*Drafted by:*  
Levi Holmstrom

**Objective**

* To understand how balanced the game logic is for each player.
* Discover variables that can tip the ratio in targeted directions.
* Possibility for logic to be trained into a model for AI opponent/difficulty.

**Approach**

1. Configure the core logic into a headless client to allow looping of the game mechanics.
2. Construct a defined framework that allows easy manipulation of variables
3. Introduce a new variable each testing phase until desired results are achieved.
4. Capture and plot results.
5. Examine results and determine if expenditure of testing is required.

**Game Logic**

**Variables (Static)**

* Rank
* Suit
* Players
* Dealer

card\_values = {'1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10, '11': 11, '12': 12, '13': 13,}  
suit\_values = {'Spades': 1, 'Hearts': 2, 'Diamonds': 3, 'Clubs': 4}

def rank\_proximity(player1\_card, player2\_card):  
 rank1 = int(player1\_card.split()[0])  
 rank2 = int(player2\_card.split()[0])  
 diff = abs(rank1 - rank2)  
 return min(diff, 13 - diff)  
  
def suit\_proximity(player\_card, dealer\_card):  
 player\_suit = player\_card.split()[2]  
 dealer\_suit = dealer\_card.split()[2]  
 if player\_suit == dealer\_suit:  
 return 0  
 else:  
 return abs(suit\_values[player\_suit] - suit\_values[dealer\_suit])  
  
def card\_distance(player1\_card, player2\_card, dealer\_card):  
 rank\_diff1 = rank\_proximity(player1\_card, dealer\_card)  
 rank\_diff2 = rank\_proximity(player2\_card, dealer\_card)  
 suit\_diff1 = suit\_proximity(player1\_card, dealer\_card)  
 suit\_diff2 = suit\_proximity(player2\_card, dealer\_card)  
 player1\_distance = rank\_diff1 + suit\_diff1  
 player2\_distance = rank\_diff2 + suit\_diff2  
 return player1\_distance, player2\_distance

**Win Condition Logic**

* Player with the shortest count to the dealer rank wins
* If rank proximity is a tie, we use a static suit value to determine the winner.
* If rank and suit proximity are both tied, a pair of dice is rolled for each player. The highest scoring dice roll determines the winner.

**Tie Determination**

* Dice Roll
* Value Comparison (Higher Wins)

player1\_dice = roll\_dice()  
player2\_dice = roll\_dice()  
  
if player1\_dice > player2\_dice:

score\_round['player1'] += 1

elif player1\_dice < player2\_dice:

score\_round['player2'] += 1

else:

tie = True

**Deck Shuffle/Deal**

* Calls static values
* 5 Card Range Randomisation
* Player & Dealer Hand Population

deck = [(f"{k} of {s}", card\_values[k], suit\_values[s]) for k in card\_values for s in suit\_values]

random.shuffle(deck)  
player1\_cards = [deck.pop() for \_ in range(5)]  
player2\_cards = [deck.pop() for \_ in range(5)]  
dealer\_cards = [deck.pop() for \_ in range(5)]

import random

# Set a time-based seed for the random algorithm  
random.seed(time.time())

**Randomiser**

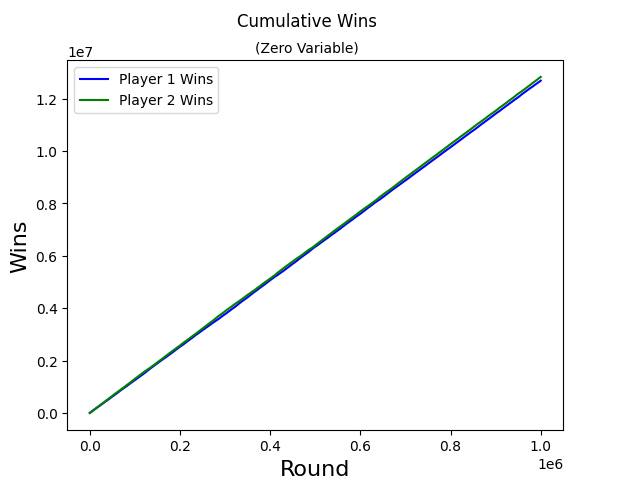
* Mersenne Twister PRNG core
* Inclusion of time-based seed for increased random probability.

**Baseline / Balance**

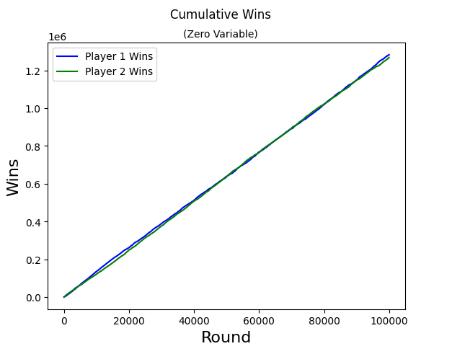
Player 1- & 2- Win Condition Totals.

3 Cycle Ratios to determine dataset accuracy.

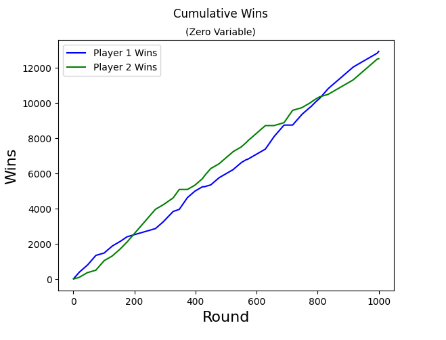
Zero game variables appended.



Cycle Count: 1,000,000



Cycle Count: 100,000



Cycle Count: 1,000



*1 x 106 provides a thoroughly accurate baseline for preliminary results, with each player having almost a 50/50 chance of winning.*

*Also observed is the geometric form the graph lines take; it almost resembles DNA strands… interesting!*

*More research into this by-result is warranted.*

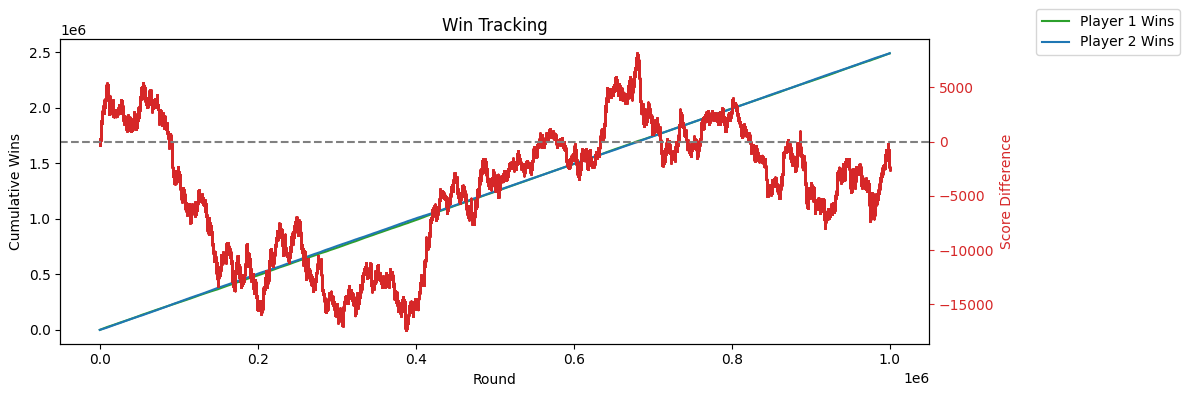
**Testing**

**Win Tracking**

Visualise the number of times players are winning on the timeline.

+0 = Player 1 (*above line*)

-0 = Player 2 (*below line*)



***Player 2*** *has a significant lead at the start of game and peaks out at around 16k score ahead at 400k games.*

***Player 1*** *has a significant lead from 400k to 700k peeking out at 7k ahead in score.*

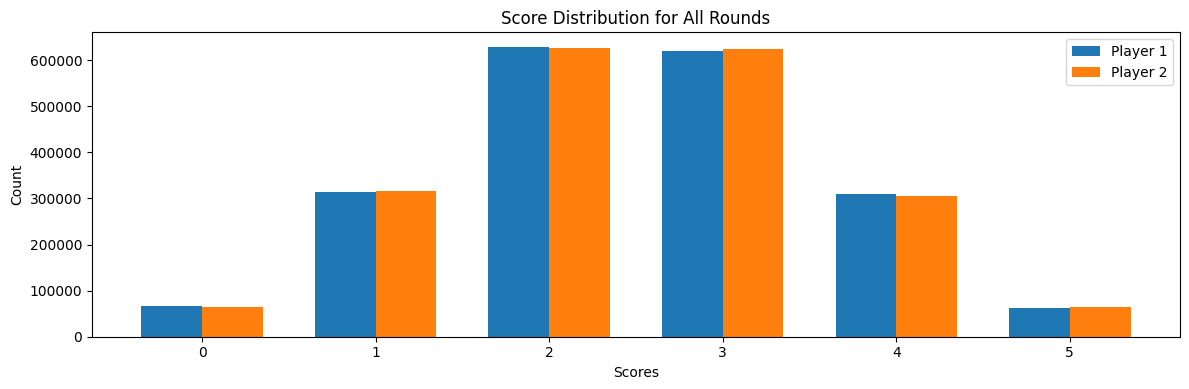
***The remaining rounds*** *play out evenly, with the score difference returning to the zero line at the end of the game, indicating a fair chance for each player*

**Testing continued…**

**Score Distribution**

Visualise the score value designation for each round.

Help determine common vs rare scores



***Testing*** *reports even score distribution for both players. This is exceptionally handy for setting a preliminary baseline betting ratio.*

*0 & 5 – 5x*

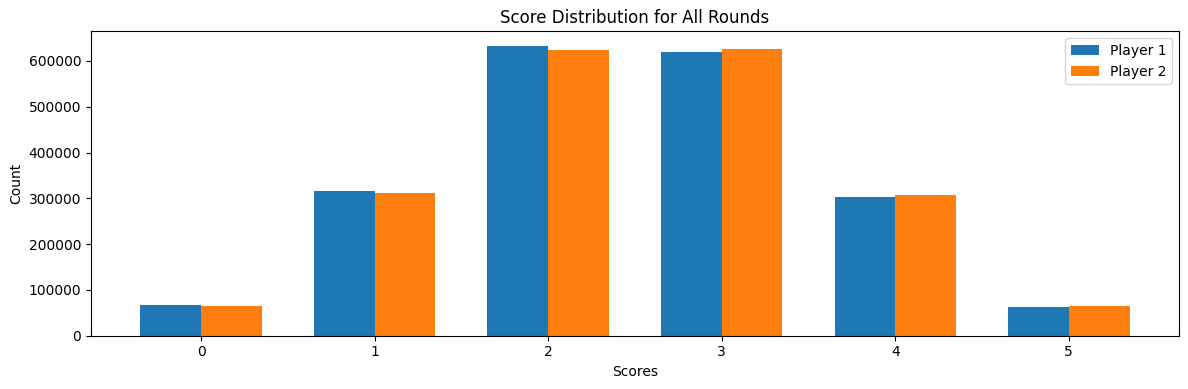
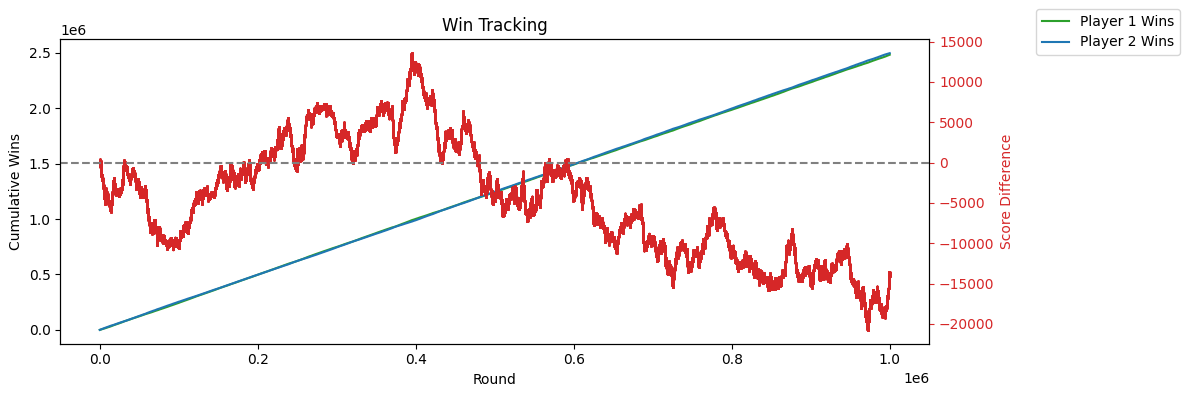
*1 & 4 – 2.5x*

*2 & 3 – 1.5x*

**Introduce Variable**

Game Mechanic Variable Initiated

5th Card Swapped cycle to simulate Blind Eye



***No noticeable deviation*** *from the initial game balance results. Reinforces game mechanics are performing as intended.*

***The results show*** *that a bias penetration is possible with a code injection.*

*This could be due to the simple nature of the Random Algorithm being used.*

*A more advanced algorithm is warranted for testing to see if a reduction in baseline deviation is achieved.*

**Outcome Manipulation**

Bias penetration prevention and detection threshold.

**100,000** Rounds x 8

Artificial Variables:

**Player 1** – More odds of receiving higher suit from dealer. (Spades)

**Player 2** – More Odds of receiving lower rank suits from dealer. (Clubs)

def deal\_biased\_cards(deck, bias\_factor=1):  
 high\_ranked\_cards = [card for card in deck if card[1] > 6]  
 low\_ranked\_cards = [card for card in deck if card[1] <= 6]  
  
 num\_high\_ranked\_cards = int(5 \* bias\_factor)  
 num\_low\_ranked\_cards = 5 - num\_high\_ranked\_cards  
  
 random.shuffle(high\_ranked\_cards)  
 random.shuffle(low\_ranked\_cards)  
  
 biased\_hand\_high = high\_ranked\_cards[:num\_high\_ranked\_cards]  
 biased\_hand\_low = low\_ranked\_cards[:num\_low\_ranked\_cards]  
 random.shuffle(biased\_hand\_high)  
 random.shuffle(biased\_hand\_low)  
  
 for card in biased\_hand\_high + biased\_hand\_low:  
 deck.remove(card)  
  
 return biased\_hand\_high + biased\_hand\_low

**Manipulation Definition**

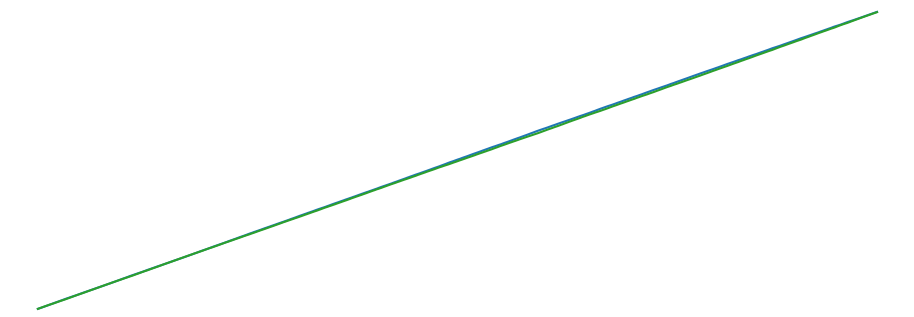
# Shuffle deck and deal cards  
 deck = [(f"{k} of {s}", card\_values[k], suit\_values[s]) for k in card\_values for s in suit\_values for \_ in range(int(k))]  
 random.shuffle(deck)  
 player1\_cards = deal\_biased\_cards(deck, bias\_factor=0.5) # You can adjust the bias\_factor to increase or decrease the bias  
 player2\_cards = deal\_biased\_cards(deck, bias\_factor=1 - 0.5) # Subtract the bias\_factor from 1 to create an inverse bias for player 2  
 dealer\_cards = [deck.pop() for \_ in range(5)]

**Manipulated Shuffle/Deal**

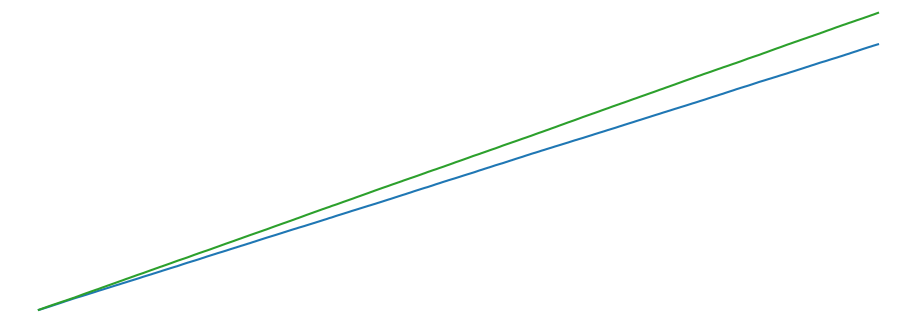
**Bias Testing Results**

**Testing continued…**

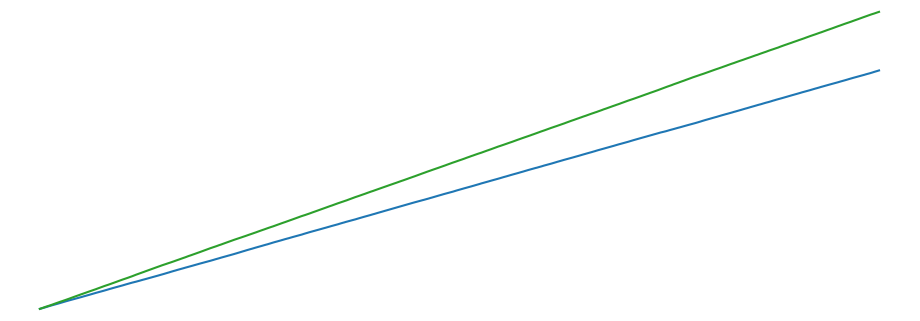
**50% Bias**: Player 1 - 247143 (49.43%), Player 2 - 246889 (49.38%), Ties - 5968 (1.19%)



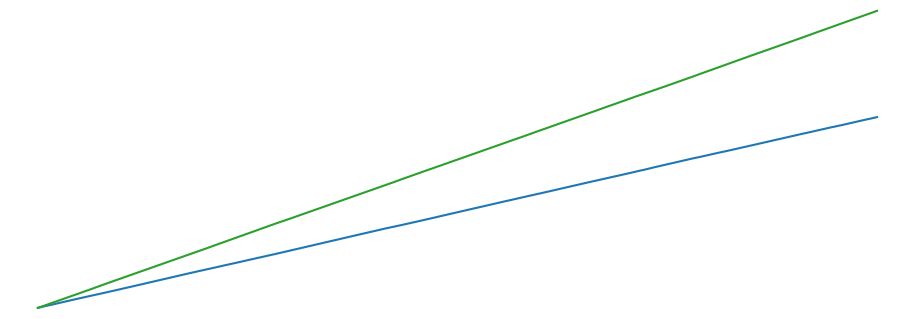
**40% Bias**: Player 1 - 233706 (46.74%), Player 2 - 261365 (52.27%), Ties - 4929 (0.99%)



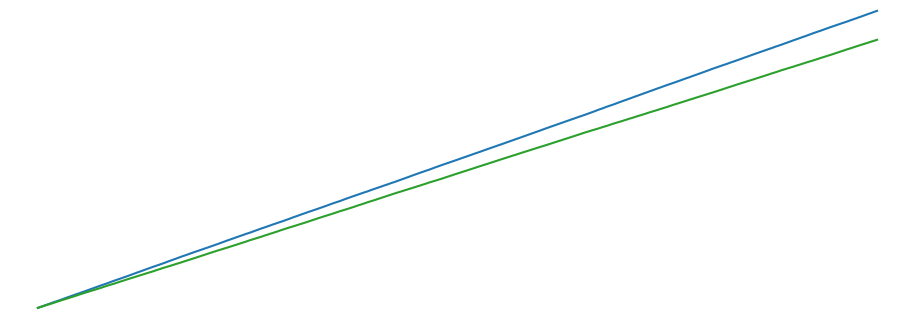
**30% Bias**: Player 1 - 220664 (44.13%), Player 2 - 274886 (54.98%), Ties - 4450 (0.89%)



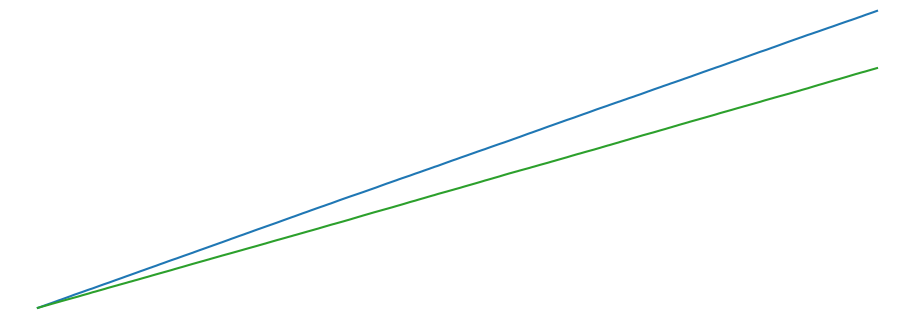
**10% Bias**: Player 1 - 194410 (38.88%), Player 2 - 302711 (60.54%), Ties - 2879 (0.58%)



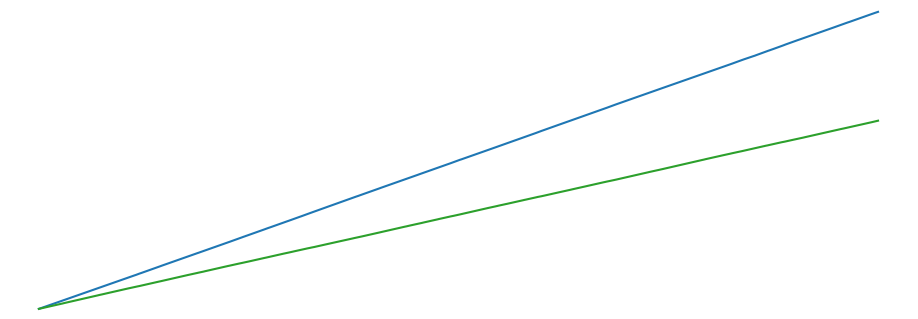
**60% Bias**: Player 1 - 260281 (52.06%), Player 2 - 234913 (46.98%), Ties - 4806 (0.96%)



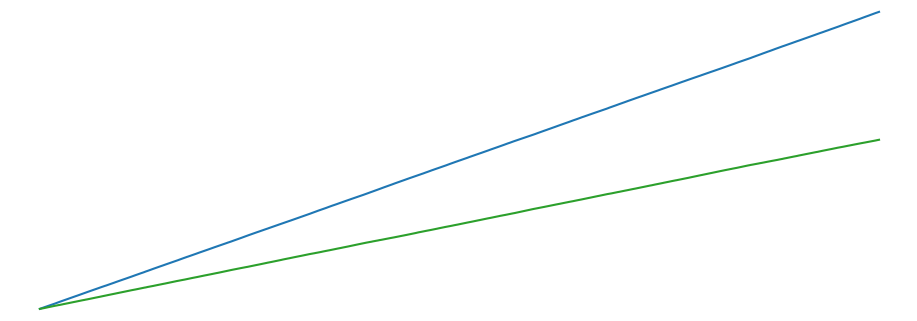
**70% Bias**: Player 1 - 274383 (54.88%), Player 2 - 221515 (44.30%), Ties - 4102 (0.82%)



**80% Bias**: Player 1 - 304389 (60.88%), Player 2 - 192837 (38.57%), Ties - 2774 (0.55%)



**100% Bias**: Player 1 - 317370 (63.47%), Player 2 - 180682 (36.14%), Ties - 1948 (0.39%)



Title: Predictive Model for [Target Variable] in [Context/Industry]

Abstract:

[Provide a brief summary of the study's goals, methods, key findings, and implications for stakeholders]

Introduction:

1.1 Background

[Provide context about the industry and the need for a predictive model]

1.2 Objective

[State the main objective of the study and the specific aims]

Literature Review:

[Summarize previous research relevant to your study, including predictive models and techniques in your field]

Methodology:

3.1 Dataset

3.1.1 Data Sources

[Describe the data sources, their credibility, and their relevance to the study]

3.1.2 Data Preprocessing

[Explain data cleaning, transformation, and feature engineering techniques]

3.2 Model Selection and Rationale

[Explain the choice of predictive model(s) and justify the selection]

3.3 Model Development and Evaluation

[Describe the process of model development, including parameter tuning, cross-validation, and evaluation metrics]

Results:

4.1 Model Performance

[Present the model's performance metrics, such as accuracy, precision, recall, F1 score, and/or RMSE]

4.2 Feature Importance

[Discuss the most influential features in the model and their relevance to the prediction task]

4.3 Model Interpretation

[Provide a general interpretation of the model's results and how they relate to the study objectives]

Discussion:

5.1 Comparison with Existing Studies

[Compare the results of your study with previous research in the field]

5.2 Limitations

[Discuss any limitations of the study, such as biases, data quality, or model assumptions]

5.3 Implications for Stakeholders

[Explain the practical implications of the model's findings for stakeholders, such as decision-making or risk management]

Conclusion:

[Summarize the main findings of the study, their significance, and potential future research directions]

References:

[List all cited sources in a consistent citation style, such as APA, MLA, or IEEE]

Appendices (optional):

[Include any supplementary material, such as code snippets, detailed data preprocessing steps, or additional analysis results]

To convey the relevant information to stakeholders, focus on the following key points:

The objective of the study and its relevance to the industry

The methodology, including data sources, model selection, and evaluation

The model's performance and its implications for decision-making or risk management

A comparison with existing research or models in the field

Any limitations of the study and their potential impact on the results

Practical recommendations for stakeholders based on the study's findings