Presentation

by Samidha Killedar

INTRODUCTION

Predictive analysis uses historical data, statistical models, and machine learning to predict outcomes, aiding decision-making in fields like finance and healthcare. Supervised learning, involving labeled data, underpins techniques like classification (categorizing) and regression (estimating trends). Algorithms like decision trees and neural networks enhance forecasting and segmentation precision.

Project Goals

- The project aims to analyze the Titanic dataset to predict passenger survival using algorithms and knowledge predictive analysis. By handling missing values, cleaning data, and identifying key survival factors, it seeks to build an accurate predictive model, offering insights into historical biases while enhancing predictive modeling expertise.

Problem Statment

- Analyze the performance of four machine learning models—KNN, Naive Bayes, Decision Tree, and Neural Network—on the Titanic dataset to determine the most accurate model for predicting passenger survival.

- Evaluate the predictive capabilities of multiple machine learning models by comparing their accuracy, precision, and recall on the Titanic dataset to identify the best-suited algorithm for survival prediction.

- Factors significantly influenced passenger survival rates during the Titanic disaster,

Methodology

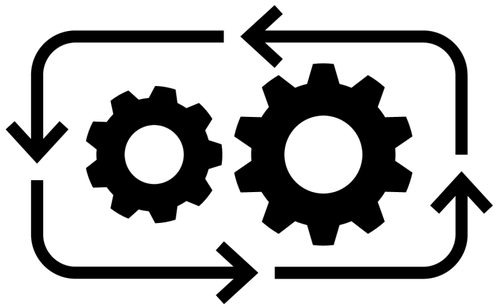
1) Data Acqusition

2) Data Preprocessing

3) Data Spliting

4) Model Selection

Process



5)Model Training

6)Model Evaluation

7) Comparison of Models

8) Deployment

Technology Stack

- Programming Language: R
- Libraries: class , caret, rpart, ggplot2 , dplyer, e1071 , rpart.plot , Neuralnet
- Algorithms: KNN, Naive Bayes, Decision Tree, Neural Network
- Visualization: ggplot2 (R), rpart.plot (R), Keras (R)
- Deployment: Github Data Source: Kaggle (https://www.kaggle.com/c/titanic/data)

Data Cleaning

Data preprocessing is the process of cleaning, transforming, and organizing raw data by handling missing values, encoding categorical variables, normalizing, and scaling to prepare it for analysis or machine learning models.

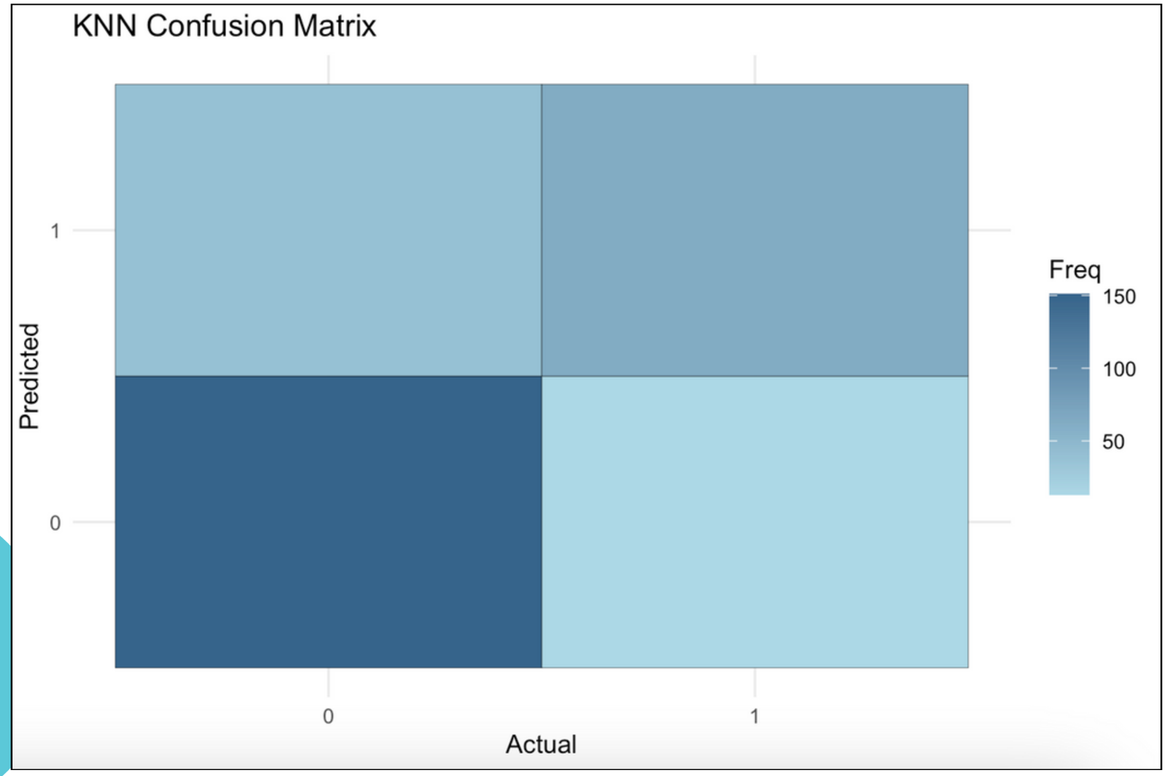
- Dropping unnecessary columns.: %>%
- Categorical data to binary numeric values: ifelse()
- Replacing na with sutaible value: is.na()
- Delete Columns: is.omit()
- Check any Na value: colSums(is.na())
- Convert to numeric data: as.Numeric()

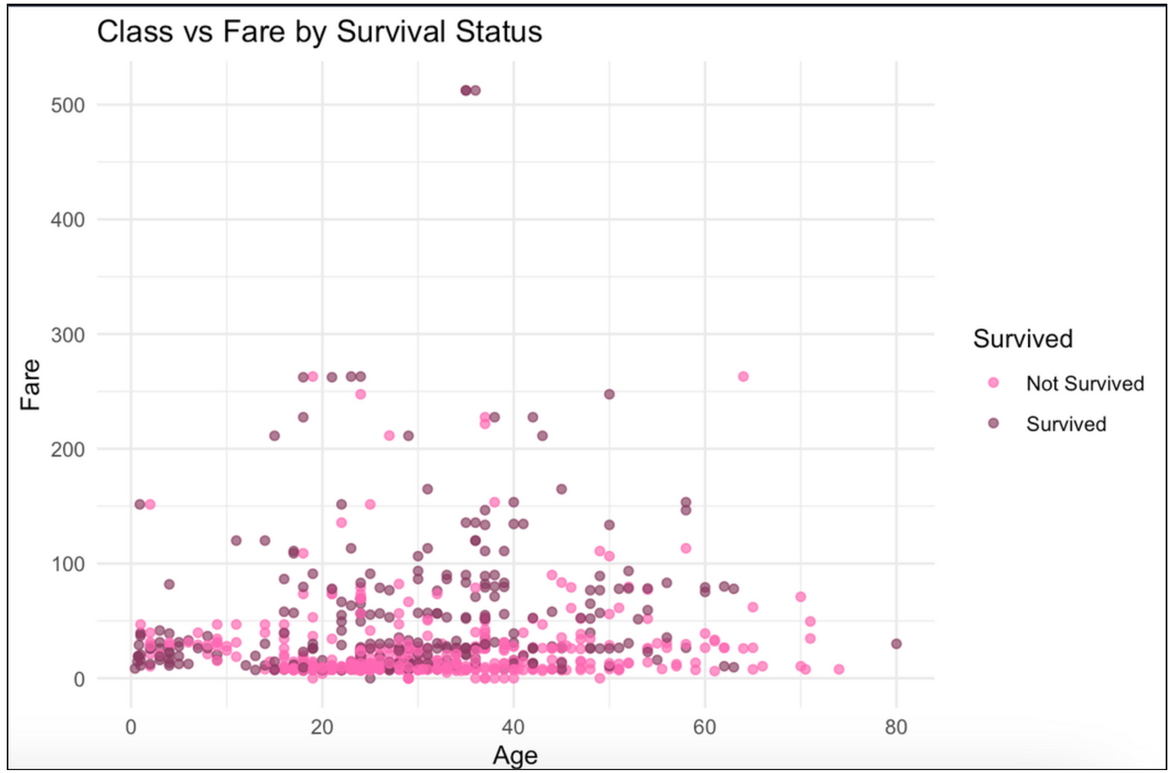
KNN

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for both classification and regression tasks. It works by identifying the k-nearest data points (neighbors) to a new, unseen data point and making predictions based on their values.

- Classification: Assigns the class most common among the k-nearest neighbors.
- Regression: Predicts the average or weighted average of k-nearest neighbors' values.
- Parameters: k: Number of neighbors (usually odd for classification).
- Distance Metric: Euclidean distances.
- Pros: Simple to implement.
  Effective for small datasets.
- Cons: Computationally expensive for large datasets.
  Sensitive to irrelevant features and noisy data.
- Applications: Image recognition, recommendation systems, and pattern classification.

KNN





	Actual	Predicted	Freq
1	0	0	151
2	1	0	13
3	0	1	39
4	1	1	63

KNN

- **70:30**

```
Accuracy: 0.6791045
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.3208955
> cat("Precision:", precision, "\n")
Precision: 0.7530864
> cat("Recall:", recall, "\n")
Recall: 0.7261905
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7393939
> # Step 6: Visualize confusion matrix using
> confusion_df <- as.data.frame(as.table(conf
```

- **80:20**

```
Accuracy: 0.8045113
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.1954887
> cat("Precision:", precision, "\n")
Precision: 0.7947368
> cat("Recall:", recall, "\n")
Recall: 0.9207317
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.8531073
>
```

- **90:10**

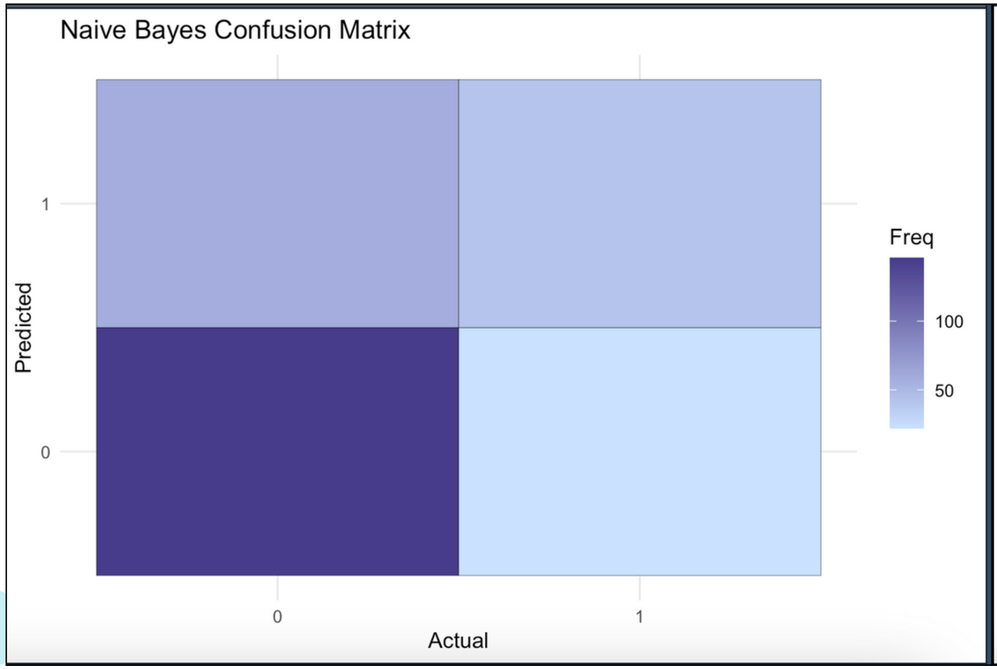
```
Accuracy: 0.6
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.4
> cat("Precision:", precision, "\n")
Precision: 0.6567164
> cat("Recall:", recall, "\n")
Recall: 0.7719298
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7096774
>
```

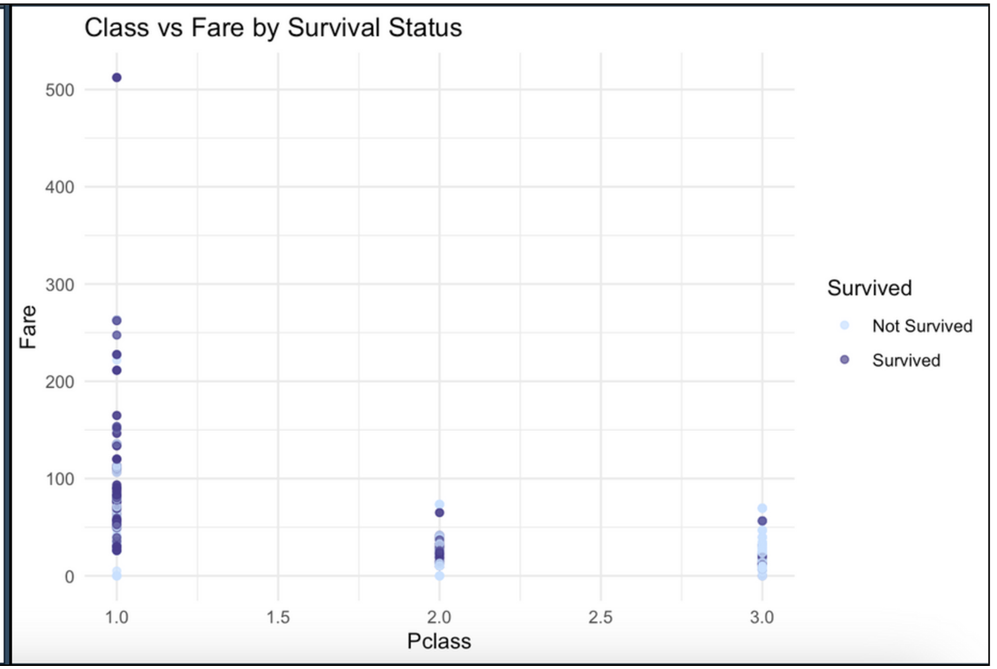
Naive Bayes

Naive Bayes is a supervised machine learning algorithm used primarily for classification tasks. It is based on Bayes' Theorem and assumes that the features are independent of each other (hence "naive"). The algorithm calculates the probability of each class for a given data point and predicts the class with the highest probability.

- Classification: Predicts the class with the highest posterior probability, calculated using Bayes' Theorem.
- Regression:Naive Bayes is not typically used for regression tasks, as it is designed for probabilistic classification.
- Parameters: Probability Calculation: Based on prior probabilities and feature likelihoods.
  Assumption: Features are independent given the class (naive assumption).
- Pros: Fast and efficient for large datasets.
  Performs well with high-dimensional data.
  Works well with categorical data.
- Cons: Assumes independence between features which is rarely true in real-world datasets.
  Struggles with datasets where feature correlations significantly affect predictions.
- Applications: Spam detection (e.g., classifying emails as spam or not spam).
  Sentiment analysis (e.g., identifying positive or negative sentiment in text).
  Document classification (e.g., categorizing articles into topics).

Naive Bayes





	Actual	Predicted	Freq
1	0	0	146
2	1	0	22
3	0	1	58
4	1	1	42

Naive Bayes

- **70:30**

```
> # Print the results
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.7014925
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.2985075
> cat("Precision:", precision, "\n")
Precision: 0.7156863
> cat("Recall:", recall, "\n")
Recall: 0.8690476
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7849462
>
```

- **80:20**

```
Accuracy: 0.6703911
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.3296089
> cat("Precision:", precision, "\n")
Precision: 0.6962963
> cat("Recall:", recall, "\n")
Recall: 0.8392857
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7611336
> # Step 7: Visualize confusion matrix using ggplot2
> confusion_df <- as.data.frame(as.table(conf_matrix$
>
```

- **90:10**

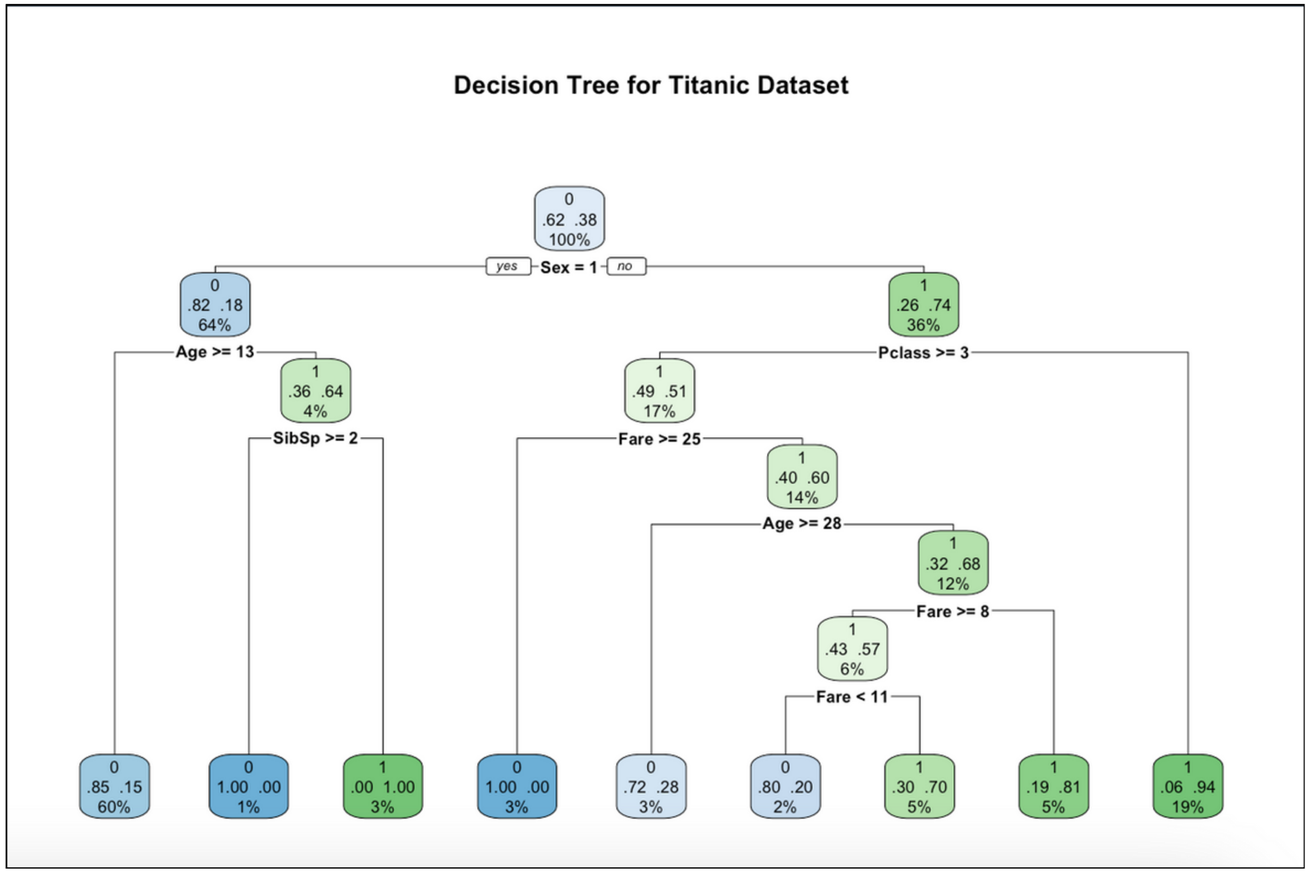
```
> # Print the results
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.6444444
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.3555556
> cat("Precision:", precision, "\n")
Precision: 0.6142857
> cat("Recall:", recall, "\n")
Recall: 0.8958333
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7288136
>
```

Decision Tree

Decision Tree is a supervised machine learning algorithm used for both classification and regression tasks. It splits the data into subsets based on feature values, forming a tree-like structure. Each internal node represents a decision based on a feature, and each leaf node represents the output. The algorithm works by recursively selecting the best feature to split the data, optimizing metrics like Gini Impurity or Information Gain.

- Classification: Predicts the class by traversing the tree from root to leaf, based on feature values and splitting criteria at each node.
- Regression: Can predict continuous values by splitting data into subsets and averaging target values in leaf nodes.
- Parameters: Splitting Criteria
- Tree Depth: Maximum allowable depth of the tree to avoid overfitting.
- Pros: Easy to interpret and visualize.
  Handles both numerical and categorical data effectively..
- Cons: Prone to overfitting, especially with deep trees.
  Can be unstable: small changes in data might result in entirely different trees.
- Applications: Credit scoring (e.g., predicting loan eligibility).
  Medical diagnosis (e.g., classifying disease types based on symptoms).
  Fraud detection (e.g., identifying fraudulent transactions).

Decision Tree




```
  Prediction Reference Freq
1          0         0   43
2          1         0    5
3          0         1   27
4          1         1   15
>
```

Decision Tree

- **70:30**

```
> # Print the results
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.6729323
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.3270677
> cat("Precision:", precision, "\n")
Precision: 0.6974359
> cat("Recall:", recall, "\n")
Recall: 0.8292683
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7576602
>
```

- **80:20**

```
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.6949153
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.3050847
> cat("Precision:", precision, "\n")
Precision: 0.7037037
> cat("Recall:", recall, "\n")
Recall: 0.8715596
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7786885
>
```

- **90:10**

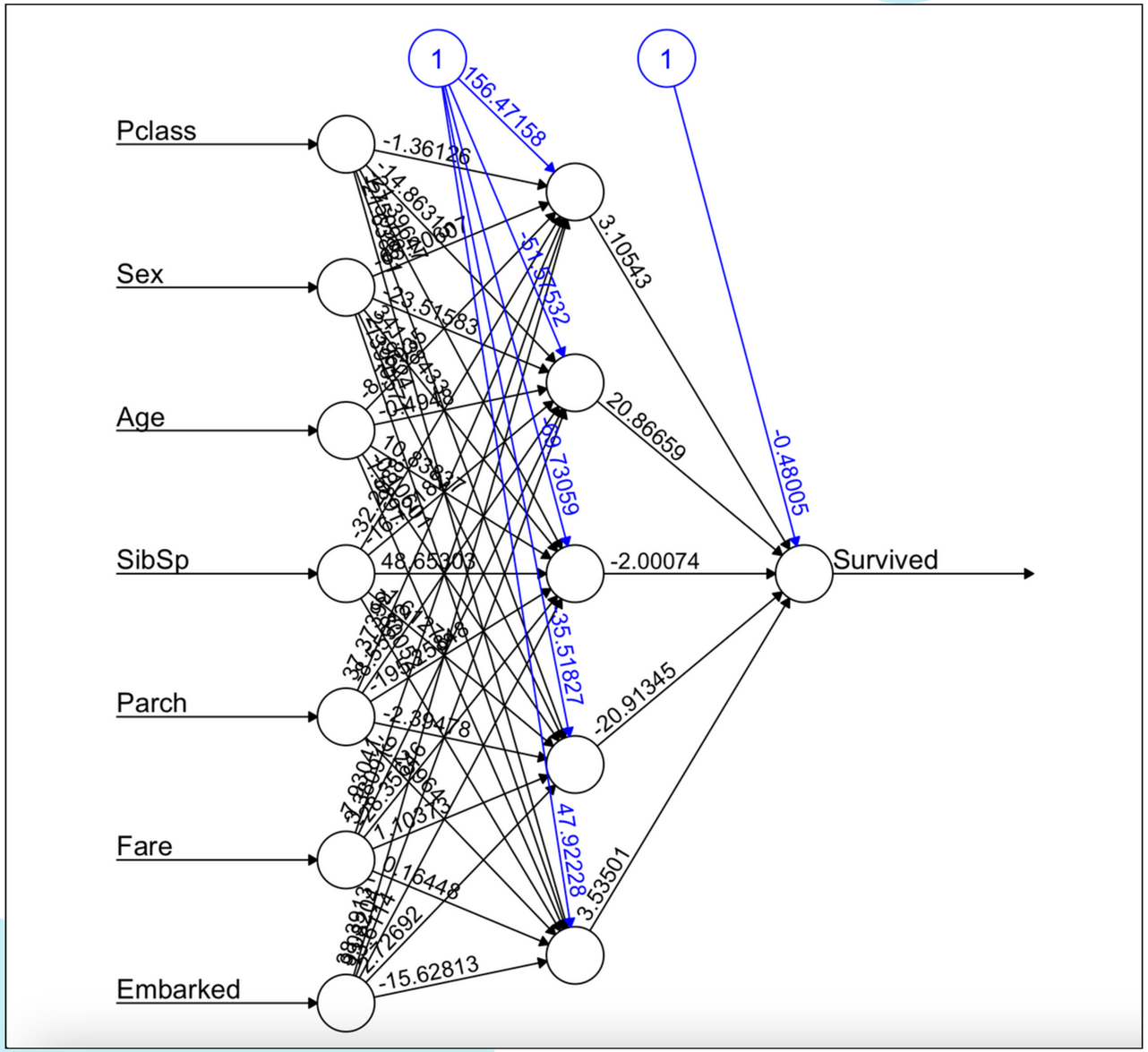
```
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.6931818
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.3068182
> cat("Precision:", precision, "\n")
Precision: 0.7014925
> cat("Recall:", recall, "\n")
Recall: 0.8703704
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7768595
```

Decision Tree

Neural Network is a supervised machine learning algorithm that mimics the human brain. It uses layers of interconnected nodes to process data, learns through backpropagation, and excels at capturing complex patterns for classification and regression tasks.

- Classification: Predicts the class by processing data through layers of interconnected neurons and using activation functions to make decisions.
- Regression: Can predict continuous values by mapping input data to output through learned weights and biases.
- Parameters: - Learning Rate: Controls the size of weight updates during training.
  - Number of Layers and Neurons: Determines the complexity of the model.
- Pros: - Handles complex, high-dimensional data effectively.
  - Excels at tasks like image recognition and natural language processing.
- Cons:- Requires significant computational resources and time.
  - Prone to overfitting on smaller datasets.
- Applications:- Image recognition (e.g., identifying objects in photos).
  Natural language processing (e.g., sentiment analysis).

Decision Tree



Decision Tree

- **70:30**

```
> # Print the results
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.6729323
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.3270677
> cat("Precision:", precision, "\n")
Precision: 0.6974359
> cat("Recall:", recall, "\n")
Recall: 0.8292683
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7576602
>
```

- **80:20**

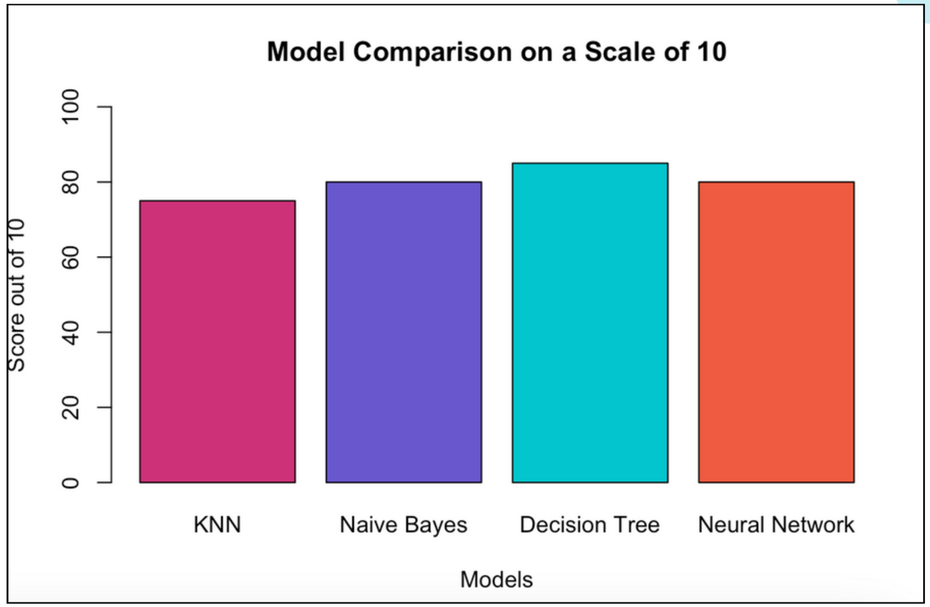
```
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.6949153
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.3050847
> cat("Precision:", precision, "\n")
Precision: 0.7037037
> cat("Recall:", recall, "\n")
Recall: 0.8715596
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7786885
>
```

- **90:10**

```
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.6931818
> cat("Error Rate:", error_rate, "\n")
Error Rate: 0.3068182
> cat("Precision:", precision, "\n")
Precision: 0.7014925
> cat("Recall:", recall, "\n")
Recall: 0.8703704
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.7768595
```

Result

- "My Decision Tree is the best fit because it effectively handles both classification and regression tasks, works well with mixed data types (numerical and categorical), and provides a clear, interpretable model for decision-making."



FUTURE SCOPE

1) Data Preprocessing Improvements.
2) Deep Dive into Unsupervised Learning
3)Interpretability and Explainability
4). Integration with Big Data Tools Objective
5)Real-time Predictions and Model Deployment Objective:

Conclusion

For the Titanic dataset, Decision Trees are the best fit due to their ability to handle categorical and numerical data, capture complex relationships, and provide interpretability. They perform particularly well with larger training sets (e.g., 80/20 or 90/10 splits) and pruning to avoid overfitting. Naive Bayes offers a simpler, computationally efficient alternative, performing adequately with smaller training sets like 70/30, though it struggles with feature dependencies. KNN requires large training data but is computationally expensive and less effective with high-dimensional features. Neural Networks, while powerful, are prone to overfitting on small datasets, making them unnecessarily complex for this task.

Thank You