EXPERIMENT 1: Exploration of WEKA (Waikato Environment for Knowledge Analysis) tool for Regression task.

WEKA (Waikato Environment for Knowledge Analysis) is a powerful open-source software suite for machine learning and data mining tasks.

Getting Started with WEKA

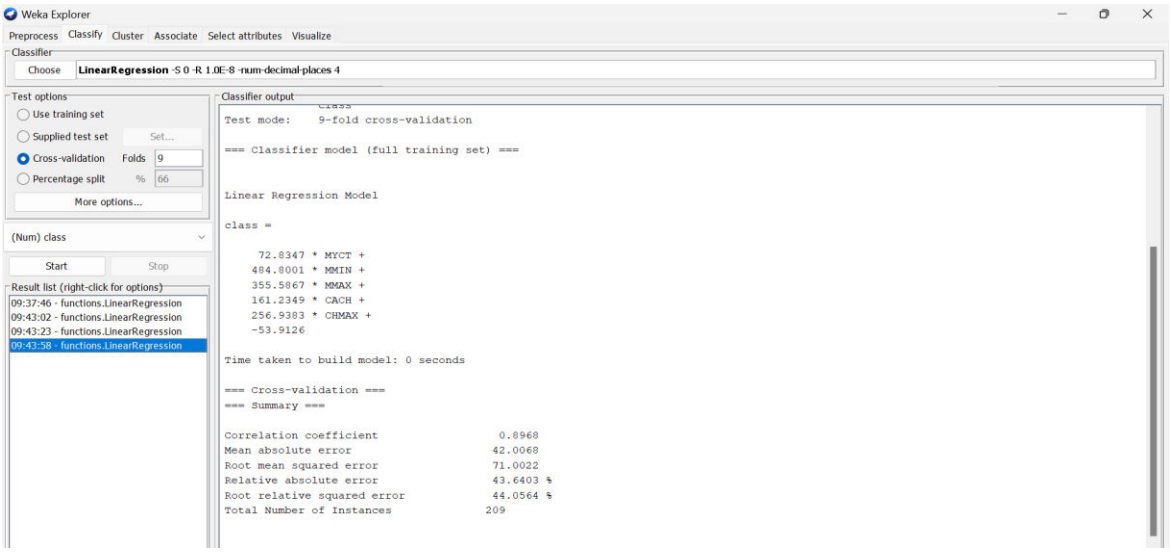
- **Download and Install:** First, download WEKA from the official website. Installation is straightforward; follow the instructions for your operating system.
- **Launch WEKA:** Open WEKA GUI and you'll see the main interface with options such as Explorer, Experimenter, Knowledge Flow, and More.

EXPERIMENT 2: Exploration of WEKA tool for Classification task.

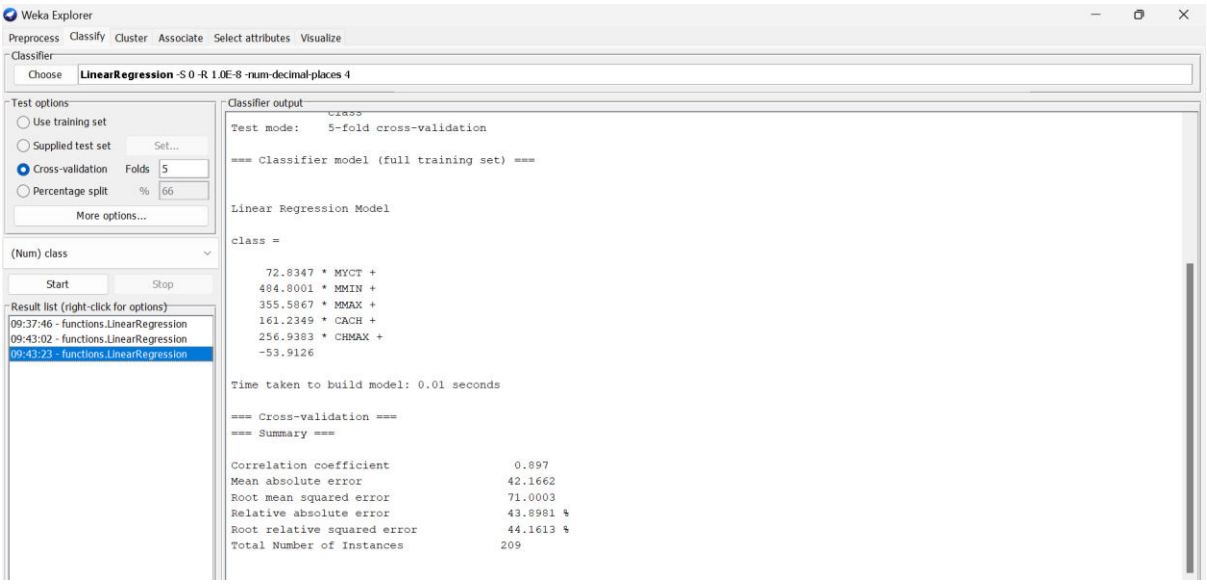
Database: CPU

Classifier: Function .Linear Regression

FOLDS: 10



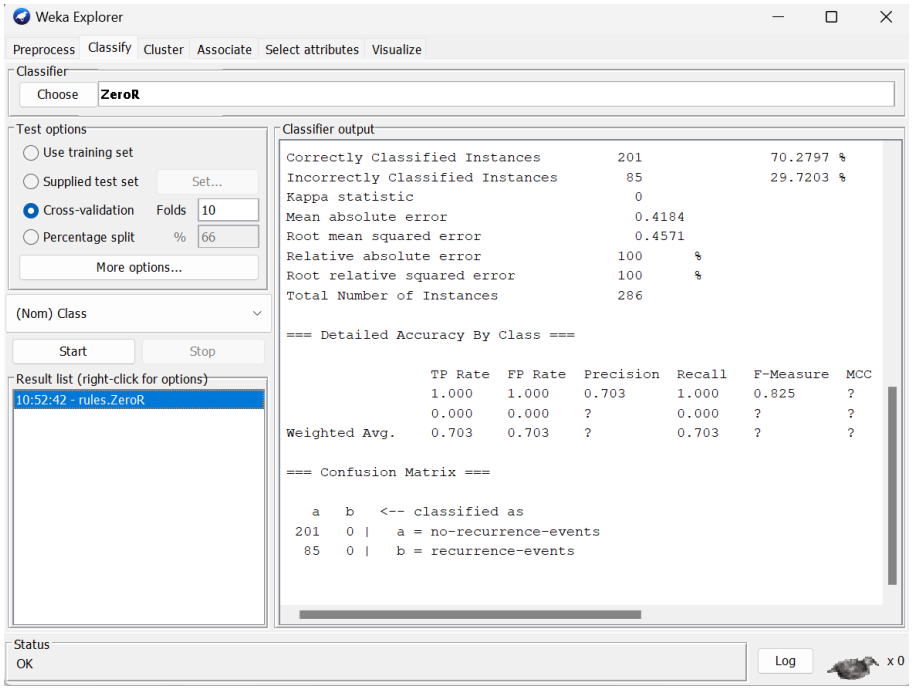
Folds: 5



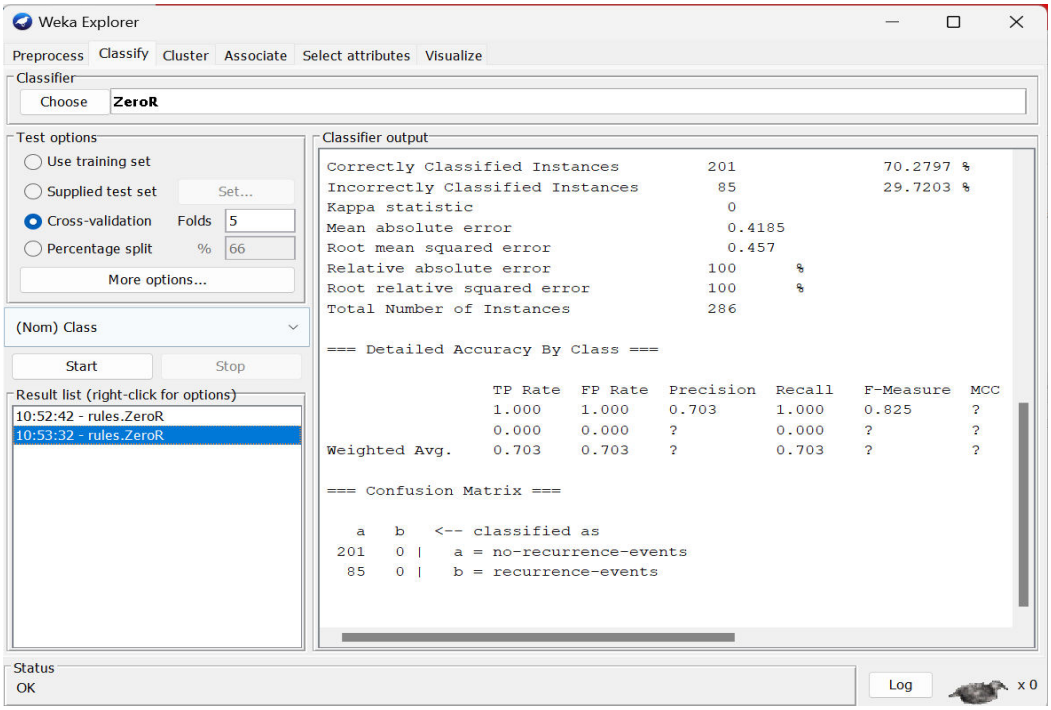
Database: BREAST CANCER EXPLORATION

Classifier: NaiveBayes

Folds: 10



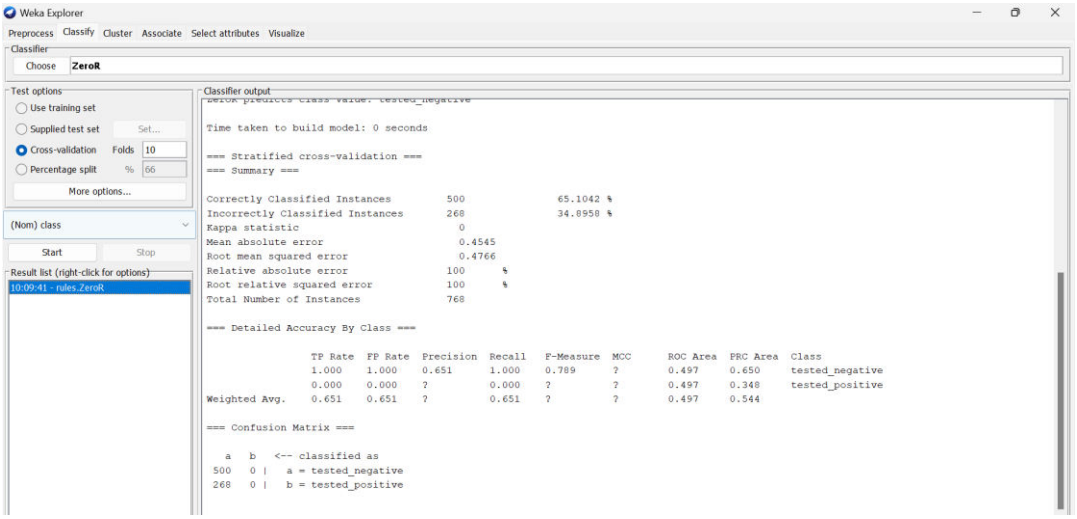
Folds: 5



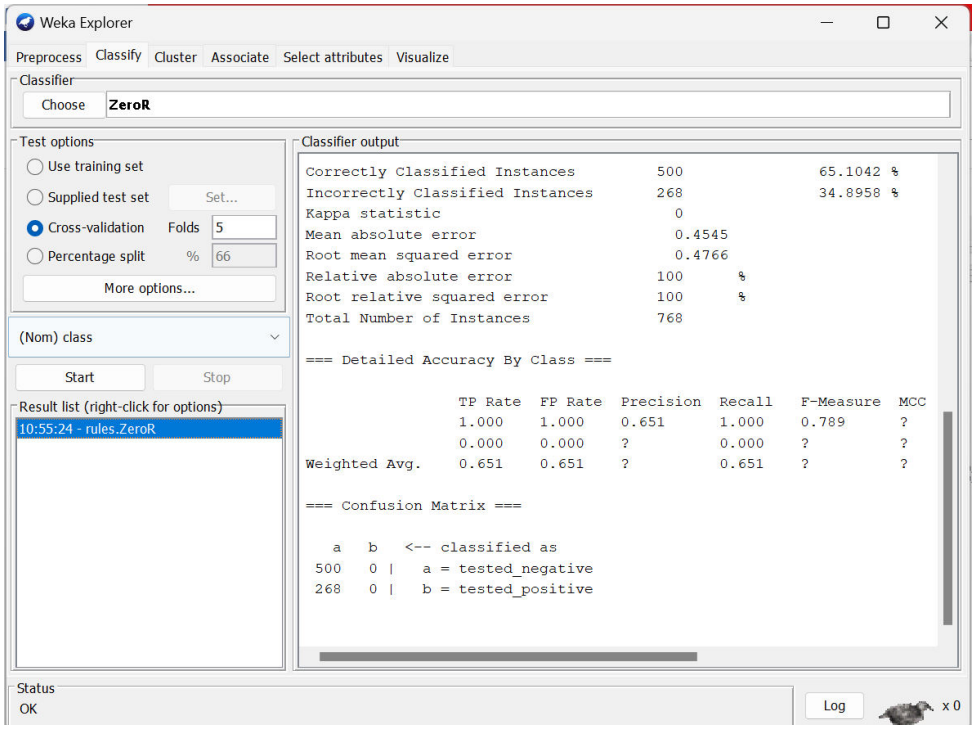
Database: Diabetes.

Classifier: ZeroR

Folds: 10



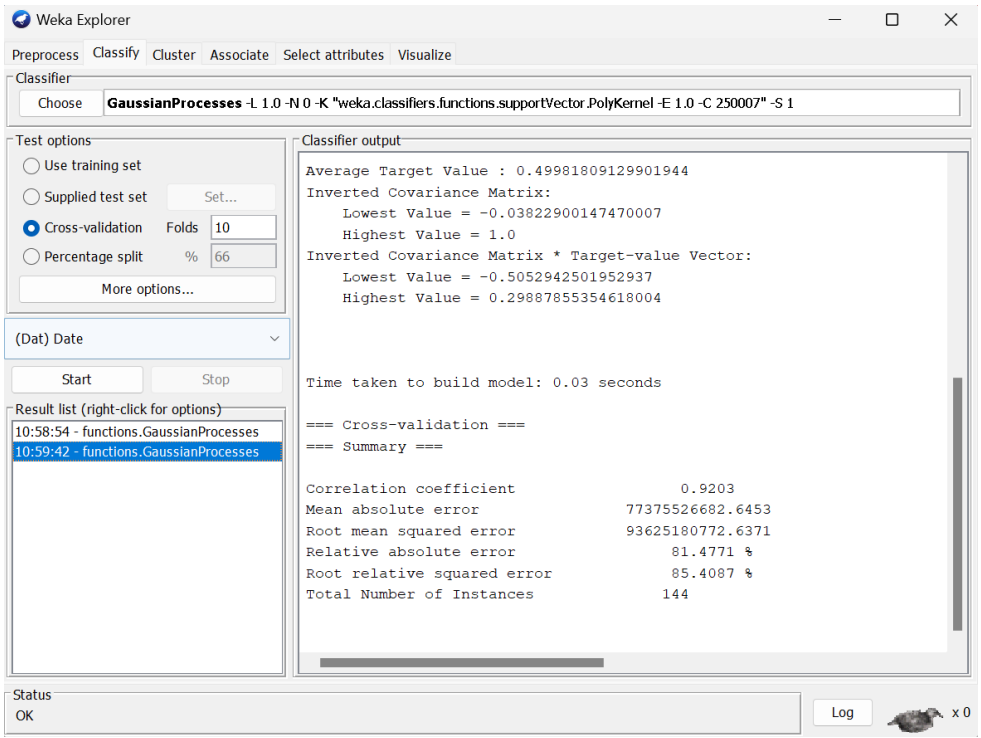
Folds:5



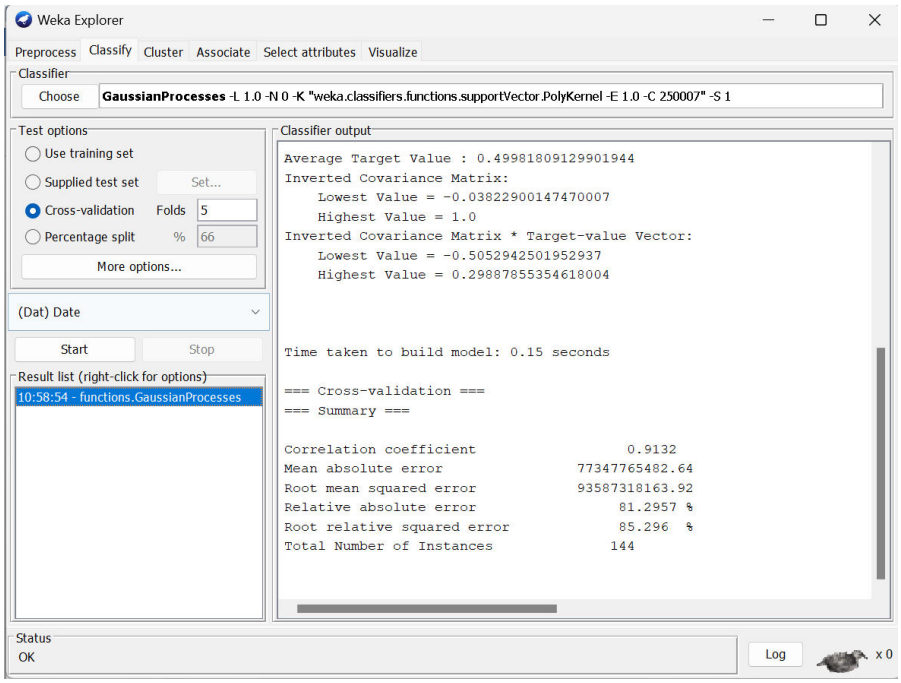
Database: airline

Classifier: GaussianProcess

Folds: 10



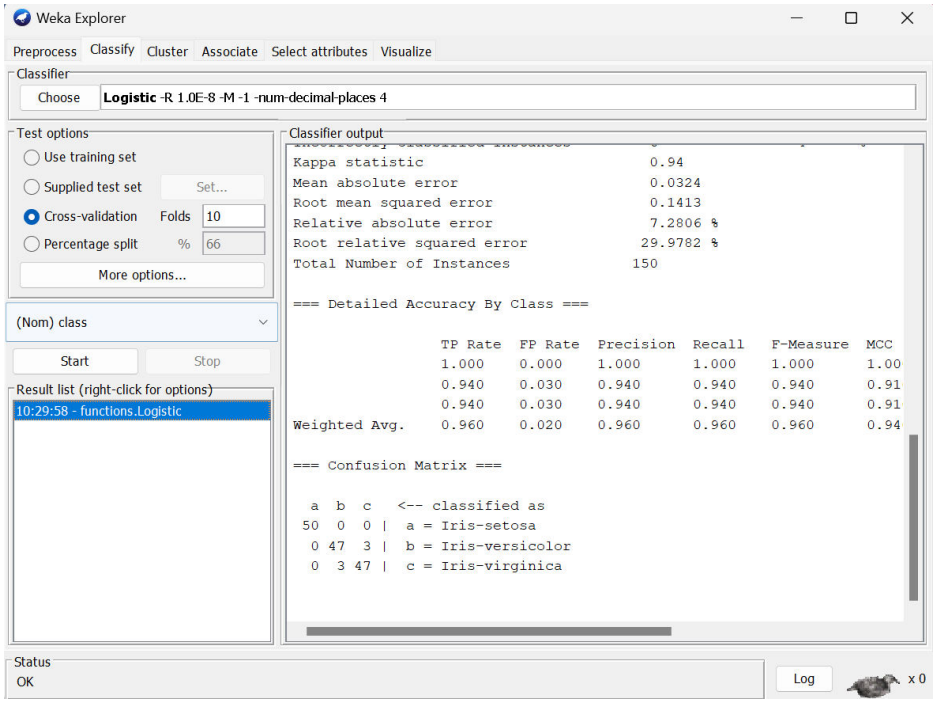
Folds : 5



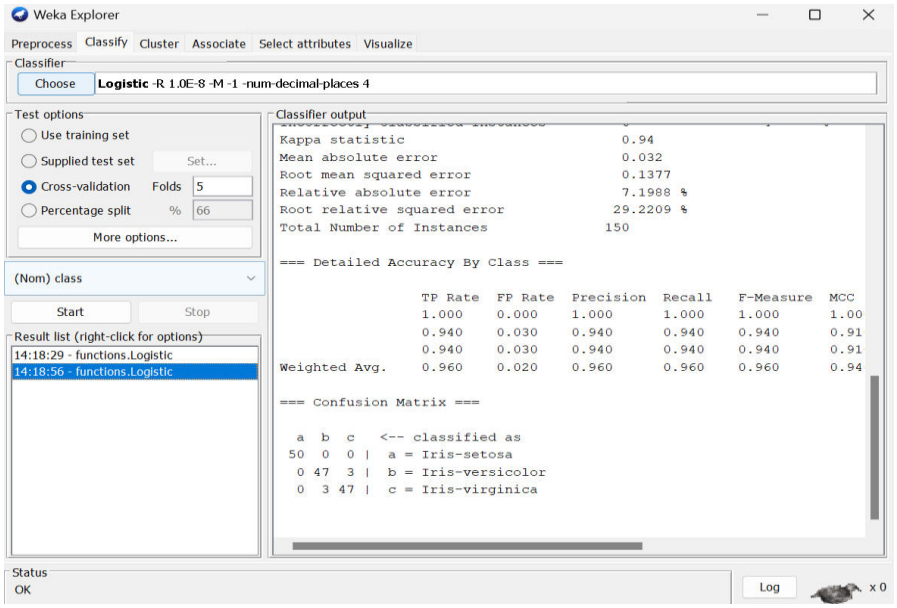
Database: Iris. 2D

Classifier: Functions.Logistic

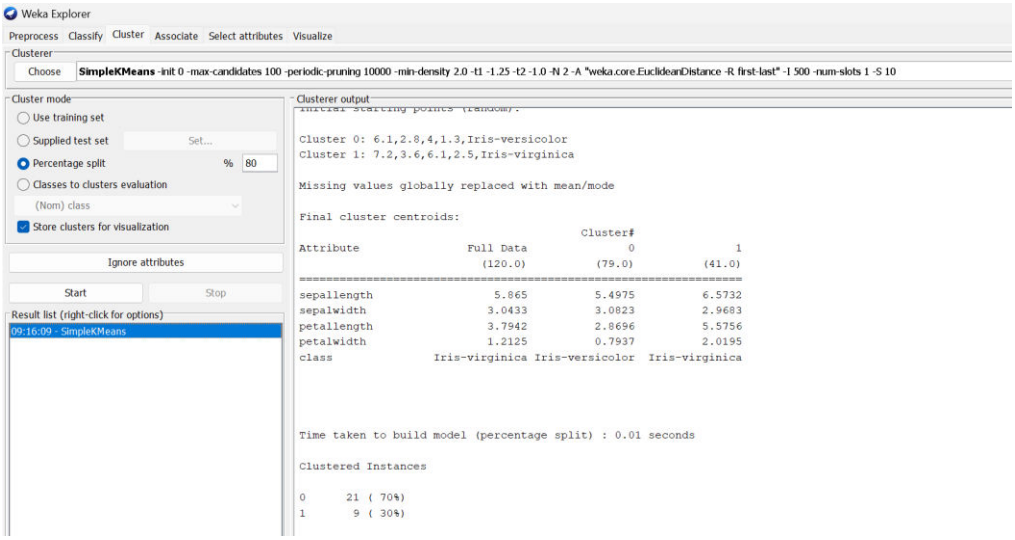
Folds: 10



Folds:5


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
Logistic -R 1.0E-8 -M -1 -num-decimal-places 4
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
5
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
14:18:29 - functions.Logistic
14:18:56 - functions.Logistic
Classifier output
Kappa statistic 0.94
Mean absolute error 0.032
Root mean squared error 0.1377
Relative absolute error 7.1988 %
Root relative squared error 29.2209 %
Total Number of Instances 150
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC
1.000 0.000 1.000 1.000 1.000 1.00
0.940 0.030 0.940 0.940 0.940 0.91
0.940 0.030 0.940 0.940 0.940 0.91
Weighted Avg. 0.960 0.020 0.960 0.960 0.960 0.94
=== Confusion Matrix ===
a b c <-- classified as
50 0 0 | a = Iris-setosa
0 47 3 | b = Iris-versicolor
0 3 47 | c = Iris-virginica
Status
OK
Log
x 0

Task 3



IRIS DATASET:

The Iris dataset is a classic dataset used in machine learning and statistics, and it's often used for testing and demonstrating algorithms. It contains measurements for three species of iris flowers: Setosa, Versicolor, and Virginica. Each flower is described by four features: sepal length, sepal width, petal length, and petal width

CLUSTERING:

Clustering is an unsupervised machine learning technique that involves grouping similar data points together without any prior labels or categories. It's like sorting objects into different boxes based on their similarities. The goal is to discover hidden patterns or structures within the data.

Key characteristics of clustering:

Unsupervised: No predefined labels or target variable.

Similarity: Data points within a cluster are similar to each other.

Dissimilarity: Data points in different clusters are dissimilar.

Types of Clustering

1. K-Means Clustering

- **Description:** Partitions data into kkk clusters, where each cluster is represented by its centroid (the mean of the points in the cluster).
- **Strengths:** Simple and efficient for large datasets.
- **Weaknesses:** Requires specifying kkk in advance; sensitive to initial cluster centers and outliers.

2. Hierarchical Clustering

- **Description:** Creates a hierarchy of clusters either through a bottom-up approach (agglomerative) or a top-down approach (divisive).
- **Strengths:** Does not require specifying the number of clusters in advance; produces a dendrogram that shows the merging or splitting of clusters.
- **Weaknesses:** Can be computationally intensive, especially for large datasets.

EXPERIMENT: 4

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('median_alpha_power.csv')
```

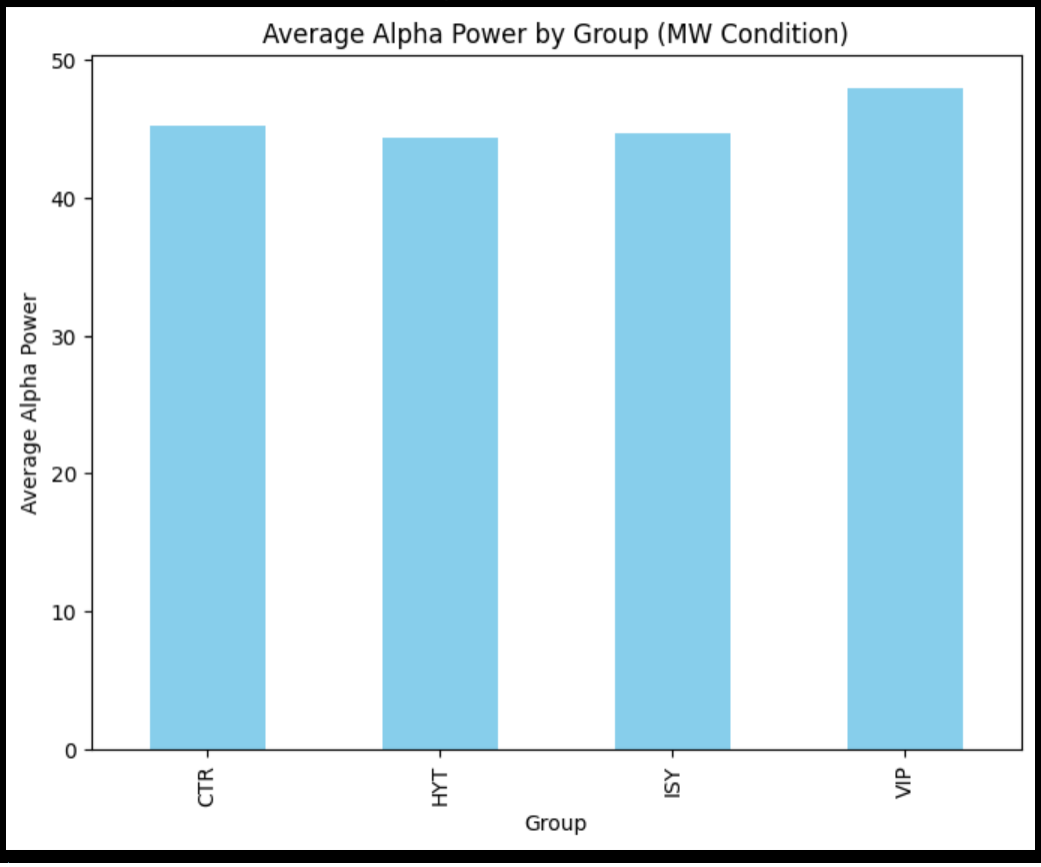
[1]

Q1. How does the average alpha power differ between different groups

```
mean_alpha_power = df[df['Condition'] == 'MW'].groupby('Group')['Alpha'].mean()

# Plot the bar chart
plt.figure(figsize=(8, 6))
mean_alpha_power.plot(kind='bar', color='skyblue')
plt.title('Average Alpha Power by Group (MW Condition)')
plt.ylabel('Average Alpha Power')
plt.xlabel('Group')
plt.show()
```

[2]

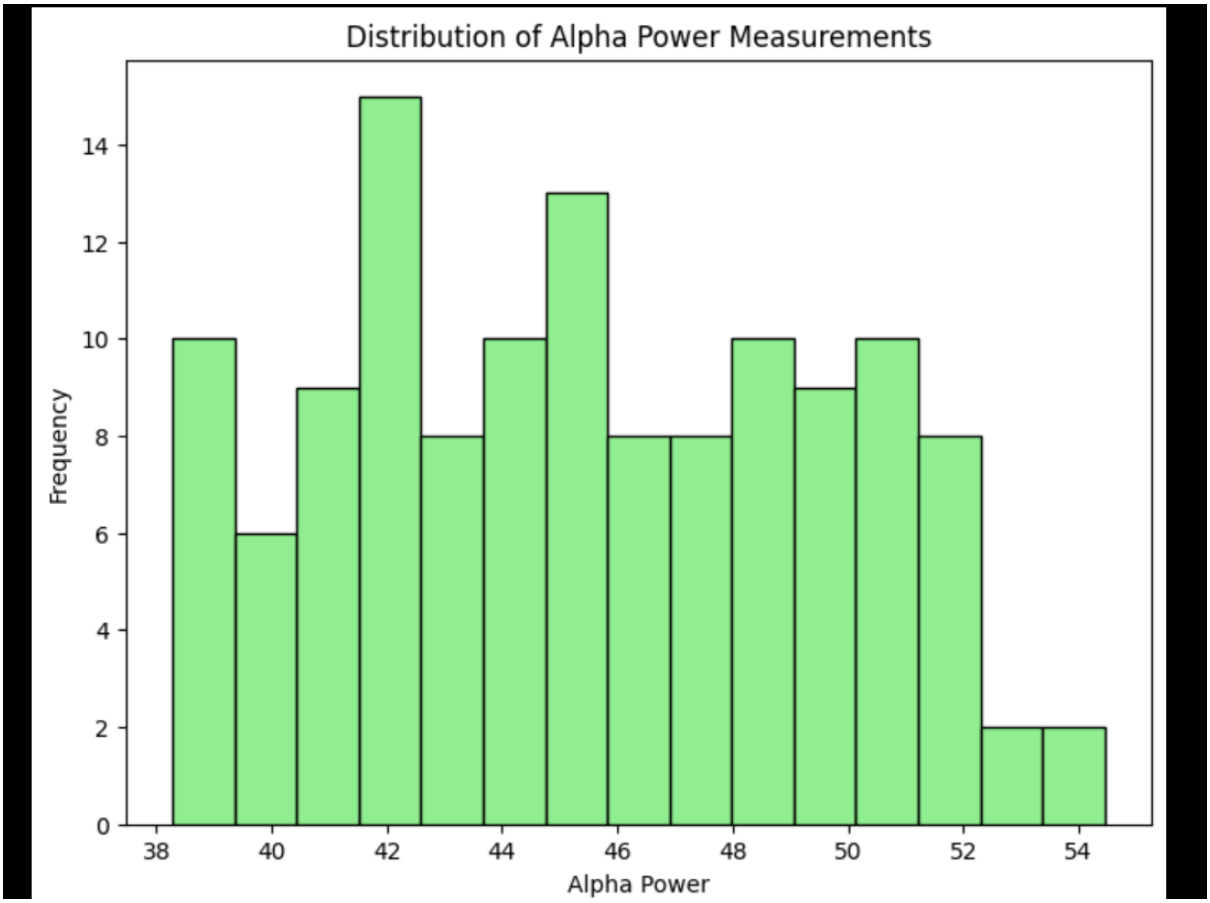
+ Code + Markdown | Run All | Clear All Outputs | Outline ...



Q2 What is the distribution of alpha power measurements across all subjects?

```
plt.figure(figsize=(8, 6))
plt.hist(df['Alpha'], bins=15, color='lightgreen', edgecolor='black')
plt.title('Distribution of Alpha Power Measurements')
plt.xlabel('Alpha Power')
plt.ylabel('Frequency')
plt.show()
```

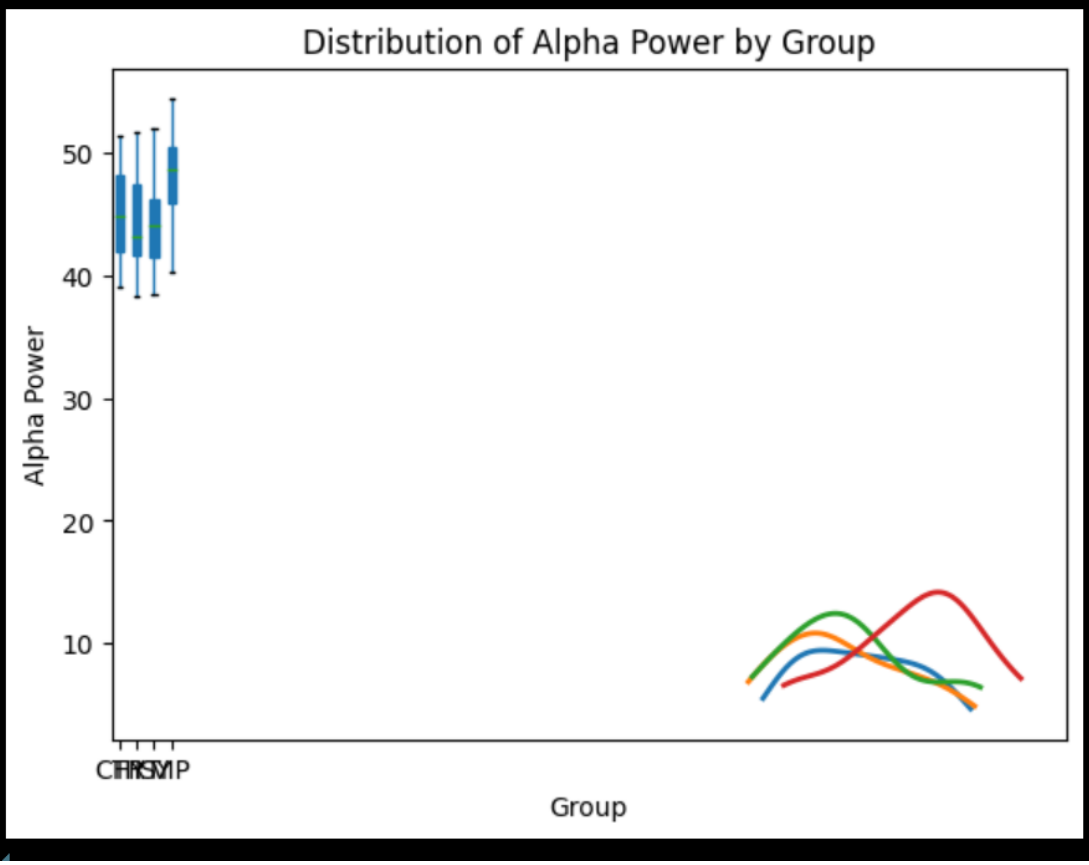
[3]



Q3 How does the distribution of alpha power measurements compare between different groups?

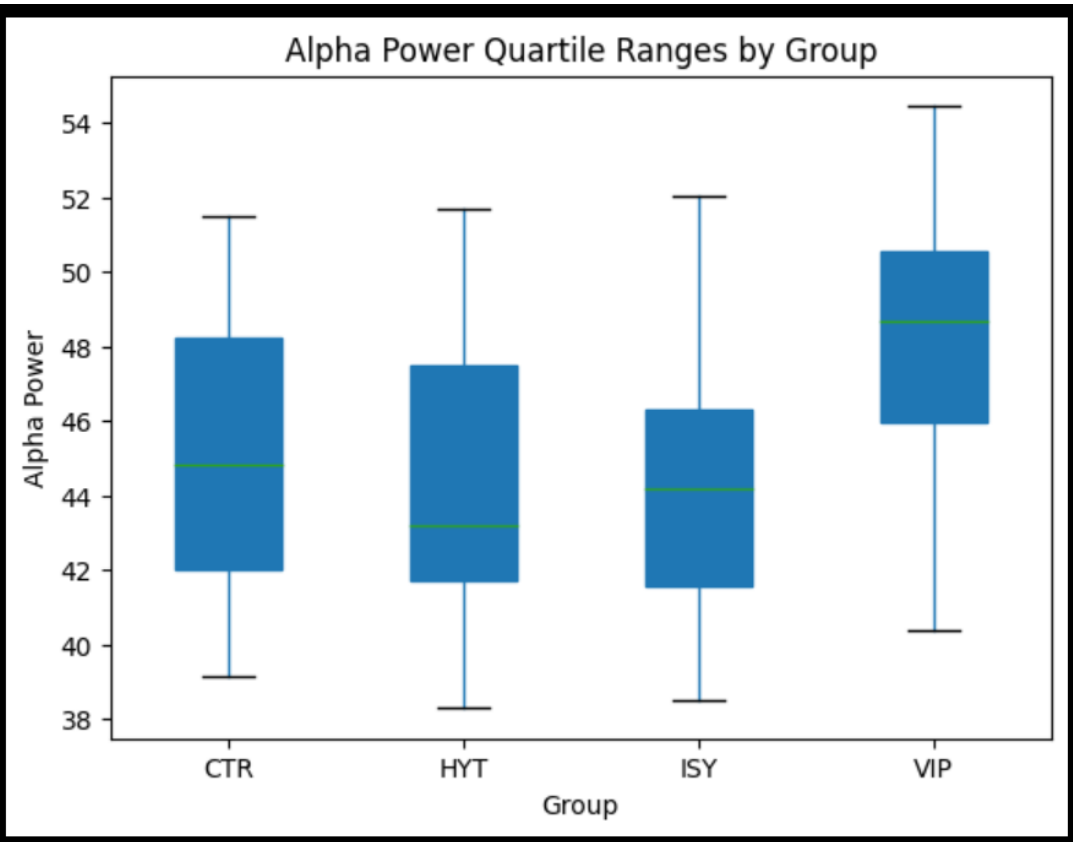
```
from scipy.stats import gaussian_kde
import numpy as np

plt.figure(figsize=(8, 6))

# Create a boxplot
df.boxplot(column='Alpha', by='Group', grid=False, patch_artist=True)

# Overlay KDE for each group
for group in df['Group'].unique():
    subset = df[df['Group'] == group]['Alpha']
    density = gaussian_kde(subset)
    x_vals = np.linspace(subset.min(), subset.max(), 100)
    plt.plot(x_vals, density(x_vals) * 100 + df['Group'].unique().tolist().index(group) + 1, lw=2)

plt.title('Distribution of Alpha Power by Group')
plt.suptitle('')
plt.xlabel('Group')
plt.ylabel('Alpha Power')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



Q4.What are the quartile ranges of alpha power measurements for each group?

```
plt.figure(figsize=(8, 6))
df.boxplot(column='Alpha', by='Group', grid=False, patch_artist=True)
plt.title('Alpha Power Quartile Ranges by Group')
plt.suptitle('')
plt.xlabel('Group')
plt.ylabel('Alpha Power')
plt.show()
```



EXPERIMENT-5

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('matches.csv')
```

1) Evaluate the frequency of most men of the match awards.

```
man_of_match_freq = df['player_of_match'].value_counts()
print(man_of_match_freq)
```

```
player_of_match
AB de Villiers       25
CH Gayle             22
RG Sharma            19
DA Warner            18
V Kohli              18
                     ..
NV Ojha               1
KV Sharma             1
Washington Sundar     1
PD Collingwood        1
Shahbaz Ahmed         1
Name: count, Length: 291, dtype: int64
```

2) Evaluate the top 10 players with most men of the match award

```
top_10_players = man_of_match_freq.head(10)
print(top_10_players)
```

```
player_of_match
AB de Villiers    25
CH Gayle          22
RG Sharma         19
DA Warner         18
V Kohli           18
MS Dhoni          17
SR Watson         16
YK Pathan         16
RA Jadeja         16
AD Russell        15
Name: count, dtype: int64
```

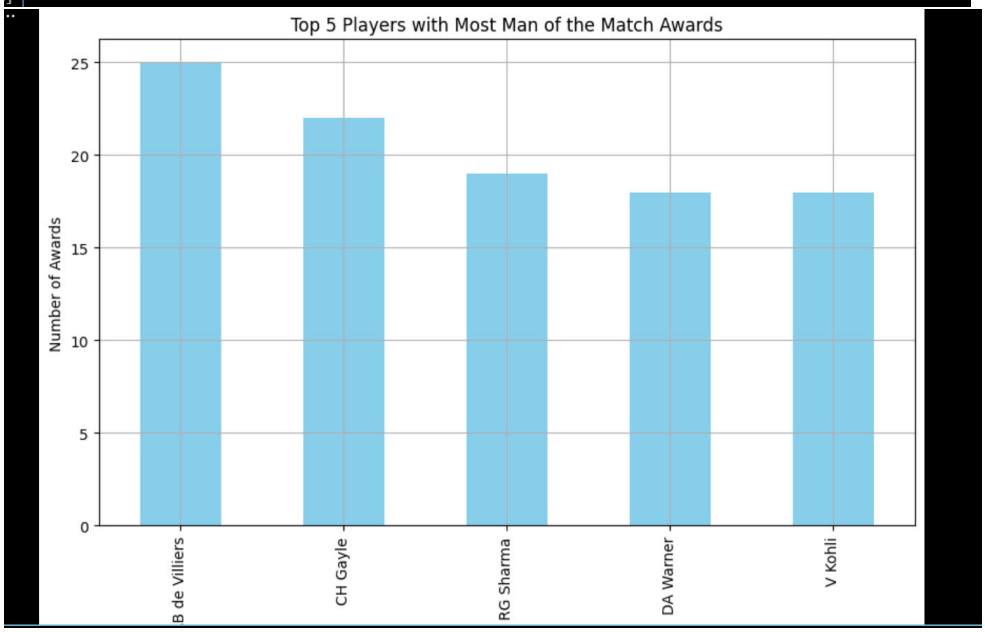
3) Evaluate the top 5 players with most men of the match award

```
top_5_players = man_of_match_freq.head(5)
print(top_5_players)
```

```
player_of_match
AB de Villiers    25
CH Gayle          22
RG Sharma         19
DA Warner         18
V Kohli           18
Name: count, dtype: int64
```

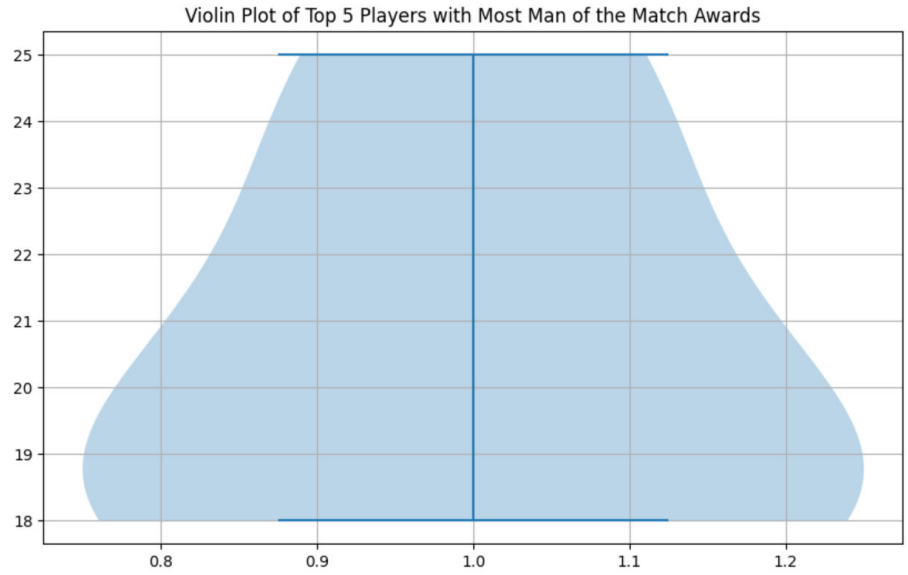
4) Drop bar plot of top 5 most man of the match award winning player

```
plt.figure(figsize=(10, 6))
top_5_players.plot(kind='bar', color='skyblue')
plt.title('Top 5 Players with Most Man of the Match Awards')
plt.xlabel('Player')
plt.ylabel('Number of Awards')
plt.grid(True)
plt.show()
```



5) Drop violin plot of top 5 most man of the match award winning player

```
plt.figure(figsize=(10, 6))
plt.violinplot(top_5_players)
plt.title('Violin Plot of Top 5 Players with Most Man of the Match Awards')
plt.grid(True)
plt.show()
```



6)Evaluate the frequency of result column

```
result_freq = df['result'].value_counts()
print(result_freq)
```

```
result
wickets      578
runs         498
tie           14
no result      5
Name: count, dtype: int64
```

7)Find the number of toss win with respect to each team

```
toss_wins = df['toss_winner'].value_counts()
print(toss_wins)
```

```
toss_winner
Mumbai Indians                143
Kolkata Knight Riders         122
Chennai Super Kings           122
Rajasthan Royals              120
Royal Challengers Bangalore   113
Sunrisers Hyderabad            88
Kings XI Punjab                85
Delhi Daredevils               80
Delhi Capitals                 50
Deccan Chargers                43
Punjab Kings                   24
Gujarat Titans                 22
Pune Warriors                  20
Lucknow Super Giants           19
Gujarat Lions                  15
Kochi Tuskers Kerala            8
Royal Challengers Bengaluru     8
Rising Pune Supergiants         7
Rising Pune Supergiant          6
Name: count, dtype: int64
```

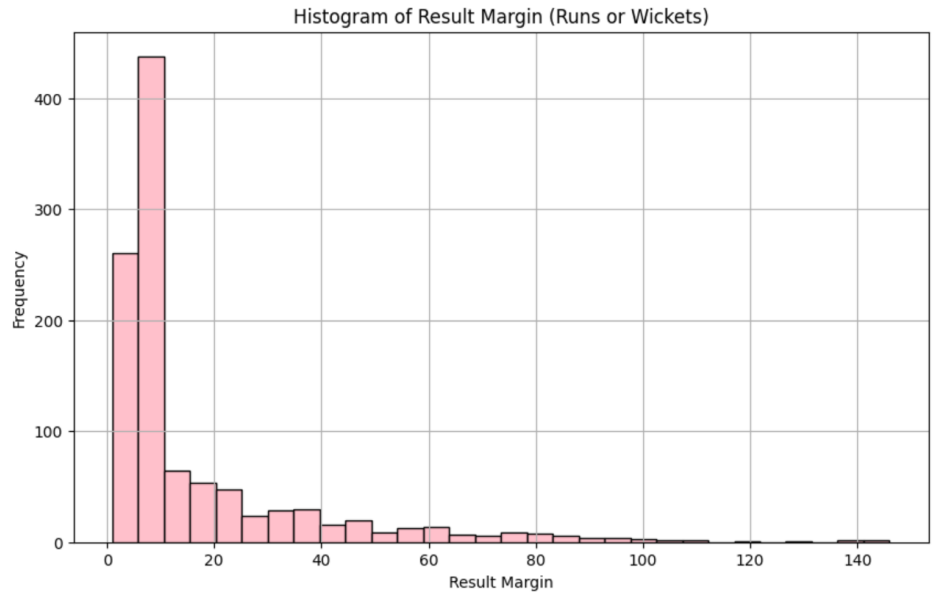
8)Extract the record where a team won batting first

```
bat_decision_matches = df[df['toss_decision'] == 'bat']

bat_and_win_matches = bat_decision_matches[bat_decision_matches['toss_winner'] == bat_decision_matches['winner']]

print("Matches where the toss decision was 'bat' and the team that decided to bat won the match:")
print(bat_and_win_matches)
```

```
Matches where the toss decision was 'bat' and the team that decided to bat won the match:
           id   season           city        date match_type player_of_match  \
1      335983  2007/08     Chandigarh  2008-04-19     League      MEK Hussey
14     335996  2007/08      Bangalore  2008-04-28     League        MS Dhoni
18     336000  2007/08         Jaipur  2008-05-01     League     SA Asnodkar
20     336003  2007/08     Chandigarh  2008-05-03     League       IK Pathan
28     336010  2007/08        Kolkata  2008-05-08     League      SC Ganguly
...       ...      ...            ...         ...        ...             ...
1036  1422131     2024  Visakhapatnam  2024-03-31     League        KK Ahmed
1039  1422134     2024  Visakhapatnam  2024-04-03     League       SP Narine
1044  1422139     2024        Lucknow  2024-04-07     League     Yash Thakur
1064  1426279     2024      Hyderabad  2024-04-25     League      RM Patidar
1073  1426288     2024      Hyderabad  2024-05-02     League         B Kumar

                                              venue  \
1            Punjab Cricket Association Stadium, Mohali
14                             M Chinnaswamy Stadium
18                            Sawai Mansingh Stadium
20           Punjab Cricket Association Stadium, Mohali
28                                   Eden Gardens
...                                           ...
1036  Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket St...
1039  Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket St...
1044  Bharat Ratna Shri Atal Bihari Vajpayee Ekana C...
1064  Rajiv Gandhi International Stadium, Uppal, Hyd...
...
1064      Nitin Menon      HAS Khalid
1073     AK Chaudhary        YC Barde

[177 rows x 20 columns]
```

9) Make the histogram of win by run column

```
plt.figure(figsize=(10, 6))
df['result_margin'].hist(bins=30, color='pink' ,edgecolor='black')
plt.title('Histogram of Result Margin (Runs or Wickets)')
plt.xlabel('Result Margin')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



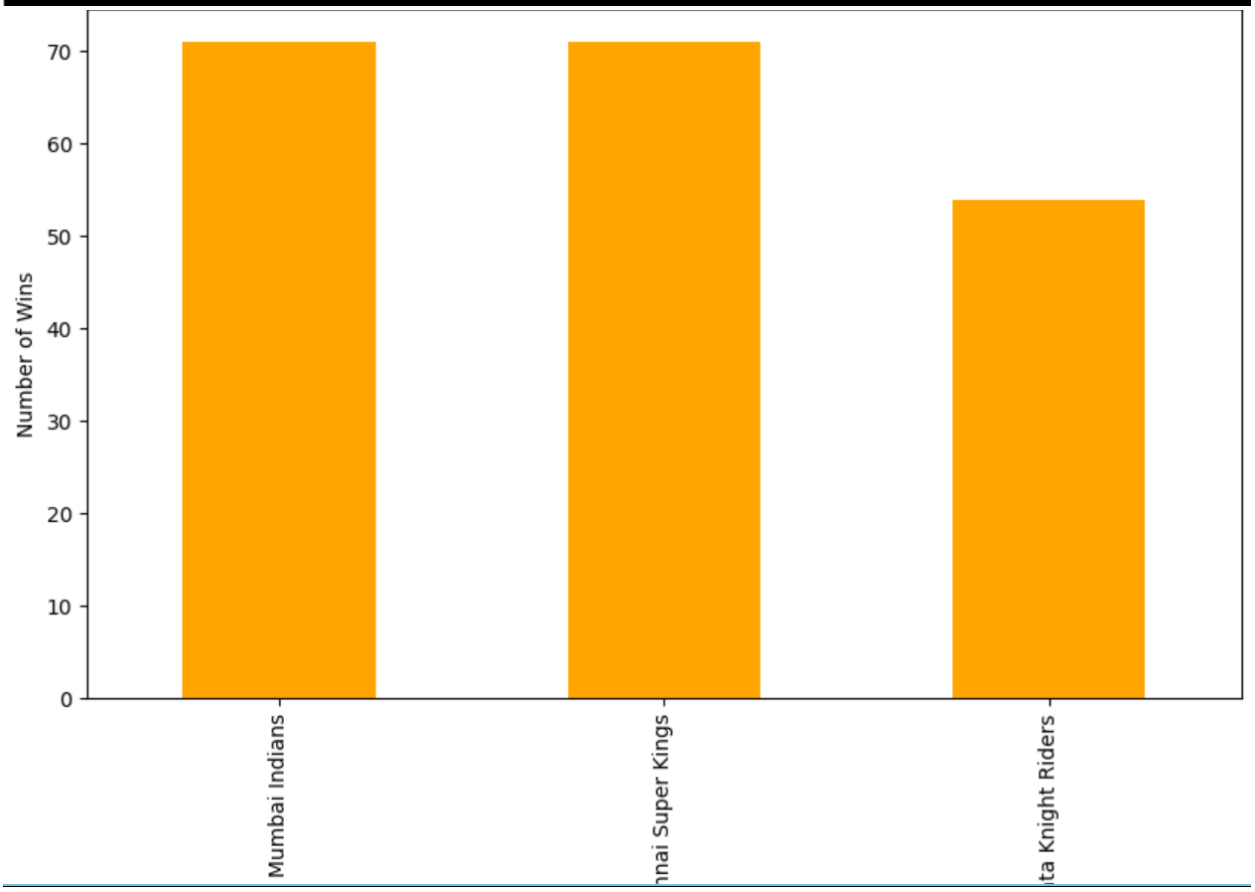
10) Find out the number of win with respect to each teams after batting first

```
batting_first_wins = df[(df['toss_decision'] == 'bat') & (df['toss_winner'] == df['winner'])]
wins_by_team = batting_first_wins['winner'].value_counts()
print(wins_by_team)
```

```
winner
Chennai Super Kings            35
Mumbai Indians                 30
Kolkata Knight Riders          20
Rajasthan Royals               18
Royal Challengers Bangalore    16
Sunrisers Hyderabad            12
Deccan Chargers                11
Delhi Daredevils               11
Delhi Capitals                  7
Kings XI Punjab                 6
Lucknow Super Giants            4
Pune Warriors                   3
Gujarat Titans                  3
Royal Challengers Bengaluru     1
Name: count, dtype: int64
```

11) Make the bar plot for top 3 teams with most wins after batting first

```
top_3_teams = wins_after_batting_first.head(3)
plt.figure(figsize=(10, 6))
top_3_teams.plot(kind='bar', color='orange')
plt.title('Top 3 Teams with Most Wins After Batting First')
plt.xlabel('Team')
plt.ylabel('Number of Wins')
plt.show()
```



Q12 Find the Number of Wins with Respect to Each Team After Batting Second

```
bat_second_wins = df[df['result'] == 'wickets']
wins_after_batting_second = bat_second_wins['winner'].value_counts()
print(wins_after_batting_second)
```

```
winner
Kolkata Knight Riders          76
Mumbai Indians                 71
Rajasthan Royals               67
Chennai Super Kings            67
Royal Challengers Bangalore    61
Kings XI Punjab                45
Sunrisers Hyderabad            43
Delhi Daredevils               42
Delhi Capitals                 24
Gujarat Titans                 17
Punjab Kings                   13
Gujarat Lions                  12
Deccan Chargers                11
Lucknow Super Giants            8
Pune Warriors                   6
Rising Pune Supergiant          5
Kochi Tuskers Kerala            4
Rising Pune Supergiants         3
Royal Challengers Bengaluru     3
Name: count, dtype: int64
```

Q13 Find the Total Number of Matches Played by Each Team

```
matches_per_team = df['team1'].value_counts() + df['team2'].value_counts()
print(matches_per_team)
```

```
Chennai Super Kings            238
Deccan Chargers                 75
Delhi Capitals                  91
Delhi Daredevils               161
Gujarat Lions                   30
Gujarat Titans                  45
Kings XI Punjab                190
Kochi Tuskers Kerala            14
Kolkata Knight Riders          251
Lucknow Super Giants            44
Mumbai Indians                 261
Pune Warriors                   46
Punjab Kings                    56
Rajasthan Royals               221
Rising Pune Supergiant          16
Rising Pune Supergiants         14
Royal Challengers Bangalore    240
Royal Challengers Bengaluru     15
Sunrisers Hyderabad            182
Name: count, dtype: int64
```

14) Calculate the Win Percentage for Each Team

```
# Calculate win percentage
total_matches = df['team1'].value_counts() + df['team2'].value_counts()
win_percentage = (df['winner'].value_counts() / total_matches) * 100

# Display the result
print(win_percentage)
```

```
Chennai Super Kings            57.983193
Deccan Chargers                38.666667
Delhi Capitals                 52.747253
Delhi Daredevils               41.614907
Gujarat Lions                  43.333333
Gujarat Titans                 62.222222
Kings XI Punjab                46.315789
Kochi Tuskers Kerala           42.857143
Kolkata Knight Riders          52.191235
Lucknow Super Giants           54.545455
Mumbai Indians                 55.172414
Pune Warriors                  26.086957
Punjab Kings                   42.857143
Rajasthan Royals               50.678733
Rising Pune Supergiant         62.500000
Rising Pune Supergiants        35.714286
Royal Challengers Bangalore    48.333333
Royal Challengers Bengaluru    46.666667
Sunrisers Hyderabad            48.351648
Name: count, dtype: float64
```

15) Find the Team that Won the Most Tosses

```
# Count the number of toss wins per team
toss_wins = df['toss_winner'].value_counts()

# Display the result
print(toss_wins)
```

```
toss_winner
Mumbai Indians                 143
Kolkata Knight Riders          122
Chennai Super Kings            122
Rajasthan Royals               120
Royal Challengers Bangalore    113
Sunrisers Hyderabad             88
Kings XI Punjab                 85
Delhi Daredevils                80
Delhi Capitals                  50
Deccan Chargers                 43
Punjab Kings                    24
Gujarat Titans                  22
Pune Warriors                   20
Lucknow Super Giants            19
Gujarat Lions                   15
Kochi Tuskers Kerala             8
Royal Challengers Bengaluru      8
Rising Pune Supergiants          7
Rising Pune Supergiant           6
Name: count, dtype: int64
```

16) Calculate the Average Result Margin for Each Team

```
avg_result_margin = df.groupby('winner')['result_margin'].mean()

print(avg_result_margin)
```

```
winner
Chennai Super Kings            20.905797
Deccan Chargers                17.000000
Delhi Capitals                 14.222222
Delhi Daredevils               14.179104
Gujarat Lions                   5.076923
Gujarat Titans                 16.928571
Kings XI Punjab                15.564706
Kochi Tuskers Kerala            8.833333
Kolkata Knight Riders          17.576923
Lucknow Super Giants           18.083333
Mumbai Indians                 19.584507
Pune Warriors                  14.583333
Punjab Kings                   12.541667
Rajasthan Royals               15.609091
Rising Pune Supergiant         15.400000
Rising Pune Supergiants        14.600000
Royal Challengers Bangalore    19.289474
Royal Challengers Bengaluru    26.571429
Sunrisers Hyderabad            15.885057
Name: result_margin, dtype: float64
```

17) Identify the Team with the Largest Winning Margin by Run

```
largest_win_margin = df[df['result'] == 'runs'].sort_values(by='result_margin', ascending=False).iloc[0]
print(largest_win_margin[['winner', 'result_margin']])
```

```
winner          Mumbai Indians
result_margin            146.0
Name: 620, dtype: object
```

18) Find the City with the Most Matches Hosted

```
matches_per_city = df['city'].value_counts()
print(matches_per_city)
```

```
city
Mumbai           173
Kolkata           93
Delhi             90
Chennai           85
Hyderabad         77
Bangalore         65
Chandigarh        61
Jaipur            57
Pune              51
Abu Dhabi         37
Ahmedabad         36
Bengaluru         29
Durban            15
Visakhapatnam     15
Lucknow           14
Dubai             13
Dharamsala        13
Centurion         12
Rajkot            10
Sharjah           10
Indore             9
Navi Mumbai        9
Johannesburg       8
Cuttack            7
...
Guwahati           3
Nagpur             3
Bloemfontein       2
Name: count, dtype: int64
```

19) Analyze the Most Common Toss Decision (Bat/Field)

```
toss_decision_count = df['toss_decision'].value_counts()
print(toss_decision_count)
```

```
toss_decision
field    704
bat      391
Name: count, dtype: int64
```

20) Identify the Venue with the Most Matches Played

```
matches_per_venue = df['venue'].value_counts()
print(matches_per_venue)
```

```
venue
Eden Gardens                                                    77
Wankhede Stadium                                                73
M Chinnaswamy Stadium                                           65
Feroz Shah Kotla                                                60
Rajiv Gandhi International Stadium, Uppal                       49
MA Chidambaram Stadium, Chepauk                                 48
Sawai Mansingh Stadium                                          47
Dubai International Cricket Stadium                             46
Wankhede Stadium, Mumbai                                        45
Punjab Cricket Association Stadium, Mohali                      35
Sheikh Zayed Stadium                                            29
Sharjah Cricket Stadium                                         28
MA Chidambaram Stadium, Chepauk, Chennai                        28
Narendra Modi Stadium, Ahmedabad                                24
Maharashtra Cricket Association Stadium                         22
Dr DY Patil Sports Academy, Mumbai                              20
Brabourne Stadium, Mumbai                                       17
Dr DY Patil Sports Academy                                      17
Eden Gardens, Kolkata                                           16
Subrata Roy Sahara Stadium                                      16
Arun Jaitley Stadium, Delhi                                     16
Rajiv Gandhi International Stadium                              15
M.Chinnaswamy Stadium                                           15
Kingsmead                                                       15
...
Barsapara Cricket Stadium, Guwahati                              3
OUTsurance Oval                                                  2
Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium, Visakhapatnam    2
Name: count, dtype: int64
```

21) Determine the Number of Super Over Matches

```
super_over_matches = df['super_over'].value_counts()
print(super_over_matches)
```

```
super_over
N    1081
Y      14
Name: count, dtype: int64
```

22) Find the Top 5 Most Frequent Matchups (Team1 vs. Team2)

```
top_5_matchups = matchup_counts.head(5)
print(top_5_matchups)
```

```
matchup
(Chennai Super Kings, Mumbai Indians)                 37
(Kolkata Knight Riders, Mumbai Indians)               34
(Mumbai Indians, Royal Challengers Bangalore)         32
(Kolkata Knight Riders, Royal Challengers Bangalore)  32
(Chennai Super Kings, Royal Challengers Bangalore)    30
Name: count, dtype: int64
```

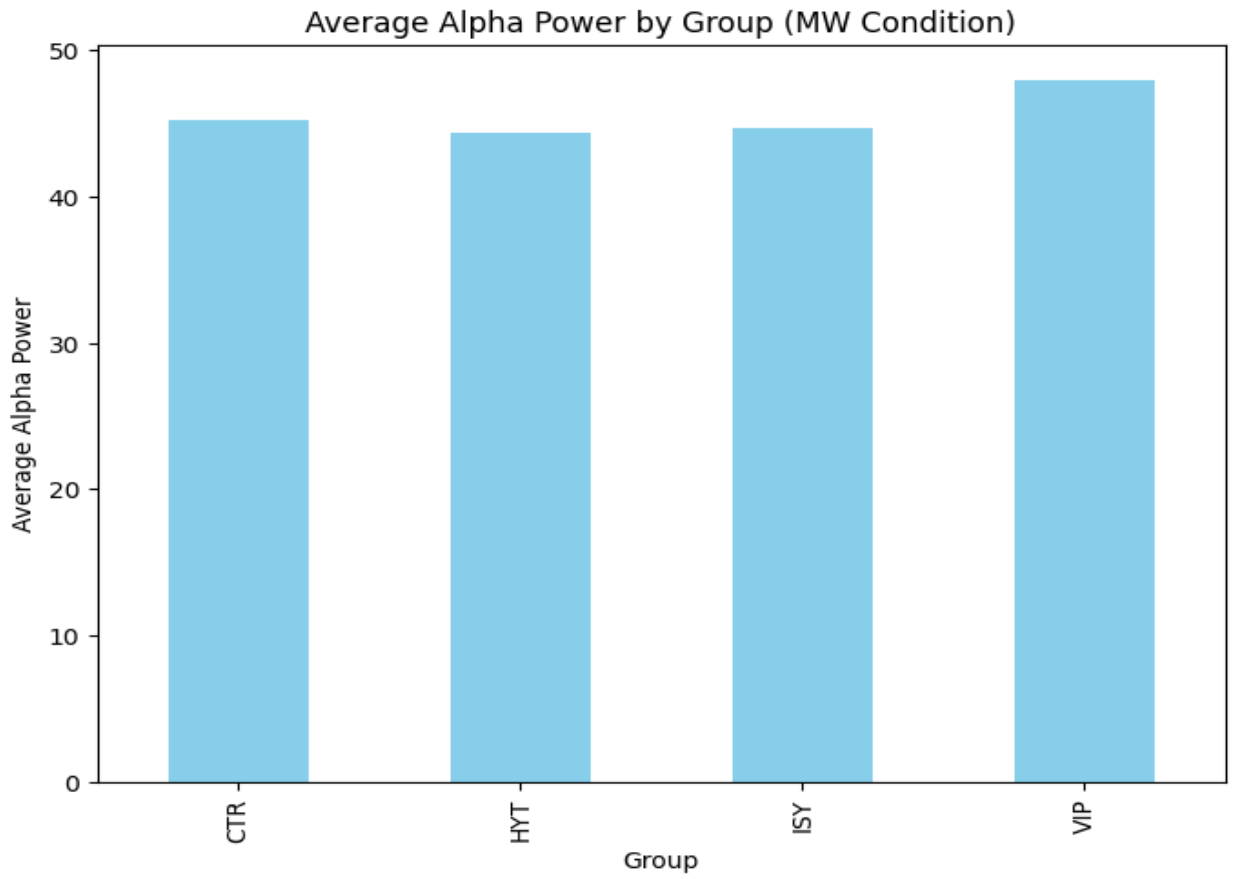
MEDIAN-ALPHA POWER

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('median_alpha_power.csv')
```

1. How does the average alpha power differ between different groups

BAR GRAPH

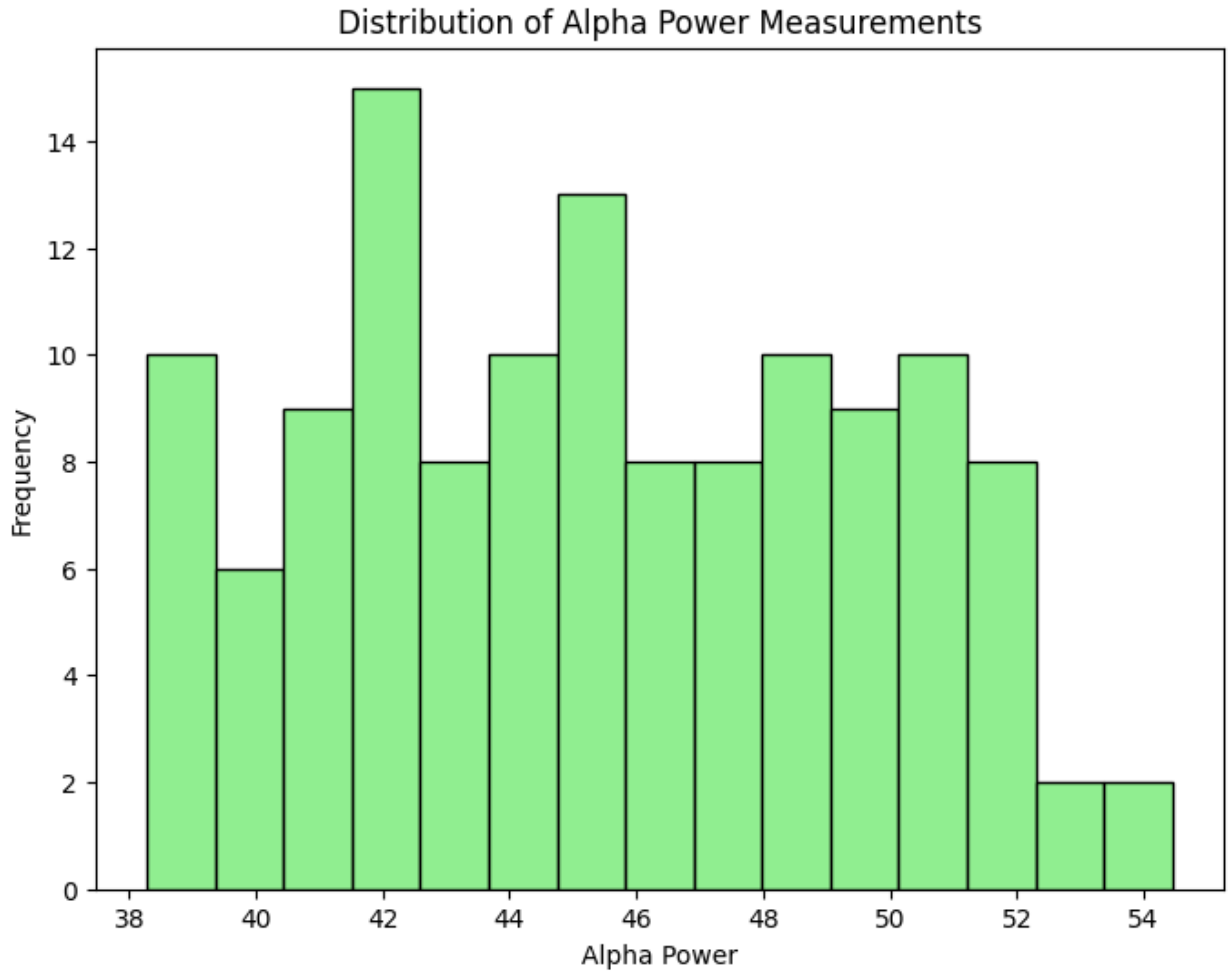
```
mean_alpha_power = df[df['Condition'] == 'MW'].groupby('Group')['Alpha'].mean()

# Plot the bar chart
plt.figure(figsize=(8, 6))
mean_alpha_power.plot(kind='bar', color='skyblue')
plt.title('Average Alpha Power by Group (MW Condition)')
plt.ylabel('Average Alpha Power')
plt.xlabel('Group')
plt.show()
```



2. What is the distribution of alpha power measurements across all subjects?

HISTOGRAM

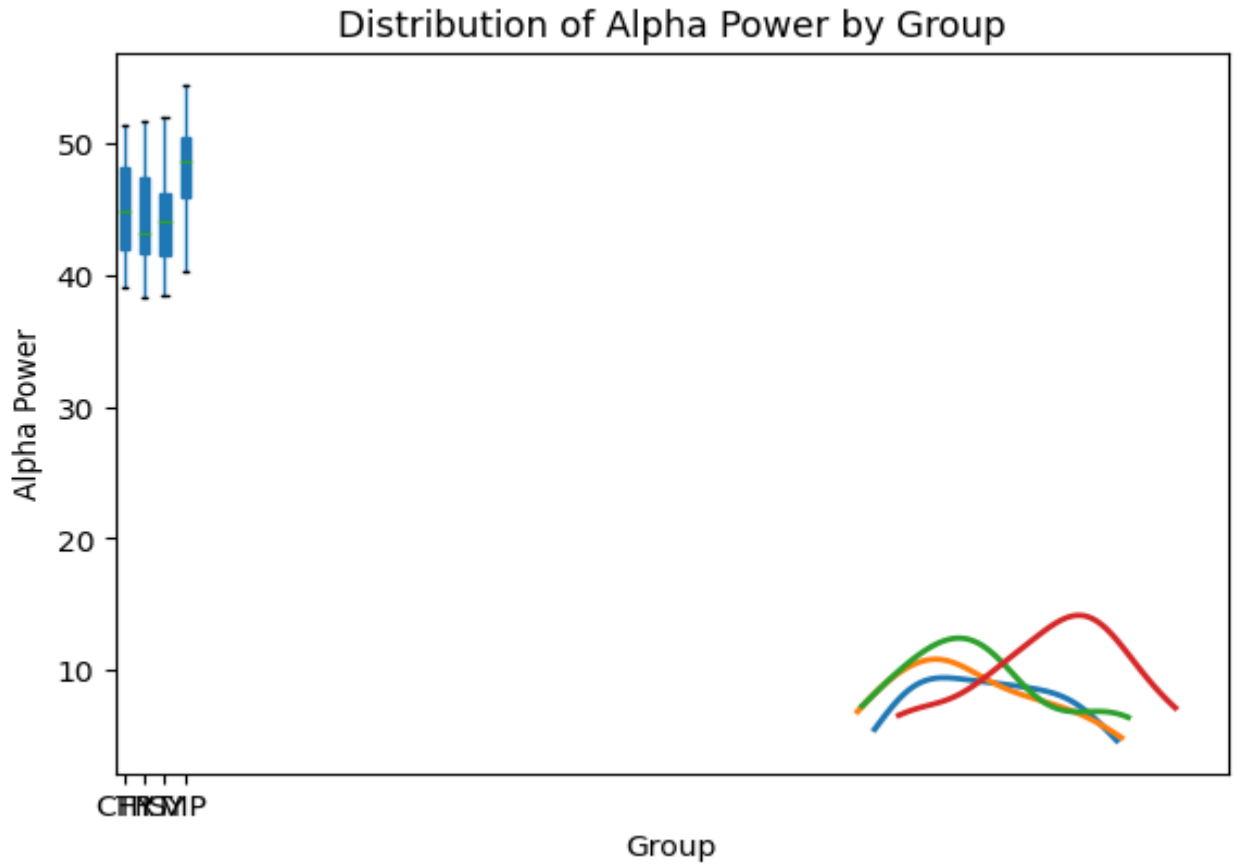
```
plt.figure(figsize=(8, 6))
plt.hist(df['Alpha'], bins=15, color='lightgreen', edgecolor='black')
plt.title('Distribution of Alpha Power Measurements')
plt.xlabel('Alpha Power')
plt.ylabel('Frequency')
plt.show()
```



Q3 How does the distribution of alpha power measurements compare between different groups?

VIOLIN PLOT

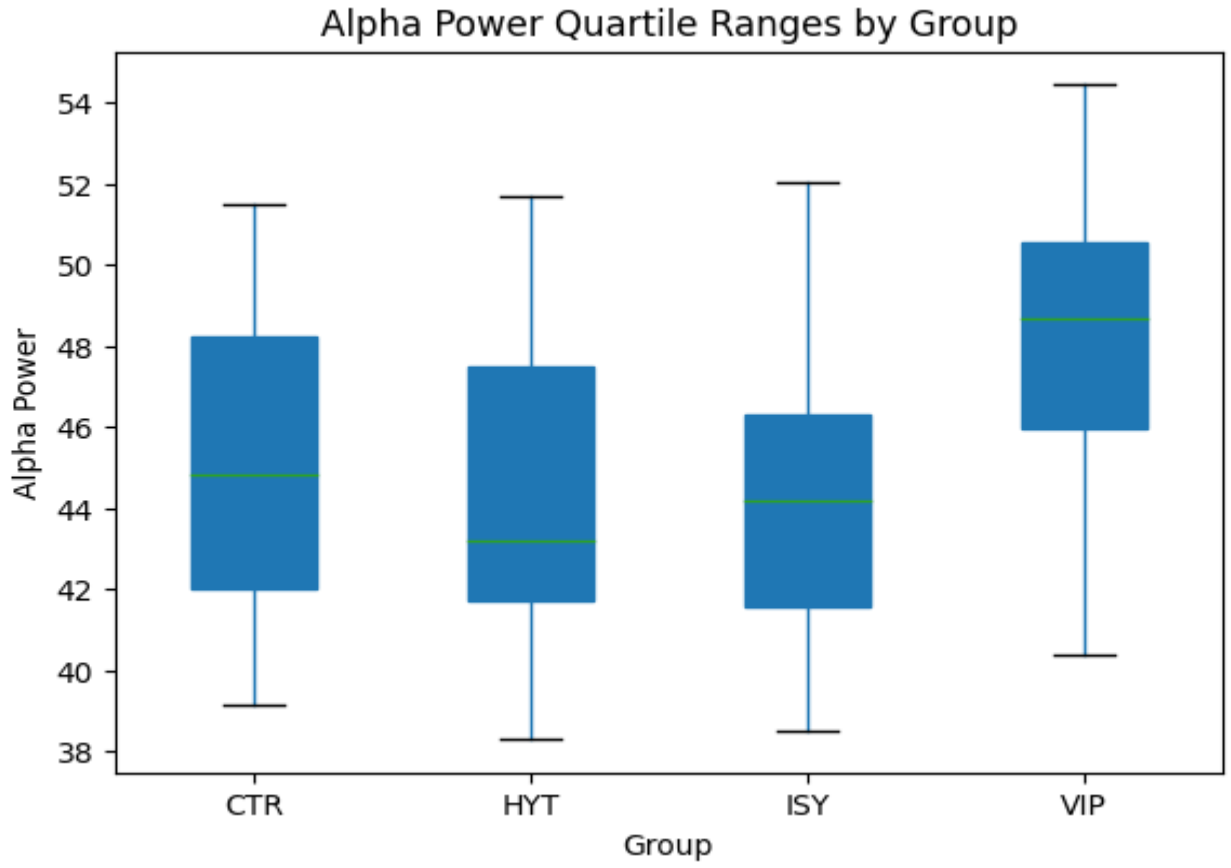
```
from scipy.stats import gaussian_kde
import numpy as np

plt.figure(figsize=(8, 6))

# Create a boxplot
df.boxplot(column='Alpha', by='Group', grid=False, patch_artist=True)

# Overlay KDE for each group
for group in df['Group'].unique():
    subset = df[df['Group'] == group]['Alpha']
    density = gaussian_kde(subset)
    x_vals = np.linspace(subset.min(), subset.max(), 100)
    plt.plot(x_vals, density(x_vals) * 100 + df['Group'].unique().tolist().index(group) + 1, lw=2)

plt.title('Distribution of Alpha Power by Group')
plt.suptitle('')
plt.xlabel('Group')
plt.ylabel('Alpha Power')
plt.show()
```



4.What are the quartile ranges of alpha power measurements for each group?

BOX PLOT

```
plt.figure(figsize=(8, 6))
df.boxplot(column='Alpha', by='Group', grid=False, patch_artist=True)
plt.title('Alpha Power Quartile Ranges by Group')
plt.suptitle('')
plt.xlabel('Group')
plt.ylabel('Alpha Power')
plt.show()
```



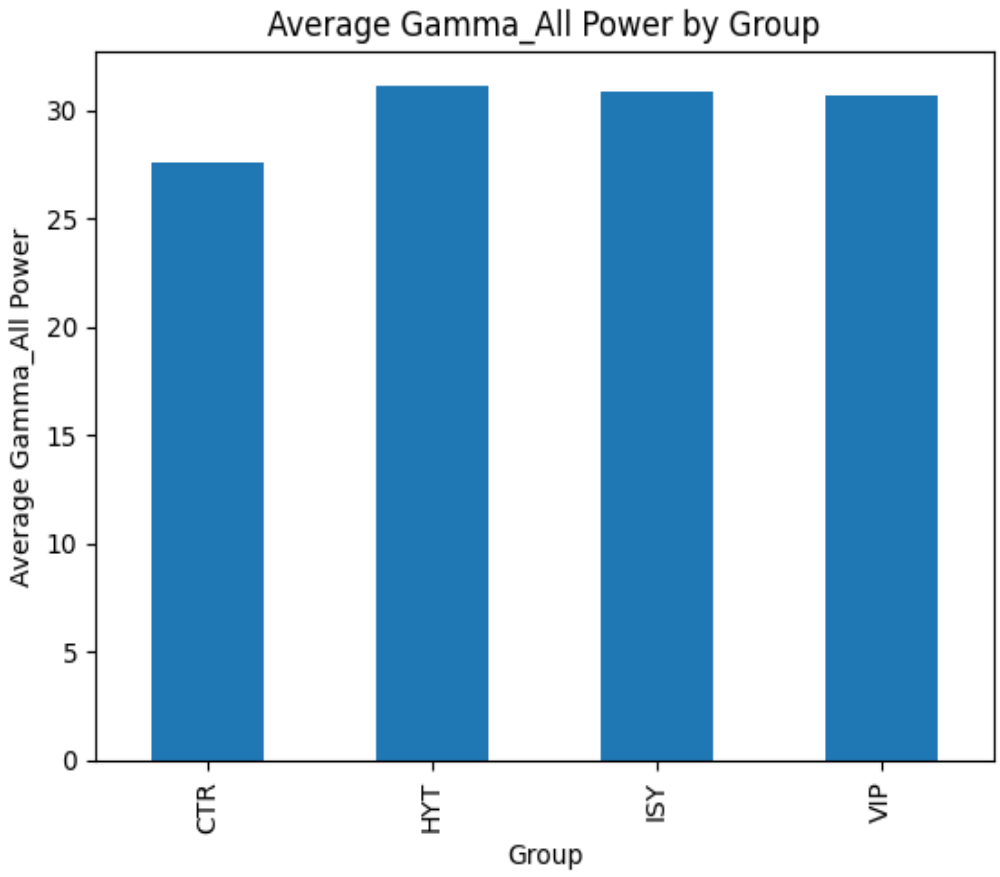
MEDIAN GAMMA POWER

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('median_gamma_power.csv')
```

1 What is the average Gamma_All power for each group

BAR GRAPH

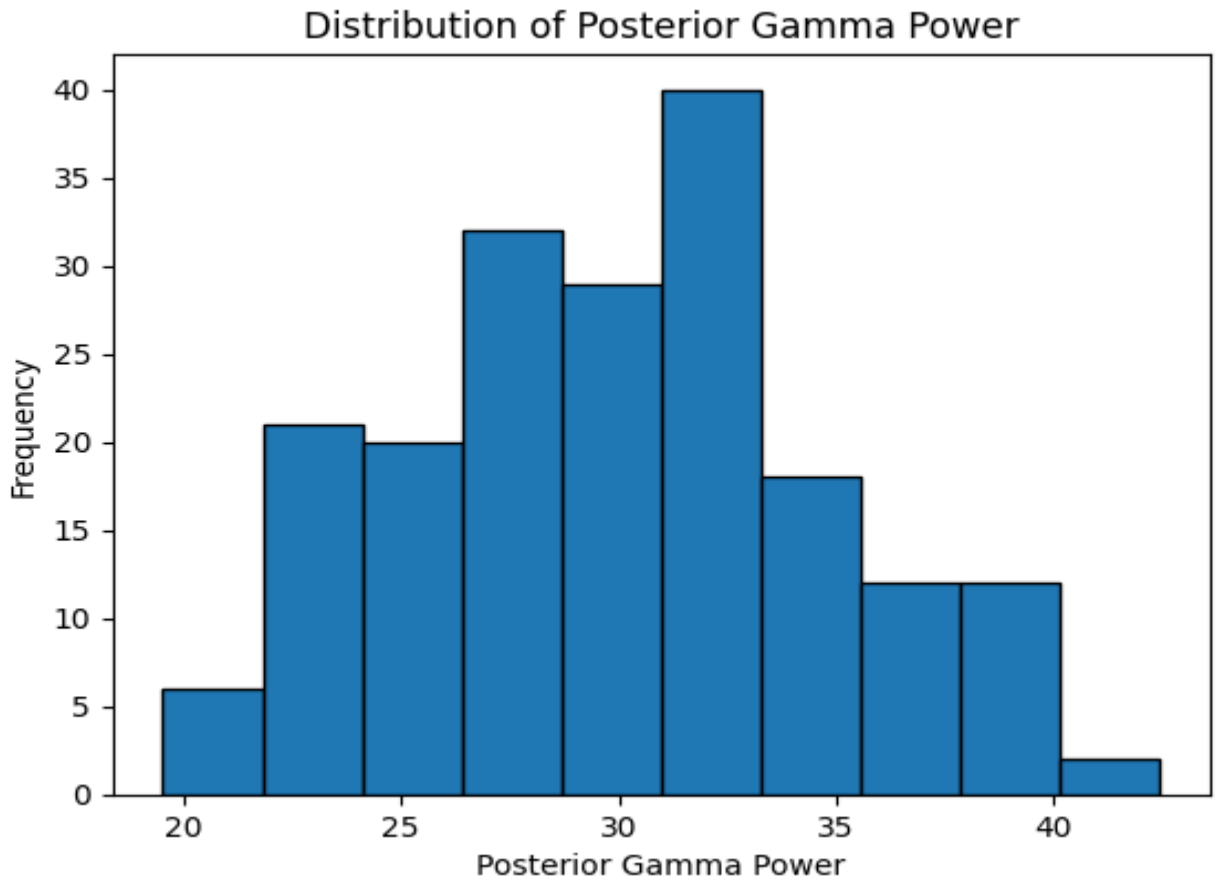
```
group_means = df.groupby('Group')['Gamma_All'].mean()

# Plot the bar chart
group_means.plot(kind='bar')
plt.title('Average Gamma_All Power by Group')
plt.ylabel('Average Gamma_All Power')
plt.show()
```



Q2 How is the distribution of Posterior gamma power across all subjects?

HISTOGRAM

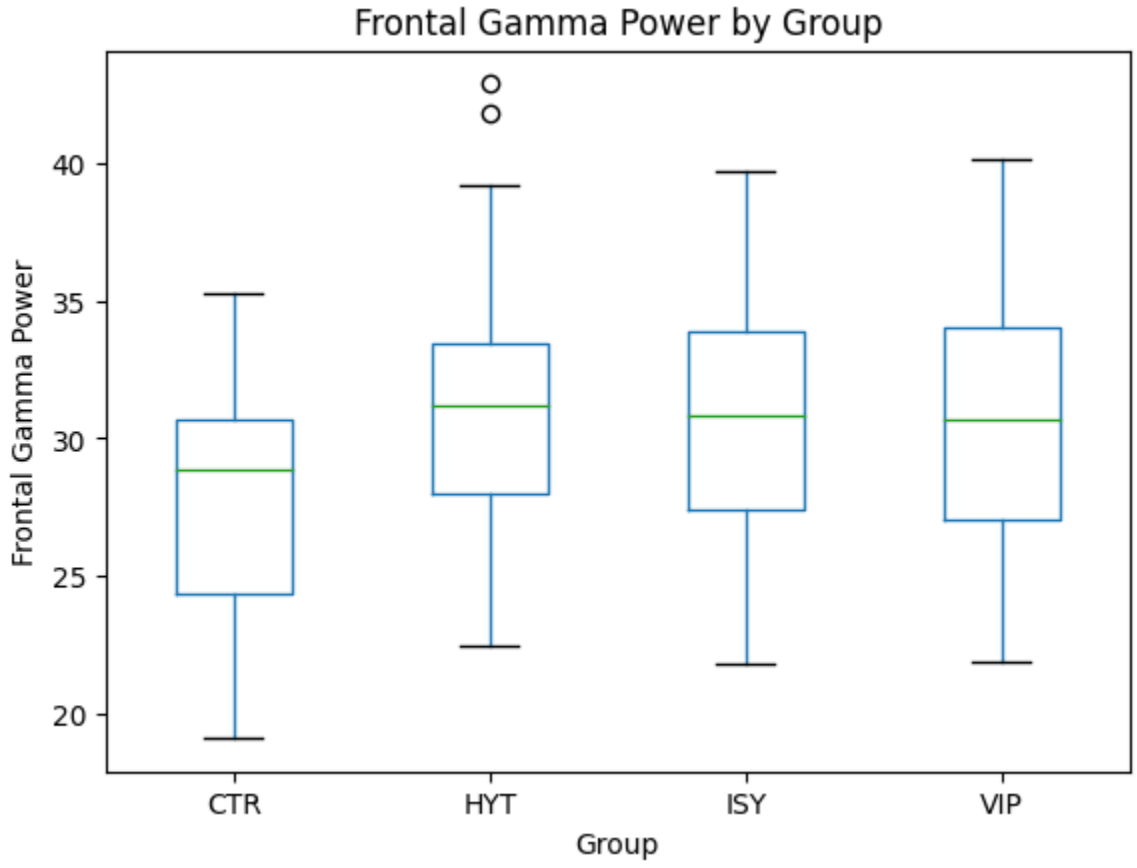
```
# Plot the histogram for Posterior gamma power
plt.hist(df['Posterior'], bins=10, edgecolor='black')
plt.title('Distribution of Posterior Gamma Power')
plt.xlabel('Posterior Gamma Power')
plt.ylabel('Frequency')
plt.show()
```



Q3 How does Frontal gamma power differ between groups?

VIOLIN PLOT

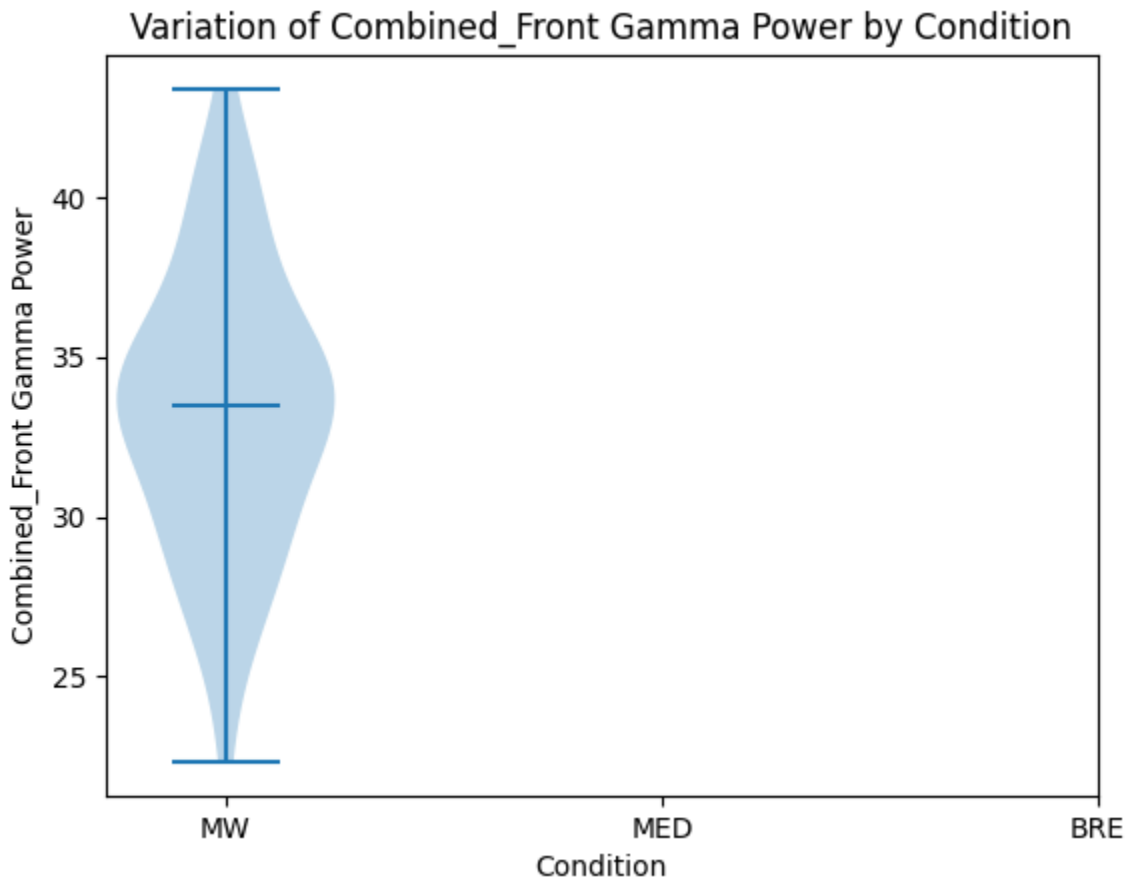
```
df.boxplot(column='Frontal', by='Group', grid=False)

# Set the title and labels
plt.title('Frontal Gamma Power by Group')
plt.suptitle('')  # Remove the default title added by pandas
plt.xlabel('Group')
plt.ylabel('Frontal Gamma Power')

plt.show()
```



Q4 How does Combined_Front gamma power vary between different conditions?

BOX PLOT

```
conditions = df['Condition'].unique()
data_by_condition = [df['Combined_Front'][df['Condition'] == condition] for condition in conditions]

# Plot the violin plot using matplotlib
fig, ax = plt.subplots()
ax.violinplot(data_by_condition, showmeans=False, showmedians=True)

# Set x-ticks and labels
ax.set_xticks(range(1, len(conditions) + 1))
ax.set_xticklabels(conditions)
ax.set_title('Variation of Combined_Front Gamma Power by Condition')
ax.set_xlabel('Condition')
ax.set_ylabel('Combined_Front Gamma Power')

# Show the plot
plt.show()
```



NAME: PREKSHA

SAP : 500120166

Experiment: 6

+ Code + Text

EXPERIMENT:6

```
[ ] import pandas as pd
    from sklearn.datasets import load_iris
```

```
[ ] df = pd.read_csv("data.csv")
```

```
[ ] df.head()
```

	Id	Colour	Country
0	1	Red	USA
1	2	Blue	UK
2	3	Green	Canada
3	4	Blue	USA
4	5	Blue	USA

+ Code + Text Connect

```
[ ] iris = load_iris()
    iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
```

```
[ ] iris_df['species'] = iris.target
```

```
[ ] iris_df['species'] = iris_df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})
```

```
[ ] print("Original DataFrame:")
    print(iris_df.head())
```

```
Original DataFrame:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

+ Code + Text Connect

```
  species
0  setosa
1  setosa
2  setosa
3  setosa
4  setosa
```

```
[ ] iris_df['species'] = iris_df['species'].map({0: 'blue', 1: 'green', 2: 'red'})
```

```
[ ] print("Original DataFrame:")
    print(iris_df.head())
```

```
Original DataFrame:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

+ Code + Text Conr

```
  species
0     NaN
1     NaN
2     NaN
3     NaN
4     NaN
```

```
[ ] iris_encoded = pd.get_dummies(iris_df, columns=['species'], drop_first=True)
```

```
[ ] print("\nDataFrame after One-Hot Encoding:")
    print(iris_encoded.head())
```

```
DataFrame after One-Hot Encoding:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

EXPERIMENT: 7

NAME: Preksha

SAP: 500120166

+ Code + Text

EXPERIMENT: 7

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

    from sklearn import datasets
    from sklearn.model_selection import train_test_split , KFold
    from sklearn.preprocessing import Normalizer
    from sklearn.metrics import accuracy_score
    from sklearn.neighbors import KNeighborsClassifier

    from collections import Counter
```

```
[ ] iris = datasets.load_iris()
    # np.c_ is the numpy concatenate function
    iris_df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                           columns= iris['feature_names'] + ['target'])
    iris_df.head()
```

+ Code + Text Connect

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

```
[ ] iris.feature_names
    iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

+ Code + Text Connect

```
iris_df.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

```
[ ] x= iris_df.iloc[:, :-1]
    y= iris_df.iloc[:, -1]
```

```
[ ] x.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
[ ] y.head()
```

	target
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

dtype: float64

```
[ ] iris_df[iris_df.target ==1].head
    iris_df[iris_df.target ==2].head
```

```
pandas.core.generic.NDFrame.head
def head(n: int=5) -> Self

Return the first `n` rows.

This function returns the first `n` rows for the object based
on position. It is useful for quickly testing if your object
has the right type of data in it.
```

```
[ ] x_train, x_test, y_train, y_test= train_test_split(x, y,
                                                       test_size= 0.2,
                                                       shuffle= True,
                                                       random_state= 0)
    x_train= np.asarray(x_train)
    y_train= np.asarray(y_train)
    x_test= np.asarray(x_test)
    y_test= np.asarray(y_test)
```

```
x_train, x_test, y_train, y_test= train_test_split(x, y,
                                                   test_size= 0.2,
                                                   shuffle= True,
                                                   random_state= 0)
x_train= np.asarray(x_train)
y_train= np.asarray(y_train)
x_test= np.asarray(x_test)
y_test= np.asarray(y_test)
```

```
[ ] print(f'training set size: {x_train.shape[0]} samples \ntest set size: {x_test.shape[0]} samples')
```

```
training set size: 120 samples
test set size: 30 samples
```

```
[ ] scaler= Normalizer().fit(x_train)
    normalized_x_train= scaler.transform(x_train)
    normalized_x_test= scaler.transform(x_test)
```

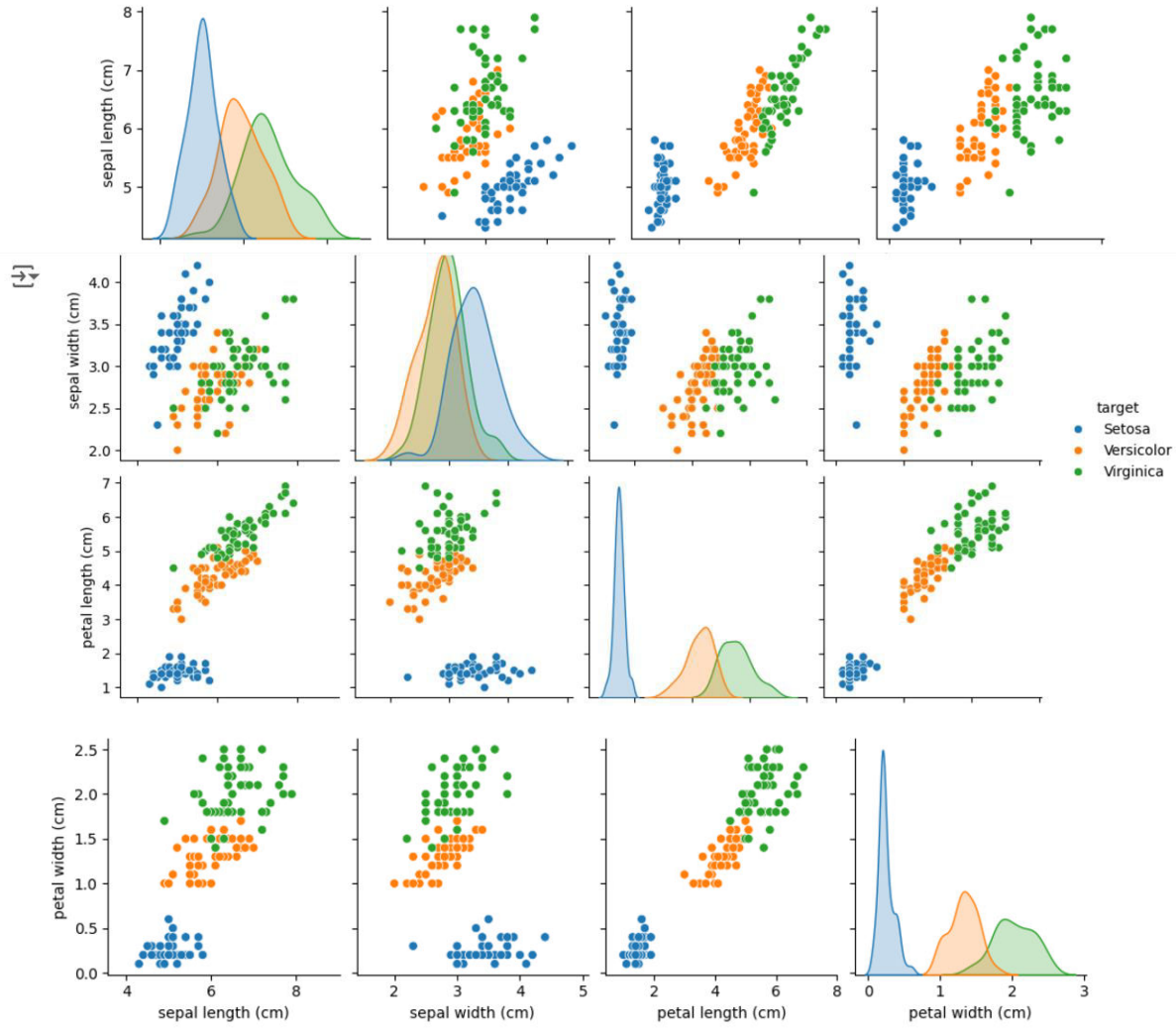
```
print('x train before Normalization')
print(x_train[0:5])
print('\nx train after Normalization')
print(normalized_x_train[0:5])
```

```
x train before Normalization
[[6.4 3.1 5.5 1.8]
 [5.4 3.  4.5 1.5]
 [5.2 3.5 1.5 0.2]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.2]]

x train after Normalization
[[0.69804799 0.338117   0.59988499 0.196326  ]
 [0.69333409 0.38518561 0.57777841 0.1925928 ]
 [0.80641965 0.54278246 0.23262105 0.03101614]
 [0.71171214 0.35002236 0.57170319 0.21001342]
 [0.69417747 0.30370264 0.60740528 0.2386235 ]]
```

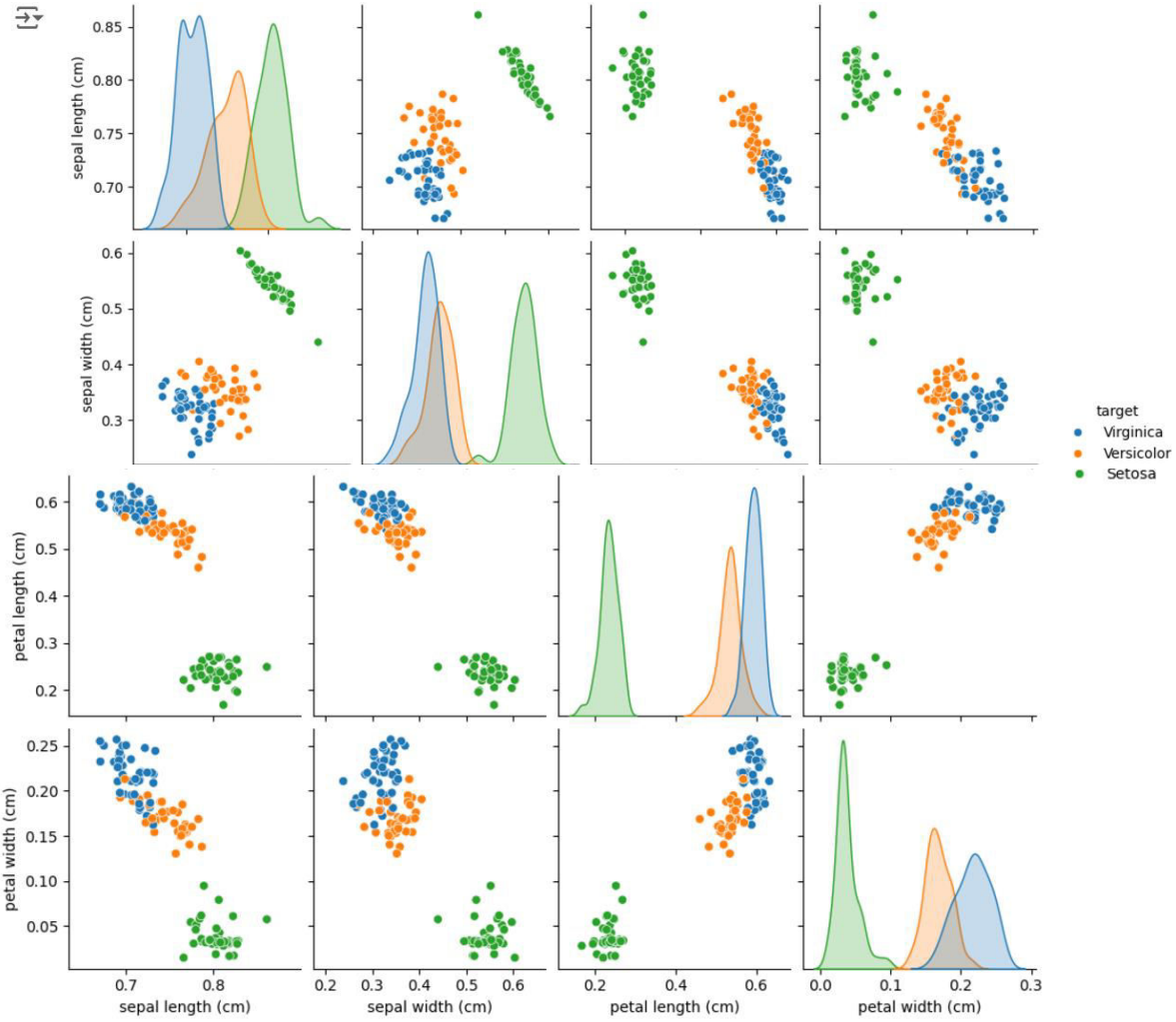
```
[ ]  ## Before
     # View the relationships between variables; color code by species type
     di= {0.0: 'Setosa', 1.0: 'Versicolor', 2.0:'Virginica'} # dictionary

     before= sns.pairplot(iris_df.replace({'target': di}), hue= 'target')
     before.fig.suptitle('Pair Plot of the dataset Before normalization', y=1.08)

     ## After
     iris_df_2= pd.DataFrame(data= np.c_[normalized_x_train, y_train],
                             columns= iris['feature_names'] + ['target'])
     di= {0.0: 'Setosa', 1.0: 'Versicolor', 2.0: 'Virginica'}
     after= sns.pairplot(iris_df_2.replace({'target':di}), hue= 'target')
     after.fig.suptitle('Pair Plot of the dataset After normalization', y=1.08)
```

```
Text(0.5, 1.08, 'Pair Plot of the dataset After normalization')
```

Pair Plot of the dataset Before normalization



Pair Plot of the dataset After normalization


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
target
Virginica
Versicolor
Setosa

```
[ ]  def distance_ecu(x_train, x_test_point):
       """
       Input:
         - x_train: corresponding to the training data
         - x_test_point: corresponding to the test point

       Output:
         -distances: The distances between the test point and each point in the training data.

       """
       distances= []  ## create empty list called distances
       for row in range(len(x_train)): ## Loop over the rows of x_train
           current_train_point= x_train[row] #Get them point by point
           current_distance= 0 ## initialize the distance by zero

           for col in range(len(current_train_point)): ## Loop over the columns of the row

               current_distance += (current_train_point[col] - x_test_point[col]) **2
               ## Or current_distance = current_distance + (x_train[i] - x_test_point[i])**2
           current_distance= np.sqrt(current_distance)

           distances.append(current_distance) ## Append the distances
```

```
[ ]
       # Store distances in a dataframe
       distances= pd.DataFrame(data=distances,columns=['dist'])
       return distances
```

```
def nearest_neighbors(distance_point, K):
    """
    Input:
        -distance_point: the distances between the test point and each point in the training data.
        -K             : the number of neighbors

    Output:
        -df_nearest: the nearest K neighbors between the test point and the training data.

    """

    # Sort values using the sort_values function
    df_nearest= distance_point.sort_values(by=['dist'], axis=0)

    ## Take only the first K neighbors
    df_nearest= df_nearest[:K]
    return df_nearest
```

```
def voting(df_nearest, y_train):
    """
    Input:
        -df_nearest: dataframe contains the nearest K neighbors between the full training dataset and the test
        -y_train: the labels of the training dataset.

    Output:
        -y_pred: the prediction based on Majority Voting

    """

    ## Use the Counter Object to get the labels with K nearest neighbors.
    counter_vote= Counter(y_train[df_nearest.index])

    y_pred= counter_vote.most_common()[0][0]   # Majority Voting

    return y_pred
```

```python
def KNN_from_scratch(x_train, y_train, x_test, K):

    """
    Input:
    -x_train: the full training dataset
    -y_train: the labels of the training dataset
    -x_test: the full test dataset
    -K: the number of neighbors

    Output:
    -y_pred: the prediction for the whole test set based on Majority Voting.

    """

    y_pred=[]

    ## Loop over all the test set and perform the three steps
    for x_test_point in x_test:
      distance_point  = distance_ecu(x_train, x_test_point)  ## Step 1
      df_nearest_point= nearest_neighbors(distance_point, K)  ## Step 2
      y_pred_point    = voting(df_nearest_point, y_train) ## Step 3
      y_pred.append(y_pred_point)

    return y_pred
```

```python
K=3
y_pred_scratch= KNN_from_scratch(normalized_x_train, y_train, normalized_x_test, K)
print(y_pred_scratch)
```

```
[2.0, 1.0, 0.0, 2.0, 0.0, 2.0, 0.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 2.0, 0.0, 0.0, 2.0, 1.0,
```

```python
knn=KNeighborsClassifier(K)
knn.fit(normalized_x_train, y_train)
y_pred_sklearn= knn.predict(normalized_x_test)
print(y_pred_sklearn)
```

```
[2. 1. 0. 2. 0. 2. 0. 1. 1. 1. 2. 1. 1. 1. 1. 0. 1. 2. 0. 0. 2. 1. 0. 0.
 2. 0. 0. 1. 1. 0.]
```

```python
print(np.array_equal(y_pred_sklearn, y_pred_scratch))
```

```
True
```

```python
print(f'The accuracy of our implementation is {accuracy_score(y_test, y_pred_scratch)}')
print(f'The accuracy of sklearn implementation is {accuracy_score(y_test, y_pred_sklearn)}')
```

```
The accuracy of our implementation is 0.9666666666666667
The accuracy of sklearn implementation is 0.9666666666666667
```

```python
n_splits= 4 ## Choose the number of splits
kf= KFold(n_splits= n_splits) ## Call the K Fold function

accuracy_k= [] ## Keep track of the accuracy for each K
k_values= list(range(1,30,2)) ## Search for the best value of K

for k in k_values: ## Loop over the K values
  accuracy_fold= 0
  for normalized_x_train_fold_idx, normalized_x_valid_fold_idx in  kf.split(normalized_x_train): ## Loop over
      normalized_x_train_fold= normalized_x_train[normalized_x_train_fold_idx] ## fetch the values
      y_train_fold= y_train[normalized_x_train_fold_idx]

      normalized_x_test_fold= normalized_x_train[normalized_x_valid_fold_idx]
      y_valid_fold= y_train[normalized_x_valid_fold_idx]
      y_pred_fold= KNN_from_scratch(normalized_x_train_fold, y_train_fold, normalized_x_test_fold, k)
```

```
print(f'The accuracy for each K value was {list ( zip (accuracy_k, k_values))}') ## creates a tuple with accur
```

```
The accuracy for each K value was [(0.9666666666666668, 1), (0.9666666666666668, 3), (0.9666666666666668, 5), (
```

```
print(f'Best accuracy was {np.max(accuracy_k)}, which corresponds to a value of K= {k_values[np.argmax(accurac
```

```
Best accuracy was 0.9666666666666668, which corresponds to a value of K= 1
```

```
k_values = [1, 3, 5, 7, 9]

results = {}

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(x_train, y_train)

    y_pred = knn.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)

    results[k] = accuracy
    print(f'k={k}, Accuracy={accuracy:.4f}')
```

```
k=1, Accuracy=1.0000
k=3, Accuracy=0.9667
k=5, Accuracy=0.9667
k=7, Accuracy=1.0000
k=9, Accuracy=1.0000
```

EXPERIMENT 8

Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

```
[ ] import numpy as np
    import pandas as pd
    from sklearn.decomposition import PCA
    from sklearn.datasets import load_iris
    from sklearn.preprocessing import StandardScaler
    import matplotlib.pyplot as plt
```

```
[ ] iris = load_iris()
```

```
[ ] X = iris.data  #features
    y = iris.target   #species
    feature_names = iris.feature_names
```

```
[ ] scaler = StandardScaler()
    iris_scaled = scaler.fit_transform(iris.data)
```

```
[ ] print("Mean of features after scaling:", np.mean(iris_scaled, axis=0))
    print("Standard deviation of features after scaling:", np.std(iris_scaled, axis=0))
```

```
Mean of features after scaling: [-1.69031455e-15 -1.84297022e-15 -1.69864123e-15 -1.40924309e-15]
Standard deviation of features after scaling: [1. 1. 1. 1.]
```

```
[ ] pca = PCA(n_components=2)
    iris_pca = pca.fit_transform(iris_scaled)
```

```
pca = PCA(n_components=2)
iris_pca = pca.fit_transform(iris_scaled)
```

```
[ ] pca_df = pd.DataFrame(data=iris_pca, columns=['PC1', 'PC2'])
    pca_df['species'] = pd.Categorical.from_codes(y, iris.target_names) # Add target  column
```

```
pca_df
```

	PC1	PC2	species
0	-2.264703	0.480027	setosa
1	-2.080961	-0.674134	setosa
2	-2.364229	-0.341908	setosa
3	-2.299384	-0.597395	setosa
4	-2.389842	0.646835	setosa
...	...	...	...
145	1.870503	0.386966	virginica
146	1.564580	-0.896687	virginica
147	1.521170	0.269069	virginica
148	1.372788	1.011254	virginica
149	0.960656	-0.024332	virginica

150 rows × 3 columns

```
plt.figure(figsize=(8, 6))
for species in iris.target_names:
    species_data = pca_df[pca_df['species'] == species]
    plt.scatter(species_data['PC1'], species_data['PC2'], label=species)

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.legend()
plt.show()
```