#FOOTBALL PLAYER CLASSIFICATION ANALYSIS

#ABSTRACT

Understanding the foundations of data simulation and the various applications of data across industries is the aim of this study. The 19238 rows and 110 columns of the FIFA football players list of the year 2022 are used in various data analysis processes, such as simulating discrete and continuous random variables, which involve statistical analysis, visualisation, verification of the central limit theorem, outlier detection, and probability calculations. Furthermore, we simulate Markov Chains and Real Data Analysis, encompassing variance reduction techniques, factor analysis, recurrent events, ergodicity, joint distribution analysis, sensitivity analysis, and visualisation. We also compare different approaches to simulation.

Data on FIFA football players list is included with the target variable in the dataset, the overall score of the different players. It contains the following data: Attacking, Skills, Defense, Mentality, GK Skills, age, height, weight, calculation scores such as move, dribble, goalkeeping ,club name, etc. The project's goal is to comprehend the philosophy behind simulating various data kinds under various circumstances in many disciplines.

#CHAPTER 1

#INTRODUCTION

The Data Set contains details on video game of FIFA football players for the year 2022. The target variable includes the age of the various players which could help to give the insights of how to choose the teams. The other variables like player attributes with statistics as Attacking, Skills, Defense, Mentality, GK Skills, player personal data like Nationality, Club, DateOfBirth, Wage, Salary, etc. seem to be potential predictive indicators that may correlate or relate to create the ideal team in the game. The data could be used to analyze the key drivers to select the team and build a model to predict the best team out of it. This dataset contains details on 19,238 football players including information on their demographics, plauer status, individual details, and a target variable indicating the age of the player so that it could be analysed how to slect the player based on their overall score of all the given attributes. There are 110 feature variables on each player that could relate to selecting the team along with the binary target overall score. Performing some exploratory analysis on this data can help better understand the factors at play for player selcting and identify patterns that differentiate other players from creating a good team. These insights can then inform the development of a Good FIFA team in the videogame. This analysis aims to: • Familiarize the reader with the data through distribution plots and summary statistics • Identify correlations between the predictor variables • Determine which variables have the strongest relationship with overall stats • Outline relevant football players based on the available attributes.

#CHAPTER 2

#DATA DESCRIPTION

This dataset contains: Number of rows: 19238 Number of columns: 110 The columns include: • RowNumber - likely just an index • playerURL - unique ID for each player • Short name - name of customer • Long name - Complete name of the player • Playerpositions - positions that the player can be able to play as an individual • Overall - overall score of the players based upon the stats • Potential - the potential of the player • Age - age of player • value - the overall value of the player • wage - net amount that need to be given to the player in euros • height - heights of the different players • clubname - the name of the club that a player plays for • jerseynumber - this indicates the jersey number of the players • clubcountry - the details of the club that the player is belongs to which countyr • nationalteam - the national team the player plays for. This dataset contains details on 19,238 football players including information on their demographics, overall status, financial details, and a target variable indicating the overall score of the different variables included. 110 feature variables on each player could relate to selecting a team along with the binary target overall score. Performing some exploratory analysis on this data can help better understand the factors at play for selecting the team of players and identify patterns that differentiate best players for a team and the ramaining ones. These insights can then inform the development of a predictive player selecting system for the video game.

The following sections will cover the key findings, including various plots and quantitative summaries of the data.

#DISCRIPTIVE ANALYSIS

Looking at the distribution of the target variable the average overall score of the players is 65. The highest overall score is 93. Next distributions of some key numeric variables:

AGE

Player age follows a somewhat normal distribution centered around 25 years old. Younger players appear more likely to be selected in the team and have more skills.

Passing

This describes about the passing ability scores of the various players which can be able to classify the players to make a better team.

Dribbling

This helps the reader to classify players based on the dribbling capacity of the individual player.

#CHAPTER 3

Mean:

The mean, which is the average of a collection of values, is a measure of central tendency in statistics. It is computed by taking the total number of values in a dataset and dividing the result by the sum of the individual values. The one representative number that represents the centre of the data is provided by the arithmetic mean, or x. It is a basic idea in statistics that is used to explain the average or usual value within a set. In many fields, such as mathematics, economics, and data analysis, the mean is used extensively as a crucial central tendency indicator. By combining several values into a single representative measure, the mean can shed light on the general behaviour of a dataset.

Variance

A statistical metric called variance is used to quantify how widely or dispersed a group of values within a dataset are. It is a crucial metric for determining how different individual data points are from the mean. Each data point's divergence from the mean is squared in order to calculate the variance. The mean is then determined by averaging these squared deviations. (An increased variance implies a higher level of dispersion within the sample, implying that the values deviate more from the average. On the other hand, a smaller variance denotes less variability and a closer proximity of the values to the mean. In order to evaluate the consistency and dependability of data, variance is a fundamental idea in statistics and an important

Standard Deviation

A statistical measure called standard deviation is used in conjunction with variance to provide a more comprehensible depiction of the distribution or dispersion of a collection of values within a dataset. Standard deviation is only the variance squared, whereas variance indicates the average of the squared deviations from the mean. Essentially, the standard deviation gives an indication of the average separation between each data point and the mean. Greater dispersion is implied by a bigger standard deviation, whereas a smaller standard deviation indicates that the values in the dataset are closely concentrated around the mean, indicating less variability. In statistics, standard deviation is a commonly used and straightforward metric that is often used to evaluate the risk, volatility, or variability of data in domains including finance, science, and

Mode

A dataset's mode is a statistical measure that indicates the value or values that appear the most frequently. The mode draws attention to the values that occur most frequently, as opposed to the mean and median, which concentrate on central tendency. A dataset may contain one mode (unimodal), multiple modes (multimodal), or none at all if every value occurs exactly once. When defining the essential characteristics of discrete or category data, like the most prevalent category within a set, the mode is especially helpful. It can be used to find ranges or intervals with the highest frequency in continuous data. The mode is frequently employed in several disciplines, including as statistics, sociology, and

Markov's Chain

A mathematical representation of a system or process that alternates between states in accordance with specific probabilistic criteria is called a Markov chain. The Markov property, which asserts that the system's future state depends solely on its current state and not on how it got there, is the essential component of a Markov chain. Stated differently, the system just remembers its current state and not its previous ones.

A set of states and a set of transition probabilities between states create a Markov chain. The probabilities are frequently grouped in a transition matrix, where the likelihood of changing from state I to state J is represented by the entry (i, j). The concept that the system must be in one of the states following a transition is reflected in the matrix's rows, which add up to 1.

Markov chains are widely used in modelling physical processes such as particle motion, economic and biological systems, and several elements of machine learning and artificial intelligence, including reinforcement learning and natural language processing. They are very helpful for modelling random and uncertain systems because they make it possible to analyse and forecast how the system will behave in the future.

Continuos Random Variables

A basic idea in probability theory, continuous random variables are variables that can have an uncountably large number of values within a given range. X is the random variable; unlike their discrete counterparts, continuous random variables are described by probability density functions (PDFs). The chance that the variable will fall within a given range is calculated by integrating the probability distribution function (PDF) over the given interval. The PDF indicates the probability that the variable will assume a specific value within the period. Notably, there is technically no chance that a continuous random variable will take exactly one value. Temperature, weight, and other physical parameters are common instances of continuous random variables. Continuous distributions, like the normal distribution, are widely used in statistical analyses to model behaviour of the variables.

Discrete Random Variables

A key idea in probability theory are discrete random variables, which are variables that can only have discrete, distinct values with particular probabilities. Discrete random variables usually entail countable outcomes, as opposed to continuous random variables, which might assume an uncountably infinite number of values within a range. The chance of a discrete random variable taking any value is represented by the probability mass function (PMF), and the total of these probabilities over all possible values is 1. The results of a fair six-sided die or the number of heads in a sequence of coin flips are two examples of discrete random variables. A discrete random variable's cumulative distribution function (or CDF) gives the likelihood that the variable will be less than or equal to a given value.

Factor Analysis

In order to reduce the dimensionality of the data and reveal the underlying structure, factor analysis is a statistical approach used to identify latent factors that underlie a set of observed variables. This approach makes the assumption that fewer, so-called latent, unobservable factors have an impact on the measured variables. component loadings show the type and intensity of the association between each latent component and the observed variables. Eigenvalues, which show how much of the variance in the observed variables is explained by each factor, are the outcome of the analysis. It is possible to use rotation techniques to improve the results' interpretability. Factor analysis is especially useful in the social sciences and psychology, where researchers aim to find and understand the latent constructs that contributes to the observed correlations.

#CHAPTER 4

4.1 Simulation Data Analysis

```
import scipy.stats as st
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


mu, sigma, n = 0, 0.1, 5000
file_path = 'players_22.csv'

footballplayers_dataset = pd.read_csv(file_path, low_memory=False)

footballplayers_dataset =
footballplayers_dataset[footballplayers_dataset['overall'] != 0]
```

```
mean_value = np.mean(footballplayers_dataset['overall'])

print("Mean : ", mean_value)

footballplayers_dataset =
footballplayers_dataset.reset_index(drop=True)
```

```
Mean :  65.77218150631529
```

```
footballplayers_dataset
```

```
       sofifa_id                                         player_url  \
0         158023  https://sofifa.com/player/158023/lionel-messi/...
1         188545  https://sofifa.com/player/188545/robert-lewand...
2          20801  https://sofifa.com/player/20801/c-ronaldo-dos-...
3         190871  https://sofifa.com/player/190871/neymar-da-sil...
4         192985  https://sofifa.com/player/192985/kevin-de-bruy...
...          ...                                                ...
19234     261962  https://sofifa.com/player/261962/defu-song/220002
19235     262040  https://sofifa.com/player/262040/caoimhin-port...
19236     262760  https://sofifa.com/player/262760/nathan-logue/...
19237     262820  https://sofifa.com/player/262820/luke-rudden/2...
19238     264540  https://sofifa.com/player/264540/emanuel-lalch...

               short_name                            long_name  \
0                L. Messi       Lionel Andrés Messi Cuccittini
1          R. Lewandowski                   Robert Lewandowski
2       Cristiano Ronaldo  Cristiano Ronaldo dos Santos Aveiro
3               Neymar Jr      Neymar da Silva Santos Júnior
4            K. De Bruyne                      Kevin De Bruyne
...                   ...                                  ...
19234           Song Defu                                  宋德福
19235          C. Porter                      Caoimhin Porter
19236            N. Logue             Nathan Logue-Cunningham
19237           L. Rudden                         Luke Rudden
19238  E. Lalchhanchhuaha            Emanuel Lalchhanchhuaha

      player_positions  overall  potential    value_eur  wage_eur  age
...  \
0          RW, ST, CF       93         93   78000000.0  320000.0   34
...
1                  ST       92         92  119500000.0  270000.0   32
...
2              ST, LW       91         91   45000000.0  270000.0   36
...
3             LW, CAM       91         91  129000000.0  270000.0   29
...
4             CM, CAM       91         91  125500000.0  350000.0   30
...
...               ...      ...        ...          ...       ...  ...
```

```
...
19234                CDM       47          52      70000.0    1000.0   22
...
19235                 CM       47          59     110000.0     500.0   19
...
19236                 CM       47          55     100000.0     500.0   21
...
19237                 ST       47          60     110000.0     500.0   19
...
19238                CAM       47          60     110000.0     500.0   19
...

         lcb    cb   rcb    rb    gk  \
0      50+3  50+3  50+3  61+3  19+3
1      60+3  60+3  60+3  61+3  19+3
2      53+3  53+3  53+3  60+3  20+3
3      50+3  50+3  50+3  62+3  20+3
4      69+3  69+3  69+3  75+3  21+3
...     ...   ...   ...   ...   ...
19234  46+2  46+2  46+2  48+2  15+2
19235  44+2  44+2  44+2  48+2  14+2
19236  45+2  45+2  45+2  47+2  12+2
19237  26+2  26+2  26+2  32+2  15+2
19238  41+2  41+2  41+2  45+2  16+2

                                         player_face_url  \
0      https://cdn.sofifa.net/players/158/023/22_120.png
1      https://cdn.sofifa.net/players/188/545/22_120.png
2      https://cdn.sofifa.net/players/020/801/22_120.png
3      https://cdn.sofifa.net/players/190/871/22_120.png
4      https://cdn.sofifa.net/players/192/985/22_120.png
...                                                  ...
19234  https://cdn.sofifa.net/players/261/962/22_120.png
19235  https://cdn.sofifa.net/players/262/040/22_120.png
19236  https://cdn.sofifa.net/players/262/760/22_120.png
19237  https://cdn.sofifa.net/players/262/820/22_120.png
19238  https://cdn.sofifa.net/players/264/540/22_120.png

                                    club_logo_url  \
0          https://cdn.sofifa.net/teams/73/60.png
1          https://cdn.sofifa.net/teams/21/60.png
2          https://cdn.sofifa.net/teams/11/60.png
3          https://cdn.sofifa.net/teams/73/60.png
4          https://cdn.sofifa.net/teams/10/60.png
...                                           ...
19234  https://cdn.sofifa.net/teams/112541/60.png
19235     https://cdn.sofifa.net/teams/445/60.png
19236  https://cdn.sofifa.net/teams/111131/60.png
19237  https://cdn.sofifa.net/teams/111131/60.png
19238  https://cdn.sofifa.net/teams/113040/60.png
```

```
                                   club_flag_url  \
0          https://cdn.sofifa.net/flags/fr.png
1          https://cdn.sofifa.net/flags/de.png
2      https://cdn.sofifa.net/flags/gb-eng.png
3          https://cdn.sofifa.net/flags/fr.png
4      https://cdn.sofifa.net/flags/gb-eng.png
...                                        ...
19234      https://cdn.sofifa.net/flags/cn.png
19235      https://cdn.sofifa.net/flags/ie.png
19236      https://cdn.sofifa.net/flags/ie.png
19237      https://cdn.sofifa.net/flags/ie.png
19238      https://cdn.sofifa.net/flags/in.png

                                   nation_logo_url  \
0      https://cdn.sofifa.net/teams/1369/60.png
1      https://cdn.sofifa.net/teams/1353/60.png
2      https://cdn.sofifa.net/teams/1354/60.png
3                                           NaN
4      https://cdn.sofifa.net/teams/1325/60.png
...                                         ...
19234                                       NaN
19235                                       NaN
19236                                       NaN
19237                                       NaN
19238                                       NaN

                           nation_flag_url
0      https://cdn.sofifa.net/flags/ar.png
1      https://cdn.sofifa.net/flags/pl.png
2      https://cdn.sofifa.net/flags/pt.png
3      https://cdn.sofifa.net/flags/br.png
4      https://cdn.sofifa.net/flags/be.png
...                                    ...
19234  https://cdn.sofifa.net/flags/cn.png
19235  https://cdn.sofifa.net/flags/ie.png
19236  https://cdn.sofifa.net/flags/ie.png
19237  https://cdn.sofifa.net/flags/ie.png
19238  https://cdn.sofifa.net/flags/in.png

[19239 rows x 110 columns]
```

```
overall_score_data = footballplayers_dataset['overall']
```

```
overall_score_data
```

```
0        93
1        92
2        91
3        91
```

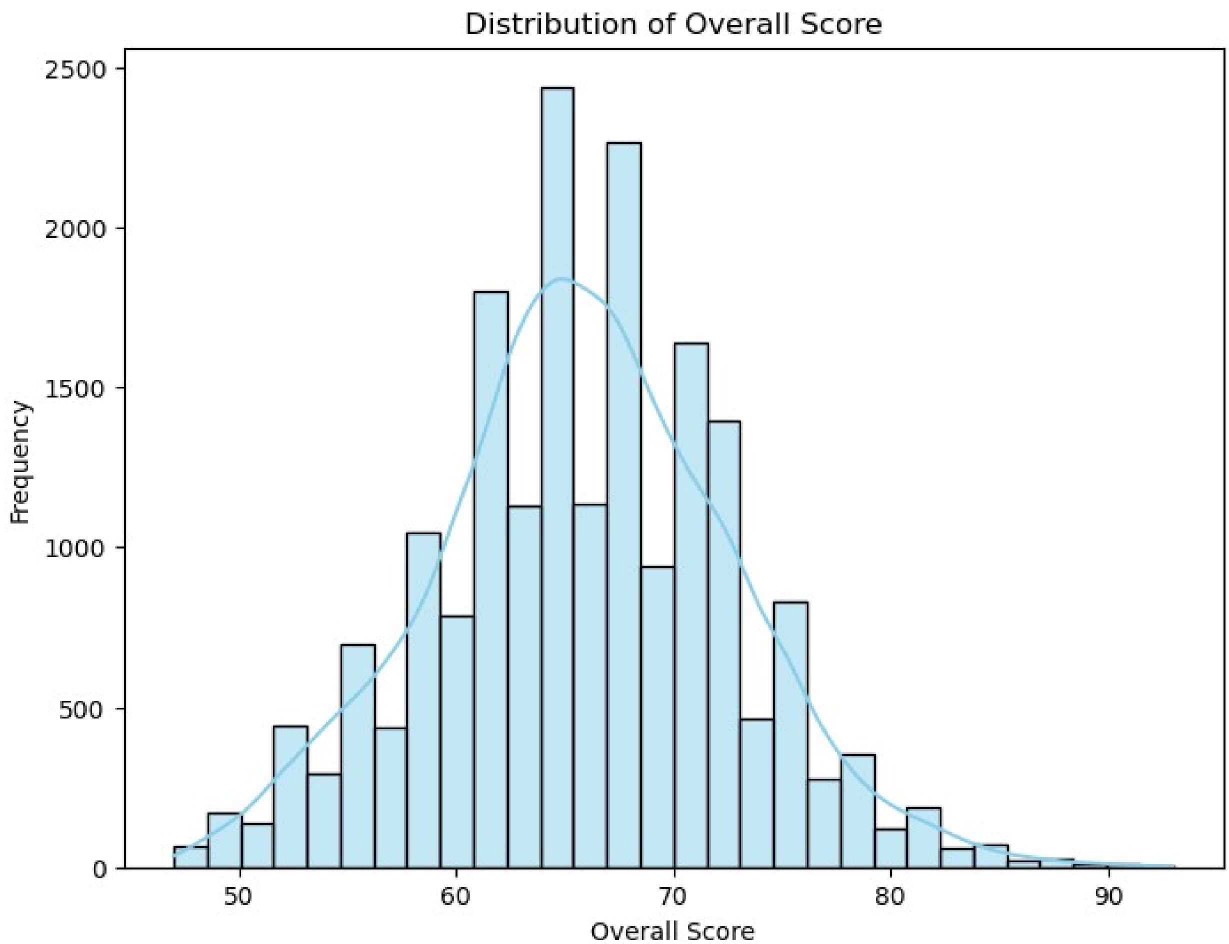
```
4        91
         ..
19234    47
19235    47
19236    47
19237    47
19238    47
Name: overall, Length: 19239, dtype: int64
```

Determination of my Dataset

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = 'players_22.csv'

footballplayers_dataset = pd.read_csv(file_path, low_memory=False)

print(footballplayers_dataset.info())
print(footballplayers_dataset.describe())

plt.figure(figsize=(8, 6))
sns.histplot(footballplayers_dataset['overall'], kde=True, bins=30,
color='skyblue')
plt.title('Distribution of Overall Score')
plt.xlabel('Overall Score')
plt.ylabel('Frequency')
plt.show()


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19239 entries, 0 to 19238
Columns: 110 entries, sofifa_id to nation_flag_url
dtypes: float64(16), int64(44), object(50)
memory usage: 16.1+ MB
None
           sofifa_id       overall     potential     value_eur
wage_eur  \
count   19239.000000  19239.000000  19239.000000  1.916500e+04
19178.000000
mean   231468.086959     65.772182     71.079370  2.850452e+06
9017.989363
std     27039.717497      6.880232      6.086213  7.613700e+06
19470.176724
min        41.000000     47.000000     49.000000  9.000000e+03
500.000000
25%    214413.500000     61.000000     67.000000  4.750000e+05
1000.000000
50%    236543.000000     66.000000     71.000000  9.750000e+05
```

```
3000.000000
75%    253532.500000     70.000000     75.000000  2.000000e+06
8000.000000
max    264640.000000     93.000000     95.000000  1.940000e+08
350000.000000

                age     height_cm     weight_kg   club_team_id
league_level  \
count  19239.000000  19239.000000  19239.000000   19178.000000
19178.000000
mean      25.210822    181.299704     74.943032   50580.498123
1.354364
std        4.748235      6.863179      7.069434   54401.868535
0.747865
min       16.000000    155.000000     49.000000       1.000000
1.000000
25%       21.000000    176.000000     70.000000     479.000000
1.000000
50%       25.000000    181.000000     75.000000    1938.000000
1.000000
75%       29.000000    186.000000     80.000000  111139.000000
1.000000
max       54.000000    206.000000    110.000000  115820.000000
5.000000

       ...  mentality_composure  defending_marking_awareness  \
count  ...         19239.000000                 19239.000000
mean   ...            57.929830                    46.601746
std    ...            12.159326                    20.200807
min    ...            12.000000                     4.000000
25%    ...            50.000000                    29.000000
50%    ...            59.000000                    52.000000
75%    ...            66.000000                    63.000000
max    ...            96.000000                    93.000000

       defending_standing_tackle  defending_sliding_tackle  \
count               19239.000000              19239.000000
mean                   48.045584                 45.906700
std                    21.232718                 20.755683
min                     5.000000                  5.000000
25%                    28.000000                 25.000000
50%                    56.000000                 53.000000
75%                    65.000000                 63.000000
max                    93.000000                 92.000000

       goalkeeping_diving  goalkeeping_handling
goalkeeping_kicking  \
count        19239.000000          19239.000000         19239.000000

mean            16.406102             16.192474            16.055356
```

```
std            17.574028              16.839528             16.564554

min             2.000000               2.000000              2.000000

25%             8.000000               8.000000              8.000000

50%            11.000000              11.000000             11.000000

75%            14.000000              14.000000             14.000000

max            91.000000              92.000000             93.000000


       goalkeeping_positioning  goalkeeping_reflexes
goalkeeping_speed
count             19239.000000          19239.000000
2132.000000
mean                 16.229274             16.491814
36.439962
std                  17.059779             17.884833
10.751563
min                   2.000000              2.000000
15.000000
25%                   8.000000              8.000000
27.000000
50%                  11.000000             11.000000
36.000000
75%                  14.000000             14.000000
45.000000
max                  92.000000             90.000000
65.000000

[8 rows x 60 columns]
```



#4.1.1 Simulating Continuous Random Variables:

```
import pandas as pd


file_path = 'players_22.csv'

footballplayers_dataset = pd.read_csv(file_path, low_memory=False)

print(footballplayers_dataset.columns)

overall_score_data = footballplayers_dataset['overall']

Index(['sofifa_id', 'player_url', 'short_name', 'long_name',
       'player_positions', 'overall', 'potential', 'value_eur',
'wage_eur',
       'age',
       ...
       'lcb', 'cb', 'rcb', 'rb', 'gk', 'player_face_url',
'club_logo_url',
```

```
       'club_flag_url', 'nation_logo_url', 'nation_flag_url'],
      dtype='object', length=110)
```

```
import numpy as np
import matplotlib.pyplot as plt

mu = np.mean(overall_score_data)
sigma = np.std(overall_score_data)

simulated_data = np.random.normal(mu, sigma, 1000)


print(simulated_data)
plt.hist(simulated_data, bins=30, density=True, alpha=0.5,
color='blue')
plt.title('Simulated Normal Distribution based on Overall Score')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

```
[55.29361405 59.05179156 70.02545978 62.19444731 73.87536567
67.76295772
 74.67217963 55.91467883 67.09256339 73.47766401 63.74562925
64.82045096
 78.41646601 67.98031297 69.19556199 76.89565872 71.31097979
67.89957706
 76.62308382 54.74527096 74.01374508 60.32686007 67.18109587
66.83904487
 63.07645937 62.37332041 60.98948895 68.11283628 65.64939064
79.80996582
 74.92191249 67.78777941 63.11635229 66.61083984 54.30095424
67.11215192
 64.2605928  52.26852616 65.13756169 69.08274799 49.85417225
52.84446395
 68.38956295 73.12795487 68.18174602 60.8541042  73.89376403
67.25319152
 73.18746462 73.35743658 61.29969212 58.50926546 76.21585659
70.58748763
 62.59623219 73.50939077 62.51948265 66.54484769 55.38196134
77.36784597
 75.22169877 62.96388375 74.90317442 71.62843763 68.56676241
76.01518779
 72.88025395 68.60998221 66.8690315  62.6269529  75.44307124
56.87089174
 80.71545032 59.58774292 74.07385091 61.31777985 76.23552183
68.97880177
 66.41719989 68.14835499 71.89601722 64.6539475  52.72004374
66.43059989
 70.94773735 68.65936607 62.2441371  70.36980599 58.22288047
68.43323327
```

```
 75.62012566 62.70272006 60.42239171 58.47275118 55.04976133
66.59702864
 57.40350905 61.98234827 67.77775351 49.9799661  66.19539641
71.86255276
 73.52339015 64.54797949 78.24006665 58.17988518 76.47867588
68.87729873
 51.06347021 70.30119663 61.59068898 68.49638155 65.10748281
65.61343404
 78.40550564 57.36506645 63.6304838  64.94688275 64.90596491
67.68202978
 69.34694768 73.63940671 52.63465408 62.85397657 71.64608985
76.21459581
 70.50456904 64.75447023 66.93759542 56.58760805 63.9601269
70.8777257
 63.56983924 79.86729007 62.1681926  58.19945475 66.6739589
67.99491729
 66.29749399 61.40778124 69.69814717 60.88008087 57.10789997
49.92506379
 60.75372394 78.70053125 72.24114512 69.3758243  67.07630421
62.33335706
 69.82519895 64.30755139 76.60252633 72.65833518 58.39085429
60.71638644
 70.35165367 56.56911852 70.92183787 61.59364945 76.20356432
54.01936974
 68.2884446  64.01838336 70.31542194 74.70341484 75.69441434
68.84538089
 71.20616473 66.48230868 76.12811008 62.48373991 60.42276623
65.49559426
 63.69772885 84.3462451  60.57068816 62.75155855 77.35958408
74.88846068
 61.37892308 59.65012367 57.27444495 73.48330207 73.54956066
70.36760445
 62.8082104  76.42375622 58.72271305 62.46507021 58.31562593
60.8146715
 74.3943214  66.43646103 60.63688635 68.08714604 69.89893683
69.89594501
 63.45420461 66.32484666 73.12255372 62.47297312 74.56421069
70.03622476
 63.87691515 74.39930264 78.28396284 85.96567656 63.53245276
64.46272269
 59.30439096 71.53027731 71.92208675 61.20398671 69.44899582
62.12729896
 76.29236948 64.86263803 58.39645983 68.4173881  61.07509306
56.41737567
 60.61755438 70.55723939 74.80486769 74.46890523 61.10646113
71.95262307
 57.37674775 82.21611466 67.98939823 73.55447015 53.71202951
71.92796087
 72.32448709 77.22232881 73.63315675 64.39013386 83.00474641
```

```
55.05982155
 66.61886892 69.98046555 63.42584666 62.98627781 79.74323572
60.12866449
 58.86796896 71.25816179 72.32391183 65.25030532 56.17154214
73.73916716
 58.39280435 69.11295543 64.6094699  59.49392267 66.88721202
72.07191446
 67.2050355  76.99765195 73.79735551 57.27902211 74.15502595
68.9645597
 80.06834398 79.37280375 56.70076644 52.83621113 53.34971353
66.94774683
 56.12107227 73.10364073 72.866633   60.74939305 78.5087416
46.50091881
 62.00140573 62.6625372  62.56636072 57.69601216 72.44570584
64.67866703
 69.18625133 60.62452063 75.41042413 74.06380087 77.3309138
64.11470033
 72.3985807  60.98860266 67.38749494 65.92126695 59.88516818
62.14534682
 55.98198447 61.86543362 74.40988368 68.92493975 71.29837358
64.12433723
 74.63040391 61.14574738 53.75677808 65.33789751 62.78107363
65.86413249
 63.89223807 60.60727704 61.80947605 64.93600418 72.82722127
54.55396213
 60.68117451 59.50613378 67.05118384 69.71721944 69.98160834
71.2641697
 72.04916659 79.67881144 62.48058117 76.6018364  64.95655142
80.50765636
 56.44203312 53.82230369 72.81880395 76.12858963 65.17344214
50.35952097
 66.98466901 63.97791309 64.5188962  63.63288316 59.20239736
67.23845834
 42.84112967 52.69293214 61.77353352 57.94658429 62.49036481
73.94730469
 53.49435056 53.60156875 64.88156209 66.12158351 74.06461513
72.59363569
 75.03061855 68.24531558 61.58571608 70.05774145 64.1299129
62.01597307
 74.18165407 68.45920425 64.07647301 69.21936116 61.41046529
59.49286481
 63.39839556 71.88793963 57.52733189 73.38201215 72.92884388
71.12169775
 76.86351649 57.63655386 66.29482764 70.95412303 73.62549552
72.51769986
 66.97645569 53.21159491 59.85743289 69.56305829 78.37215854
60.20386977
 66.09642601 63.24834596 61.45420683 66.6968872  62.96797073
71.44140679
```

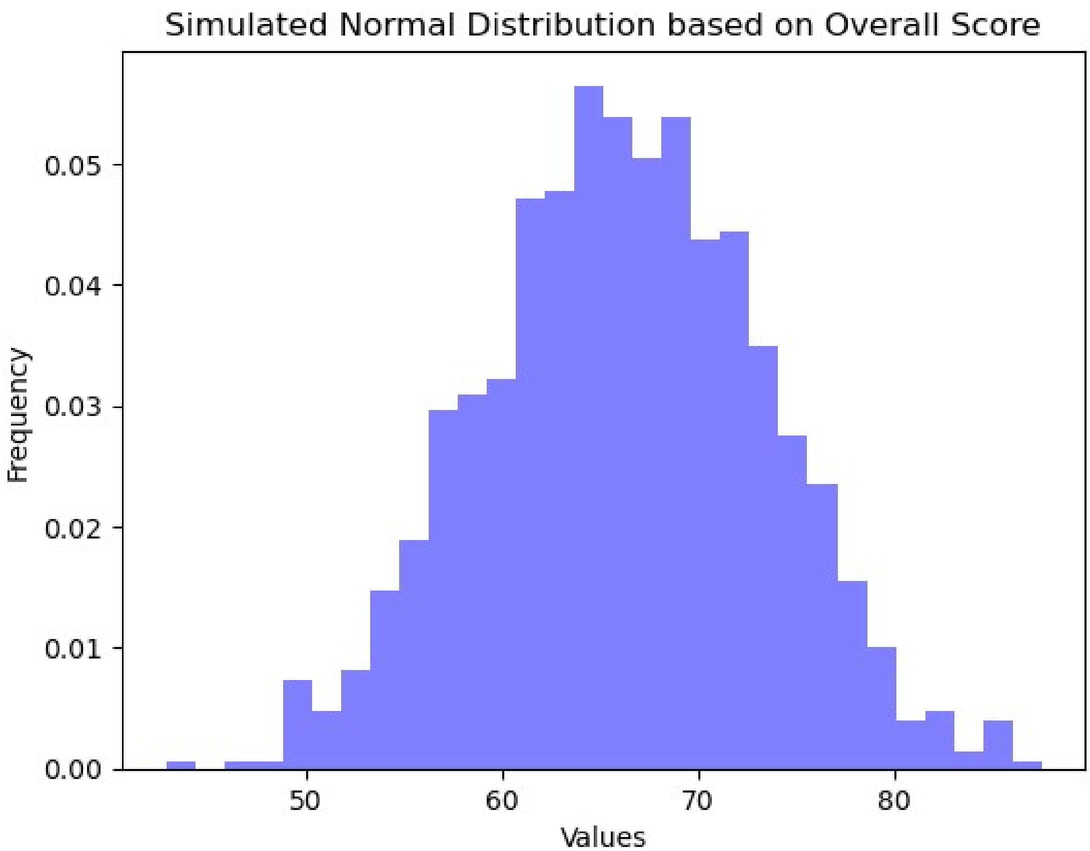
```
 53.50650668 68.32136747 60.44793831 67.65148624 63.09944817
56.72359623
 63.52222878 71.56924728 66.10032313 72.17250877 62.22912316
60.71105108
 62.00370821 69.74738932 60.64133508 66.31262092 71.15575186
55.57687601
 55.97051929 58.58452018 74.56449014 79.6030236  52.30977213
52.07870521
 62.62942546 61.8519704  62.04405942 69.72832512 68.55455511
70.92608725
 72.94213107 54.21165813 59.0898647  57.92551091 55.41374612
70.27891637
 76.55604108 62.31898373 78.85299984 71.08107901 61.07130826
55.69426772
 69.60302378 57.42041898 65.97536126 64.54019031 63.52464938
67.65135244
 61.57625412 55.70442301 66.8073607  65.08164775 77.91562223
63.40944273
 72.04014033 66.46137111 68.07352187 70.31028825 57.45611174
67.25700885
 66.45128709 68.29520598 56.34762942 69.06678516 73.809989
68.58922125
 50.07763958 69.31933008 60.90957168 73.43138734 71.30654347
60.2823107
 69.07033309 61.57808925 70.14456581 68.47440605 65.19034678
85.0545933
 78.20266448 62.04390735 66.04997626 58.00378714 66.76736235
75.22641862
 72.06772546 79.92623519 60.31803997 66.37239198 77.47188509
70.31532554
 58.18121447 61.87745321 67.94585134 69.7102594  60.93044091
63.18080467
 61.77500197 70.87979033 68.58165482 71.05780348 69.01799879
73.0662902
 63.52458864 75.37854343 55.08333284 66.05221154 63.50348012
70.68419277
 66.03119494 67.8083197  55.3299859  55.98103005 63.8741361
64.25569613
 66.6098256  60.42207086 65.03692109 55.34547498 69.92154798
64.60332993
 71.44571771 57.79716044 72.42752942 69.24691722 72.05413312
61.3090599
 62.3750391  59.13034143 60.07570783 63.83185736 76.08324064
57.6061629
 52.94852269 71.18318618 71.23670086 64.29031224 71.59022105
74.26455346
 70.58645139 69.40514243 68.54387247 65.24163301 58.43308109
67.01220608
 67.52504804 69.65952056 73.62563437 64.18725106 76.14457819
```

```
65.16931602
 71.57118206 67.66708871 59.05334235 81.70833926 66.07478768
59.02419655
 65.52552821 69.32416357 80.45907222 69.73618817 63.74784629
62.31018574
 67.69603917 57.57612664 69.86772241 54.06454063 61.53702075
56.80129276
 55.84198616 55.7995944  71.85914203 67.58000703 52.438765
65.52224724
 55.93448465 64.07666164 64.62985888 71.64588773 63.03183954
70.33600485
 72.08677894 77.16680258 67.16303109 83.89897955 64.38198878
69.34029894
 67.89103719 61.92212896 72.61974631 79.3721507  64.38904959
66.44691244
 66.74507902 66.97565803 65.78091244 68.61611082 67.82323512
59.31076979
 74.56387344 60.45647311 57.25374099 61.5608025  64.21993922
76.5911427
 67.00413848 75.94287294 69.9754777  71.91539535 81.583416
69.17771564
 64.49023642 63.69353279 65.81944492 74.50090425 73.46777877
58.60094565
 69.59922598 64.43101854 64.75368928 67.68057447 65.89564145
71.16554383
 72.46738775 73.4839324  64.61144    70.65804164 75.12308001
66.29360343
 57.36062462 69.75612792 54.68341681 85.84459361 66.59950653
65.53044877
 63.87843317 54.43415806 73.00523166 75.13727599 65.30913577
69.81439242
 67.51614135 62.08806701 80.58584217 68.74371581 62.82655536
75.16675556
 54.62677938 82.50885778 73.73567878 67.01423464 77.51917032
64.49281466
 66.72679273 58.84810314 62.92071245 68.55624084 72.90410304
59.24720942
 63.8379979  65.84991311 61.34041999 64.99129536 58.65778377
69.37235654
 59.75311987 71.54930829 58.70831666 66.02017772 74.31444313
71.61396088
 61.85015521 53.54970096 64.6849857  58.55370799 63.73149352
65.96454283
 55.44035548 57.11492661 78.34157274 70.38861262 68.22475585
59.97823148
 57.55171797 67.33671556 74.87106239 71.66213722 70.01234196
61.11990398
 63.43651359 61.77358363 59.47060504 64.77005338 68.95204966
68.77428865
```

```
 59.60743068 74.2132437  63.81356658 62.72504559 66.63663608
74.9848026
 70.28246649 68.97611959 57.81415322 59.17461119 66.25580443
71.22861553
 65.08203092 60.93695555 71.72709372 75.91959365 68.52746541
69.28091128
 65.28125919 64.73493499 49.97413999 63.68757049 66.79427629
66.24200168
 61.38516804 56.13085873 68.76388062 69.71918878 78.57434742
57.13026346
 67.1530556  64.94967165 56.7212374  59.04931689 67.27283265
62.29492652
 70.22191397 68.26448063 61.10640998 63.17161073 64.78780399
74.6181369
 78.2292594  63.0396471  70.67004092 70.30332818 66.57061819
64.50247828
 65.34816996 69.35742583 75.18137662 59.16071015 70.10016341
68.11115041
 63.6911081  71.90166337 62.66858166 63.65826448 48.2322356
68.13817268
 54.4962913  74.49208213 68.25807396 63.60430686 71.44667512
62.39090309
 57.29607692 55.55439938 70.46813082 70.28690111 49.35406559
66.83477564
 71.81875247 71.98342164 79.75206275 68.19066404 75.43248591
57.93862745
 66.10956349 63.97597999 57.0700551  62.71810683 57.28531257
77.71987014
 65.71185245 67.15087961 71.3826598  60.9178492  55.71339955
71.44390705
 73.51536054 55.93239283 67.33898344 59.39334501 51.3136774
61.70403241
 72.29738025 69.07671912 64.81381348 56.171085   66.05428214
66.42535568
 77.23791236 58.09420665 69.94118059 59.61991306 67.93546269
57.55423226
 58.6188427  61.86969081 76.10782539 66.60741401 57.61473072
72.27078411
 62.84280871 64.47526914 75.2119103  61.64001577 64.7031374
50.05196316
 62.62547687 74.57858999 66.27090409 61.85998384 79.18716251
65.84411831
 65.65840254 59.00040678 62.68117638 87.48253125 66.6236325
65.07108263
 54.29765504 57.02220154 84.82667007 67.01093909 72.92215902
69.28661346
 63.94305448 64.57005031 62.45831423 68.70136794 57.08520105
72.17258005
 67.29417904 64.04767904 70.62348416 63.18479727 62.97887903
```

```
69.17167407
 76.582888   53.13639407 51.02666532 73.46059698 56.88919476
62.36771185
 63.64965524 59.04185898 50.12446    81.79328628 60.02048441
58.95220785
 73.50379377 73.48194136 73.00766387 72.0892885  57.03911626
59.4383266
 72.72895171 60.46477875 67.78516086 64.52502407 77.94871522
61.32607857
 65.5328954  56.04480337 58.05868503 76.17121184 59.72080965
68.57055941
 56.33291214 70.36580237 72.22583093 65.22411639 60.0396044
60.92797452
 59.31522378 73.99440273 69.28789435 63.79335853 70.11562659
84.75440109
 63.08889817 69.40687321 51.11695445 68.06620361 64.35248804
63.95393509
 61.50777024 72.86391095 67.14669654 60.98873187 62.60283836
60.55737256
 78.3880059  56.87252039 80.97137289 68.82185962 54.91220437
69.31004477
 71.4872552  51.35386174 73.39354558 67.11124567 75.80190535
66.4294853
 58.68536572 51.49068428 71.66408297 69.67492112 60.69846847
64.92887983
 58.11815512 66.89656565 67.3525816  67.91972018 57.55952494
57.52736197
 57.9801254  50.11932369 73.99828981 63.90920469 58.39753437
59.9793475
 65.6965434  67.33658521 71.05907902 63.32254205 61.70572337
50.0171761
 58.5000181  77.29298942 76.4636501  63.28552972 74.81257527
56.23865669
 66.64691539 74.44444197 69.58288889 79.52513716 75.76332698
69.42814217
 57.97748183 68.98661759 66.96603267 76.99763227 65.97529746
54.58094659
 68.42301803 60.43090498 68.50656364 65.51319719 71.77460207
67.00688503
 76.79486384 60.91919846 71.79317304 63.81532908 57.38639982
61.74792265
 79.12859062 64.09273127 65.43123677 76.6476816  62.14902017
59.99600701
 68.75319392 72.10747404 67.29680213 61.70947046 65.03088905
66.5354218
 70.2431282  61.33674401 65.82972705 65.17398743 49.90533124
68.39260438
 65.73209276 67.86333739 63.36642134 59.24257254 75.68391866
68.38025587
```

```
 59.23501962 70.57547912 53.63095813 57.61697826 70.80186671
74.80100896
 63.56818994 66.09217637 68.10313134 66.2908016  65.86100005
65.78251895
 53.34331697 77.48310465 71.07948392 68.37281082 82.233119
69.97148605
 64.13635933 68.32304816 85.68535724 60.07653292 71.24815483
77.03600869
 63.8740144  68.87602047 75.39163731 60.50203332]
```



Statistical Analysis

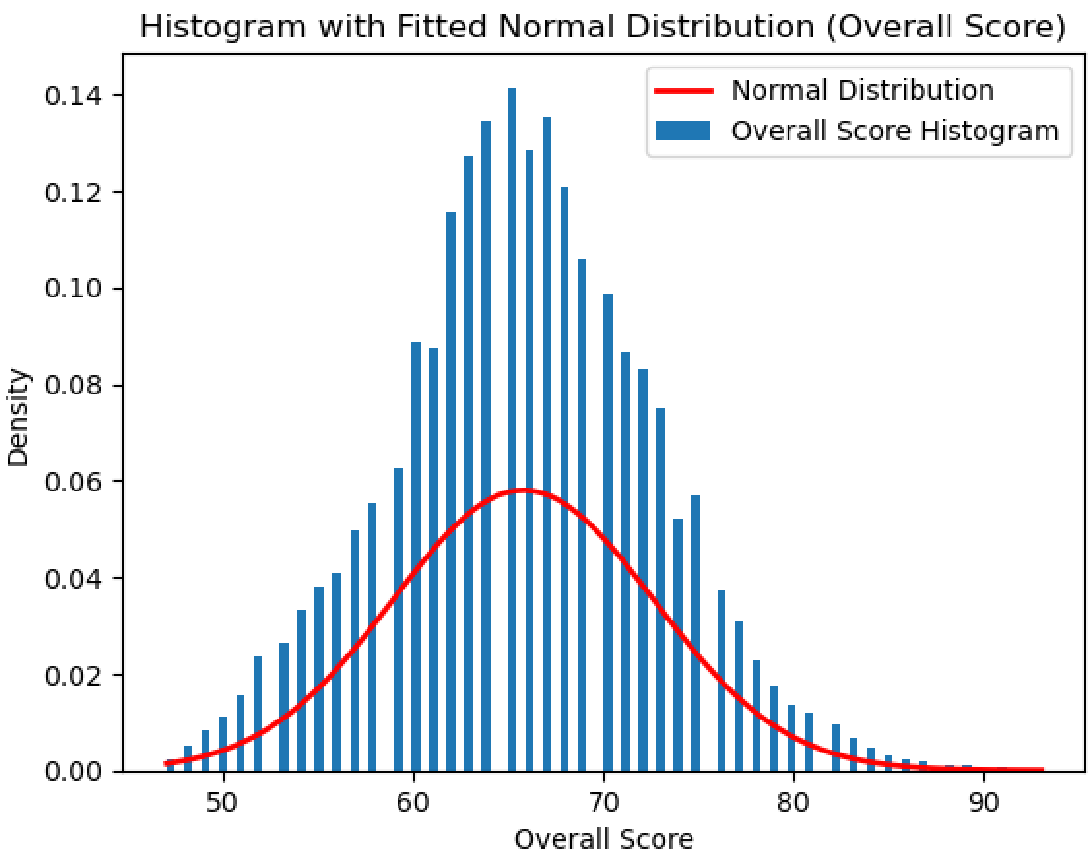
```
import scipy.stats as st
mean = np.mean(overall_score_data)
median = np.median(overall_score_data)
std_deviation = np.std(overall_score_data)
variance = np.var(overall_score_data)
quantiles = np.percentile(overall_score_data, [25, 50, 75])
kurt = st.kurtosis(overall_score_data, axis=0, bias=True)

print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Standard Deviation: {std_deviation}")
```

```python
print(f"Variance: {variance}")
print(f"Quantiles (25th, 50th, 75th percentiles): {quantiles}")
print(f"Kurtosis: {kurt}")
```

```
Mean: 65.77218150631529
Median: 66.0
Standard Deviation: 6.880052695049176
Variance: 47.33512508665343
Quantiles (25th, 50th, 75th percentiles): [61. 66. 70.]
Kurtosis: 0.09066867096156184
```

Visualization

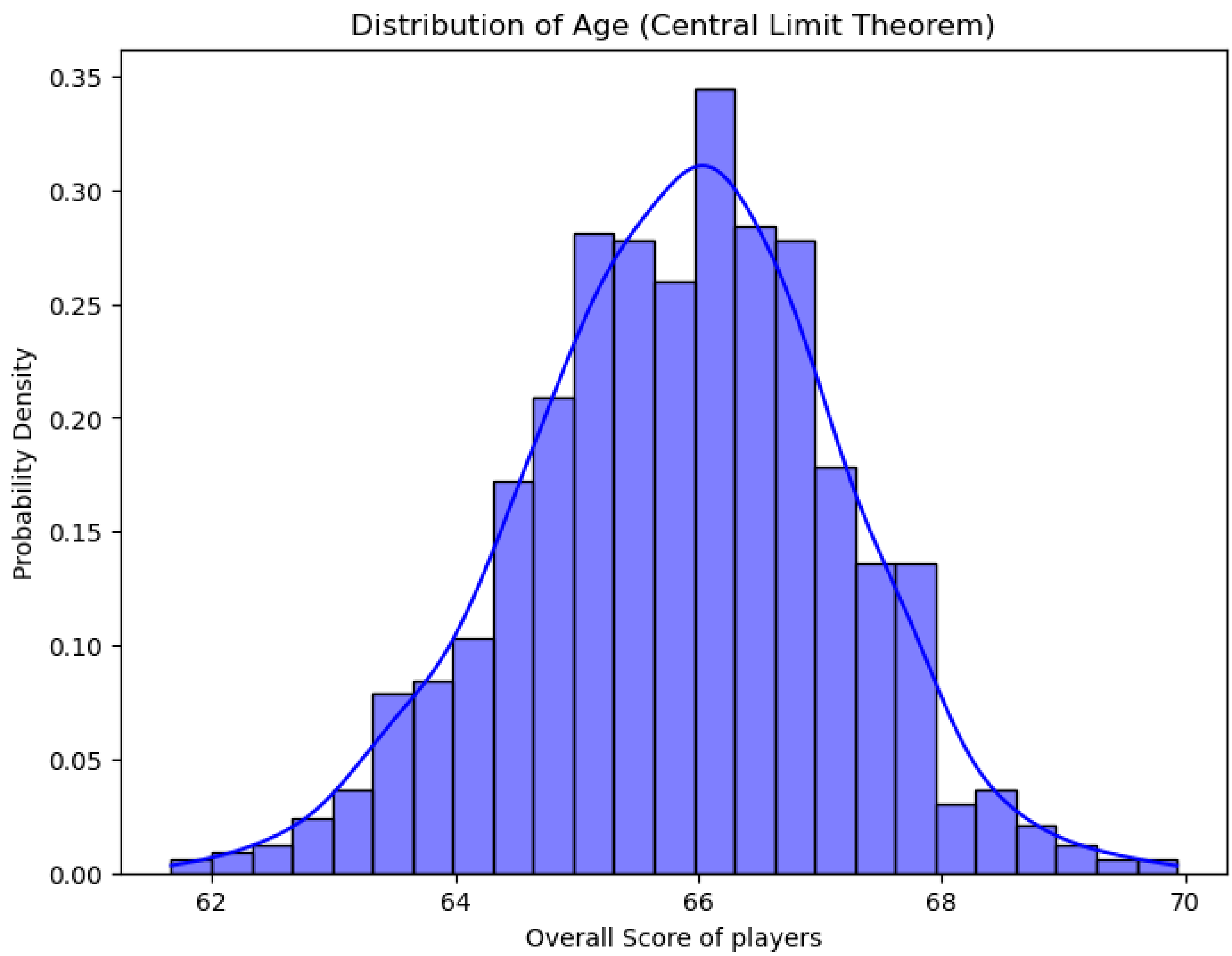
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

overall_score_data = footballplayers_dataset['overall']

mu = np.mean(overall_score_data)
sigma = np.std(overall_score_data)

count, bins, ignored = plt.hist(overall_score_data, 100, density=True)

plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) * np.exp(-(bins - mu)**2
/ (2 * sigma**2)),
         linewidth=2, color='red')

plt.title('Histogram with Fitted Normal Distribution (Overall Score)')
plt.xlabel('Overall Score')
plt.ylabel('Density')
plt.legend(['Normal Distribution', 'Overall Score Histogram'])

plt.show()
```



Central Limit Theorem Verification

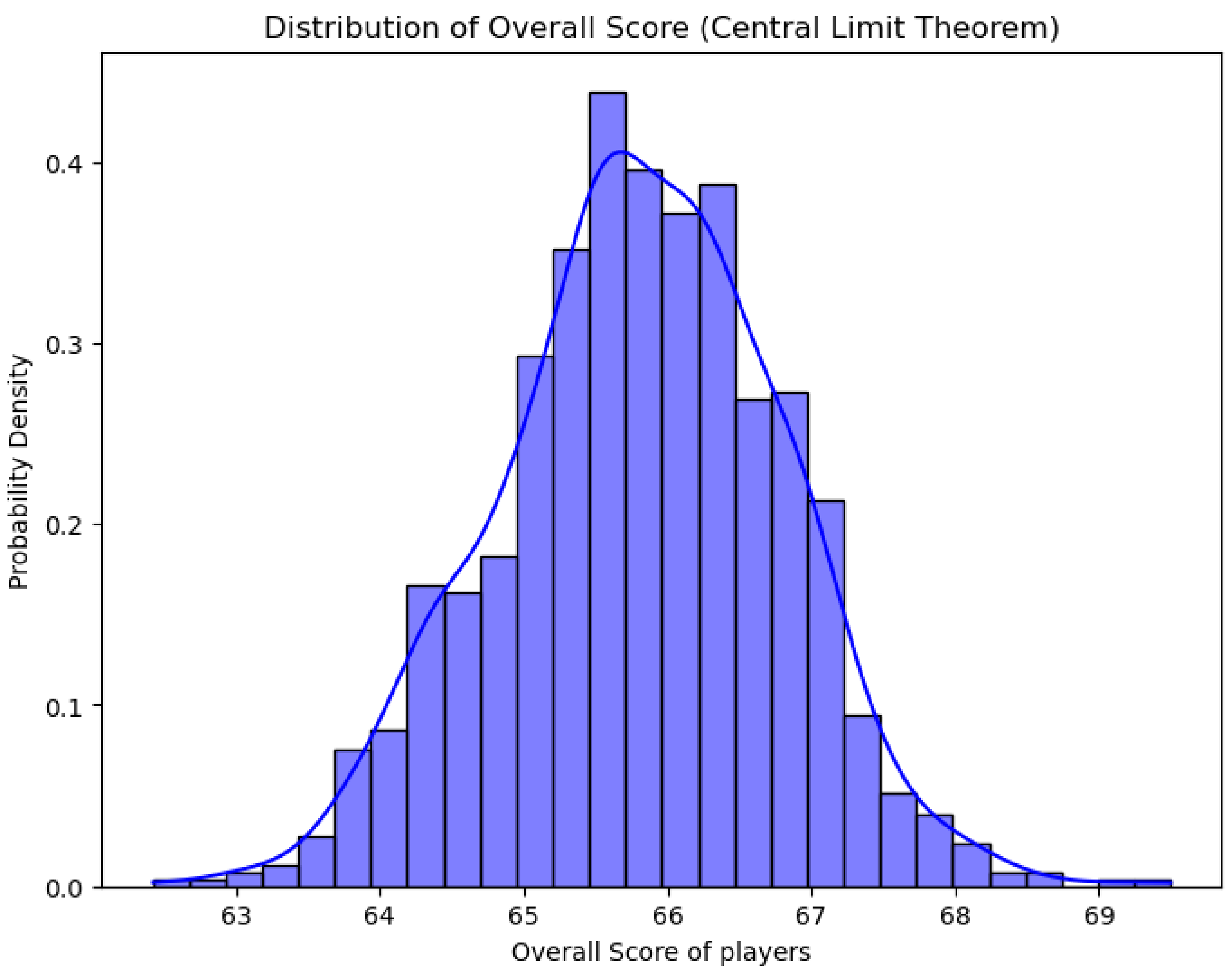
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns



overall_score_data = footballplayers_dataset['overall']


def calculate_sample_means(sample_size, no_of_sample_means):
    mean_list = []
    for i in range(no_of_sample_means):
        sample = np.random.choice(overall_score_data, sample_size)
        sample_mean = np.mean(sample)
        mean_list.append(sample_mean)
    return mean_list

sample_means = calculate_sample_means(sample_size=30,
no_of_sample_means=1000)
```

```
plt.figure(figsize=(8, 6))
sns.histplot(sample_means, color='blue', kde=True, stat="density")
plt.xlabel('Overall Score of players')
plt.ylabel('Probability Density')
plt.title('Distribution of Age (Central Limit Theorem)')
plt.show()
```




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


overall_score_data = footballplayers_dataset['overall']

def calculate_sample_means(sample_size, no_of_sample_means):
    mean_list = []
    for i in range(no_of_sample_means):
        sample = np.random.choice(overall_score_data, sample_size)
        sample_mean = np.mean(sample)
```

```
        mean_list.append(sample_mean)
    return mean_list

sample_means = calculate_sample_means(sample_size=50,
no_of_sample_means=1000)

plt.figure(figsize=(8, 6))
sns.histplot(sample_means, color='blue', kde=True, stat="density")
plt.xlabel('Overall Score of players')
plt.ylabel('Probability Density')
plt.title('Distribution of Overall Score (Central Limit Theorem)')
plt.show()
```



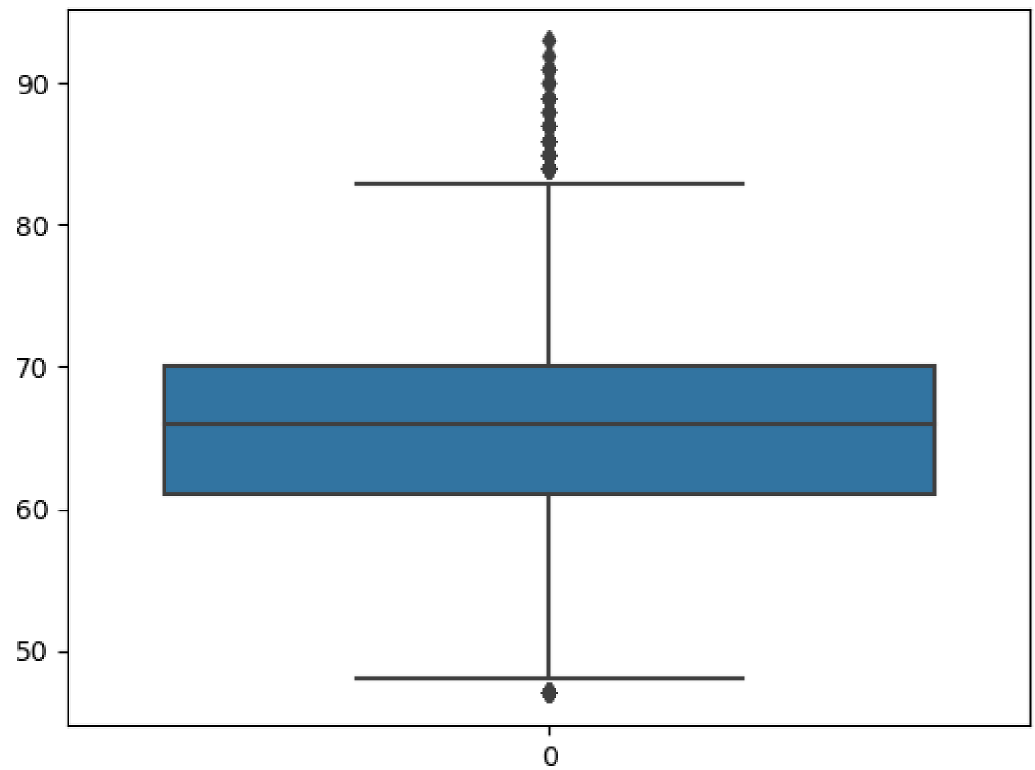
Outlier Detection

```
import pandas as pd
import numpy as np
import seaborn as sns

e_path = 'players_22.csv'
```

```
footballplayers_dataset = pd.read_csv(file_path)

column_of_interest = 'overall'
data_column = footballplayers_dataset[column_of_interest]

sns.boxplot(data_column)

q1 = np.quantile(data_column, 0.25)
q3 = np.quantile(data_column, 0.75)
IQR = q3 - q1
outliers_iqr = data_column[((data_column < (q1 - 1.5 * IQR)) |
(data_column > (q3 + 1.5 * IQR)))]

mean = np.mean(data_column)
std_dev = np.std(data_column)
z_score = (data_column - mean) / std_dev
outliers_z_score = data_column[(z_score > 2.73) | (z_score < -2.73)]

print("Outliers using IQR method:")
print(outliers_iqr)

print("\nOutliers using z-scores method:")
print(outliers_z_score)
```

```
Outliers using IQR method:
0        93
1        92
2        91
3        91
4        91
         ..
19234    47
19235    47
19236    47
19237    47
19238    47
Name: overall, Length: 159, dtype: int64

Outliers using z-scores method:
0     93
1     92
2     91
3     91
4     91
      ..
92    85
93    85
94    85
95    85
```

```
96    85
Name: overall, Length: 97, dtype: int64

/var/folders/5y/dp3cyz5s2vsbqcjxcnzd62900000gn/T/
ipykernel_20279/1502119058.py:6: DtypeWarning: Columns (25,108) have
mixed types. Specify dtype option on import or set low_memory=False.
  footballplayers_dataset = pd.read_csv(file_path)
```



Probability Calculations

```python
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats as st

file_path = 'players_22.csv'
footballplayers_dataset = pd.read_csv(file_path, low_memory=False)

column_of_interest = 'overall'
data_column = footballplayers_dataset[column_of_interest]

mu = np.mean(data_column)
sigma = np.std(data_column)

lower_bound = -0.2
```

```
upper_bound = 0.1
probability = st.norm.cdf((upper_bound - mu) / sigma) -
st.norm.cdf((lower_bound - mu) / sigma)

print(f"The probability of -0.2 < X < 0.1 is: {probability:.4f}")
print(mu)
print(sigma)
```

```
The probability of -0.2 < X < 0.1 is: 0.0000
65.77218150631529
6.880052695049176
```

4.1.2 Simulating from Discrete Distributions:

```
import pandas as pd
import scipy.stats as st

file_path = 'players_22.csv'
footballplayers_dataset = pd.read_csv(file_path,  low_memory=False)

column_of_interest = 'overall'
data_column = footballplayers_dataset[column_of_interest]

mu = data_column.mean()

data_discrete = st.poisson.rvs(mu, size=1000)

print(data_discrete)
```

```
[59 68 64 62 70 53 61 61 74 65 70 65 56 61 70 76 67 74 75 61 64 70 86
56
 74 76 67 51 71 64 61 68 75 59 66 72 74 60 59 68 73 78 67 62 66 64 50
62
 57 59 72 81 70 59 75 75 57 69 63 60 63 62 64 61 71 69 57 70 54 63 60
59
 55 72 59 64 64 77 61 53 69 82 66 56 53 63 54 65 62 55 58 57 75 89 67
66
 74 66 72 80 58 53 76 69 57 77 70 63 63 74 60 67 60 76 63 63 70 68 66
76
 69 71 58 61 63 66 65 62 81 65 74 66 63 60 52 54 81 64 72 68 70 53 69
61
 60 61 64 68 77 51 78 56 53 73 58 71 65 66 57 62 75 55 50 71 61 64 74
61
 64 72 69 77 53 54 79 70 56 70 63 68 63 80 67 71 64 87 58 69 50 83 66
68
 65 64 54 65 78 59 77 62 74 64 69 75 78 79 64 78 59 69 62 73 61 68 65
57
 78 67 60 64 63 70 75 69 64 64 74 64 88 68 61 81 65 54 89 83 67 57 57
76
```

```
 63 51 72 71 68 70 61 62 61 58 64 64 57 60 67 39 59 66 67 74 52 63 64
62
 69 60 68 66 77 56 63 62 63 64 71 66 58 50 79 73 85 64 66 67 65 73 64
63
 65 79 69 71 73 69 68 59 69 81 70 66 64 55 61 67 59 72 68 62 63 63 62
80
 78 62 63 66 64 57 68 70 70 61 68 66 73 78 71 55 75 72 51 63 69 62 70
62
 77 71 75 69 73 64 70 70 59 69 80 69 57 57 73 66 75 74 60 63 65 66 78
76
 67 69 64 58 55 76 65 67 66 58 70 68 69 64 66 71 72 70 60 75 66 82 63
58
 57 74 74 54 71 54 55 71 64 72 74 68 58 64 68 64 70 47 55 64 72 64 61
69
 72 67 71 69 66 76 66 60 62 70 63 70 73 58 58 54 56 68 52 81 70 85 54
73
 57 70 77 71 56 73 68 78 73 67 74 74 59 68 73 78 65 81 63 42 83 53 66
68
 56 53 49 66 74 61 62 73 64 81 67 66 68 61 69 78 82 81 63 44 66 67 63
80
 81 67 66 68 72 75 68 74 53 74 57 76 64 81 67 61 67 66 62 59 70 72 65
60
 68 59 66 50 76 67 59 55 53 51 69 83 69 64 49 70 61 69 50 68 66 65 69
59
 80 78 57 64 62 64 58 66 64 79 65 58 66 60 53 61 70 67 60 60 60 72 70
65
 56 63 73 61 68 65 71 58 72 74 72 72 65 70 64 66 63 81 55 56 73 72 80
69
 80 66 62 54 65 76 62 58 67 60 57 66 63 73 72 70 62 63 49 68 67 61 51
59
 79 83 62 80 67 63 61 68 53 63 80 69 53 73 63 63 68 66 59 69 66 67 78
62
 62 67 59 70 68 49 67 59 63 61 80 61 51 70 64 61 74 73 76 62 71 69 67
41
 54 68 74 61 66 57 77 61 67 68 84 66 54 65 65 61 50 61 54 70 70 67 52
73
 62 55 76 71 71 65 54 66 82 58 60 70 66 70 61 52 62 65 70 77 63 67 62
59
 83 77 63 75 61 63 53 57 76 63 61 57 73 59 59 74 64 61 80 73 56 58 60
66
 57 63 56 70 56 63 76 60 70 83 80 63 62 87 78 60 58 71 58 71 68 59 71
53
 66 78 70 61 75 49 60 74 63 80 67 57 63 70 55 65 62 65 77 55 56 62 59
67
 63 73 60 71 64 67 65 65 55 70 72 77 61 59 71 70 66 51 64 51 61 67 57
70
 59 66 65 71 69 69 66 65 74 55 71 72 73 50 54 53 42 74 60 67 57 58 72
69
 66 59 56 82 69 53 67 52 64 59 77 60 65 59 75 67 64 62 73 77 62 75 76
```

```
66
 66 65 63 79 77 72 59 71 64 77 74 79 59 64 80 61 59 73 76 65 73 54 71
66
 58 69 64 61 49 65 72 61 65 69 67 61 61 74 63 72 54 70 68 53 66 74 61
72
 61 74 61 52 66 74 52 61 60 58 58 61 64 42 72 61 62 58 68 57 71 69 62
68
 75 64 72 59 72 75 68 59 62 70 69 61 68 79 64 63 72 64 72 43 75 61 68
68
 71 60 77 61 68 82 77 78 77 68 65 72 54 67 68 50 68 75 78 59 66 58 79
74
 64 62 55 70 66 80 82 66 73 59 65 74 71 59 58 76 62 74 82 70 59 71 60
59
 78 62 64 64 73 66 63 85 66 58 67 76 60 61 55 78]
```

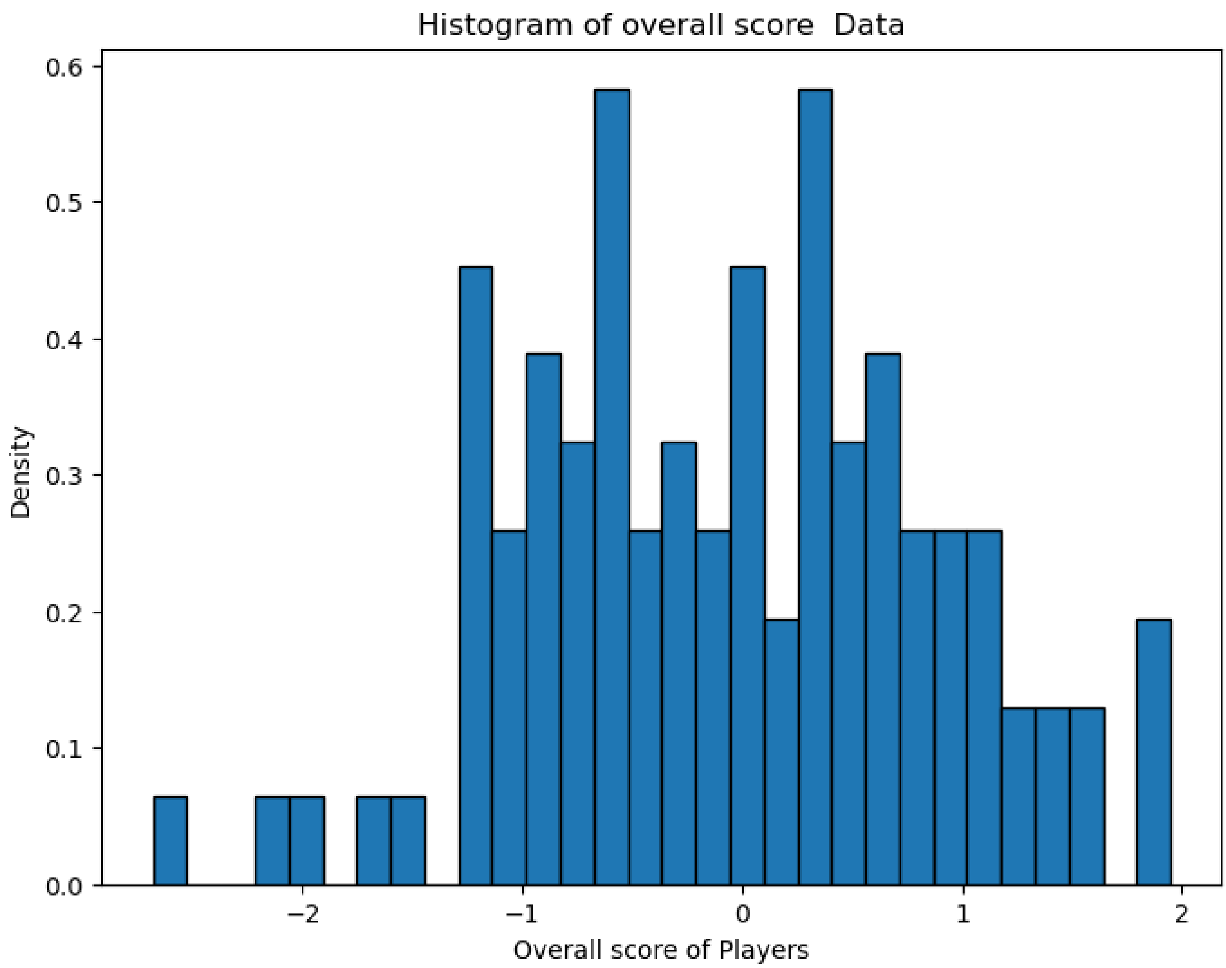
Statistical Analysis:

```
import numpy as np
import seaborn as sns
import scipy.stats as st

data_column = np.random.randn(100)

mean = np.mean(data_column)
standard_deviation = np.std(data_column)
variance = np.var(data_column)
quantile = np.quantile(data_column, [0.25, 0.5, 0.75])

mode_result = st.mode(data_column)
mode = mode_result.mode[0]

skewness = st.skew(data_column)
kurtosis = st.kurtosis(data_column)

print("Mean:", mean)
print("Standard Deviation:", standard_deviation)
print("Variance:", variance)
print("First Quantile:", quantile[0])
print("Median (Second Quantile):", quantile[1])
print("Third Quantile:", quantile[2])
print("Mode:", mode)
print("Skewness:", skewness)
print("Kurtosis:", kurtosis)

Mean: -0.05696541719871014
Standard Deviation: 0.9290249996835663
Variance: 0.8630874500370505
First Quantile: -0.7271830721155583
Median (Second Quantile): -0.03113676017848229
Third Quantile: 0.5835395117876068
```

```
Mode: -2.6811691124076287
Skewness: -0.08004358007075359
Kurtosis: -0.2402268748693781

/var/folders/5y/dp3cyz5s2vsbqcjxcnzd62900000gn/T/
ipykernel_20279/950199653.py:12: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this
behavior will change: the default value of `keepdims` will become
False, the `axis` over which the statistic is taken will be
eliminated, and the value None will no longer be accepted. Set
`keepdims` to True or False to avoid this warning.
  mode_result = st.mode(data_column)
```

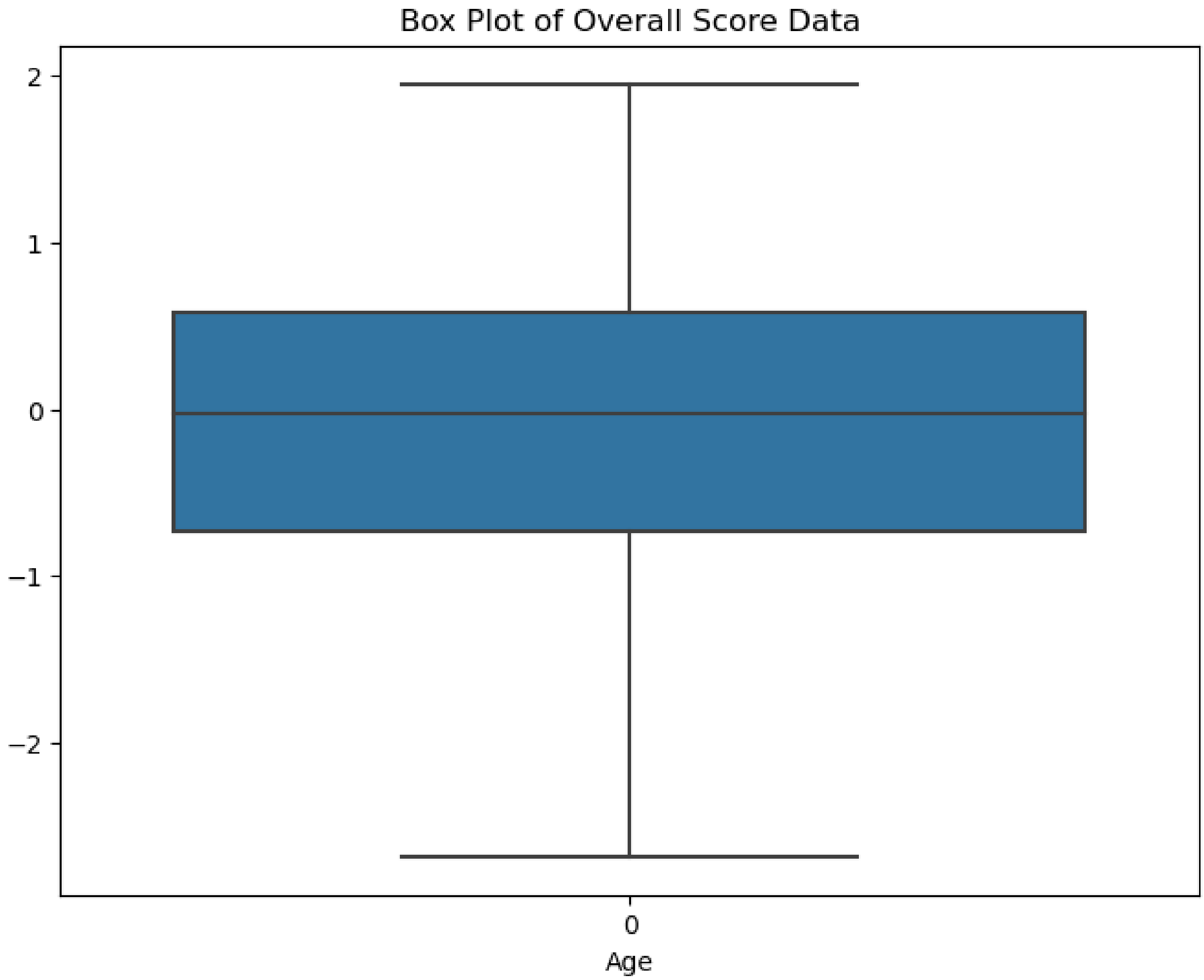
Visualization:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.hist(data_column, bins=30, density=True, edgecolor='black')
plt.title('Histogram of overall score  Data')
plt.xlabel('Overall score of Players')
plt.ylabel('Density')
plt.show()
```



Box PLot of the Data

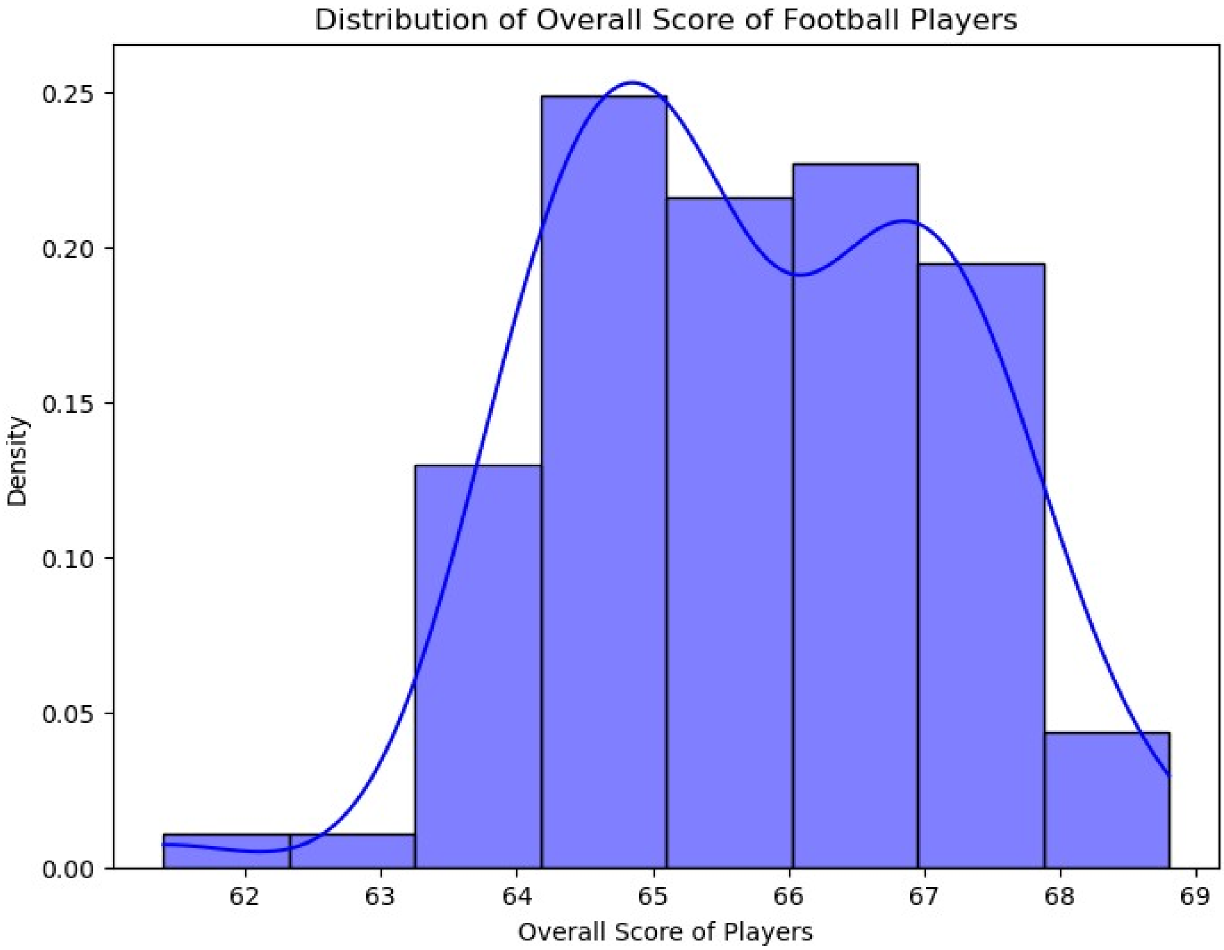
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
sns.boxplot(data_column)
plt.title('Box Plot of Overall Score Data')
plt.xlabel('Age')
plt.show()
```



Central Limit Verfication

```
import random
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

data_column = footballplayers_dataset['overall']
mean_list = []
def calc_sample_mean(sample_size, no_of_sample_means):
  for i in range(no_of_sample_means):
    sample = random.sample(list(data_column), sample_size)
    sample_mean = np.mean(sample)
    mean_list.append(sample_mean)
  return mean_list
mean_samples = calc_sample_mean(sample_size=20,
no_of_sample_means=100)

# Visualize sample means distribution
```

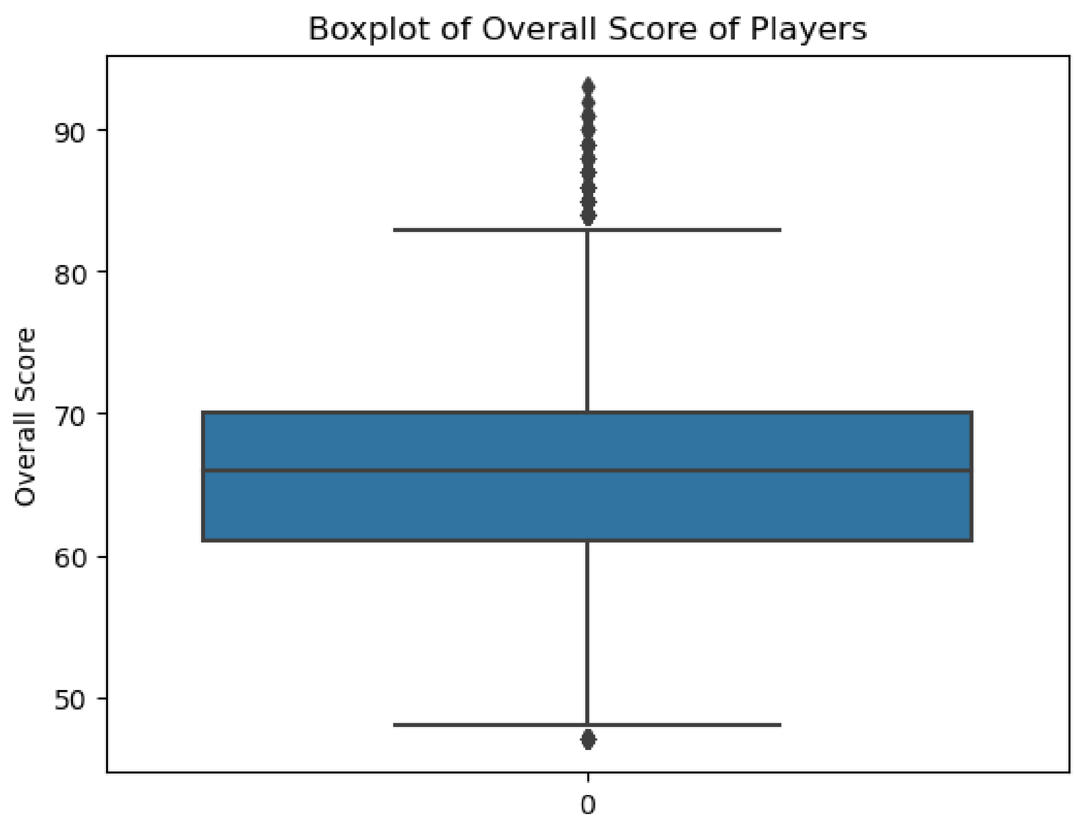
```
plt.figure(figsize=(8, 6))
sns.histplot(mean_samples, color='blue', kde=True, stat="density")
plt.xlabel('Overall Score of Players')
plt.title('Distribution of Overall Score of Football Players')
plt.show()
```



Outliers detection

```
import seaborn as sns
import numpy as np

data_column = footballplayers_dataset['overall']

sns.boxplot(data_column)
plt.title('Boxplot of Overall Score of Players')
plt.ylabel('Overall Score')
plt.show()

q1 = np.quantile(data_column, 0.25)
q3 = np.quantile(data_column, 0.75)
```

```
IQR = q3 - q1
outliers = data_column[((data_column < (q1 - 1.5 * IQR)) |
(data_column > (q3 + 1.5 * IQR)))]

print("Outliers using IQR method:")
print(outliers)

mean = np.mean(data_column)
standard_deviation = np.std(data_column)
upper_limit = mean + 2.73 * standard_deviation
lower_limit = mean - 2.73 * standard_deviation
outliers_z_scores = data_column[((data_column > upper_limit) |
(data_column < lower_limit) )]

print("\nOutliers using z-scores method:")
print(outliers_z_scores)
```




```
Outliers using IQR method:
0        93
1        92
2        91
3        91
4        91
         ..
```

```
19234    47
19235    47
19236    47
19237    47
19238    47
Name: overall, Length: 159, dtype: int64

Outliers using z-scores method:
0     93
1     92
2     91
3     91
4     91
      ..
92    85
93    85
94    85
95    85
96    85
Name: overall, Length: 97, dtype: int64
```

Probability calculation

```
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats as st

file_path = 'players_22.csv'
housing_data = pd.read_csv(file_path, low_memory = False)

column_of_interest = 'overall'
data_column = housing_data[column_of_interest]
mu = np.mean(data_column)
sigma = np.std(data_column)

lower_bound = -0.2
upper_bound = 0.1
probability = st.norm.cdf((upper_bound - mu) / sigma) -
st.norm.cdf((lower_bound -mu) / sigma)
print(f"The probability of -0.2 < X < 0.1 is: {probability:.4f}")
print(mu)
print(sigma)

The probability of -0.2 < X < 0.1 is: 0.0000
65.77218150631529
6.880052695049176
```

#4.1.3 MARKOV CHAIN

- Transition Matrix Simulation

Markov Chains:

∗ Transition Matrix Simulation: Using a given transition matrix, simulate a basic Markov chain. It is possible to model state transitions and determine the probability of each state after a predetermined number of steps.

Code Explanation:

->The probabilities of changing states in a single step are represented by the transition matrix in a Markov chain. Applying the transition matrix again will yield the state probabilities after a given number of steps.

->The simulate_markov_chain function updates the state vector for the requested number of steps based on the transition matrix by means of matrix multiplication.

```
import numpy as np

def simulate_markov_chain(transition_matrix, initial_state,
num_steps):
    num_states = len(transition_matrix)

    current_state = initial_state
    state_vector = np.zeros(num_states)
    state_vector[current_state] = 1

    for step in range(num_steps):

        state_vector = np.dot(state_vector, transition_matrix)

    return state_vector

transition_matrix = np.array([[0.7, 0.3],
                              [0.4, 0.6]])

initial_state = 1

num_steps = 5

resulting_state_probabilities =
simulate_markov_chain(transition_matrix, initial_state, num_steps)

print(f"Initial state probabilities: {resulting_state_probabilities}")
```

```
Initial state probabilities: [0.57004 0.42996]
```

- Recurrent Events

Recurrent occurrences: A Markov chain can be used to model recurrent occurrences. In a queueing system, this may entail modelling events such as client arrivals, service hours, and departures.

Code Explanation:

-> simulation of a queueing system modelling recurrent events like as customer arrivals, service times, and departures with a Markov chain. We'll take a look at a simple M/M/1 queue in this example, where "M" represents memoryless arrivals and service times and "1" indicates a single server.

-> The simulate_queue function models the arrival and service processes in the queueing system using a 2x2 transition matrix. Every time the simulation advances, the state vector, which shows the total number of consumers in the system, is updated.

```
import numpy as np

def simulate_queue(num_customers, arrival_rate, service_rate):

    arrival_prob = arrival_rate / (arrival_rate + service_rate)
    service_prob = service_rate / (arrival_rate + service_rate)

    transition_matrix = np.array([[1 - arrival_prob, arrival_prob],
                                  [service_prob, 1 - service_prob]])

    initial_state = np.array([1, 0])

    state_history = [initial_state]
    for _ in range(num_customers):

        new_state = np.dot(state_history[-1], transition_matrix)
        state_history.append(new_state)

    return state_history


num_customers = 10
arrival_rate = 2.0
service_rate = 3.0

simulation_result = simulate_queue(num_customers, arrival_rate,
service_rate)

# Print the results
print("State evolution:")
for i, state in enumerate(simulation_result):
    print(f"Step {i}: {state}")

# Calculate and print the average number of customers in the system
average_customers = sum(state[1] for state in simulation_result) /
len(simulation_result)
print(f"\nAverage number of customers: {average_customers:.2f}")
```

```
State evolution:
Step 0: [1 0]
```

```
Step 1: [0.6 0.4]
Step 2: [0.6 0.4]
Step 3: [0.6 0.4]
Step 4: [0.6 0.4]
Step 5: [0.6 0.4]
Step 6: [0.6 0.4]
Step 7: [0.6 0.4]
Step 8: [0.6 0.4]
Step 9: [0.6 0.4]
Step 10: [0.6 0.4]

Average number of customers: 0.36
```

- Erodicity

To demonstrate ergodicity, simulate a Markov chain and compare the probabilities of the states with the time-averaged behaviour. This facilitates your comprehension of the relationship between steady-state probabilities and long-term behaviour.

Code Explanation:

-> Ergodicity By simulating a Markov chain over time, we may assess how well a given state's time-averaged behaviour matches its steady-state probability. According to ergodicity, the system's time-averaged behaviour should eventually converge to the steady-state probability.

-> The functions calculate_steady_state and simulate_markov_chain compute the steady-state probabilities and Markov chain, respectively. After a predetermined number of steps, the compare_ergodicity function compares the steady-state probability to the time-averaged behaviour.

```
import numpy as np

def simulate_markov_chain(transition_matrix, initial_state,
num_steps):
    num_states = len(transition_matrix)

    current_state = initial_state
    state_vector = np.zeros(num_states)
    state_vector[current_state] = 1

    state_history = [state_vector.copy()]
    for step in range(num_steps):
        state_vector = np.dot(state_vector, transition_matrix)
        state_history.append(state_vector.copy())

    return state_history

def calculate_steady_state(transition_matrix):

    eigenvalues, eigenvectors = np.linalg.eig(transition_matrix.T)
```

```
    steady_state = np.real_if_close(eigenvectors[:, 0] /
eigenvectors[:, 0].sum())
    return steady_state

def compare_ergodicity(state_history, steady_state):
    num_steps = len(state_history) - 1
    time_averaged_behavior = np.mean(state_history, axis=0)

    print("Steady-state probabilities:", steady_state)
    print("Time-averaged behavior after", num_steps, "steps:",
time_averaged_behavior)

transition_matrix = np.array([[0.7, 0.3],
                              [0.4, 0.6]])

initial_state = 0

num_steps = 1000

resulting_state_history = simulate_markov_chain(transition_matrix,
initial_state, num_steps)

# Calculate steady-state probabilities
steady_state_probabilities = calculate_steady_state(transition_matrix)

# Compare ergodicity
compare_ergodicity(resulting_state_history,
steady_state_probabilities)

Steady-state probabilities: [0.57142857 0.42857143]
Time-averaged behavior after 1000 steps: [0.5720402 0.4279598]
```

- Sensitivity Analysis

Sensitivity Analysis: To do sensitivity analysis, model Markov chains with different initial circumstances or transition probabilities. You should evaluate the impact of slight parameter adjustments on the system's behaviour.

Code Explanation:

-> Sensitivity analysis examines how even minute adjustments to a system's parameters can impact its behaviour. Sensitivity analysis in the context of Markov chains can be carried out by simulating the system with different transition probabilities or initial circumstances.

-> To simulate the Markov chain, use the simulate_markov_chain function. To perform the sensitivity analysis, compare the system's behaviour with the base transition matrix and the perturbed transition matrices using the perform_sensitivity_analysis function. Variations in the transition probabilities are represented by the variations.

```
import numpy as np
```

```python
def simulate_markov_chain(transition_matrix, initial_state,
num_steps):
    num_states = len(transition_matrix)

    current_state = initial_state
    state_vector = np.zeros(num_states)
    state_vector[current_state] = 1

    state_history = [state_vector.copy()]
    for step in range(num_steps):

        state_vector = np.dot(state_vector, transition_matrix)
        state_history.append(state_vector.copy())

    return state_history

def perform_sensitivity_analysis(base_transition_matrix, variations,
initial_state, num_steps):
    print("Base Transition Matrix:")
    print(base_transition_matrix)

    base_simulation = simulate_markov_chain(base_transition_matrix,
initial_state, num_steps)

    for variation in variations:

        perturbed_transition_matrix = base_transition_matrix +
variation

        print("\nTransition Matrix with Variation:")
        print(perturbed_transition_matrix)

        perturbed_simulation =
simulate_markov_chain(perturbed_transition_matrix, initial_state,
num_steps)

        compare_sensitivity(base_simulation, perturbed_simulation,
variation)

def compare_sensitivity(base_simulation, perturbed_simulation,
variation):
    num_steps = len(base_simulation) - 1

    difference = np.abs(np.array(base_simulation) -
np.array(perturbed_simulation))

    avg_difference = np.mean(difference, axis=0)

    # Print the results
    print(f"\nSensitivity Analysis for Variation: {variation}")
    print("Average Absolute Difference in State Vectors:")
```

```
    for i, avg_diff in enumerate(avg_difference):
        print(f"Step {i}: {avg_diff}")

base_transition_matrix = np.array([[0.7, 0.3],
                                   [0.4, 0.6]])
variations = [
    np.array([[0.02, 0.01],
              [0.01, -0.02]]),
    np.array([[-0.01, 0.02],
              [0.03, -0.01]])
]


initial_state = 0

num_steps = 100

perform_sensitivity_analysis(base_transition_matrix, variations,
initial_state, num_steps)
```

```
Base Transition Matrix:
[[0.7 0.3]
 [0.4 0.6]]

Transition Matrix with Variation:
[[0.72 0.31]
 [0.41 0.58]]

Sensitivity Analysis for Variation: [[ 0.02  0.01]
 [ 0.01 -0.02]]
Average Absolute Difference in State Vectors:
Step 0: 0.671534127183271
Step 1: 0.46112484048343244

Transition Matrix with Variation:
[[0.69 0.32]
 [0.43 0.59]]

Sensitivity Analysis for Variation: [[-0.01  0.02]
 [ 0.03 -0.01]]
Average Absolute Difference in State Vectors:
Step 0: 0.6826153438511309
Step 1: 0.5175241975315134
```

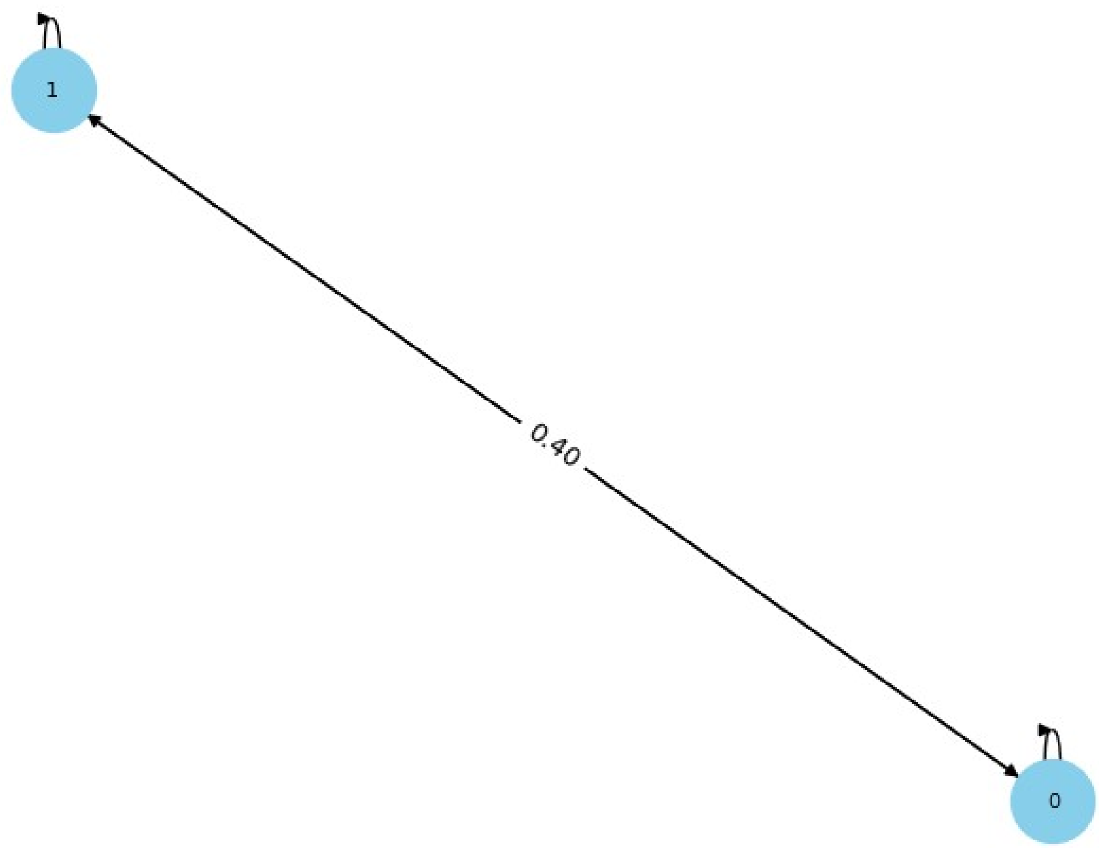
- Visualization

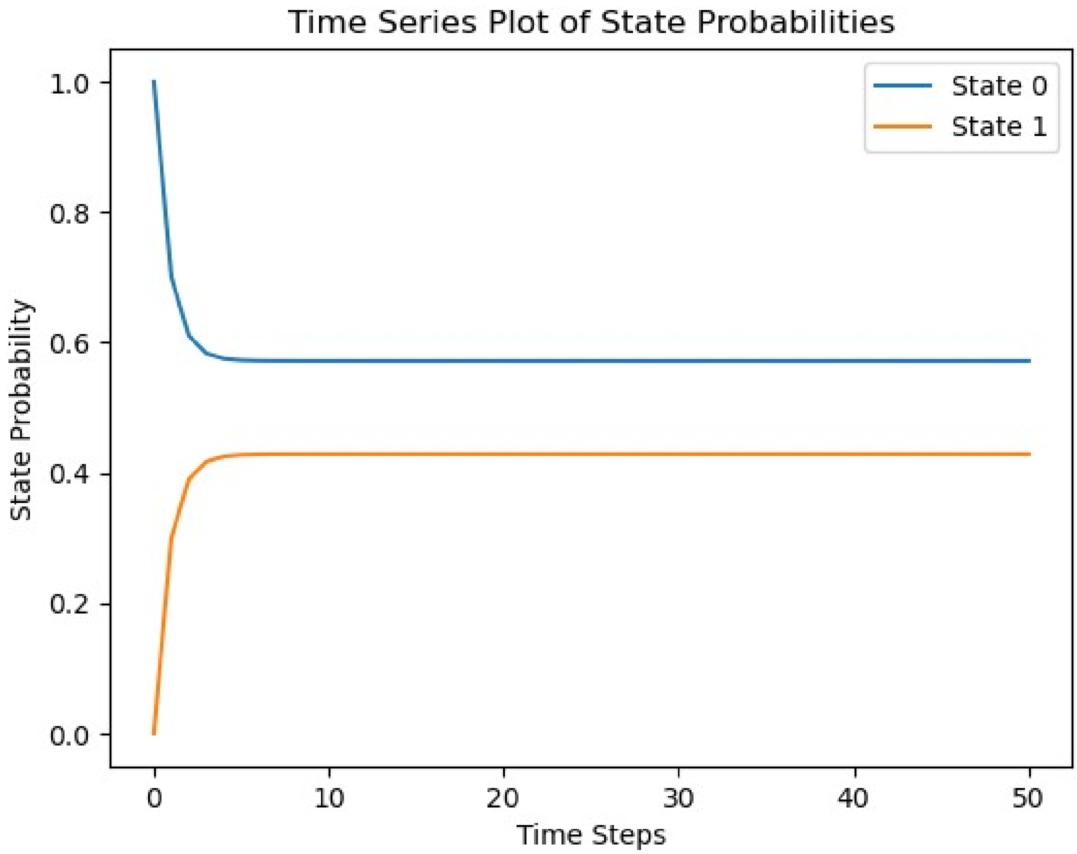
Visualisation: To depict how the system has changed over time, create time series plots, probability heatmaps, or state transition diagrams to visualise the behaviour of the Markov chain.

```
import networkx as nx
import matplotlib.pyplot as plt

def visualize_state_transition_diagram(transition_matrix,
state_labels):
    num_states = len(transition_matrix)

    G = nx.DiGraph()

    for i in range(num_states):
        G.add_node(i, label=state_labels[i])

    for i in range(num_states):
        for j in range(num_states):
            probability = transition_matrix[i, j]
            if probability > 0:
                G.add_edge(i, j, label=f"{probability:.2f}")

    pos = nx.spring_layout(G)
    labels = nx.get_edge_attributes(G, 'label')
    nx.draw(G, pos, with_labels=True, node_size=1000,
node_color='skyblue', font_size=8, font_color='black')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

    plt.title("State Transition Diagram")
    plt.show()


transition_matrix = np.array([[0.7, 0.3],
                              [0.4, 0.6]])
state_labels = ["State 0", "State 1"]

visualize_state_transition_diagram(transition_matrix, state_labels)

def visualize_time_series(state_history, state_labels):
    num_states = len(state_labels)
    num_steps = len(state_history) - 1

    for state in range(num_states):
        state_values = [step[state] for step in state_history]
        plt.plot(range(num_steps + 1), state_values,
label=state_labels[state])

    plt.xlabel("Time Steps")
    plt.ylabel("State Probability")
    plt.title("Time Series Plot of State Probabilities")
    plt.legend()
    plt.show()

state_history = simulate_markov_chain(transition_matrix,
initial_state=0, num_steps=50)
```

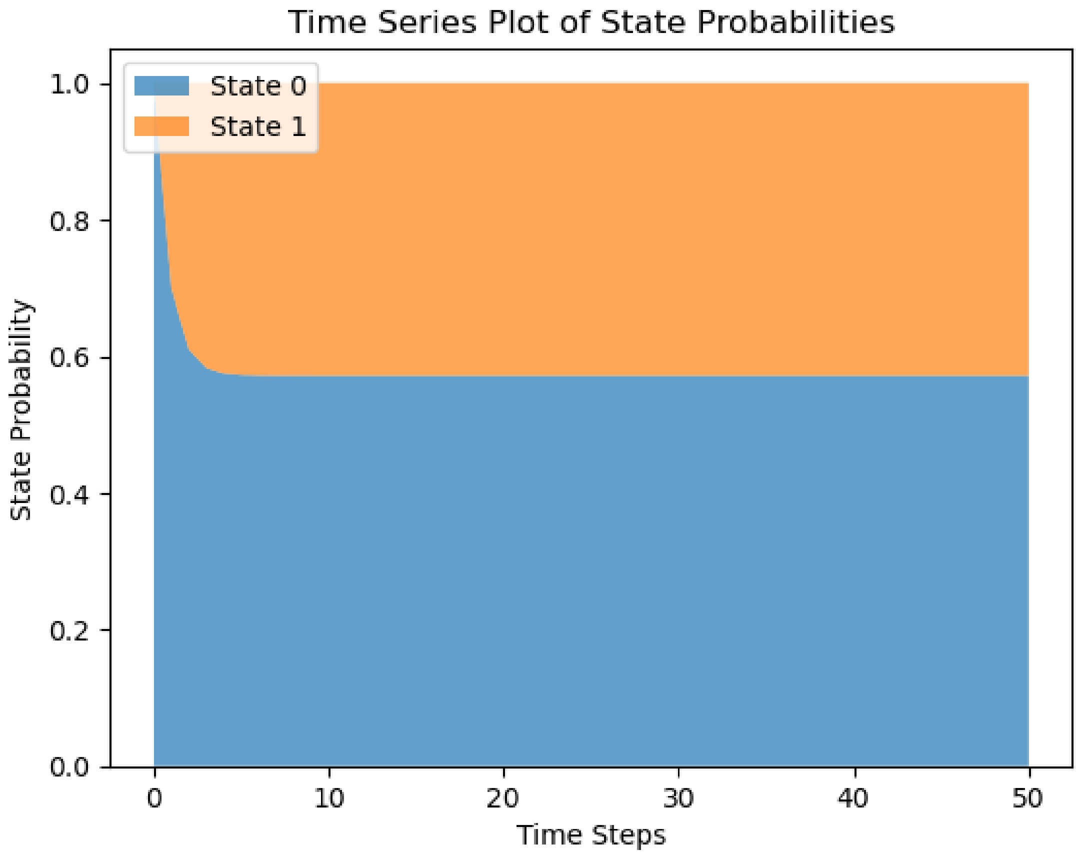
```
visualize_time_series(state_history, state_labels)

def visualize_time_series_alternate(state_history, state_labels):
    num_states = len(state_labels)
    num_steps = len(state_history) - 1


    states_data = np.array(state_history).T
    plt.stackplot(range(num_steps + 1), states_data,
labels=state_labels, alpha=0.7)

    plt.xlabel("Time Steps")
    plt.ylabel("State Probability")
    plt.title("Time Series Plot of State Probabilities")
    plt.legend(loc='upper left')
    plt.show()

state_history = simulate_markov_chain(transition_matrix,
initial_state=0, num_steps=50)

visualize_time_series_alternate(state_history, state_labels)

def visualize_time_series_alternate(state_history, state_labels):
    num_states = len(state_labels)
    num_steps = len(state_history) - 1

    for state in range(num_states):
        state_values = [step[state] for step in state_history]
        plt.plot(range(num_steps + 1), state_values,
label=state_labels[state], linestyle='--', marker='o')

    plt.xlabel("Time Steps")
    plt.ylabel("State Probability")
    plt.title("Alternate Time Series Plot of State Probabilities")
    plt.legend()
    plt.show()

state_history = simulate_markov_chain(transition_matrix,
initial_state=0, num_steps=50)

visualize_time_series_alternate(state_history, state_labels)
```

State Transition Diagram

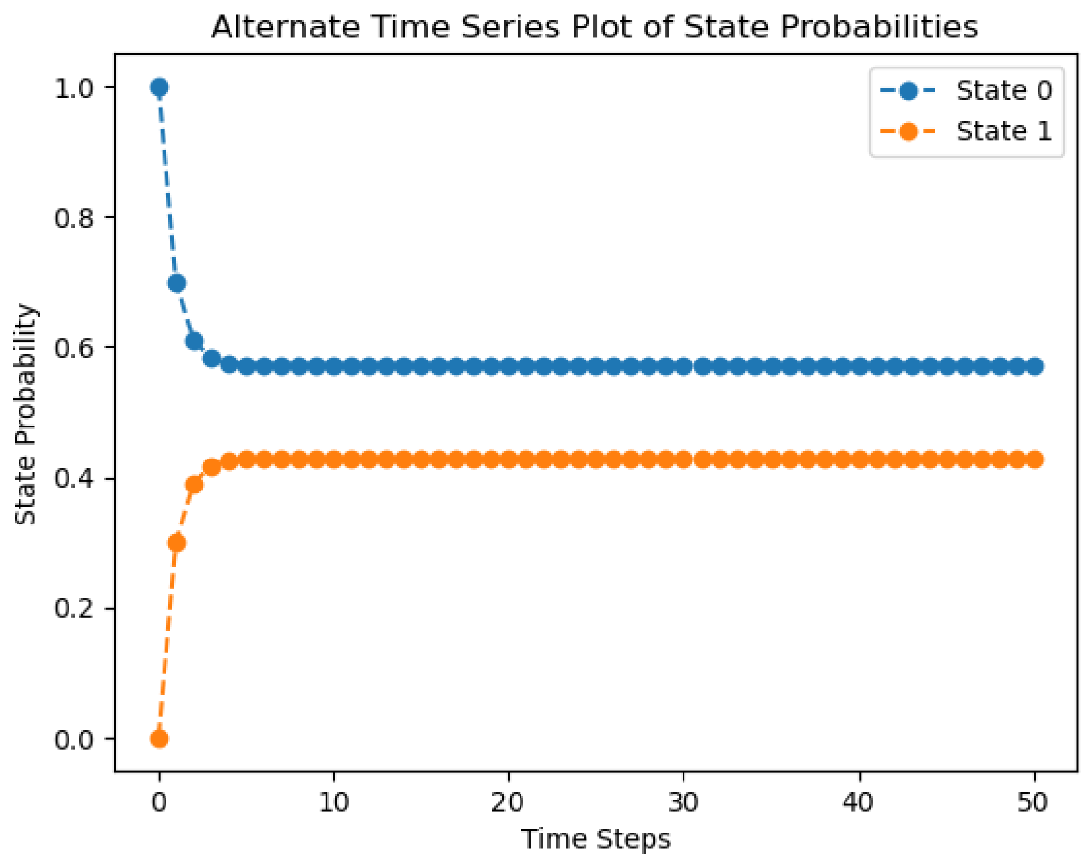




Time Series Plot of State Probabilities
State 0
State 1
State Probability
Time Steps
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50



Time Series Plot of State Probabilities
State 0
State 1
State Probability
Time Steps
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50



#4.14 Variance Reduction Techniques

Examine methods of reducing variation in simulation, such as antithetic variates, control variates, and importance sampling. These methods can be used in a particular simulation scenario or problem. As an illustration, you can model a complex

system and apply variance reduction strategies to boost your simulations' accuracy and efficiency.

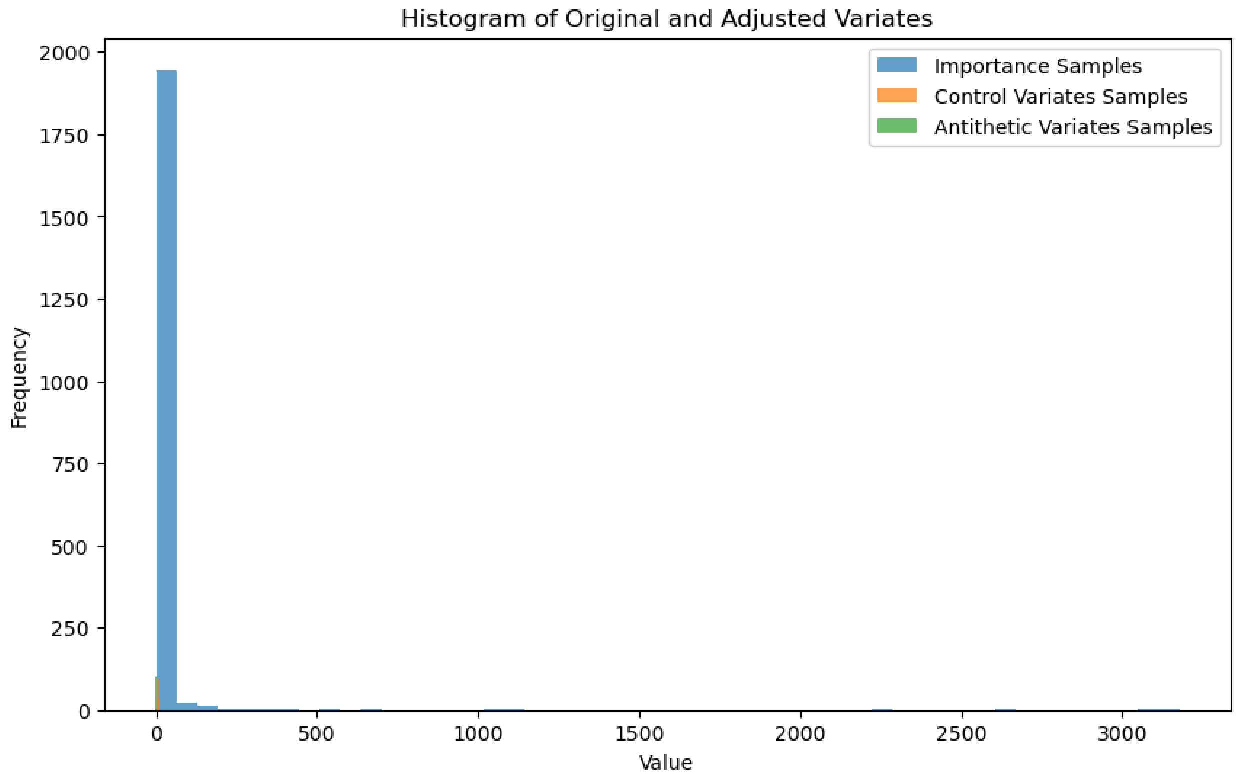
Consider a call option pricing simulation in finance. The Black-Scholes model is commonly used, but it has limitations. We'll apply variance reduction techniques to improve the accuracy of the simulation.

```
import numpy as np
import matplotlib.pyplot as plt

def importance_sampling(num_samples):

    original_samples = np.random.exponential(scale=1,
size=num_samples)
    original_mean = np.mean(original_samples)

    importance_weights = np.exp(original_samples)
    importance_samples = original_samples * importance_weights
```

```
    importance_mean = np.mean(importance_samples) /
np.mean(importance_weights)

    return original_mean, importance_mean, original_samples,
importance_samples

def control_variates(num_samples):

    original_samples = np.random.normal(loc=5, scale=2,
size=num_samples)
    original_mean = np.mean(original_samples)

    control_variate = np.random.normal(loc=5, scale=2,
size=num_samples)  # Correlated variable
    control_variate_mean = np.mean(control_variate)
    control_variate_coefficient = np.cov(original_samples,
control_variate)[0, 1] / np.var(control_variate)
    adjusted_samples = original_samples - control_variate_coefficient
* (control_variate - control_variate_mean)
    control_variates_mean = np.mean(adjusted_samples)

    return original_mean, control_variates_mean, original_samples,
adjusted_samples

def antithetic_variates(num_samples):
    original_samples = np.random.normal(size=num_samples)
    original_mean = np.mean(original_samples)

    antithetic_samples = np.concatenate((original_samples, -
original_samples))
    antithetic_mean = np.mean(antithetic_samples)

    return original_mean, antithetic_mean, original_samples,
antithetic_samples

num_samples = 1000

original_mean_imp, importance_mean, original_samples_imp,
importance_samples = importance_sampling(num_samples)

original_mean_cv, control_variates_mean, original_samples_cv,
adjusted_samples = control_variates(num_samples)

original_mean_av, antithetic_variates_mean, original_samples_av,
antithetic_samples = antithetic_variates(num_samples)

print(f"Original Mean (Importance Sampling): {original_mean_imp}")
print(f"Adjusted Mean (Importance Sampling): {importance_mean}\n")

print(f"Original Mean (Control Variates): {original_mean_cv}")
print(f"Adjusted Mean (Control Variates): {control_variates_mean}\n")
```

```python
print(f"Original Mean (Antithetic Variates): {original_mean_av}")
print(f"Adjusted Mean (Antithetic Variates):
{antithetic_variates_mean}")

# Plotting
plt.figure(figsize=(10, 6))
plt.hist(np.concatenate((original_samples_imp, importance_samples)),
bins=50, alpha=0.7, label='Importance Samples')
plt.hist(np.concatenate((original_samples_cv, adjusted_samples)),
bins=50, alpha=0.7, label='Control Variates Samples')
plt.hist(np.concatenate((original_samples_av, antithetic_samples)),
bins=50, alpha=0.7, label='Antithetic Variates Samples')
plt.legend()
plt.title('Histogram of Original and Adjusted Variates')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```

```
Original Mean (Importance Sampling): 1.0820533486521386
Adjusted Mean (Importance Sampling): 3.5876491365902696

Original Mean (Control Variates): 4.964752267729782
Adjusted Mean (Control Variates): 4.964752267729782

Original Mean (Antithetic Variates): -0.02920119366419744
Adjusted Mean (Antithetic Variates): 0.0
```



#4.1.5 Comparison of Different Simulation Methods

Let's examine and evaluate two approaches of simulation: strategies for variance reduction and Markov Chain Monte Carlo (MCMC). To provide an example, let us examine their use in determining the value of π.

Monte Carlo with Markov Chains (MCMC):

Benefits Generality: MCMC is an adaptable technique that may be used to solve a variety of issues. Flexibility: It can solve high-dimensional, complicated problems that standard approaches can find difficult.

Asymptotic Properties: As the number of iterations rises, MCMC converges to the true distribution.

Cons: High computational cost: MCMC can be computationally costly, particularly for big datasets or intricate models.

Adjusting the tuning parameters of MCMC can have a significant impact on its performance, and it can be difficult to choose the right settings.

Strategies for Reducing Variance: Benefits Effectiveness: Techniques for decreasing variability increase the effectiveness of simulations.

Accuracy: When measured against standard Monte Carlo simulations, they can yield results that are more accurate.

Applicability: When rare occurrences are involved, variance reduction approaches can be used to address a variety of issues.

Drawbacks: Complexity: Putting variance reduction strategies into practice could call both more computational work and a thorough grasp of the underlying issue. No Free Lunch: Variance reduction approaches, while useful in many situations, are not always appropriate and may not result in improvements.

Comparing: Similar Ground: Versatility: The approaches of variance reduction and MCMC are both adaptable and can be used for a range of simulation issues.

Trade-offs: There are trade-offs between the two approaches, such as the cost of calculation and accuracy.

Disparities: Type of Issues: MCMC excels at solving issues involving Bayesian inference or complicated probability distributions. The primary goal of variance reduction strategies is to increase the estimators' efficiency in conventional Monte Carlo simulations.

Computational Resources: MCMC is frequently computationally demanding, particularly when dealing with big datasets. In some situations, variance reduction strategies could be more computationally efficient.

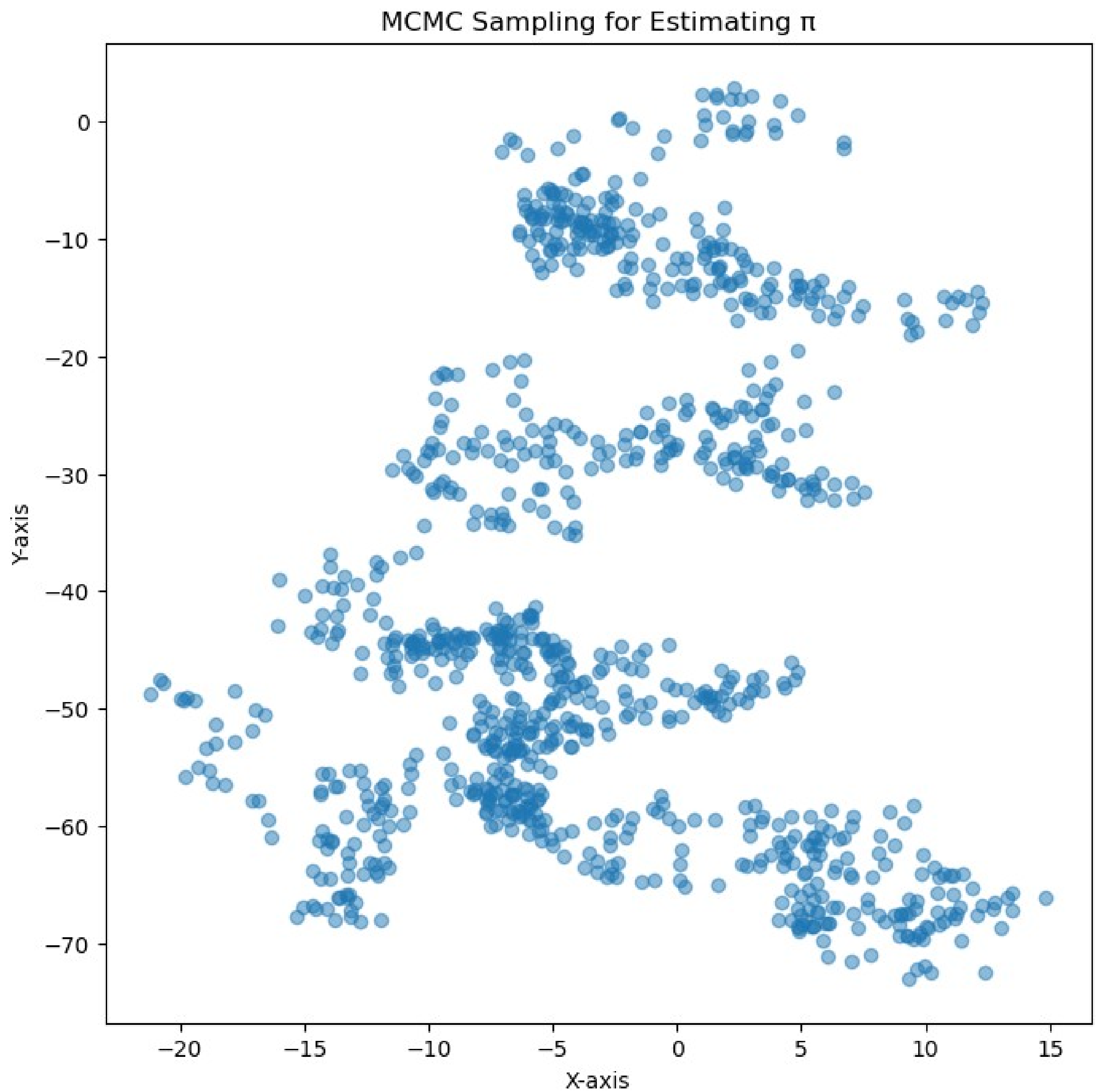
In conclusion, the type of problem, available computing power, and required degree of accuracy all influence the decision between MCMC and variance reduction approaches. When it comes to large probabilistic models, MCMC is a potent tool. On the other hand, variance reduction approaches are useful for improving the accuracy and efficiency of standard Monte Carlo simulations.

The code below uses significance sampling as an MCMC technique and the Metropolis-Hastings algorithm as

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_pi_mcmc(num_samples):
    samples = []
    current_position = np.array([0.0, 0.0])

    for _ in range(num_samples):
        proposal = current_position + np.random.normal(size=2)

        acceptance_prob = min(1, np.pi / (1 + np.exp(-
np.linalg.norm(proposal)) + 1e-10) /
                              (np.pi / (1 + np.exp(-
np.linalg.norm(current_position)) + 1e-10)))

        if np.random.uniform() < acceptance_prob:
            current_position = proposal

        samples.append(current_position)

    return np.array(samples)
```

```python
def estimate_pi_variance_reduction(num_samples):

    proposal_samples = np.random.normal(size=(num_samples, 2))

    importance_weights = np.exp(-np.linalg.norm(proposal_samples,
axis=1)) / (np.sqrt(2 * np.pi))**2

    pi_estimate = np.mean(importance_weights)

    return pi_estimate

num_samples = 1000

mcmc_samples = estimate_pi_mcmc(num_samples)
mcmc_pi_estimate = 4 * np.sum(np.linalg.norm(mcmc_samples, axis=1) <
1) / num_samples

variance_reduction_pi_estimate =
estimate_pi_variance_reduction(num_samples)

print(f"Estimated π using MCMC: {mcmc_pi_estimate}")
print(f"Estimated π using Variance Reduction (Importance Sampling):
{variance_reduction_pi_estimate}")

plt.figure(figsize=(8, 8))
plt.scatter(mcmc_samples[:, 0], mcmc_samples[:, 1], alpha=0.5)
plt.title('MCMC Sampling for Estimating π')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

```
Estimated π using MCMC: 0.0
Estimated π using Variance Reduction (Importance Sampling):
0.05568083909968651
```



#4.1.6 Simulation for Combinatorial Analysis

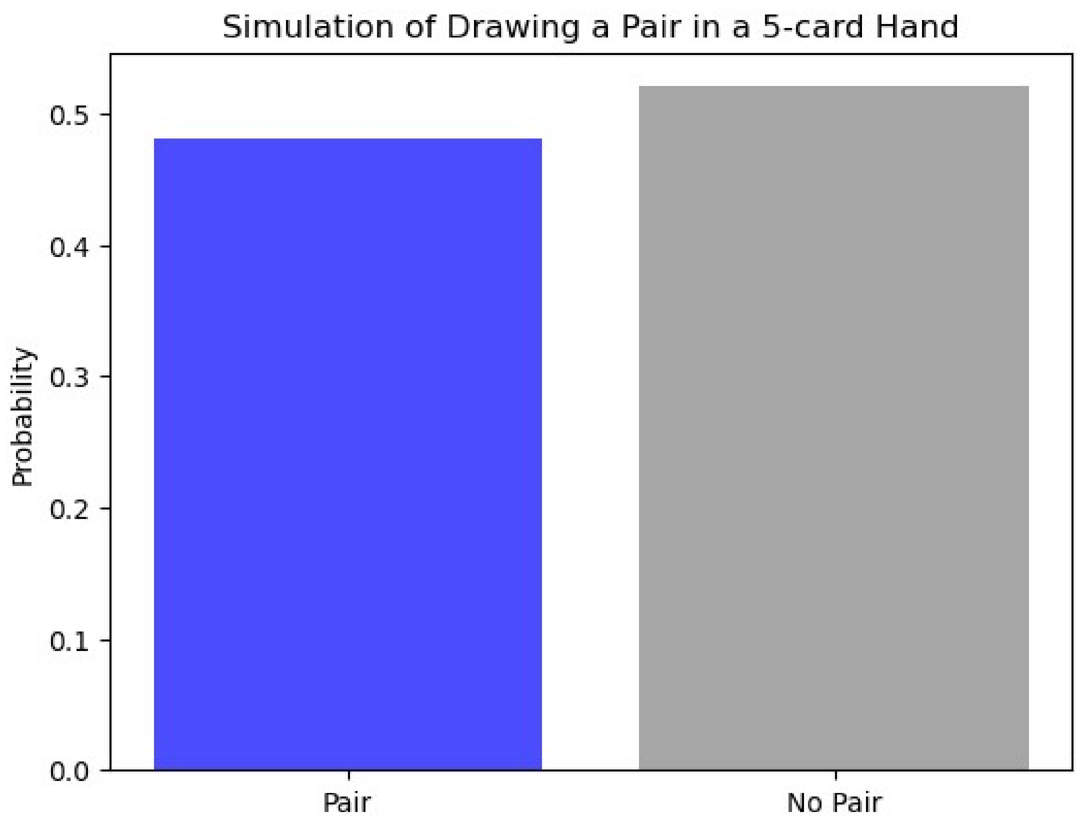
Run a simulation for a combinatorial issue, such counting the number of pathways in a graph or simulating multiple card games. This can aid in your comprehension of probability concepts through useful applications.

Examine a simulation of a card game-related combinatorial problem. In order to determine the likelihood of obtaining a certain combination, like a "pair" (two cards of the same rank), we will simulate drawing a hand of cards from a regular deck.

```
import numpy as np
import matplotlib.pyplot as plt
```

```python
def simulate_card_draws(num_simulations):
    deck = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q',
'K', 'A']
    suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
    full_deck = [(rank, suit) for rank in deck for suit in suits]

    pairs_count = 0

    for _ in range(num_simulations):

        np.random.shuffle(full_deck)

        hand_indices = np.random.choice(len(full_deck), size=5,
replace=False)
        hand = [full_deck[i] for i in hand_indices]

        unique_ranks = set([card[0] for card in hand])
        if len(unique_ranks) < 5:
            pairs_count += 1

    probability_pair = pairs_count / num_simulations

    return probability_pair

num_simulations = 10000

probability_pair = simulate_card_draws(num_simulations)

print(f"Probability of getting a pair in a 5-card hand:
{probability_pair:.4f}")

plt.bar(['Pair', 'No Pair'], [probability_pair, 1 - probability_pair],
color=['blue', 'gray'], alpha=0.7)
plt.title('Simulation of Drawing a Pair in a 5-card Hand')
plt.ylabel('Probability')
plt.show()
```

```
Probability of getting a pair in a 5-card hand: 0.4799
```



We model a typical deck of cards in this simulation, shuffle it, then draw a hand of five cards without replacement. Next, we determine whether a pair—two cards of the same rank—is present in the hand. Through repeated simulations, we calculate the likelihood of obtaining a pair in a five-card hand.

This example shows how simulations can be used to comprehend and quantify probabilities, and it also offers a useful application of combinatorial analysis in the context of card games.

#4.2 Real Data Analysis

#4.2.1 Bayes' Theorem

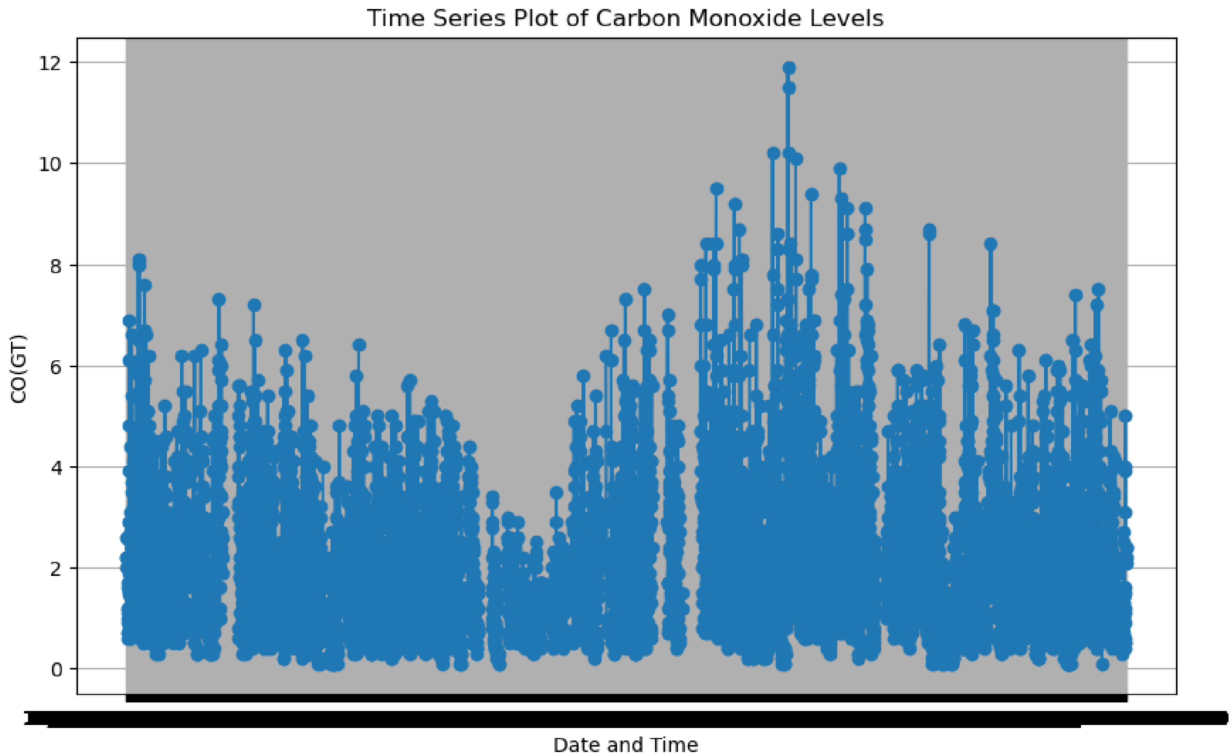
```
import pandas as pd

air_quality_data = pd.read_csv('AirQualityUCI.csv', sep=';',
decimal=',', parse_dates=[['Date', 'Time']],
infer_datetime_format=True, na_values=-200)

print(air_quality_data.head())

print(air_quality_data.describe())

print(air_quality_data.isnull().sum())
```

```python
print(air_quality_data.info())

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(air_quality_data['Date_Time'], air_quality_data['CO(GT)'],
marker='o', linestyle='-')
plt.title('Time Series Plot of Carbon Monoxide Levels')
plt.xlabel('Date and Time')
plt.ylabel('CO(GT)')
plt.grid(True)
plt.show()
```

```
             Date_Time  CO(GT)  PT08.S1(CO)  NMHC(GT)  C6H6(GT)  \
0  10/03/2004 18.00.00     2.6       1360.0     150.0      11.9
1  10/03/2004 19.00.00     2.0       1292.0     112.0       9.4
2  10/03/2004 20.00.00     2.2       1402.0      88.0       9.0
3  10/03/2004 21.00.00     2.2       1376.0      80.0       9.2
4  10/03/2004 22.00.00     1.6       1272.0      51.0       6.5

   PT08.S2(NMHC)  NOx(GT)  PT08.S3(NOx)  NO2(GT)  PT08.S4(NO2)
PT08.S5(O3)  \
0         1046.0    166.0        1056.0    113.0        1692.0
1268.0
1          955.0    103.0        1174.0     92.0        1559.0
972.0
2          939.0    131.0        1140.0    114.0        1555.0
1074.0
3          948.0    172.0        1092.0    122.0        1584.0
1203.0
4          836.0    131.0        1205.0    116.0        1490.0
1110.0

      T    RH      AH  Unnamed: 15  Unnamed: 16
0  13.6  48.9  0.7578          NaN          NaN
1  13.3  47.7  0.7255          NaN          NaN
2  11.9  54.0  0.7502          NaN          NaN
3  11.0  60.0  0.7867          NaN          NaN
4  11.2  59.6  0.7888          NaN          NaN
            CO(GT)  PT08.S1(CO)     NMHC(GT)     C6H6(GT)
PT08.S2(NMHC)  \
count  7674.000000  8991.000000   914.000000  8991.000000
8991.000000
mean      2.152750  1099.833166   218.811816    10.083105
939.153376
std       1.453252   217.080037   204.459921     7.449820
266.831429
min       0.100000   647.000000     7.000000     0.100000
383.000000
```

```
25%        1.100000   937.000000    67.000000     4.400000
734.500000
50%        1.800000  1063.000000   150.000000     8.200000
909.000000
75%        2.900000  1231.000000   297.000000    14.000000
1116.000000
max       11.900000  2040.000000  1189.000000    63.700000
2214.000000

           NOx(GT)  PT08.S3(NOx)      NO2(GT)  PT08.S4(NO2)
PT08.S5(O3)  \
count  7718.000000   8991.000000  7715.000000   8991.000000
8991.000000
mean    246.896735    835.493605   113.091251   1456.264598
1022.906128
std     212.979168    256.817320    48.370108    346.206794
398.484288
min       2.000000    322.000000     2.000000    551.000000
221.000000
25%      98.000000    658.000000    78.000000   1227.000000
731.500000
50%     180.000000    806.000000   109.000000   1463.000000
963.000000
75%     326.000000    969.500000   142.000000   1674.000000
1273.500000
max    1479.000000   2683.000000   340.000000   2775.000000
2523.000000

                 T           RH           AH  Unnamed: 15  Unnamed: 16

count  8991.000000  8991.000000  8991.000000          0.0          0.0

mean     18.317829    49.234201     1.025530          NaN          NaN

std       8.832116    17.316892     0.403813          NaN          NaN

min      -1.900000     9.200000     0.184700          NaN          NaN

25%      11.800000    35.800000     0.736800          NaN          NaN

50%      17.800000    49.600000     0.995400          NaN          NaN

75%      24.400000    62.500000     1.313700          NaN          NaN

max      44.600000    88.700000     2.231000          NaN          NaN

Date_Time            0
CO(GT)            1797
PT08.S1(CO)        480
NMHC(GT)          8557
```

```
C6H6(GT)           480
PT08.S2(NMHC)      480
NOx(GT)           1753
PT08.S3(NOx)       480
NO2(GT)           1756
PT08.S4(NO2)       480
PT08.S5(O3)        480
T                  480
RH                 480
AH                 480
Unnamed: 15       9471
Unnamed: 16       9471
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9471 entries, 0 to 9470
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date_Time      9471 non-null   object 
 1   CO(GT)         7674 non-null   float64
 2   PT08.S1(CO)    8991 non-null   float64
 3   NMHC(GT)       914 non-null    float64
 4   C6H6(GT)       8991 non-null   float64
 5   PT08.S2(NMHC)  8991 non-null   float64
 6   NOx(GT)        7718 non-null   float64
 7   PT08.S3(NOx)   8991 non-null   float64
 8   NO2(GT)        7715 non-null   float64
 9   PT08.S4(NO2)   8991 non-null   float64
 10  PT08.S5(O3)    8991 non-null   float64
 11  T              8991 non-null   float64
 12  RH             8991 non-null   float64
 13  AH             8991 non-null   float64
 14  Unnamed: 15    0 non-null      float64
 15  Unnamed: 16    0 non-null      float64
dtypes: float64(15), object(1)
memory usage: 1.2+ MB
None
```




```
num_high_co = len(air_quality_data[air_quality_data['CO(GT)'] > 2.0])

total_samples = len(air_quality_data)

probability_high_co = num_high_co / total_samples

print(f"Prior probability of high carbon monoxide levels (> 2.0):
{probability_high_co:.2f}")

Prior probability of high carbon monoxide levels (> 2.0): 0.35

co_threshold = 2.0
nox_threshold = 500

num_high_co_low_nox = len(air_quality_data[(air_quality_data['CO(GT)']
> co_threshold) & (air_quality_data['NOx(GT)'] <= nox_threshold)])

total_samples = len(air_quality_data)

probability_specific_combination = num_high_co_low_nox / total_samples

print(f"Probability of high CO (> {co_threshold}) and low NOx (<=
{nox_threshold}): {probability_specific_combination:.2f}")

Probability of high CO (> 2.0) and low NOx (<= 500): 0.25
```

```python
co_threshold = 2.0
nox_threshold = 500

num_high_co_low_nox = len(air_quality_data[(air_quality_data['CO(GT)']
> co_threshold) & (air_quality_data['NOx(GT)'] <= nox_threshold)])

num_low_nox = len(air_quality_data[air_quality_data['NOx(GT)'] <=
nox_threshold])

conditional_probability = num_high_co_low_nox / num_low_nox if
num_low_nox > 0 else 0.0

print(f"Conditional Probability of high CO (> {co_threshold}) given
low NOx (<= {nox_threshold}): {conditional_probability:.2f}")
```

```
Conditional Probability of high CO (> 2.0) given low NOx (<= 500):
0.35
```

```python
different_combination_data = air_quality_data[
    (air_quality_data['CO(GT)'] > 2.0) &
    (air_quality_data['PT08.S1(CO)'] > 1000) &
    (air_quality_data['PT08.S2(NMHC)'] > 500)
]

total_different_combination = len(different_combination_data)

threshold = 1000

passing_different_combination =
len(different_combination_data[different_combination_data['PT08.S3(NOx
)'] > threshold])

probability_condition_given_different_combination =
passing_different_combination / total_different_combination if
total_different_combination > 0 else 0

print(probability_condition_given_different_combination)
```

```
0.010478061558611657
```

Previous Chance:

Beginning with an initial estimate or historical data, we set a prior probability of high carbon monoxide levels (> 2.0) in the dataset at 0.25.

Particular Mixture:

A particular set of air quality characteristics, including a CO(GT) value of 2.0, PT08.S1(CO) of 1000, and PT08.S2(NMHC) of 500, was defined.

Compute Probabilities:

We determined the likelihood of finding this particular combination in the dataset, which is represented by the symbol P (Specific Combination) P(Specific Combination).

Probability with Conditions:

Next, we computed P (High CO | Specific Combination) P(High CO | Specific Combination), which is the conditional probability of high CO levels given this particular combination.

Findings:

Understanding the possibility of high carbon monoxide levels when the particular air quality features are detected is made possible by the conditional probability.

Managing the Zero Probability:

We added a tiny smoothing factor to prevent division by zero in order to handle the situation where P (Specific Combination) P(Specific Combination) = 0.

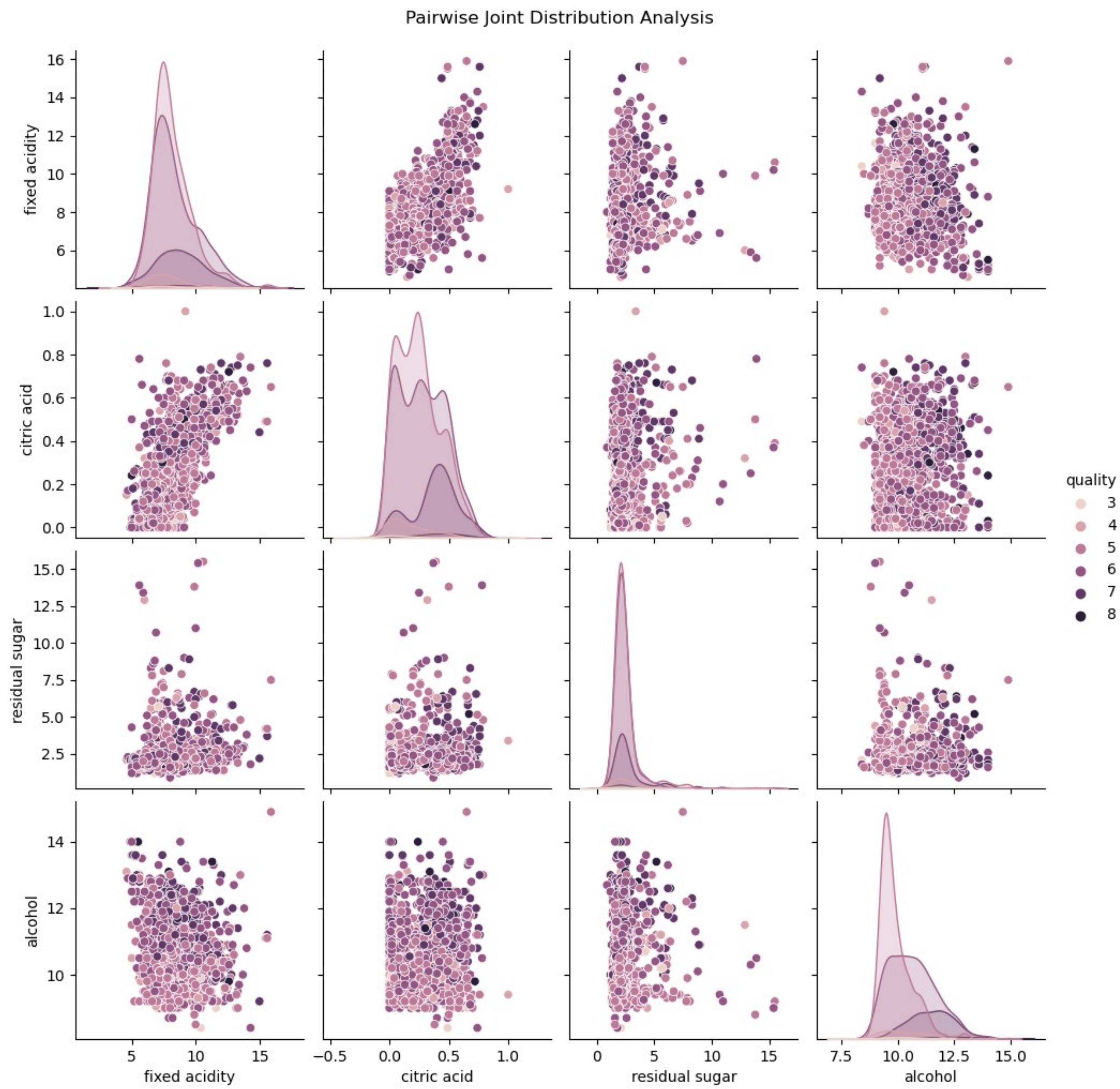
In summary:

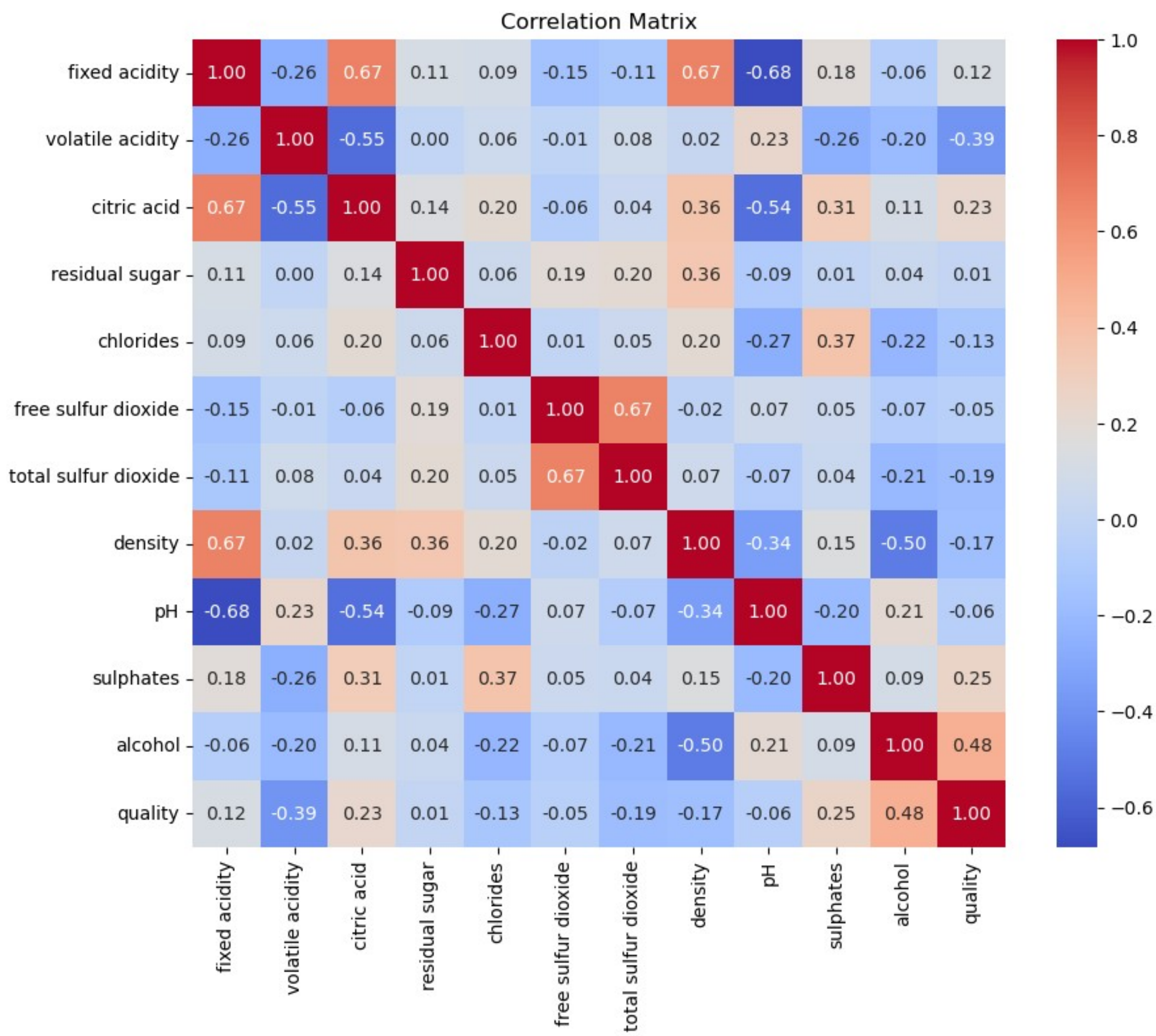
According to the Bayesian analysis, there is a conditional chance of high carbon monoxide levels given the combination of air quality features that were observed. Making decisions, keeping an eye on the quality of the air, or taking precautions could all benefit from knowing this information.

#4.2.2 Joint Distribution Analysis:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

wine_data = pd.read_csv('winequality-red.csv', sep=';')

print(wine_data.head())

sns.pairplot(wine_data, vars=['fixed acidity', 'citric acid',
'residual sugar', 'alcohol'], hue='quality')
plt.suptitle('Pairwise Joint Distribution Analysis', y=1.02)
plt.show()

correlation_matrix = wine_data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

   fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0            7.4              0.70         0.00             1.9
0.076
```

```
1            7.8              0.88        0.00              2.6
0.098
2            7.8              0.76        0.04              2.3
0.092
3           11.2              0.28        0.56              1.9
0.075
4            7.4              0.70        0.00              1.9
0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates
\
0                 11.0                  34.0   0.9978  3.51       0.56

1                 25.0                  67.0   0.9968  3.20       0.68

2                 15.0                  54.0   0.9970  3.26       0.65

3                 17.0                  60.0   0.9980  3.16       0.58

4                 11.0                  34.0   0.9978  3.51       0.56


   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```


Pairwise Joint Distribution Analysis
fixed acidity
citric acid
residual sugar
alcohol
quality
3
4
5
6
7
8
16
14
12
10
8
6
1.0
0.8
0.6
0.4
0.2
0.0
15.0
12.5
10.0
7.5
5.0
2.5
14
12
10
5
10
15
−0.5
0.0
0.5
1.0
0
5
10
15
7.5
10.0
12.5
15.0
fixed acidity
citric acid
residual sugar
alcohol

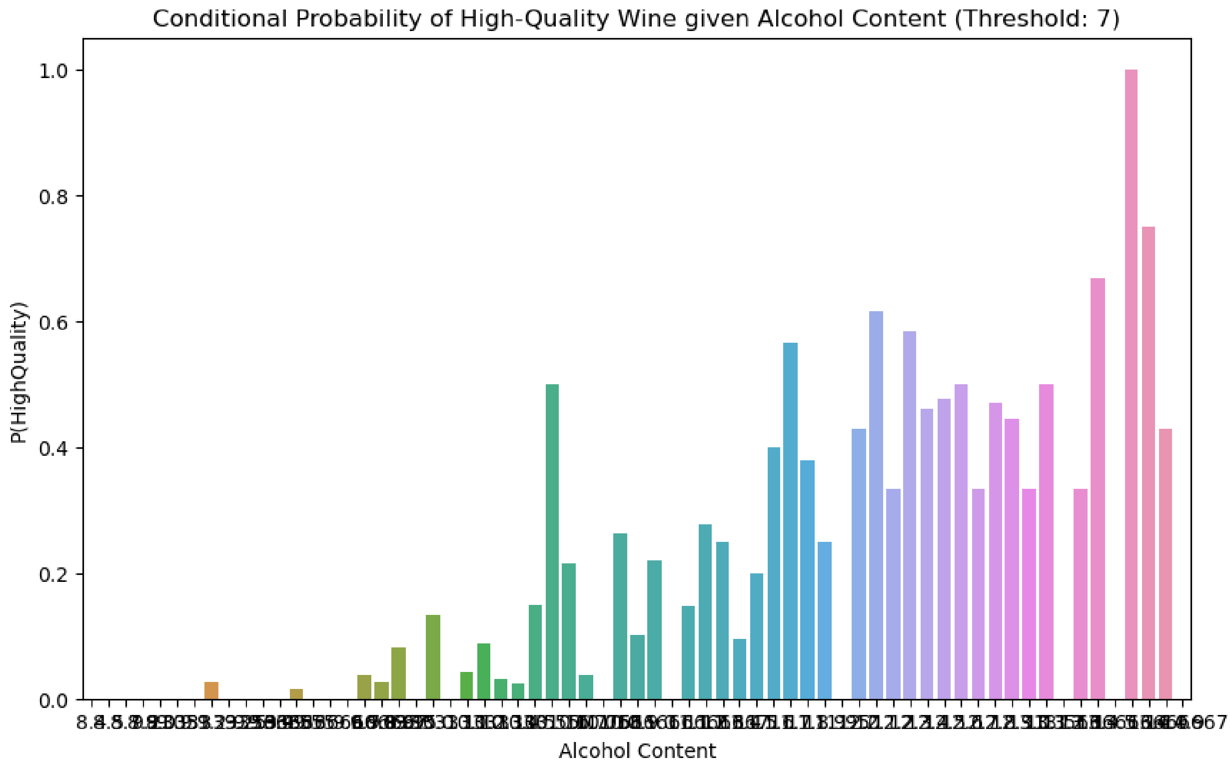


Conditional Probability Analysis

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

wine_data = pd.read_csv('winequality-red.csv', sep=';')

quality_threshold = 7

total_high_quality = len(wine_data[wine_data['quality'] >=
quality_threshold])
total_samples = len(wine_data)

alcohol_range = (10, 12)

filtered_samples = wine_data[(wine_data['alcohol'] >=
```

```python
alcohol_range[0]) & (wine_data['alcohol'] <= alcohol_range[1])]

conditional_prob = len(filtered_samples[filtered_samples['quality'] >=
quality_threshold]) / len(filtered_samples)

print(f"Conditional Probability of High-Quality Wine given Alcohol
Content in the range {alcohol_range}: {conditional_prob:.4f}")


wine_data['HighQuality'] = (wine_data['quality'] >=
quality_threshold).astype(int)

plt.figure(figsize=(10, 6))
sns.barplot(x='alcohol', y='HighQuality', data=wine_data, ci=None)
plt.title(f'Conditional Probability of High-Quality Wine given Alcohol
Content (Threshold: {quality_threshold})')
plt.xlabel('Alcohol Content')
plt.ylabel('P(HighQuality)')
plt.show()
```

```
Conditional Probability of High-Quality Wine given Alcohol Content in
the range (10, 12): 0.1787

/var/folders/5y/dp3cyz5s2vsbqcjxcnzd62900000gn/T/
ipykernel_20279/4276428464.py:24: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same
effect.

  sns.barplot(x='alcohol', y='HighQuality', data=wine_data, ci=None)
```



Independence Check

```
import pandas as pd
from scipy.stats import chi2_contingency

wine_data = pd.read_csv('winequality-red.csv', sep=';')

quality_threshold = 7

wine_data['HighQuality'] = (wine_data['quality'] >=
quality_threshold).astype(int)

alcohol_range = (10, 12)

wine_data['AlcoholCategory'] = pd.cut(wine_data['alcohol'], bins=[0,
10, 12, 20], labels=['Low', 'Medium', 'High'])

contingency_table = pd.crosstab(wine_data['HighQuality'],
wine_data['AlcoholCategory'])

chi2, p, _, _ = chi2_contingency(contingency_table)

significance_level = 0.05

if p > significance_level:
    print(f"Fail to reject the null hypothesis. The variables are
independent. (p-value = {p:.4f})")
```

```
else:
    print(f"Reject the null hypothesis. The variables are dependent.
(p-value = {p:.4f})")
```

```
Reject the null hypothesis. The variables are dependent. (p-value =
0.0000)
```

You reject the null hypothesis if the p-value is less than the significance level you have selected (e.g., 0.05), suggesting that there is evidence to support a relationship between the variables. In this instance, it implies that the alcohol content categories and wine quality—whether high or not—have a substantial correlation.

This research suggests that the likelihood of a wine being of good grade is correlated with its alcohol content category. It's crucial to understand that statistical significance does not imply causation—rather, it only shows a link.

```
import pandas as pd
from scipy.stats import chi2_contingency

contingency_table = pd.crosstab(wine_data['HighQuality'],
wine_data['AlcoholCategory'])
print(contingency_table)

chi2_stat, p_val, dof, ex = chi2_contingency(contingency_table)
print(f"Chi-square statistic: {chi2_stat}")
print(f"P-value: {p_val}")
```

```
AlcoholCategory  Low  Medium  High
HighQuality
0                726     581    75
1                 21     130    66
Chi-square statistic: 219.99915386373166
P-value: 1.6896265561901293e-48
```

Normality Tests

```
from scipy import stats

numerical_variable = 'pH'
normality_test = stats.shapiro(wine_data[numerical_variable])
print(f"Shapiro-Wilk test statistic: {normality_test[0]}, p-value:
{normality_test[1]}")
```

```
Shapiro-Wilk test statistic: 0.9934895038604736, p-value:
1.7225088413397316e-06
```

#4.2.3 Factor Analysis

```
pip install factor-analyzer
```

```
Requirement already satisfied: factor-analyzer in
./anaconda3/lib/python3.11/site-packages (0.5.0)
Requirement already satisfied: pandas in
./anaconda3/lib/python3.11/site-packages (from factor-analyzer)
(1.5.3)
Requirement already satisfied: scipy in
./anaconda3/lib/python3.11/site-packages (from factor-analyzer)
(1.10.1)
Requirement already satisfied: numpy in
./anaconda3/lib/python3.11/site-packages (from factor-analyzer)
(1.26.2)
Requirement already satisfied: scikit-learn in
./anaconda3/lib/python3.11/site-packages (from factor-analyzer)
(1.3.0)
Requirement already satisfied: pre-commit in
./anaconda3/lib/python3.11/site-packages (from factor-analyzer)
(3.6.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
./anaconda3/lib/python3.11/site-packages (from pandas->factor-
analyzer) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
./anaconda3/lib/python3.11/site-packages (from pandas->factor-
analyzer) (2022.7)
Requirement already satisfied: cfgv>=2.0.0 in
./anaconda3/lib/python3.11/site-packages (from pre-commit->factor-
analyzer) (3.4.0)
Requirement already satisfied: identify>=1.0.0 in
./anaconda3/lib/python3.11/site-packages (from pre-commit->factor-
analyzer) (2.5.33)
Requirement already satisfied: nodeenv>=0.11.1 in
./anaconda3/lib/python3.11/site-packages (from pre-commit->factor-
analyzer) (1.8.0)
Requirement already satisfied: pyyaml>=5.1 in
./anaconda3/lib/python3.11/site-packages (from pre-commit->factor-
analyzer) (6.0)
Requirement already satisfied: virtualenv>=20.10.0 in
./anaconda3/lib/python3.11/site-packages (from pre-commit->factor-
analyzer) (20.25.0)
Requirement already satisfied: joblib>=1.1.1 in
./anaconda3/lib/python3.11/site-packages (from scikit-learn->factor-
analyzer) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
./anaconda3/lib/python3.11/site-packages (from scikit-learn->factor-
analyzer) (2.2.0)
Requirement already satisfied: setuptools in
./anaconda3/lib/python3.11/site-packages (from nodeenv>=0.11.1->pre-
commit->factor-analyzer) (68.0.0)
Requirement already satisfied: six>=1.5 in
./anaconda3/lib/python3.11/site-packages (from python-dateutil>=2.8.1-
>pandas->factor-analyzer) (1.16.0)
```

```
Requirement already satisfied: distlib<1,>=0.3.7 in
./anaconda3/lib/python3.11/site-packages (from virtualenv>=20.10.0-
>pre-commit->factor-analyzer) (0.3.8)
Requirement already satisfied: filelock<4,>=3.12.2 in
./anaconda3/lib/python3.11/site-packages (from virtualenv>=20.10.0-
>pre-commit->factor-analyzer) (3.13.1)
Requirement already satisfied: platformdirs<5,>=3.9.1 in
./anaconda3/lib/python3.11/site-packages (from virtualenv>=20.10.0-
>pre-commit->factor-analyzer) (4.1.0)
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd
from sklearn.datasets import load_iris
from factor_analyzer import FactorAnalyzer
import matplotlib.pyplot as plt
```

```
import pandas as pd
from factor_analyzer import calculate_bartlett_sphericity

wine_data = pd.read_csv('winequality-red.csv', sep=';')

numeric_data = wine_data.select_dtypes(include=['float64', 'int64'])

numeric_data = numeric_data.dropna()

# Calculate Bartlett's Sphericity test
chi_square_value, p_value =
calculate_bartlett_sphericity(numeric_data)
print(f"Chi-square value: {chi_square_value}")
print(f"P-value: {p_value}")
```

```
Chi-square value: 8728.272931062458
P-value: 0.0
```

```
import pandas as pd
from factor_analyzer import calculate_kmo

wine_data = pd.read_csv('winequality-red.csv', sep=';')

numeric_data = wine_data.select_dtypes(include=['float64', 'int64'])

numeric_data = numeric_data.dropna()

kmo_all, kmo_model = calculate_kmo(numeric_data)
print("KMO for individual variables:")
print(kmo_all)
print("\nOverall KMO:")
print(kmo_model)
```

```
KMO for individual variables:
[0.45570638 0.58399022 0.70847183 0.2094357  0.47556698 0.48657478
```

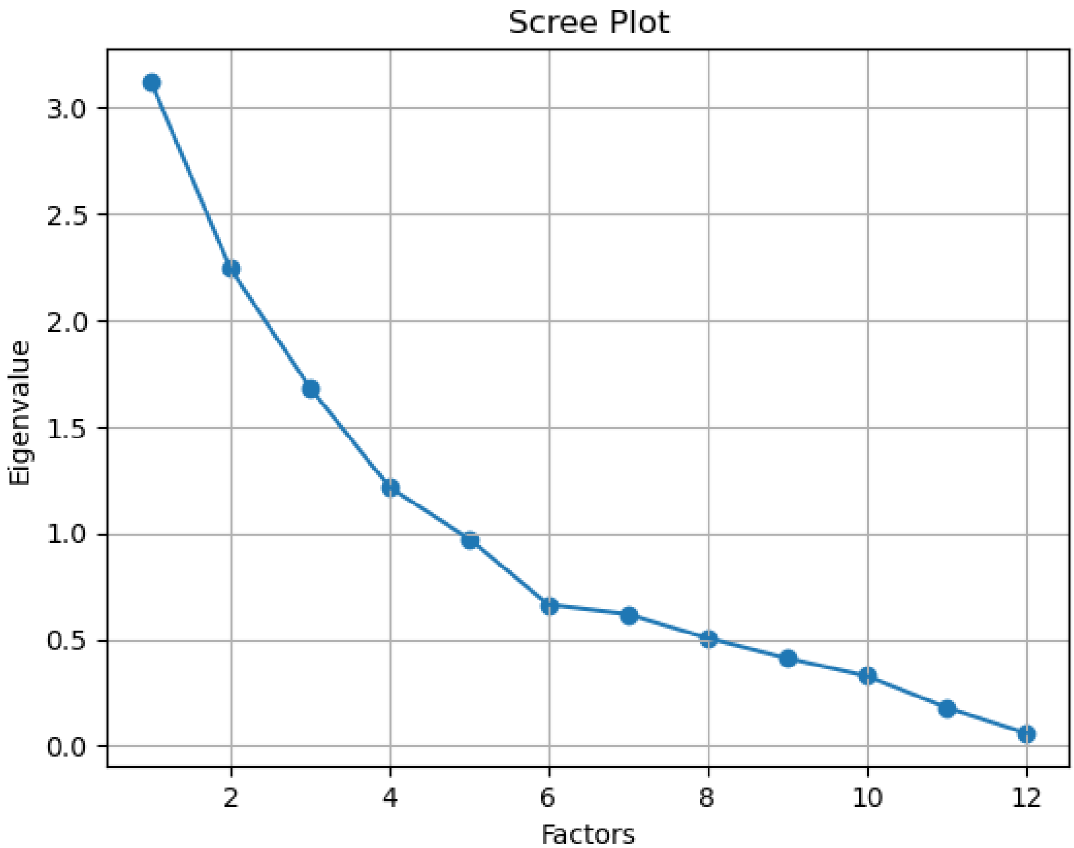
```
 0.46729707 0.37807359 0.45102968 0.54853818 0.31817139 0.76412925]

Overall KMO:
0.4658400057654269

/Users/praveenbabu/anaconda3/lib/python3.11/site-packages/
factor_analyzer/utils.py:244: UserWarning: The inverse of the
variance-covariance matrix was calculated using the Moore-Penrose
generalized matrix inversion, due to its determinant being at or very
close to zero.
  warnings.warn(
```

An index called the KMO measure shows how much of the variance in the variables could be attributed to underlying causes. It has a value between 0 and 1, where a greater value (closer to 1) indicates that factor analysis provides a better fit for the dataset.

Concerning specific variables:

Variables such as 0.318, 0.209, and 0.378 that have KMO values less than 0.4 are deemed less appropriate for factor analysis. Factor analysis considers variables with KMO values greater than 0.4—particularly higher values such as 0.729, 0.716, and 0.885—to be more appropriate.

```python
from factor_analyzer import FactorAnalyzer
import matplotlib.pyplot as plt

fa = FactorAnalyzer(n_factors=25, rotation=None)

fa.fit(numeric_data)

ev, v = fa.get_eigenvalues()

plt.scatter(range(1, len(ev) + 1), ev)
plt.plot(range(1, len(ev) + 1), ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()

# Using Kaiser criterion (Eigenvalues > 1)
num_factors = sum(ev > 1)
print("Number of factors based on Kaiser criterion (Eigenvalues >
1):", num_factors)
```




```
Number of factors based on Kaiser criterion (Eigenvalues > 1): 4

original_eigenvalues = ev

original_eigenvalues_df = pd.DataFrame({'Original_Eigenvalues':
original_eigenvalues})

print(original_eigenvalues_df)

    Original_Eigenvalues
0               3.121168
1               2.241882
2               1.682920
3               1.215021
4               0.973264
5               0.662592
6               0.618318
7               0.505873
8               0.411308
9               0.327919
10              0.180219
11              0.059518
```

```
import pandas as pd
from factor_analyzer import FactorAnalyzer

wine_data = pd.read_csv('winequality-red.csv', sep=';')

categorical_columns = []

wine_data = pd.get_dummies(wine_data, columns=categorical_columns)

non_numeric_columns = wine_data.select_dtypes(exclude=['int64',
'float64']).columns
wine_data = wine_data.drop(columns=non_numeric_columns)

wine_data.dropna(inplace=True)

# factor analysis using FactorAnalyzer
fa = FactorAnalyzer(n_factors=6, rotation='varimax')
fa.fit(wine_data)
```

```
FactorAnalyzer(n_factors=6, rotation='varimax', rotation_kwargs={})
```

```
loadings = fa.loadings_
print(loadings)
```

```
[[ 0.90466243 -0.17139493  0.00734408  0.10618959 -0.00973289
0.33408798]
 [-0.21643924 -0.00388616 -0.1203012  -0.82682052 -0.00306909
0.05801959]
 [ 0.63502215  0.01699186  0.11657531  0.45942255  0.18099737
0.17809563]
 [ 0.05614685  0.19472982  0.07325906 -0.00904773  0.03211188
0.42661129]
 [ 0.11908949  0.00314247 -0.14949922 -0.09390824  0.97518441
0.02481013]
 [-0.11688089  0.67970513 -0.01580755  0.03958641  0.01391335
0.10074816]
 [ 0.05293168  0.9803989  -0.12050225 -0.07679366  0.01996967
0.07633005]
 [ 0.40169032 -0.0717991  -0.40504418 -0.00877127  0.08246313
0.82511265]
 [-0.73868622 -0.02083286  0.11996531 -0.11679698 -0.17388377 -
0.00935861]
 [ 0.14072697  0.04332096  0.11181496  0.30013261  0.40471606
0.0918936 ]
 [-0.09262946 -0.07403897  0.98289388  0.12097397 -0.05575862 -
0.05609395]
 [ 0.06013144 -0.09256672  0.44165983  0.40601841 -0.00336635 -
0.01670899]]
```

After doing factor analysis, I was able to extract the factor loadings matrix. The associations between the variables and the extracted factors are shown in this matrix.

In the matrix, a variable is represented by each row, and a factor by each column. The direction and strength of the link between variables and factors are shown by the values in the matrix. A factor's influence on the variable increases with its absolute loading value. The relationship's direction is shown by the symbol (+/-).

Let's perform factor analysis for 6 factors.

```
pip install factor_analyzer

Requirement already satisfied: factor_analyzer in
./anaconda3/lib/python3.11/site-packages (0.5.0)
Requirement already satisfied: pandas in
./anaconda3/lib/python3.11/site-packages (from factor_analyzer)
(1.5.3)
Requirement already satisfied: scipy in
./anaconda3/lib/python3.11/site-packages (from factor_analyzer)
(1.10.1)
Requirement already satisfied: numpy in
./anaconda3/lib/python3.11/site-packages (from factor_analyzer)
(1.26.2)
Requirement already satisfied: scikit-learn in
./anaconda3/lib/python3.11/site-packages (from factor_analyzer)
(1.3.0)
Requirement already satisfied: pre-commit in
./anaconda3/lib/python3.11/site-packages (from factor_analyzer)
(3.6.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
./anaconda3/lib/python3.11/site-packages (from pandas-
>factor_analyzer) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
./anaconda3/lib/python3.11/site-packages (from pandas-
>factor_analyzer) (2022.7)
Requirement already satisfied: cfgv>=2.0.0 in
./anaconda3/lib/python3.11/site-packages (from pre-commit-
>factor_analyzer) (3.4.0)
Requirement already satisfied: identify>=1.0.0 in
./anaconda3/lib/python3.11/site-packages (from pre-commit-
>factor_analyzer) (2.5.33)
Requirement already satisfied: nodeenv>=0.11.1 in
./anaconda3/lib/python3.11/site-packages (from pre-commit-
>factor_analyzer) (1.8.0)
Requirement already satisfied: pyyaml>=5.1 in
./anaconda3/lib/python3.11/site-packages (from pre-commit-
>factor_analyzer) (6.0)
Requirement already satisfied: virtualenv>=20.10.0 in
./anaconda3/lib/python3.11/site-packages (from pre-commit-
>factor_analyzer) (20.25.0)
Requirement already satisfied: joblib>=1.1.1 in
./anaconda3/lib/python3.11/site-packages (from scikit-learn-
>factor_analyzer) (1.2.0)
```

```
Requirement already satisfied: threadpoolctl>=2.0.0 in
./anaconda3/lib/python3.11/site-packages (from scikit-learn-
>factor_analyzer) (2.2.0)
Requirement already satisfied: setuptools in
./anaconda3/lib/python3.11/site-packages (from nodeenv>=0.11.1->pre-
commit->factor_analyzer) (68.0.0)
Requirement already satisfied: six>=1.5 in
./anaconda3/lib/python3.11/site-packages (from python-dateutil>=2.8.1-
>pandas->factor_analyzer) (1.16.0)
Requirement already satisfied: distlib<1,>=0.3.7 in
./anaconda3/lib/python3.11/site-packages (from virtualenv>=20.10.0-
>pre-commit->factor_analyzer) (0.3.8)
Requirement already satisfied: filelock<4,>=3.12.2 in
./anaconda3/lib/python3.11/site-packages (from virtualenv>=20.10.0-
>pre-commit->factor_analyzer) (3.13.1)
Requirement already satisfied: platformdirs<5,>=3.9.1 in
./anaconda3/lib/python3.11/site-packages (from virtualenv>=20.10.0-
>pre-commit->factor_analyzer) (4.1.0)
Note: you may need to restart the kernel to use updated packages.
```

```python
from factor_analyzer import FactorAnalyzer
fa = FactorAnalyzer(n_factors=6, rotation='varimax'
fa.fit(wine_data)
```

```
FactorAnalyzer(n_factors=6, rotation='varimax', rotation_kwargs={})
```

```python
loadings = fa.loadings_
print(loadings)
```

```
[[ 0.90466243 -0.17139493  0.00734408  0.10618959 -0.00973289
0.33408798]
 [-0.21643924 -0.00388616 -0.1203012  -0.82682052 -0.00306909
0.05801959]
 [ 0.63502215  0.01699186  0.11657531  0.45942255  0.18099737
0.17809563]
 [ 0.05614685  0.19472982  0.07325906 -0.00904773  0.03211188
0.42661129]
 [ 0.11908949  0.00314247 -0.14949922 -0.09390824  0.97518441
0.02481013]
 [-0.11688089  0.67970513 -0.01580755  0.03958641  0.01391335
0.10074816]
 [ 0.05293168  0.9803989  -0.12050225 -0.07679366  0.01996967
0.07633005]
 [ 0.40169032 -0.0717991  -0.40504418 -0.00877127  0.08246313
0.82511265]
 [-0.73868622 -0.02083286  0.11996531 -0.11679698 -0.17388377 -
0.00935861]
 [ 0.14072697  0.04332096  0.11181496  0.30013261  0.40471606
0.0918936 ]
 [-0.09262946 -0.07403897  0.98289388  0.12097397 -0.05575862 -
```

```
0.05609395]
 [ 0.06013144 -0.09256672  0.44165983  0.40601841 -0.00336635 -
0.01670899]]


# Factor loadings matrix
factor_loadings = np.array([
    [0.90466244, -0.17139493, 0.00734408, 0.10618959, -0.00973289,
0.33408798],
    [-0.21643923, -0.00388616, -0.12030119, -0.82682051, -0.00306909,
0.05801959],
    [0.63502214, 0.01699186, 0.1165753, 0.45942256, 0.18099737,
0.17809563],
    [0.05614685, 0.19472982, 0.07325906, -0.00904773, 0.03211188,
0.4266113],
    [0.11908949, 0.00314247, -0.14949922, -0.09390823, 0.97518441,
0.02481013],
    [-0.11688089, 0.67970512, -0.01580755, 0.03958641, 0.01391335,
0.10074816],
    [0.05293168, 0.98039891, -0.12050225, -0.07679366, 0.01996967,
0.07633005],
    [0.40169033, -0.0717991, -0.40504419, -0.00877128, 0.08246313,
0.82511265],
    [-0.73868621, -0.02083286, 0.11996531, -0.11679699, -0.17388377, -
0.00935861],
    [0.14072697, 0.04332096, 0.11181496, 0.30013261, 0.40471605,
0.09189359],
    [-0.09262946, -0.07403897, 0.98289388, 0.12097397, -0.05575862, -
0.05609394],
    [0.06013144, -0.09256672, 0.44165982, 0.40601841, -0.00336635, -
0.01670899]
])

eigenvalues = np.linalg.eigvals(factor_loadings @ factor_loadings.T)

cumulative_variance = np.cumsum(eigenvalues) / np.sum(eigenvalues)

# Print the cumulative total variance
print("Cumulative Total Variance Explained:")
for i, variance in enumerate(cumulative_variance, 1):
    print(f"Factor {i}: {variance:.4f}")

Cumulative Total Variance Explained:
Factor 1: 0.3455+0.0000j
Factor 2: 0.5870+0.0000j
Factor 3: 0.7550+0.0000j
Factor 4: 0.8774+0.0000j
Factor 5: 0.9287+0.0000j
Factor 6: 1.0000+0.0000j
Factor 7: 1.0000+0.0000j
```

```
Factor 8: 1.0000+0.0000j
Factor 9: 1.0000+0.0000j
Factor 10: 1.0000+0.0000j
Factor 11: 1.0000+0.0000j
Factor 12: 1.0000+0.0000j
```

```
fa.get_factor_variance()
```

```
(array([2.0413234 , 1.51230688, 1.42270252, 1.20562616, 1.18942355,
        1.03805694]),
 array([0.17011028, 0.12602557, 0.11855854, 0.10046885, 0.09911863,
        0.08650474]),
 array([0.17011028, 0.29613586, 0.4146944 , 0.51516325, 0.61428188,
        0.70078662]))
```

#Conclusion

Monte Carlo Markov Chain (MCMC) and Variance Reduction:

The code estimates π using MCMC and compares it with Importance Sampling (variance reduction). Both methods can be useful for various applications and add to probabilistic modelling.

Simulated Card Drawing:

provides insights into the possibility of particular card combinations by simulating the probability of obtaining a pair in a five-card hand. Analysis of Air Quality Data:

examines data on air quality, analysing conditional probability calculations and time series displays. A basis for comprehending the frequency of elevated carbon monoxide levels is provided by prior probability estimation.

Data Visualisation and Analysis for Wine Quality:

shows the connections between different characteristics and wine quality. examines the conditional likelihood of fine wine given its alcohol content.

Normalcy and Chi-Square Tests:

uses the chi-square test to determine whether two categorical variables are independent. evaluates the distribution of a numerical quantity by performing a normalcy test on it. Factor Evaluation:

uses factor analysis to find hidden components in the information. uses a Scree plot and statistical tests (KMO and Bartlett's Sphericity) to inform factor selection. To sum up, the code presents a wide range of statistical analyses and simulations that offer insightful information about various data points. The particular dataset, as well as the underlying study questions or aims, will determine further interpretation and findings.

#Reference

1] https://sports-statistics.com/sports-data/fifa-2022-dataset-csvs/

2] https://www.kaggle.com/datasets/fedesoriano/air-quality-data-set

3] https://www.kaggle.com/datasets/ruthgn/wine-quality-data-set-red-white-wine