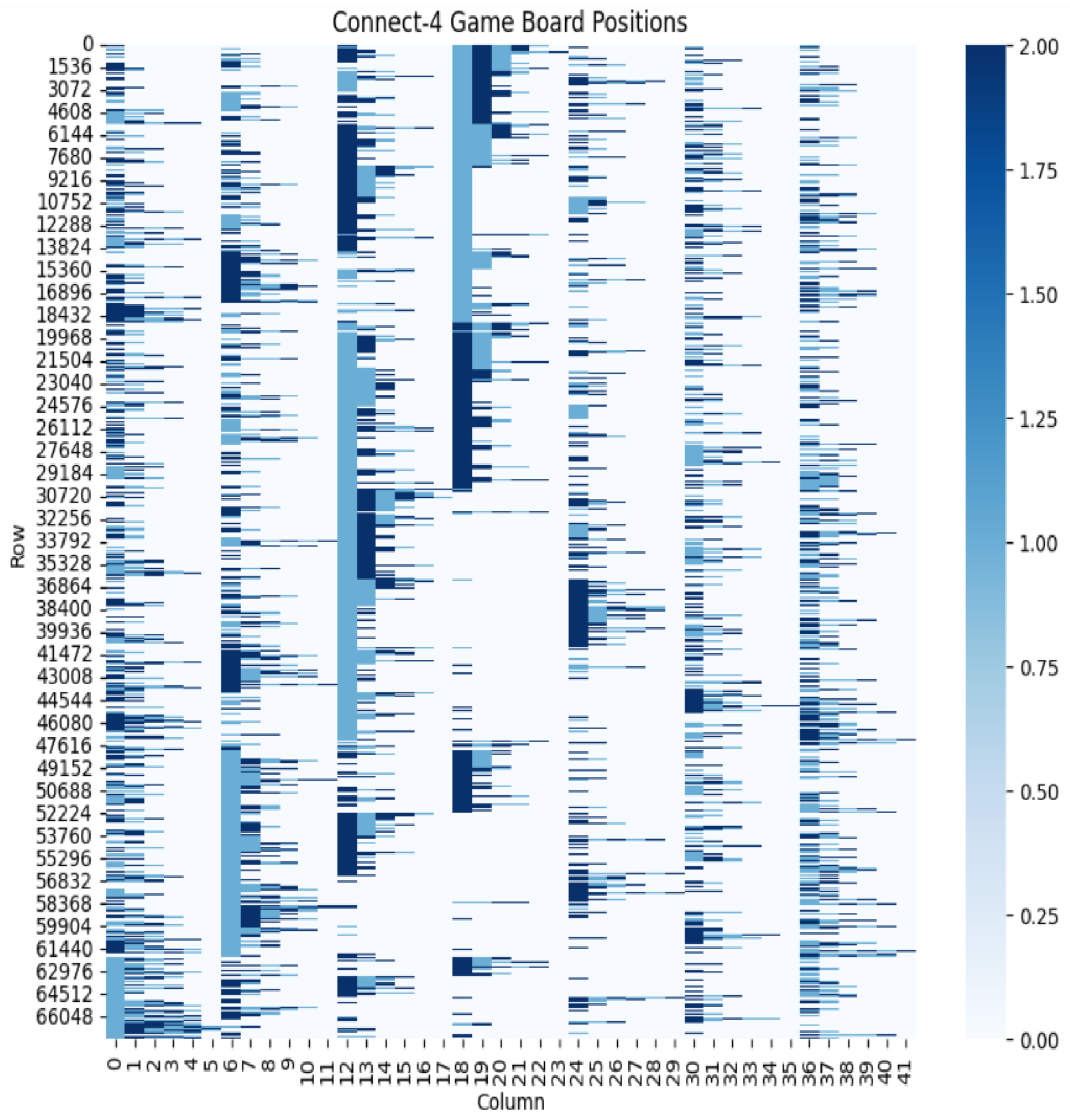
The table provides insights into the performance of different models and the clustering algorithm on the given dataset. The accuracy metric represents the percentage of correctly classified samples, while the silhouette score and completeness score measures how well the clusters capture the original classes. Based on the results, the random forest model achieved the highest accuracy, indicating its effectiveness in predicting the target variable. The linear SVM and clustering algorithm performed relatively poorly compared to other models, suggesting that the dataset may not exhibit clear linear separability or distinct clustering patterns.

6. DISCUSSION

6.1 Visualizations:



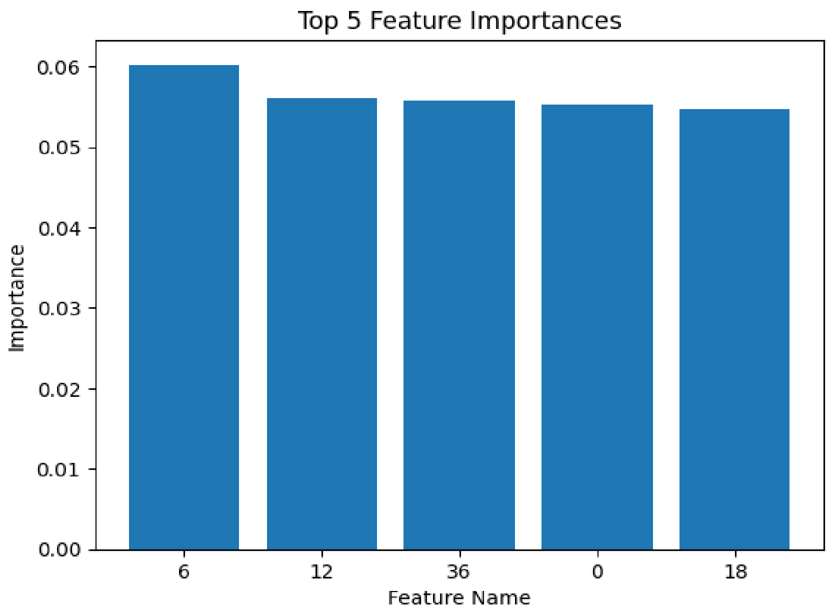
The given figure illustrates the Outcome Distribution for the connect-4 dataset, revealing an imbalance in the data distribution. However, it is noteworthy that an ample number of data points are available for each outcome category. Consequently, we can confidently proceed with this project.



The provided visualization depicts the Connect-4 game board positions. In this representation, the number 0 corresponds to the empty spaces on the board. Player 1's positions are represented by a lighter shade of blue, indicated by the number 1. Similarly, player 2's positions are denoted by a darker shade of blue, represented by the number 2. This visualization offers a clear overview of the distribution and placement of the players' moves on the Connect-4 game board.

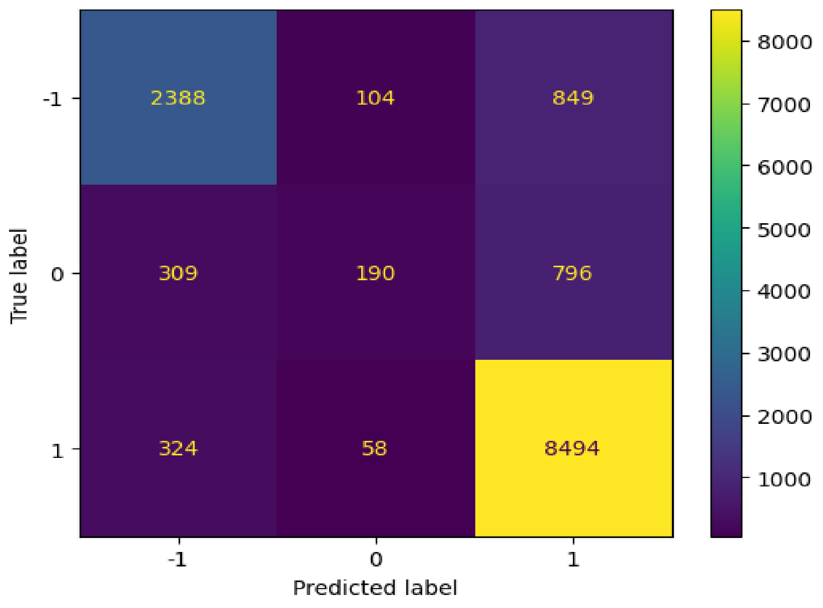
6.2 Random Forest Results:

Random Forest is the best model according to the accuracy values.

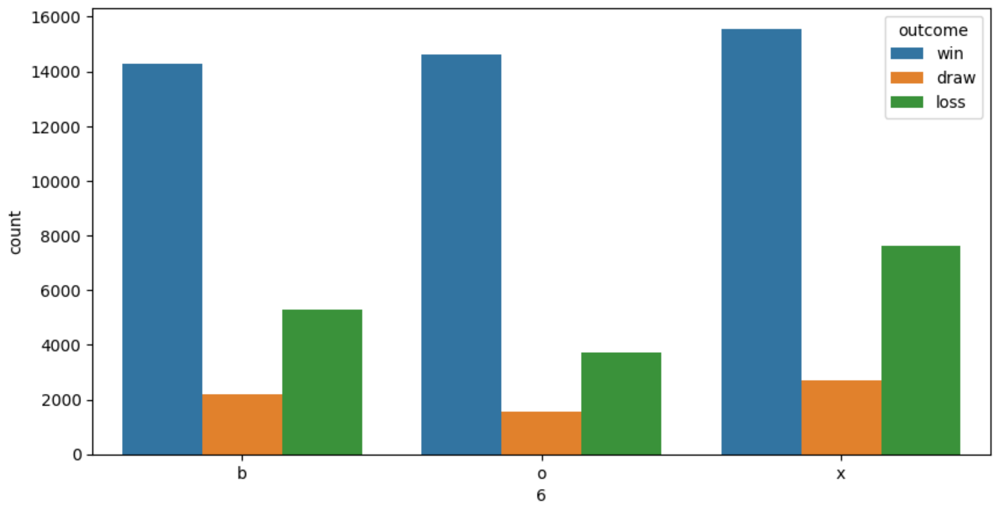


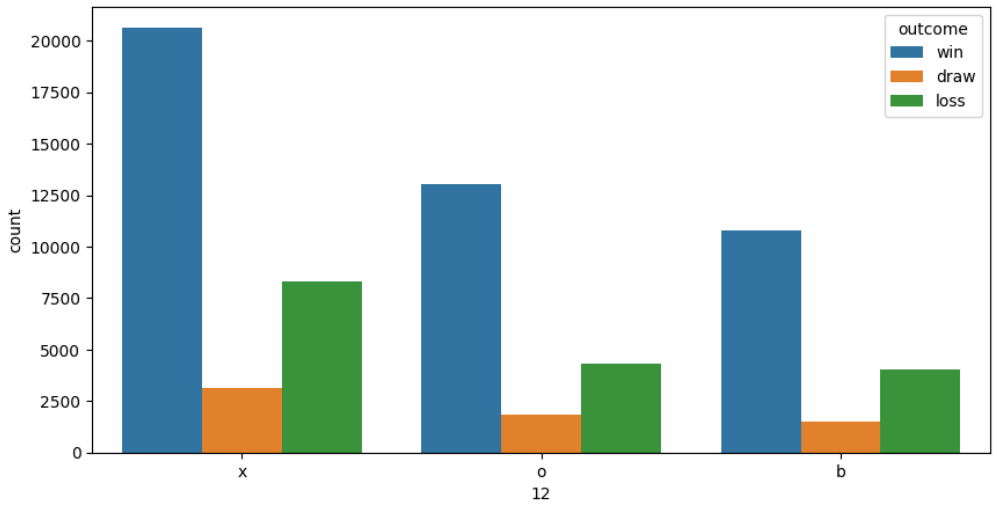
The plot displays the top 5 feature importances obtained from the random forest model. The x-axis represents the feature names, while the y-axis represents the importance scores assigned to each feature. The heights of the bars in the plot indicate the relative importance of the corresponding features. The higher the bar, the more significant the feature is in predicting the target variable. These features have been ranked based on their contribution to the model's predictive performance.

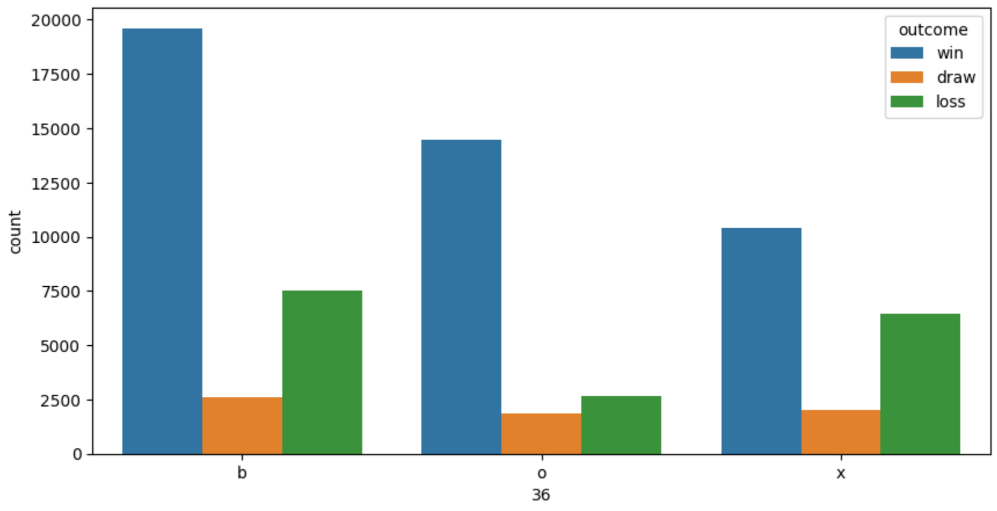
Analyzing feature importances is essential for understanding the underlying factors that drive the model's predictions. It helps identify which features have the most influence on the target variable and can aid in feature selection or feature engineering processes.

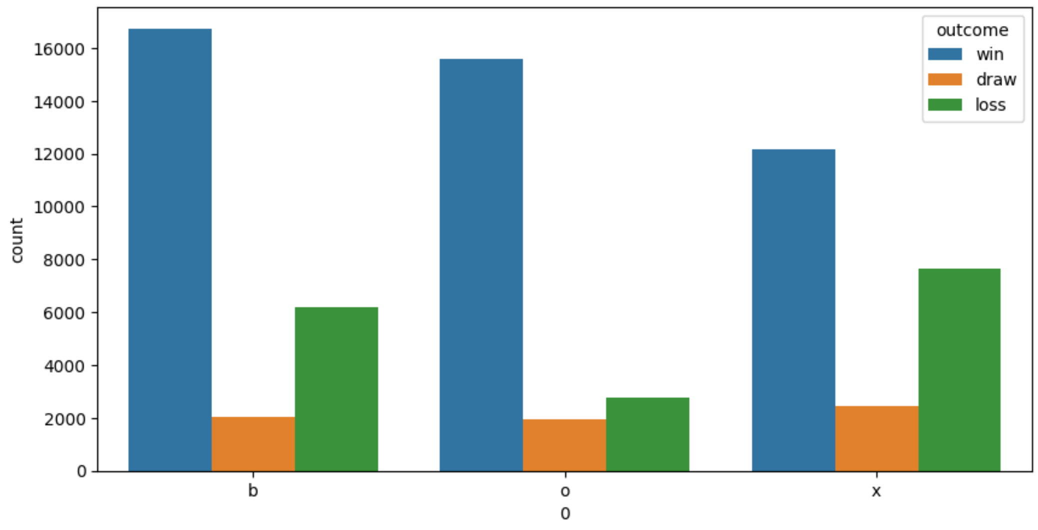


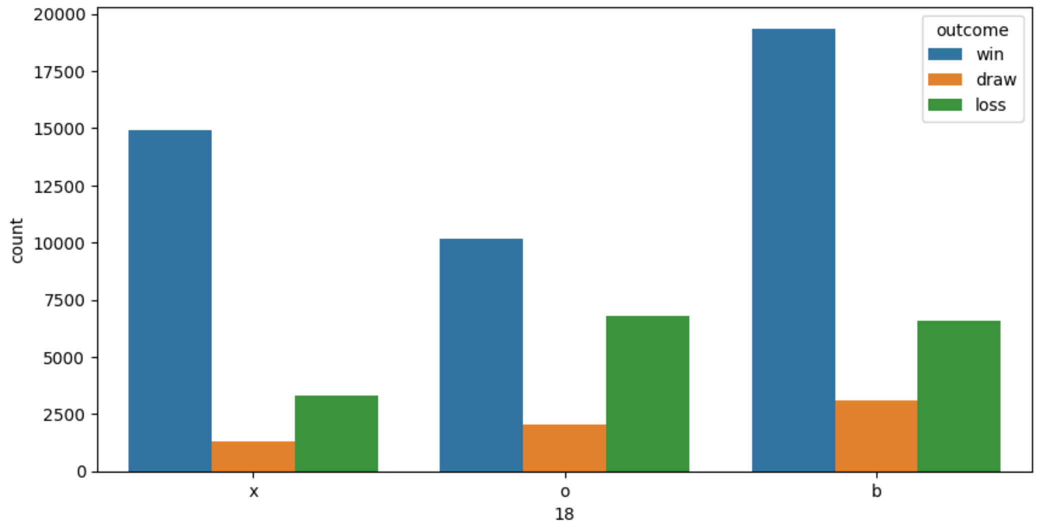
The confusion matrix plot provides an overview of the performance of the random forest model by visually representing the predicted and actual values of the target variable, The diagonal elements of the confusion matrix represent the instances that were correctly classified, while the off-diagonal elements represent the instances that were misclassified.









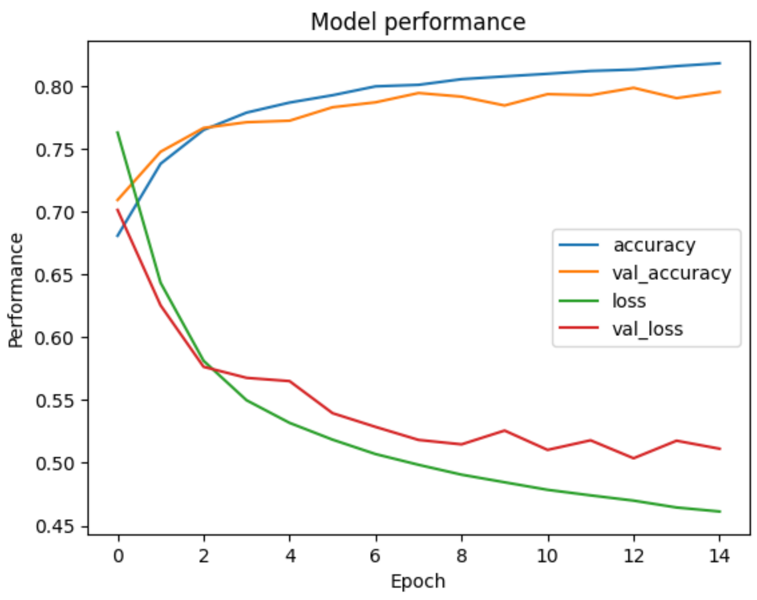


The above plots represent countplot for the top 5 important variables based on the feature importances obtained from the random forest model. Each countplot visualizes the distribution of values for a specific variable, grouped by the outcome variable. In each countplot, the x-axis represents the values of the selected variable, and the y-axis represents the count of occurrences. The countplot is further differentiated by color, with each color representing a different outcome category.

By examining these countplots, we can gain insights into the relationship between the top variables and the outcome variable. We can observe the distribution of values for each variable across different outcome categories. This allows us to explore potential associations and gain a better understanding of how these variables contribute to the prediction of the outcome.

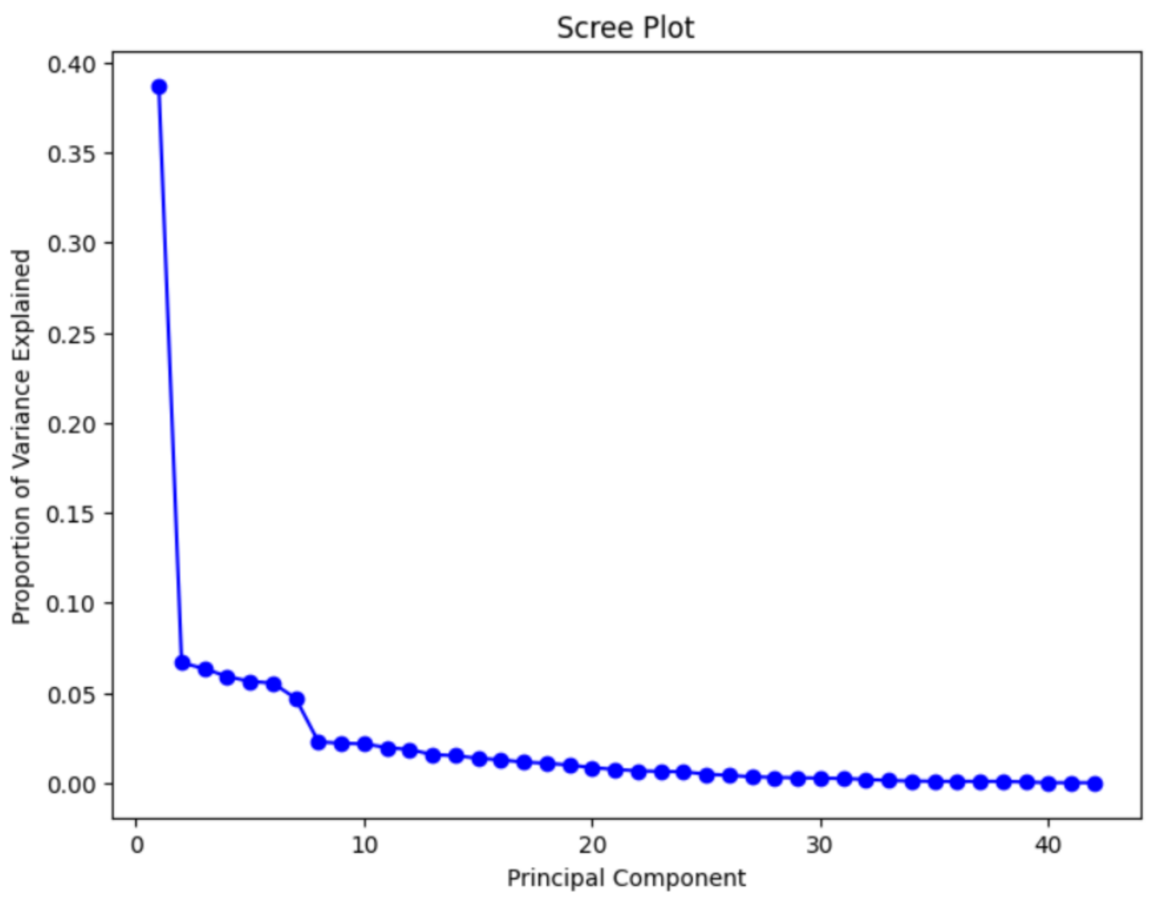
The count plots suggest that if player 1 marks positions 6 and 12 on the game board, their chances of winning are higher. Additionally, if positions 0, 18, and 36 are empty (blank spaces), it increases the likelihood of player 1 winning.

6.3 Neural Network Performance Visualization:



The neural network model that was trained achieved an impressive test accuracy of 79.55%. There is potential for further improvement by either developing a more robust model or making modifications to the existing model. Exploring different neural network architectures in future endeavours could lead to more advanced analyses and insights for the Connect-4 game.

6.4 Singular Value Decomposition (SVD):



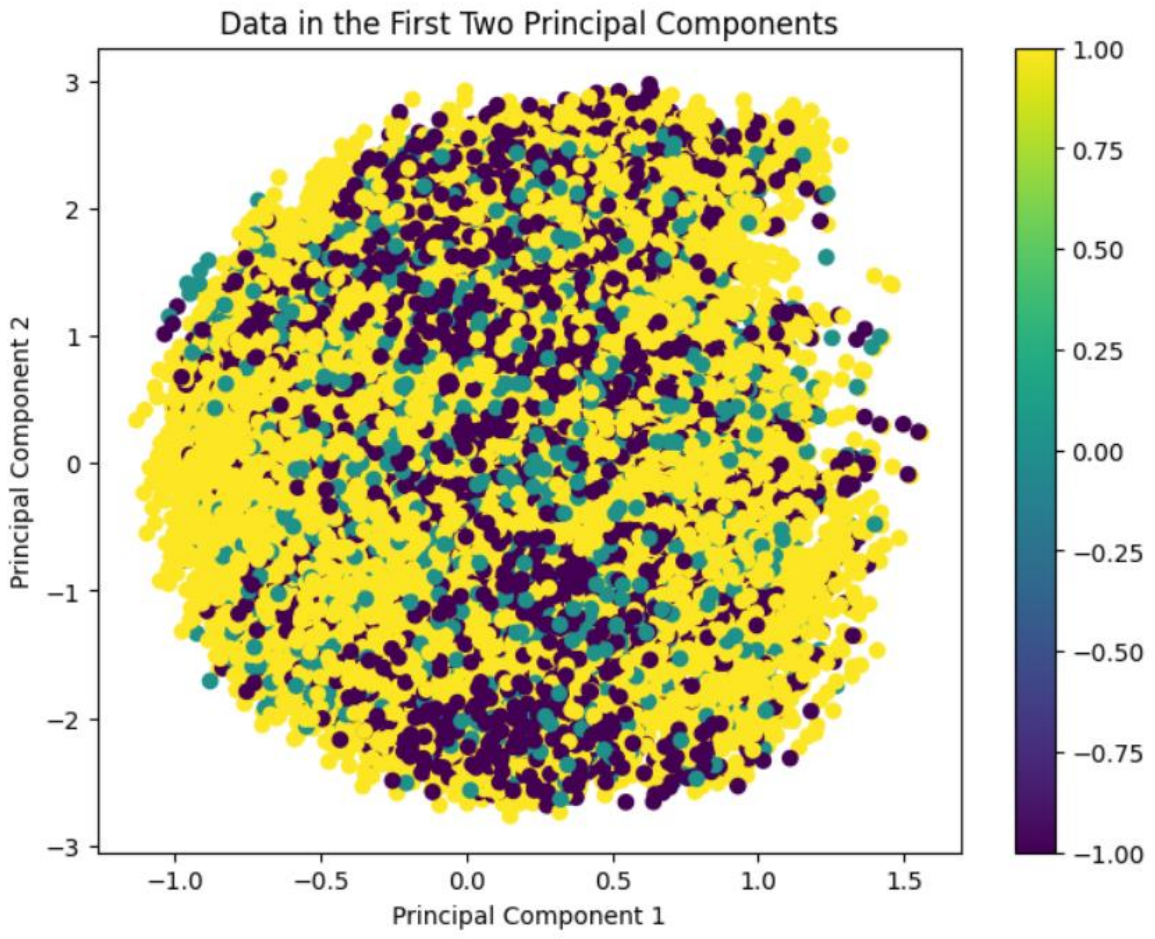
The scree plot is a graphical representation of the eigenvalues of the principal components, showing the amount of variance explained by each component.

If you observe that the scree plot has a clear elbow or cutoff point, where the eigenvalues sharply drop after a certain number of components, it suggests that only a few principal components contribute significantly to the variance explained. In this case, it appears that the first eight principal components contribute to a larger proportion of the total variance, while the remaining components have minimal effect on the variance.
This finding can be useful for dimensionality reduction purposes. Instead of using all the components, you may consider retaining only the first eight components that explain the majority of the variance. By doing so, you can reduce the dimensionality of your data while still preserving most of the relevant information.

# Discuss the interpretation of the U and V* matrices
The U matrix from SVD represents the left singular vectors and can be interpreted as the transformed feature matrix. Each row of U corresponds to a data point, and each column

represents a linear combination of the original features that capture the underlying patterns in the data.

The V* matrix from SVD represents the right singular vectors and can be interpreted as the transformed variable matrix. Each row of V* represents a principal component, and each column corresponds to a feature in the original dataset. The entries in V* represent the weight or importance of each feature in the corresponding principal component.

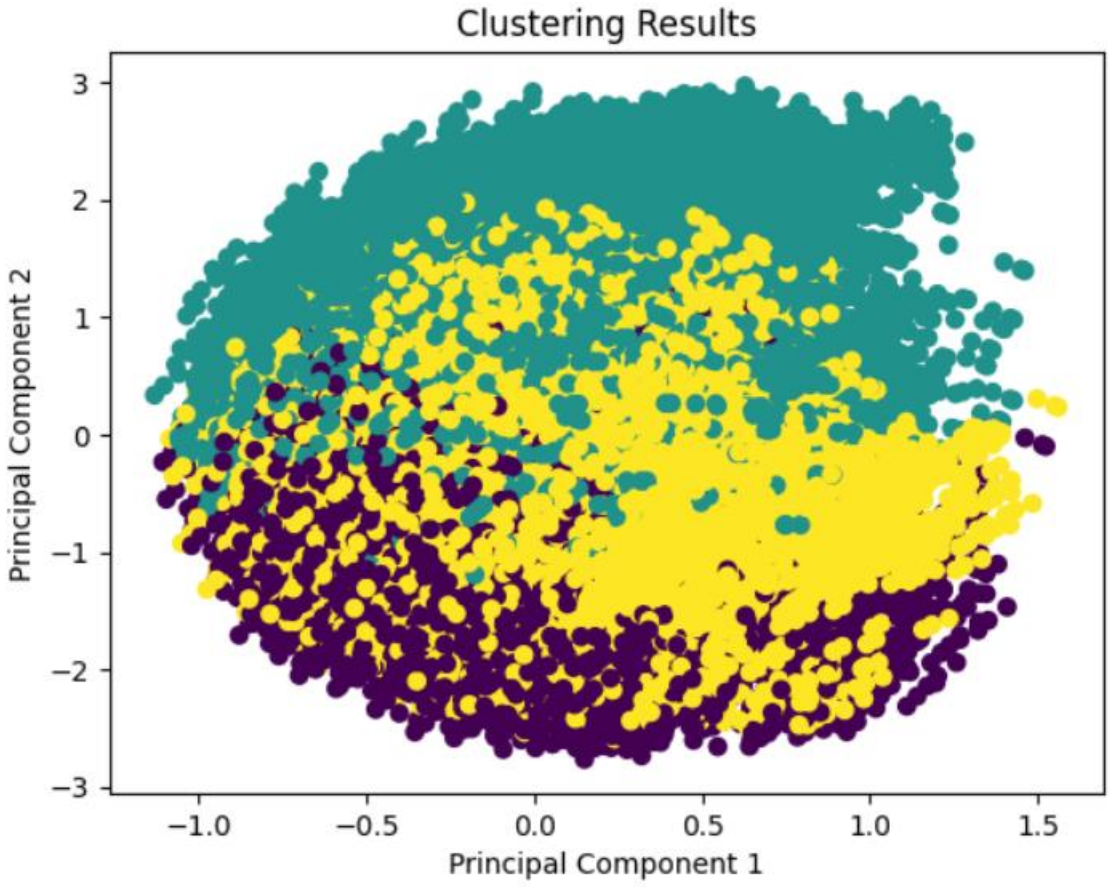


# Discuss the structure within the data and the difference from plotting on the original features

The plot of the data in the first two principal components shows the inherent structure or patterns in the data. Points that are close to each other in the plot are more similar in terms of their underlying patterns. This structure captures the relationships between the original features in a reduced-dimensional space.

Compared to plotting on the two original features, the plot in the principal components provides a more condensed representation of the data, as it combines multiple original features into the principal components. This can help in identifying clusters or groups of similar data points more easily.

6.5. Clustering:



Overall, the low values for both the clustering completeness score and clustering silhouette score imply that the clustering algorithm used in this context is not performing well in accurately capturing the true class assignments and creating distinct, well-defined clusters for the Connect-4 dataset.

The low values of the clustering completeness score and clustering silhouette score suggest that there may not be clear and discernible patterns in the data that can be effectively captured by the clustering algorithm. In such cases, it becomes challenging for the algorithm to identify meaningful clusters that correspond to the underlying structure or class assignments in the data. The absence of distinct patterns or structures may hinder the performance of the clustering algorithm, resulting in low scores. It could indicate that the Connect-4 dataset does not exhibit easily separable clusters or that other factors, such as noise or overlapping data points, are present, making it difficult for the algorithm to accurately cluster the data.

7. CONCLUSION

In conclusion, this report presented a comprehensive methodology for analyzing the Connect-4 dataset using various machine learning techniques. The dataset was pre-processed and cleaned, ensuring data quality and appropriate formatting for analysis. Several models were applied, including logistic regression, decision trees, random forest, boosting, SVMs, and neural networks, to predict the outcome of the Connect-4 game. Additionally, a clustering algorithm (K-means) was used to group similar samples together based on their features.

The computational results demonstrated the performance of each model and the clustering algorithm. The random forest model achieved the highest accuracy of 81.94%, indicating its effectiveness in predicting the outcome. The neural network model also performed well with an accuracy of 79.55%. However, linear SVM and the clustering algorithm showed relatively lower accuracies, suggesting challenges in linearly separating classes or identifying distinct clusters in the dataset.

Visualizations were provided to gain insights into the data distribution, feature importances, and model performance. The count plots of the top important variables from the random forest model highlighted specific game board positions that were more likely to lead to a win for player 1. The scatter plot for clustering showcased the separation of different outcomes based on two selected features, although the clustering algorithm's performance was relatively low, indicating challenges in identifying clear patterns.

Overall, this analysis demonstrated the capabilities and limitations of different machine learning models in predicting the Connect-4 game outcome. Further improvements and explorations can be made by refining existing models, exploring different architectures, or considering alternative clustering algorithms. The findings of this report contribute to a better understanding of the Connect-4 dataset and provide insights into the predictive capabilities of different models for similar classification tasks.

8. BIBLIOGRAPHY

1. Connect-4 Data Set. UCI Machine Learning Repository. (n.d.). Retrieved from https://archive.ics.uci.edu/ml/datasets/Connect-4

2. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, https://link.springer.com/book/10.1007/978-0-387-84858-7

3. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer, https://link.springer.com/book/9780387310732

4. Mitchell, T. M. (1997). Machine Learning. McGraw-Hill, https://www.cs.cmu.edu/~tom/mlbook.html

5. Burges, C. J. C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery, 2(2), 121-167, https://link.springer.com/article/10.1023/A:1009715923555

6. Haykin, S. (1999). Neural Networks: A Comprehensive Foundation. Prentice Hall, https://drive.google.com/file/d/0B2iRDvP8jUuAUnpfaDBnQTBWLUU/edit?resourcekey=0-bq1kH6l5hurYT7TtvylSCQ

7. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press, https://www.researchgate.net/publication/320703571_Ian_Goodfellow_Yoshua_Bengio_and_Aaron_Courville_Deep_learning_The_MIT_Press_2016_800_pp_ISBN_0262035618

8. Aggarwal, C. C. (2018). Neural Networks and Deep Learning: A Textbook. Springer, https://link.springer.com/book/10.1007/978-3-319-94463-0

9. Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345-1359, https://ieeexplore.ieee.org/document/5288526

10. Bengio, Y. (2012). Deep Learning of Representations for Unsupervised and Transfer Learning. Journal of Machine Learning Research, 13, 2493-2537, http://proceedings.mlr.press/v27/bengio12a/bengio12a.pdf

11. Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. ACM Computing Surveys (CSUR), 31(3), 264-323

12. Shlens, J. (2014). A tutorial on principal component analysis. arXiv preprint arXiv:1404.1100

13. Golub, G. H., & Van Loan, C. F. (2012). Matrix computations. JHU Press.'

9. APPENDIX

```
In [ ]: #Import all the required libraries
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import random

        from sklearn.preprocessing import OneHotEncoder, LabelBinarizer, StandardSca
        from sklearn.compose import ColumnTransformer
        from sklearn.metrics import mean_squared_error, confusion_matrix, ConfusionM
        from sklearn.model_selection import train_test_split, GridSearchCV

        from sklearn.linear_model import LogisticRegression

        from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
        from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassif
        from sklearn.inspection import permutation_importance

        from sklearn.svm import SVC
        from sklearn.feature_selection import SelectKBest, f_classif

        from keras.utils import to_categorical
        import tensorflow as tf
        import tensorflow
        from tensorflow.keras import layers
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.optimizers import Adam
        from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dr
        from sklearn.cluster import KMeans
        from sklearn.metrics import silhouette_score

        import warnings
        warnings.filterwarnings("ignore")
```

Reading the data

```
In [ ]: #Mounting the drive
        from google.colab import drive
        drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
In [ ]: # changing the working directory
        import os
        os.chdir('/content/drive/MyDrive/5322/')
```

```
In [ ]: #Metadata about the dataset
        names_data = pd.read_csv('connect-4.names', header=None, skiprows=0, delimit
```

x is player 1
o is player 2
b is blank
The outcome class is the game theoretical value for the first player.
This database contains all legal 8-ply positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced.

In [ ]:
```
# reading the data
data = pd.read_csv('connect-4.data', header=None)
# adding the columns
data.columns = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11'
data.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	33	34	35	36	37	38	39	40	41	outcome
0	b	b	b	b	b	b	b	b	b	b	...	b	b	b	b	b	b	b	b	b	win
1	b	b	b	b	b	b	b	b	b	b	...	b	b	b	b	b	b	b	b	b	win
2	b	b	b	b	b	b	o	b	b	b	...	b	b	b	b	b	b	b	b	b	win
3	b	b	b	b	b	b	b	b	b	b	...	b	b	b	b	b	b	b	b	b	win
4	o	b	b	b	b	b	b	b	b	b	...	b	b	b	b	b	b	b	b	b	win

5 rows × 43 columns

In [ ]:
```
data.shape
```

Out[ ]:
```
(67557, 43)
```

In [ ]:
```
data.isnull().sum().sum()
```

Out[ ]:
```
0
```

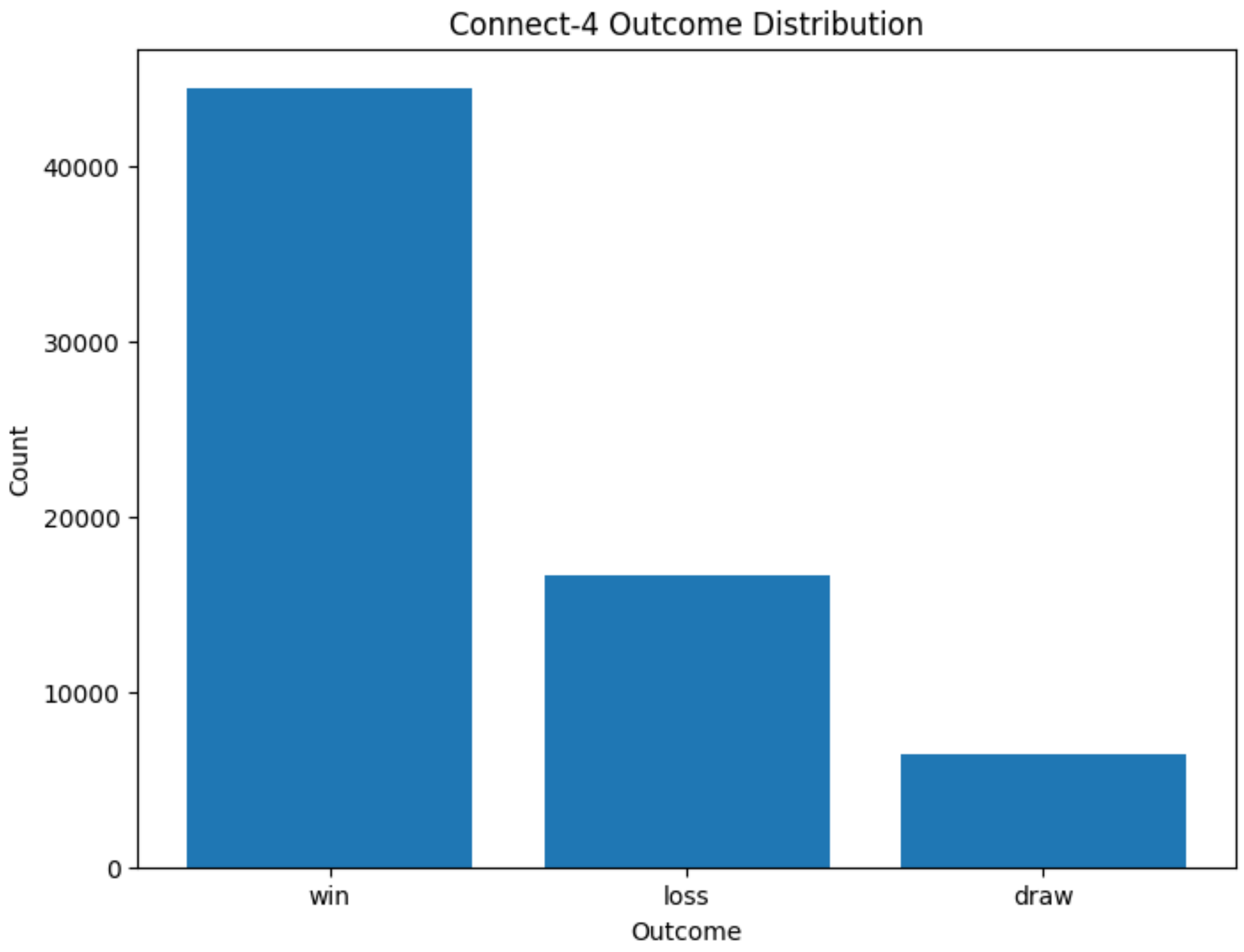
There are no null values in the data.

Visualizations

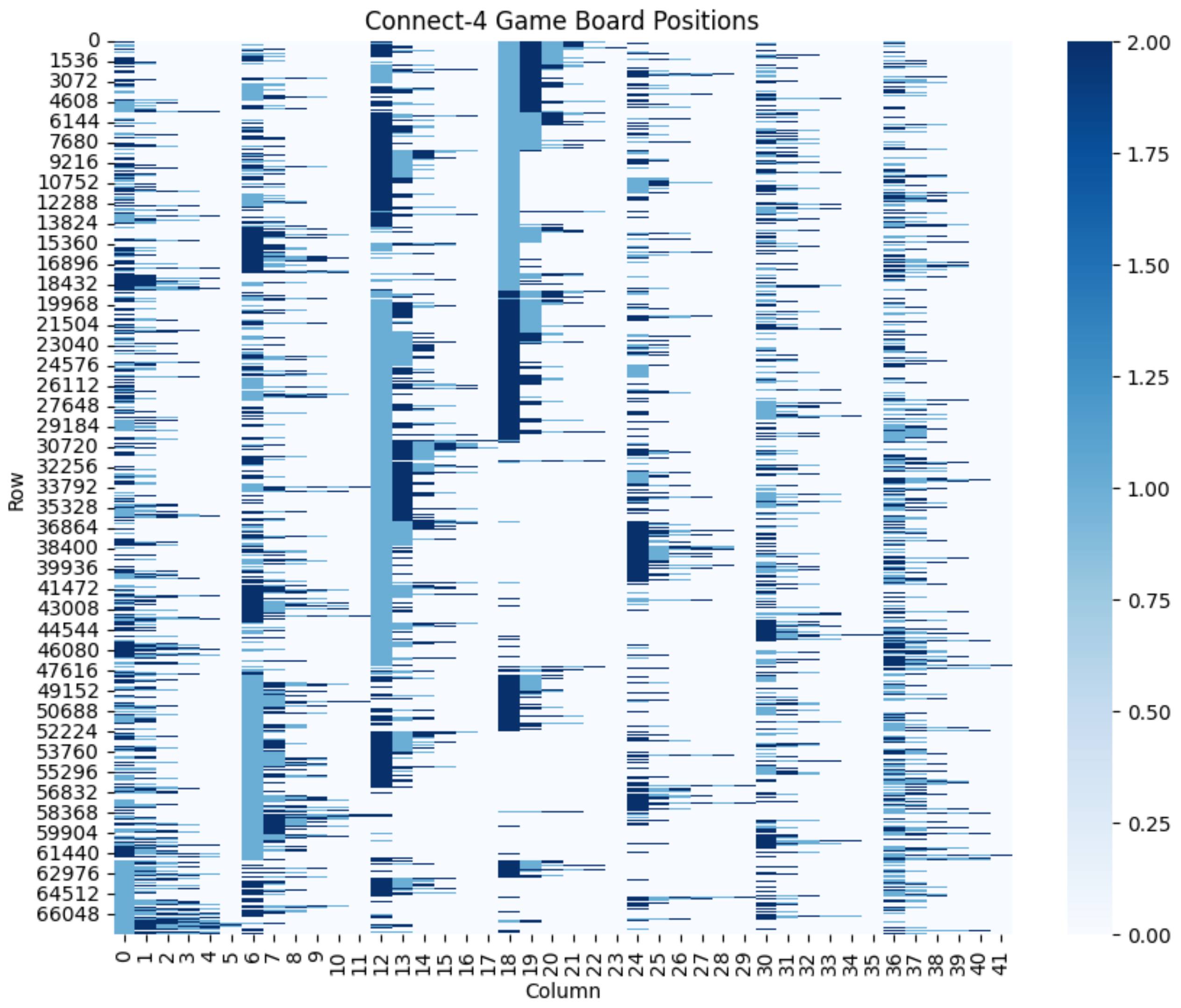
In [ ]:
```
data['outcome'].value_counts()
```

Out[ ]:
```
win     44473
loss    16635
draw     6449
Name: outcome, dtype: int64
```

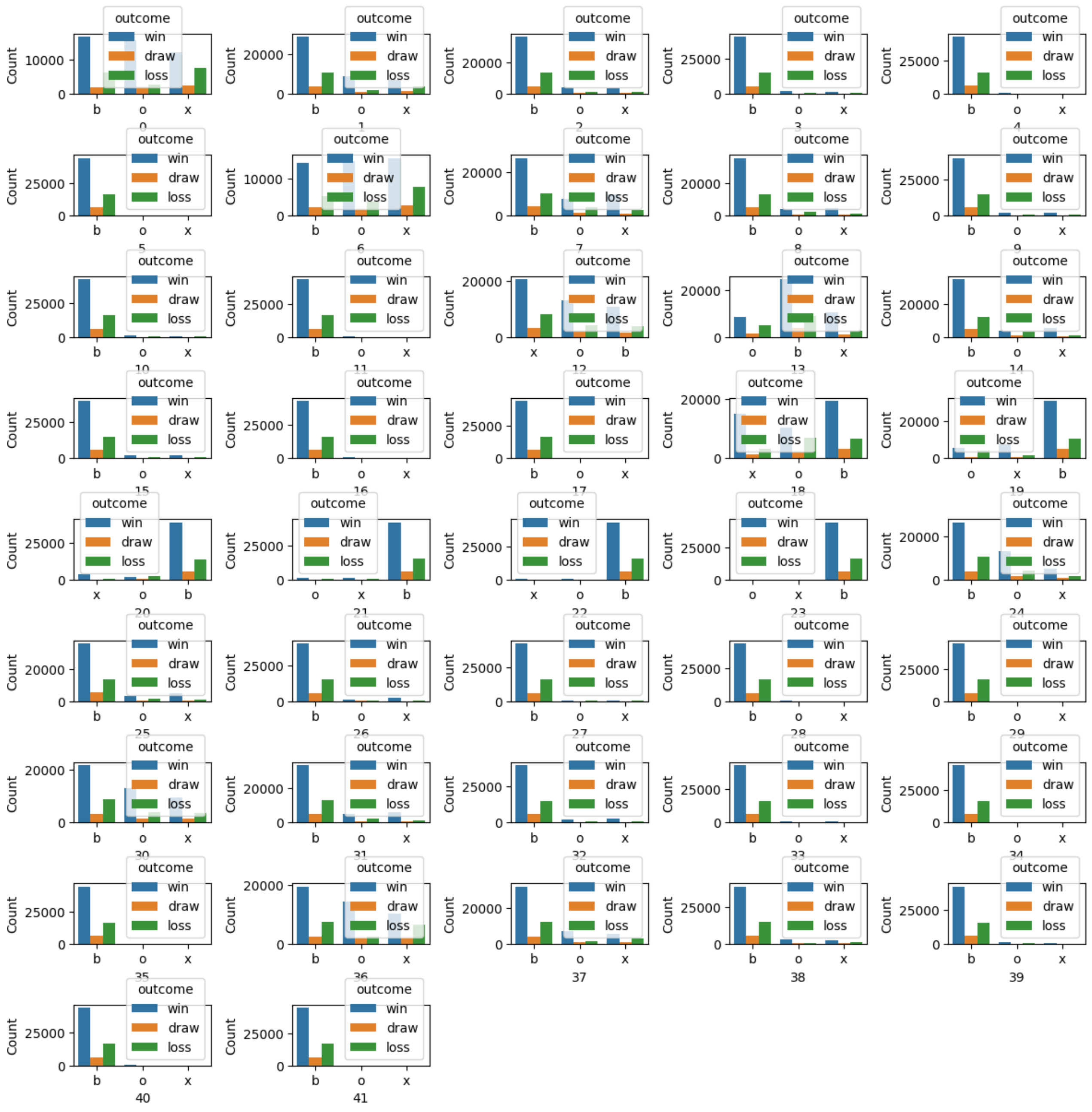
In [ ]:
```
# Barplot to understand the output distribution.
outcome_counts = data['outcome'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(outcome_counts.index, outcome_counts.values)
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.title('Connect-4 Outcome Distribution')
plt.show()
```



In [ ]:
```
# heatmap to visualize the board positions
board_positions = data.iloc[:, :-1]  # Exclude the outcome column

# Map categorical values to numerical values
board_positions_encoded = board_positions.replace({'b': 0, 'x': 1, 'o': 2})

plt.figure(figsize=(10, 8))
sns.heatmap(board_positions_encoded, cmap='Blues')
plt.xlabel('Column')
plt.ylabel('Row')
plt.title('Connect-4 Game Board Positions')
plt.show()
```



In [ ]:
```
# pairplot to visualize the relationship between each individual feature (ga
features = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '1

# Your code for creating the pairplots here...
plt.figure(figsize=(12, 12))
for i, feature in enumerate(features):
    plt.subplot(9, 5, i+1)
    sns.countplot(x=feature, hue='outcome', data=data)
    plt.xlabel(feature)
    plt.ylabel('Count')

plt.tight_layout()
plt.show()
```



Preprocessing Data

```
In [ ]: # Split the data into features (X) and target variable (y)
        X = data.iloc[:, :-1]
        y = data.iloc[:, -1]
```

```
In [ ]: # Apply one-hot encoding to categorical features
        X_encoded = X.replace({'b': 0, 'x': 1, 'o': 2})
        X_encoded
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	32	33	34	35	36	37	38	39	40	41
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	2	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
67552	1	1	0	0	0	0	2	1	2	0	...	0	0	0	0	2	2	1	0	0	0
67553	1	1	0	0	0	0	2	0	0	0	...	0	0	0	0	2	1	2	2	1	0
67554	1	1	0	0	0	0	2	2	0	0	...	0	0	0	0	2	1	1	2	0	0
67555	1	2	0	0	0	0	2	0	0	0	...	0	0	0	0	2	1	2	1	1	0
67556	1	2	2	2	1	0	2	0	0	0	...	0	0	0	0	1	0	0	0	0	0

67557 rows × 42 columns

In [ ]:

```
# Map class labels to numerical values
class_mapping = {'win': 1, 'draw': 0, 'loss': -1}
y = y.map(class_mapping)
y.head(5)
```

Out[ ]:

```
0    1
1    1
2    1
3    1
4    1
Name: outcome, dtype: int64
```

In [ ]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=
print(X_train.shape, X_test.shape)
```

```
(54045, 42) (13512, 42)
```

Logistic Regression

In [ ]:
```
# Logistic Regression
logistic_regression = LogisticRegression(max_iter=1000)
logistic_regression.fit(X_train, y_train)
logistic_regression_predictions = logistic_regression.predict(X_test)
logistic_regression_accuracy = accuracy_score(y_test, logistic_regression_pr

print("Logistic Regression Accuracy: {:.2f}%".format(logistic_regression_acc
```

```
Logistic Regression Accuracy: 65.75%
```

Decision Trees

In [ ]:
```
# Fitting a decision tree.
tree1 = DecisionTreeClassifier(random_state=1)
tree1.fit(X_train, y_train)

# Calculate the accuracy of the decision tree on the test data
accuracy = tree1.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 75.87%
```

In [ ]:
```
importances = pd.DataFrame({'feature_name': X_train.columns, 'importance': t
importances = importances.sort_values('importance', ascending=False).reset_i
importances.head(5)
```

Out[ ]:

	feature_name	importance
0	6	0.066177
1	7	0.056426
2	30	0.056181
3	0	0.055566
4	13	0.055475

Random Forest

In [ ]:
```
# Decision Trees with Random Forest
random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)
random_forest_predictions = random_forest.predict(X_test)
random_forest_accuracy = accuracy_score(y_test, random_forest_predictions)
print("Random Forest Accuracy: {:.2f}%".format(random_forest_accuracy*100))
```

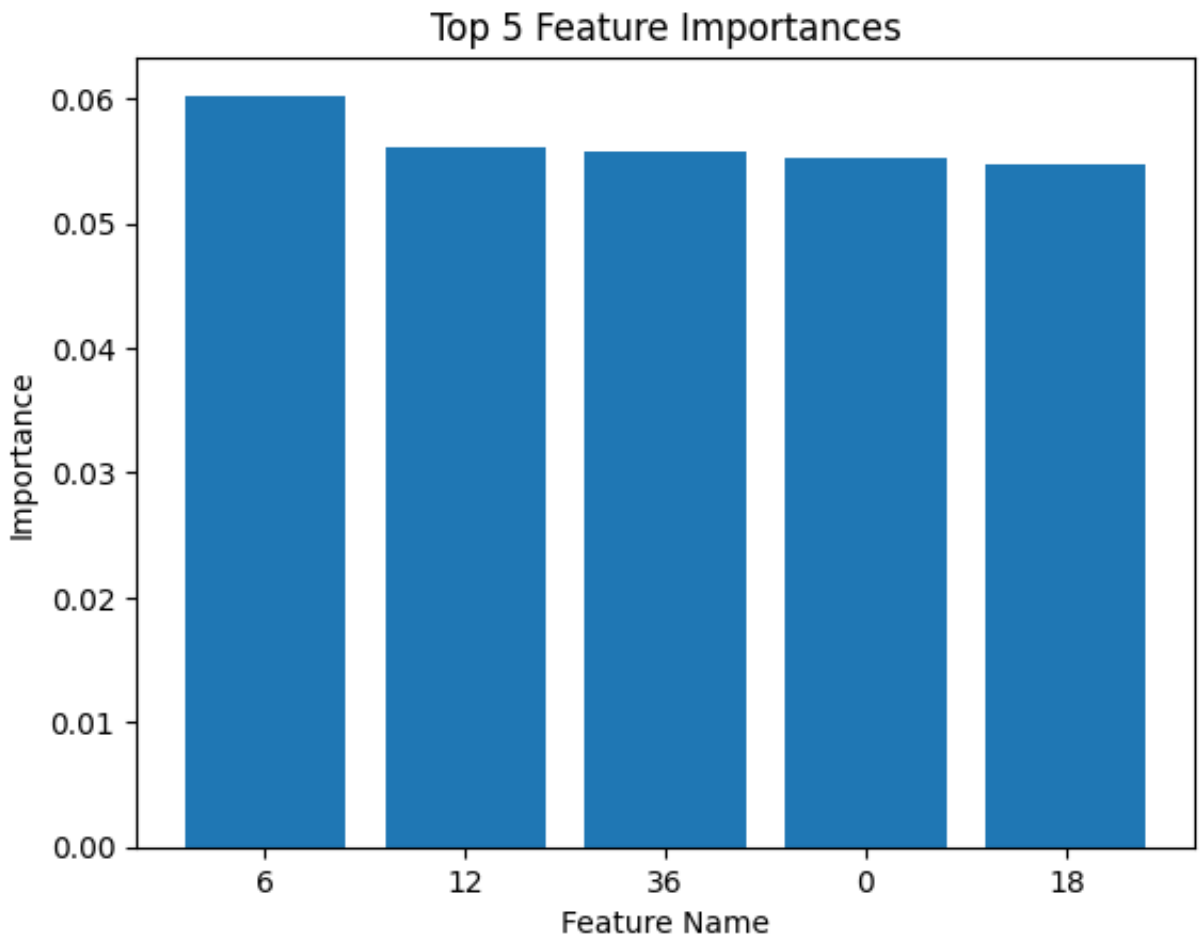
```
Random Forest Accuracy: 81.94%
```

In [ ]:
```
# create a dataframe of feature importances and their corresponding column n
importances_rf = pd.DataFrame({'Feature': X_train.columns, 'Importance': ran
importances_rf = importances_rf.sort_values('Importance', ascending=False).r
importances_rf.head(5)
```

Out[ ]:

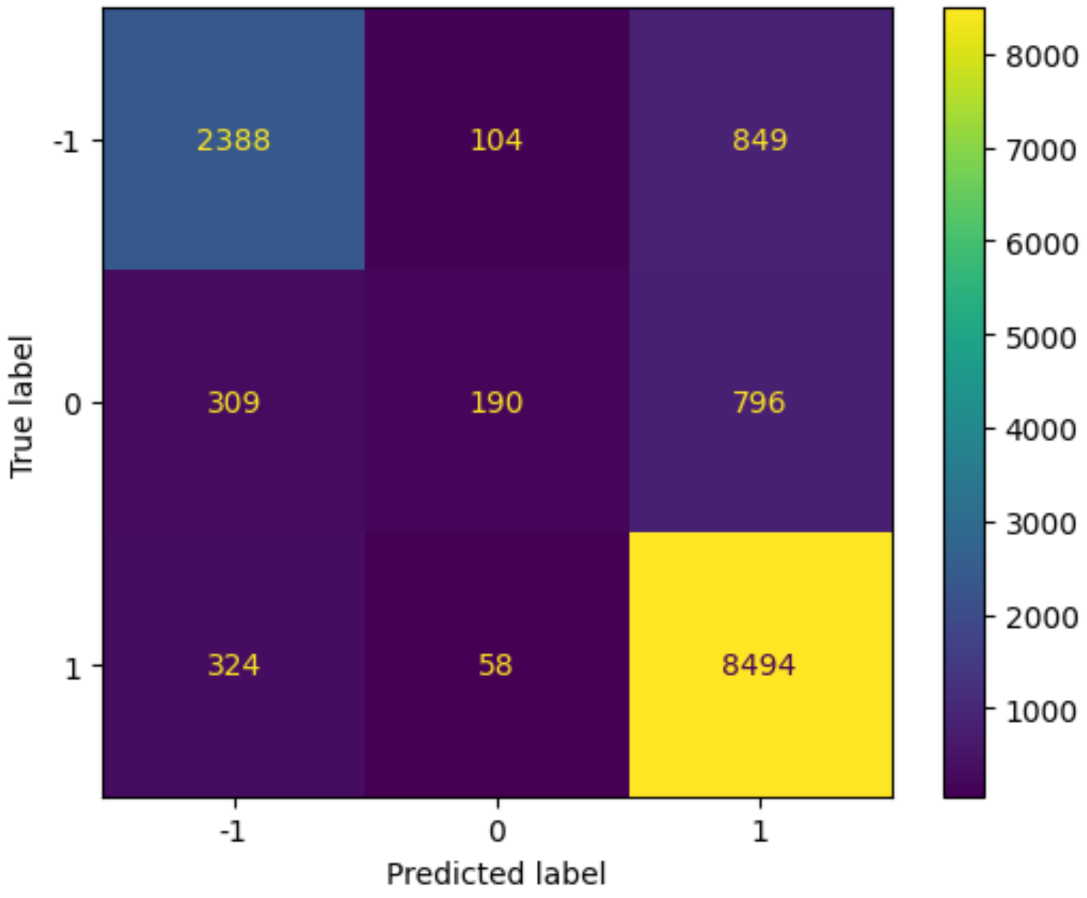
	Feature	Importance
0	6	0.060264
1	12	0.056114
2	36	0.055727
3	0	0.055322
4	18	0.054773

In [ ]:
```
# create a bar plot of feature importances
plt.bar(importances_rf.Feature[:5], importances_rf.Importance[:5])
plt.xlabel('Feature Name')
plt.ylabel('Importance')
plt.title('Top 5 Feature Importances')
plt.show()
```

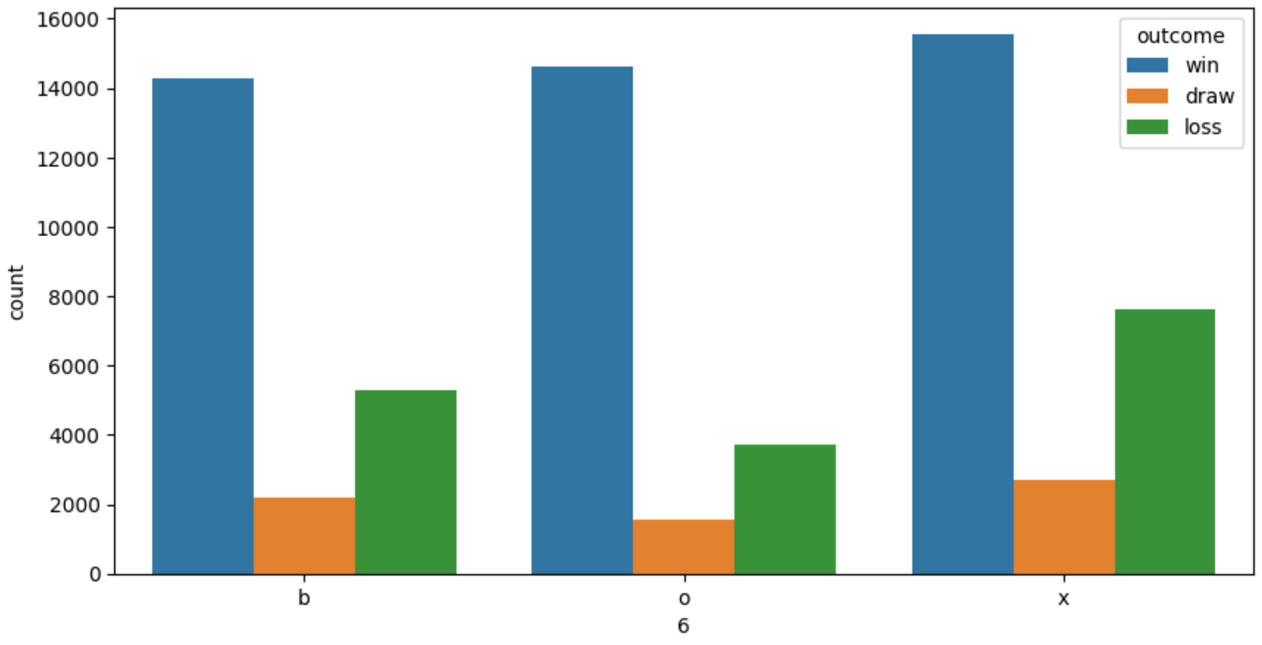


In [ ]:
```
# plot the confusion matrix
disp = ConfusionMatrixDisplay.from_predictions(y_test, random_forest_predict
disp
```

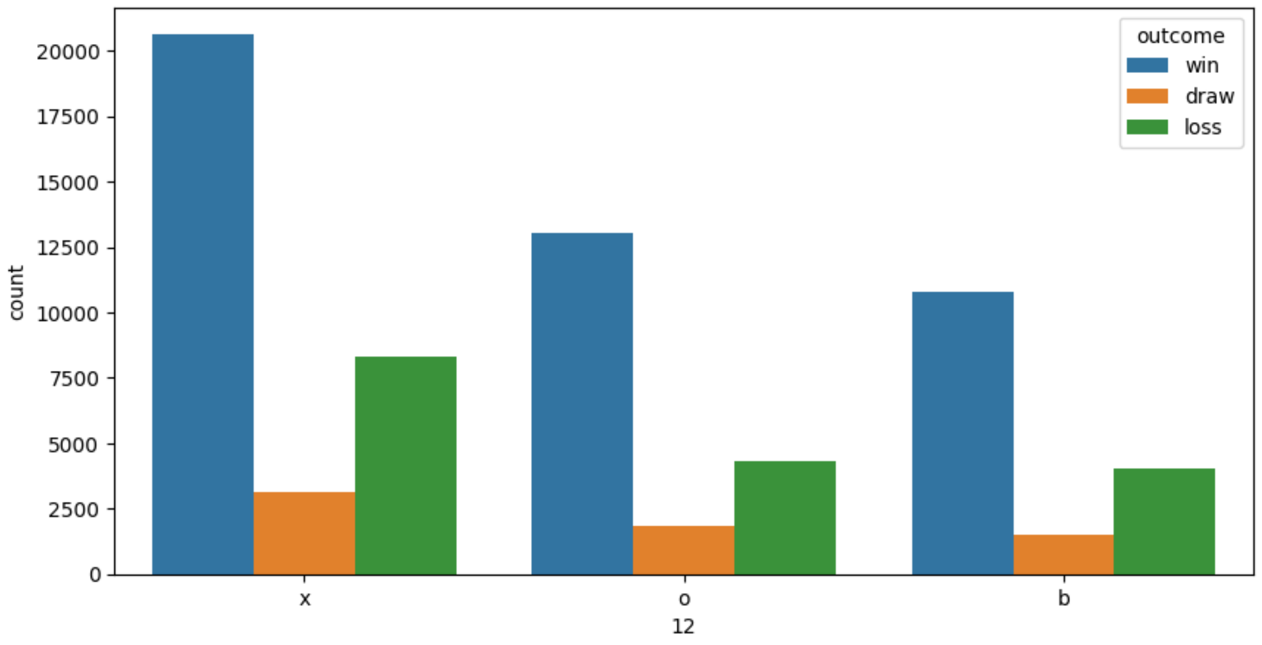
Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f30767f
eb60>
```



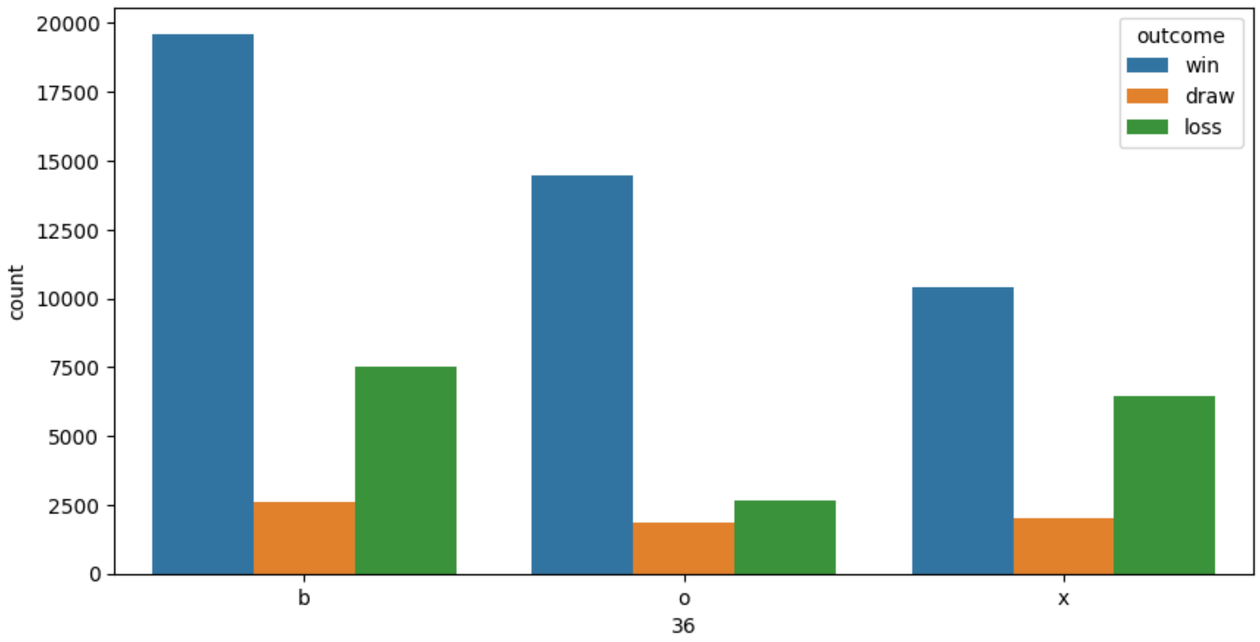
In [ ]:
```
import seaborn as sns

# Get the top 5 important variables
top_5_vars = importances_rf.iloc[:5]['Feature'].tolist()

# Plot countplots for each of the top 5 variables
for var in top_5_vars:
    plt.figure(figsize=(10, 5))
    sns.countplot(x=var, hue='outcome', data=data)
    plt.show()
```

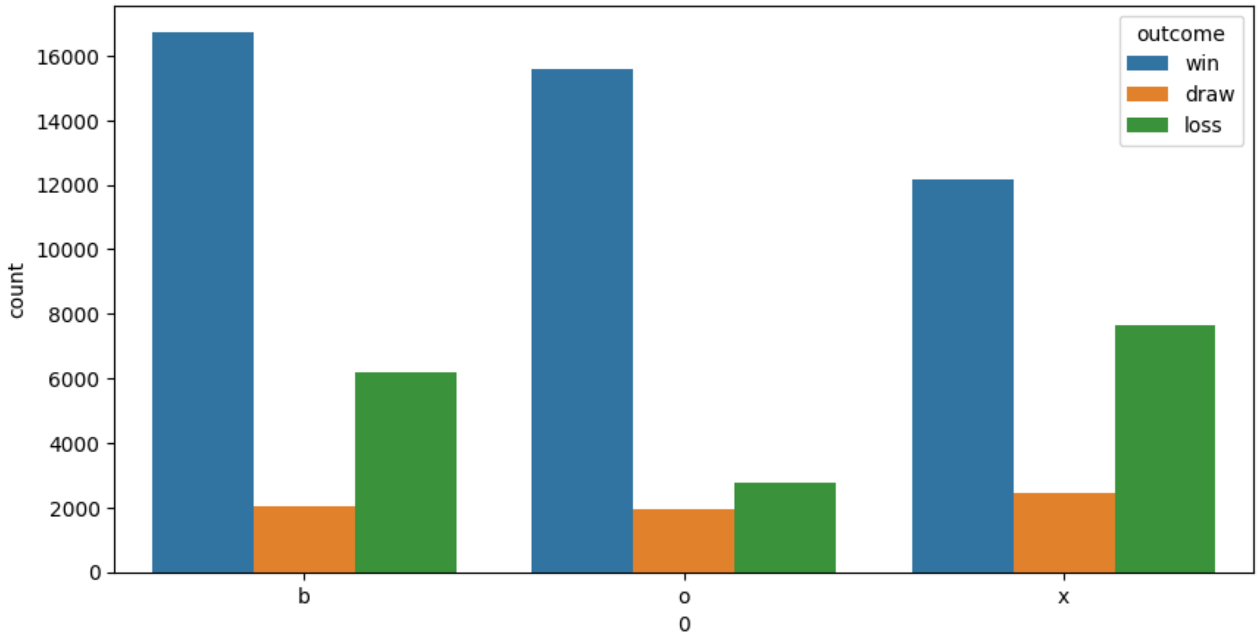


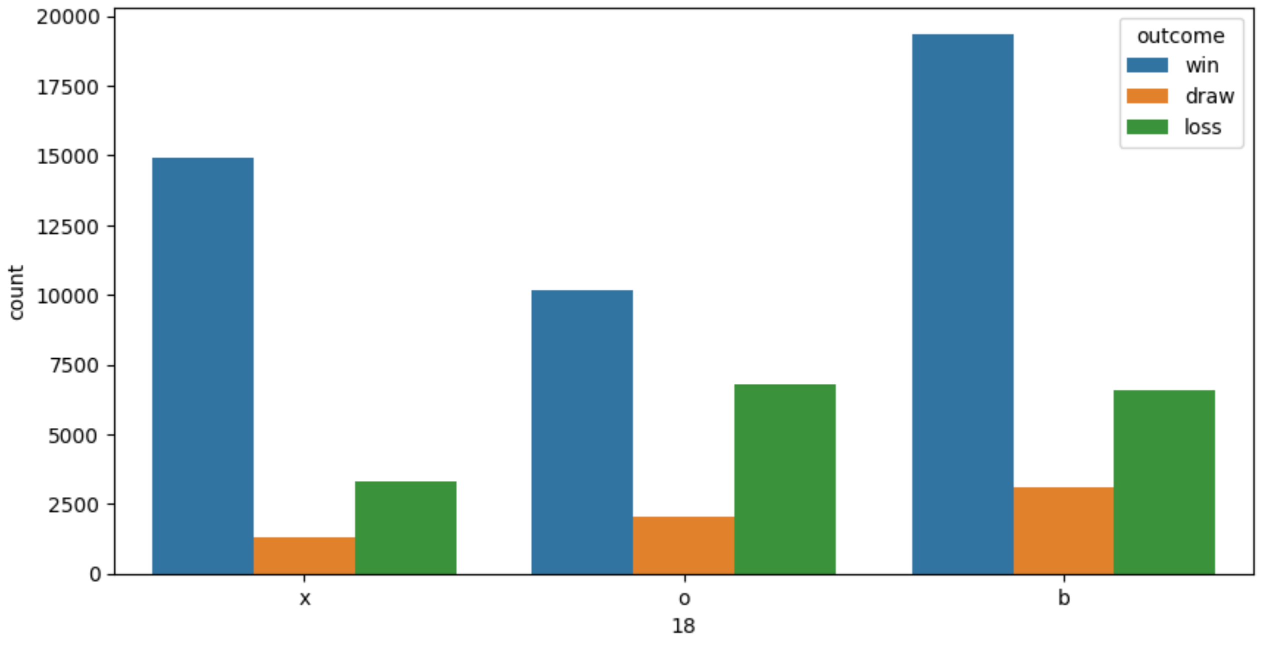
count
16000
14000
12000
10000
8000
6000
4000
2000
0
b
o
x
6
outcome
win
draw
loss



count
20000
17500
15000
12500
10000
7500
5000
2500
0
x
o
b
12
outcome
win
draw
loss


count
20000
17500
15000
12500
10000
7500
5000
2500
0
b
o
x
36
outcome
win
draw
loss


count
16000
14000
12000
10000
8000
6000
4000
2000
0
b
o
x
0
outcome
win
draw
loss



Boosting

In [ ]:
```
# Create a gradient boosting classifier
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_dep

# Fit the model on the training data
gb.fit(X_train, y_train)
```

Out[ ]:
```
▾                    GradientBoostingClassifier
GradientBoostingClassifier(max_depth=4, random_state=1)
```

In [ ]:
```
# Predict the target variable on the test data
y_pred = gb.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 78.06%
```

In [ ]:
```
# create a dataframe of feature importances and their corresponding column n
importances_boost = pd.DataFrame({'Feature': X_train.columns, 'Importance':
importances_boost = importances_boost.sort_values('Importance', ascending=Fa

importances_boost.head()
```

Out[ ]:

	Feature	Importance
0	18	0.094775
1	13	0.087695
2	20	0.087271
3	19	0.084545
4	0	0.083536

SVM

Linear SVM

In [ ]:
```
# Perform feature selection
k = 15  # number of top features to select
selector = SelectKBest(f_classif, k=k)
selector.fit(X_train, y_train)
X_train_new = selector.transform(X_train)
X_test_new = selector.transform(X_test)
```

In [ ]:
```
linear_svm = SVC(kernel='linear', cache_size=1000, verbose = True, max_iter

# Fit the model to the training data
linear_svm.fit(X_train_new, y_train)

svm_predictions = linear_svm.predict(X_test_new)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print("Linear SVM Accuracy: %.2f%%" % (svm_accuracy * 100))
```

```
[LibSVM]Linear SVM Accuracy: 49.59%
```

Radial SVM

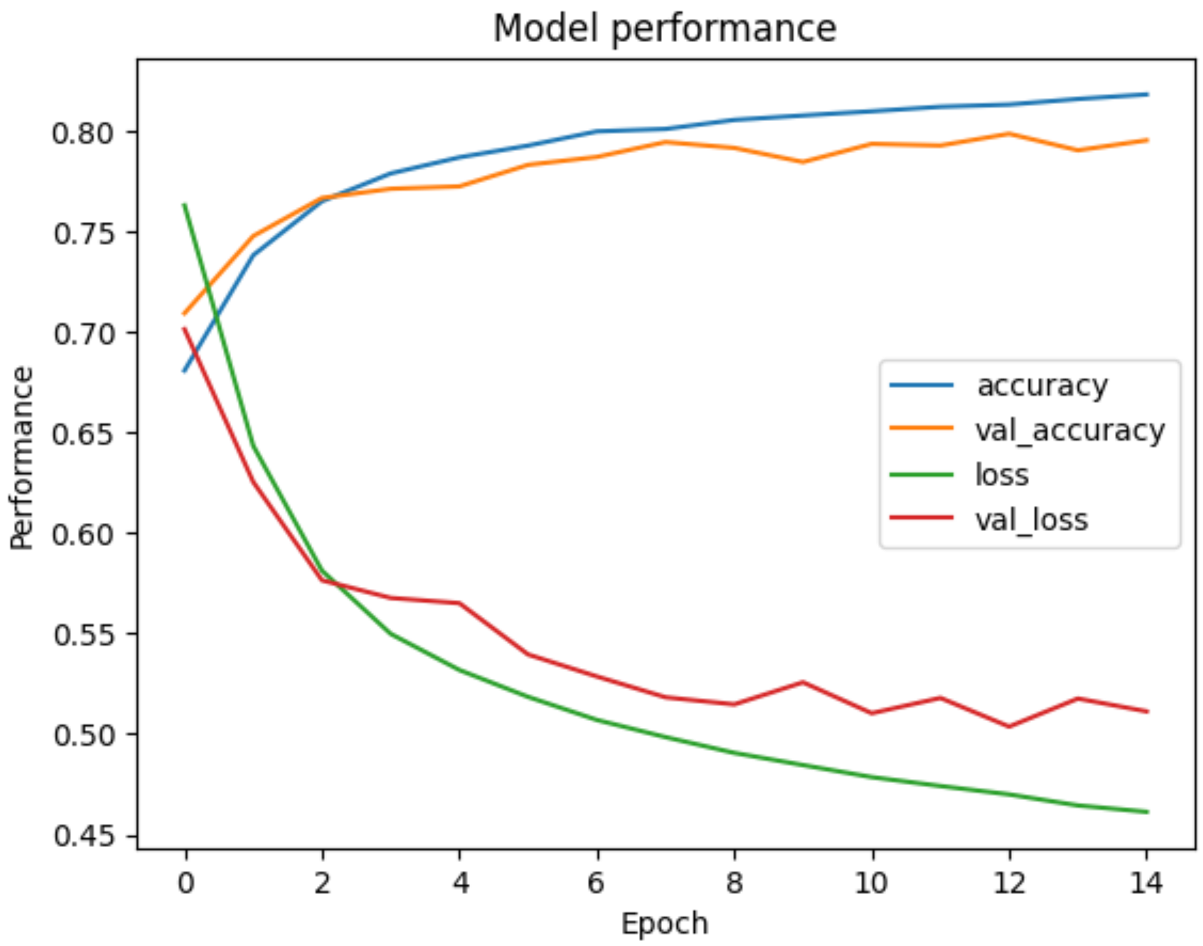
In [ ]:
```
radial_svm = SVC(kernel='rbf', cache_size=1000, verbose = True, max_iter = 1
# Fit the model to the training data
radial_svm.fit(X_train_new, y_train)

radial_svm_predictions = radial_svm.predict(X_test_new)
radial_svm_accuracy = accuracy_score(y_test, radial_svm_predictions)
print("Radial SVM Accuracy: %.2f%%" % (radial_svm_accuracy * 100))
```

```
[LibSVM]Radial SVM Accuracy: 74.52%
```

Neural Networks

```
In [ ]: # Set the seed for reproducibility
        np.random.seed(123)
        tf.random.set_seed(123)

        # Convert target variable to categorical format
        num_classes = len(class_mapping)
        y_train_categorical = to_categorical(y_train, num_classes)
        y_test_categorical = to_categorical(y_test, num_classes)

        # Define the model architecture
        model = Sequential()
        model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
        model.add(Dense(32, activation='relu'))
        model.add(Dense(num_classes, activation='softmax'))

        # Compile the model
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a

        # Train the model
        history = model.fit(X_train, y_train_categorical, epochs=15, batch_size=32,
        Epoch 1/15
        1689/1689 [==============================] - 7s 4ms/step - loss: 0.7632 - ac
        curacy: 0.6809 - val_loss: 0.7015 - val_accuracy: 0.7094
        Epoch 2/15
        1689/1689 [==============================] - 4s 2ms/step - loss: 0.6435 - ac
        curacy: 0.7383 - val_loss: 0.6255 - val_accuracy: 0.7479
        Epoch 3/15
        1689/1689 [==============================] - 4s 2ms/step - loss: 0.5813 - ac
        curacy: 0.7652 - val_loss: 0.5766 - val_accuracy: 0.7668
        Epoch 4/15
        1689/1689 [==============================] - 6s 4ms/step - loss: 0.5499 - ac
        curacy: 0.7790 - val_loss: 0.5678 - val_accuracy: 0.7714
        Epoch 5/15
        1689/1689 [==============================] - 4s 3ms/step - loss: 0.5319 - ac
        curacy: 0.7870 - val_loss: 0.5651 - val_accuracy: 0.7726
        Epoch 6/15
        1689/1689 [==============================] - 4s 3ms/step - loss: 0.5185 - ac
        curacy: 0.7929 - val_loss: 0.5396 - val_accuracy: 0.7833
        Epoch 7/15
        1689/1689 [==============================] - 6s 3ms/step - loss: 0.5070 - ac
        curacy: 0.7999 - val_loss: 0.5287 - val_accuracy: 0.7872
        Epoch 8/15
        1689/1689 [==============================] - 4s 3ms/step - loss: 0.4985 - ac
        curacy: 0.8011 - val_loss: 0.5183 - val_accuracy: 0.7946
        Epoch 9/15
        1689/1689 [==============================] - 4s 3ms/step - loss: 0.4906 - ac
        curacy: 0.8057 - val_loss: 0.5148 - val_accuracy: 0.7917
        Epoch 10/15
        1689/1689 [==============================] - 6s 4ms/step - loss: 0.4846 - ac
        curacy: 0.8079 - val_loss: 0.5258 - val_accuracy: 0.7847
        Epoch 11/15
        1689/1689 [==============================] - 4s 3ms/step - loss: 0.4786 - ac
        curacy: 0.8100 - val_loss: 0.5104 - val_accuracy: 0.7937
        Epoch 12/15
        1689/1689 [==============================] - 4s 3ms/step - loss: 0.4742 - ac
        curacy: 0.8122 - val_loss: 0.5180 - val_accuracy: 0.7929
        Epoch 13/15
        1689/1689 [==============================] - 6s 3ms/step - loss: 0.4700 - ac
        curacy: 0.8133 - val_loss: 0.5037 - val_accuracy: 0.7988
        Epoch 14/15
        1689/1689 [==============================] - 4s 3ms/step - loss: 0.4645 - ac
        curacy: 0.8161 - val_loss: 0.5177 - val_accuracy: 0.7906
        Epoch 15/15
        1689/1689 [==============================] - 4s 3ms/step - loss: 0.4613 - ac
        curacy: 0.8183 - val_loss: 0.5113 - val_accuracy: 0.7955
```

In [ ]:
```
# Plot the accuracy and loss
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()
```

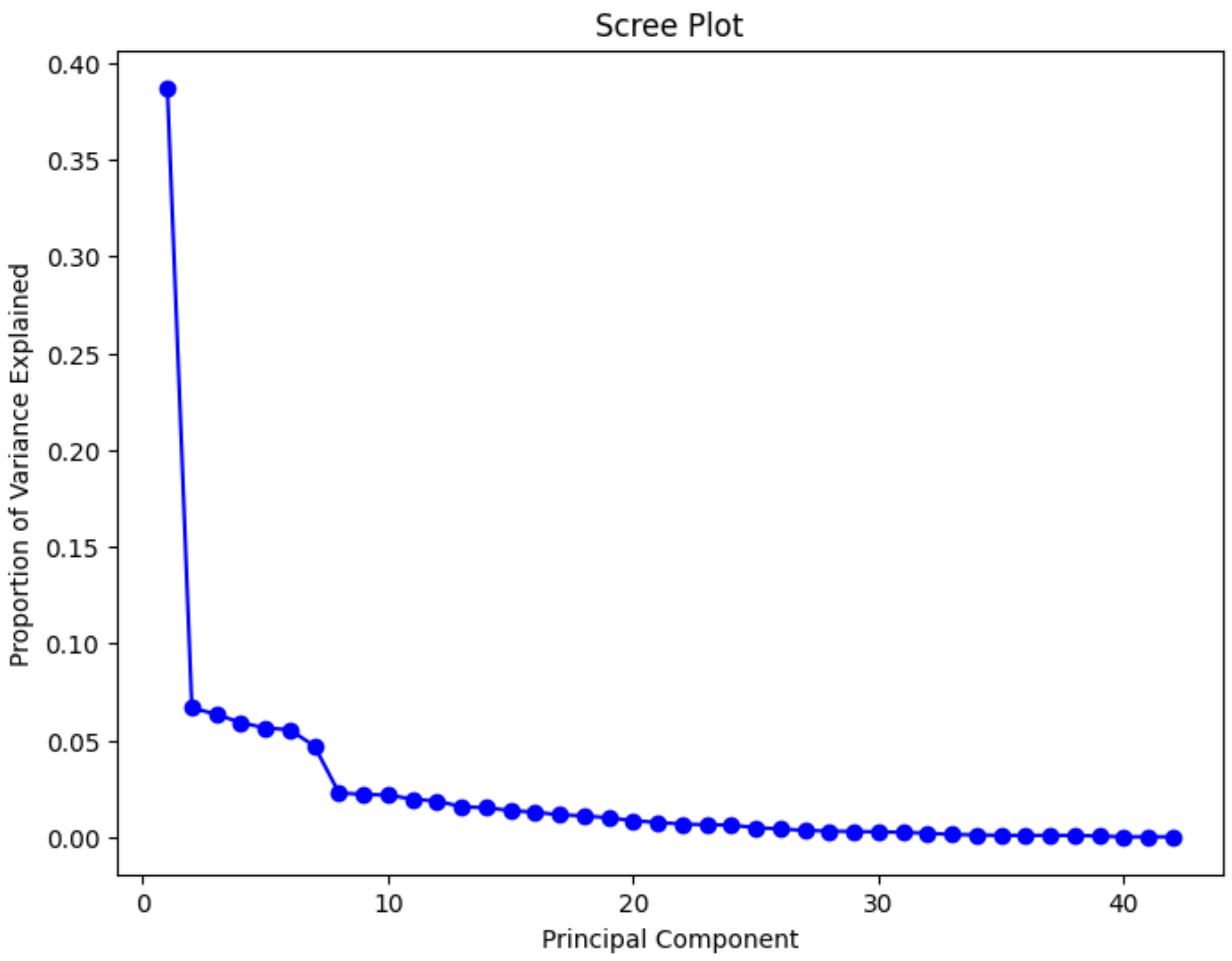


In [ ]:
```
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test_categorical)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')
```

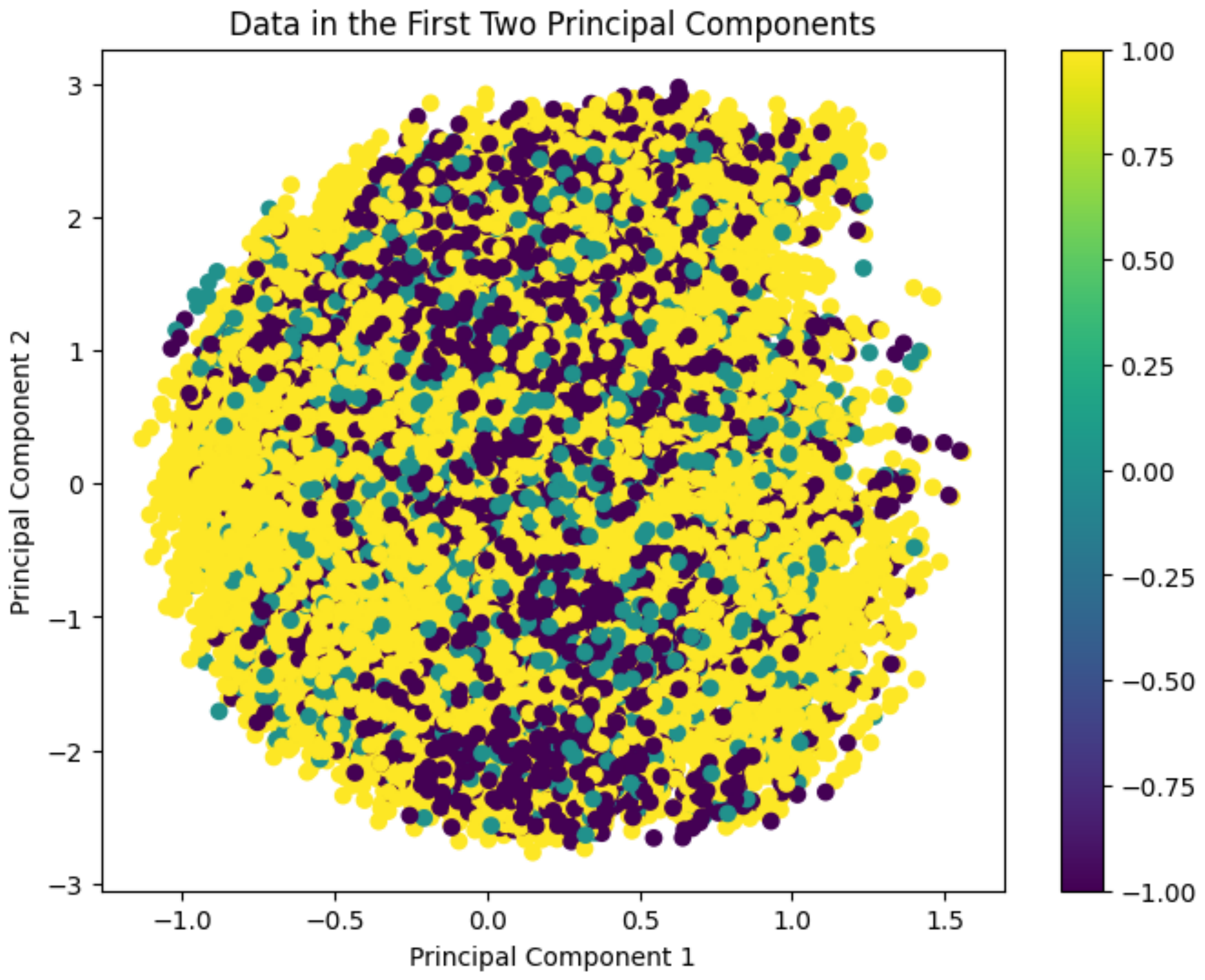
```
423/423 [==============================] - 1s 2ms/step - loss: 0.5113 - accu
racy: 0.7955
Test Loss: 0.5113
Test Accuracy: 0.7955
```

SVD

In [ ]:
```
# Perform SVD on the encoded data
U, s, Vt = np.linalg.svd(X_encoded, full_matrices=False)

# Calculate the proportion of variance explained
explained_variance = np.square(s) / np.sum(np.square(s))
cumulative_variance = np.cumsum(explained_variance)

# Plot the scree plot
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(explained_variance) + 1), explained_variance, 'bo-')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.title('Scree Plot')
plt.show()
```



In [ ]:
```python
# Project the data onto the first two principal components
X_pca = np.dot(X_encoded - np.mean(X_encoded, axis=0), Vt.T[:, :2])

# Plot the data in the first two principal components
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Data in the First Two Principal Components')
plt.colorbar()
plt.show()
```

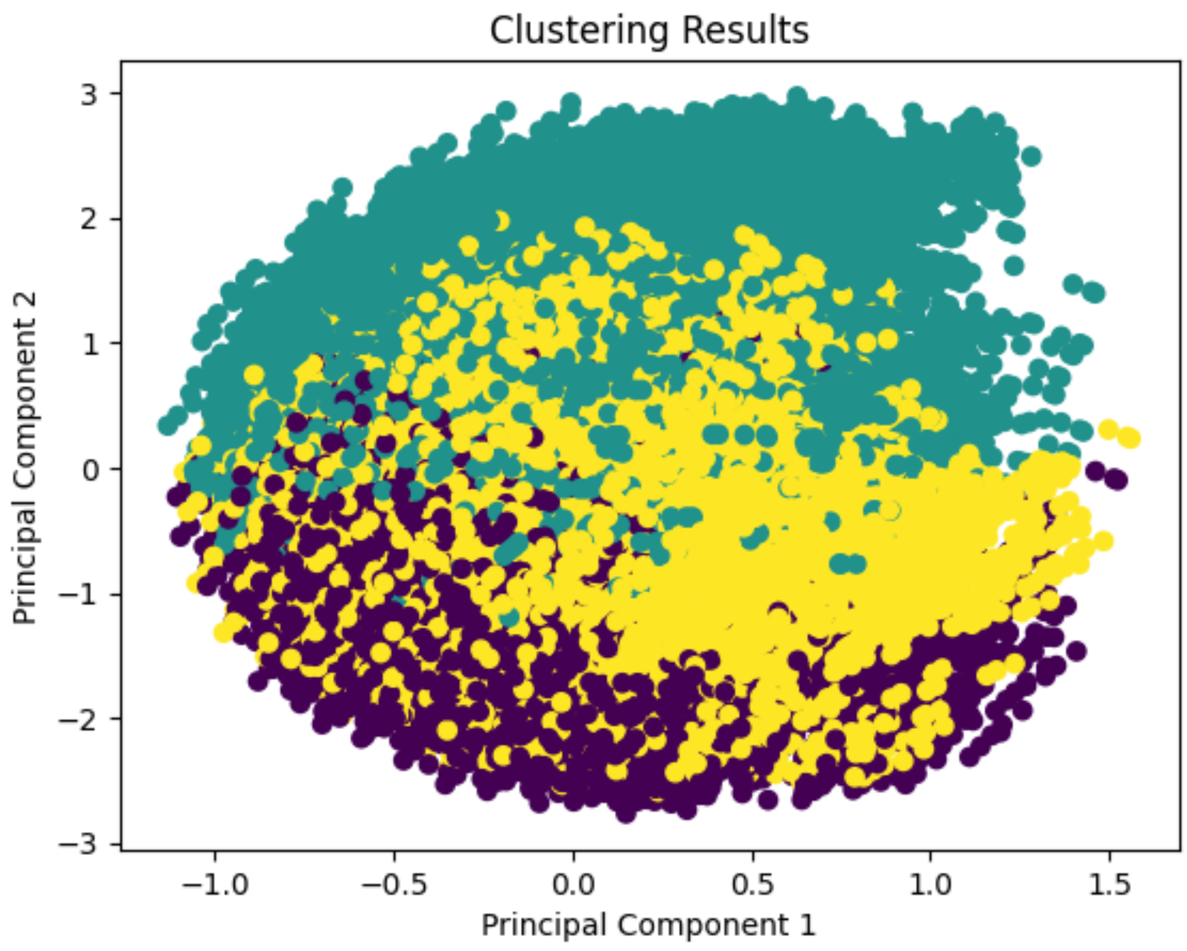


Clustering

In [ ]:
```
# Perform clustering using K-means algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_encoded)

# Assign cluster labels to the original data
data['cluster'] = kmeans.labels_

# Display observations within the same cluster
for cluster_label in data['cluster'].unique():
    cluster_data = data[data['cluster'] == cluster_label]
    print(f"Observations in Cluster {cluster_label}:")
    print(cluster_data.head())
    print()
```

```
Observations in Cluster 1:
   0  1  2  3  4  5  6  7  8  9  ... 34 35 36 37 38 39 40 41 outcome cluster
0  b  b  b  b  b  b  b  b  b  b  ...  b  b  b  b  b  b  b  b     win       1
1  b  b  b  b  b  b  b  b  b  b  ...  b  b  b  b  b  b  b  b     win       1
2  b  b  b  b  b  b  o  b  b  b  ...  b  b  b  b  b  b  b  b     win       1
3  b  b  b  b  b  b  b  b  b  b  ...  b  b  b  b  b  b  b  b     win       1
4  o  b  b  b  b  b  b  b  b  b  ...  b  b  b  b  b  b  b  b     win       1

[5 rows x 44 columns]

Observations in Cluster 0:
    0  1  2  3  4  5  6  7  8  9  ... 34 35 36 37 38 39 40 41 outcome cluste
r
8   b  b  b  b  b  b  x  o  b  b  ...  b  b  b  b  b  b  b  b     win       
0
39  b  b  b  b  b  b  x  o  b  b  ...  b  b  b  b  b  b  b  b    loss       
0
59  b  b  b  b  b  b  o  o  b  b  ...  b  b  b  b  b  b  b  b    loss       
0
64  b  b  b  b  b  b  o  o  b  b  ...  b  b  b  b  b  b  b  b    loss       
0
69  b  b  b  b  b  b  o  o  b  b  ...  b  b  b  b  b  b  b  b     win       
0

[5 rows x 44 columns]

Observations in Cluster 2:
    0  1  2  3  4  5  6  7  8  9  ... 34 35 36 37 38 39 40 41 outcome cluste
r
16  x  o  b  b  b  b  b  b  b  b  ...  b  b  b  b  b  b  b  b     win       
2
52  x  o  b  b  b  b  b  b  b  b  ...  b  b  b  b  b  b  b  b     win       
2
85  x  o  b  b  b  b  o  b  b  b  ...  b  b  b  b  b  b  b  b    draw       
2
90  o  o  b  b  b  b  b  b  b  b  ...  b  b  b  b  b  b  b  b     win       
2
93  o  o  b  b  b  b  b  b  b  b  ...  b  b  b  b  b  b  b  b     win       
2

[5 rows x 44 columns]
```

In [ ]:
```
from sklearn.metrics import completeness_score
# Perform clustering using K-means algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_encoded)

# Assign cluster labels to the original data
data['cluster'] = kmeans.labels_

# Plot the data points colored by the cluster labels
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=data['cluster'], cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Clustering Results')
plt.show()
```



In [ ]:
```
# Calculate the completeness score
completeness = completeness_score(y, data['cluster'])
print(f"Completeness Score: {completeness}")
```

```
Completeness Score: 0.002307015334908293
```

In [ ]:
```
# Evaluate the model performance using silhouette score
silhouette = silhouette_score(X_encoded, kmeans.labels_)
print("Silhouette Score:", silhouette)
```

```
Silhouette Score: 0.08820663846626903
```

In [ ]: