```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import f_oneway

from sklearn.metrics import balanced_accuracy_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import VarianceThreshold


from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, classification_report

from scipy.stats.mstats import winsorize
```

Data Reading

```
# Reading the newly uploaded CSV file
file_path = 'PCOS_Likelihood.csv'
original_data = pd.read_csv(file_path)

# Show the glimps of dataset
pd.set_option('display.max_columns', None)
display(original_data.head())

# Checking the number of columns and rows
rows, columns = original_data.shape
print("\nNumber of Rows:",rows, "\n\nNumber of Columns:", columns,'\
n')

# Identifying data types of each feature
data_types = original_data.info()
```

```
   Sl. No  Patient File No.  PCOS (Y/N)   Age (yrs)  Weight (Kg)
Height(Cm)   \
0       1                 1           0          28         44.6
152.0
1       2                 2           0          36         65.0
161.5
```

```
2       3                  3           1          33         68.8
165.0
3       4                  4           0          37         65.0
148.0
4       5                  5           0          25         52.0
161.0

     BMI  Blood Group  Pulse rate(bpm)   RR (breaths/min)  Hb(g/dl)  \
0  19.30           15                78                22     10.48
1  24.92           15                74                20     11.70
2  25.27           11                72                18     11.80
3  29.67           13                72                20     12.00
4  20.06           11                72                18     10.00

   Cycle(R/I)  Cycle length(days)  Marraige Status (Yrs)
Pregnant(Y/N)  \
0           2                   5                   7.0
0
1           2                   5                  11.0
1
2           2                   5                  10.0
1
3           2                   5                   4.0
0
4           2                   5                   1.0
1

   No. of aborptions    I   beta-HCG(mIU/mL)  II
beta-HCG(mIU/mL)  \
0                   0                  1.99                    1.99

1                   0                 60.80                    1.99

2                   0                494.08                  494.08

3                   0                  1.99                    1.99

4                   0                801.45                  801.45


   FSH(mIU/mL)  LH(mIU/mL)  FSH/LH  Hip(inch)  Waist(inch)  Waist:Hip
Ratio  \
0         7.95        3.68    2.16         36           30
0.83
1         6.73        1.09    6.17         38           32
0.84
2         5.54        0.88    6.30         40           36
0.90
3         8.06        2.36    3.42         42           36
0.86
```

```
4         3.98        0.90    4.42         37           30
0.81

   TSH (mIU/L)  AMH(ng/mL)  PRL(ng/mL)  Vit D3 (ng/mL)  PRG(ng/mL)  \
0         0.68        2.07       45.16            17.1        0.57
1         3.16        1.53       20.09            61.3        0.97
2         2.54        6.63       10.52            49.7        0.36
3        16.41        1.22       36.90            33.4        0.36
4         3.57        2.26       30.09            43.8        0.38

   RBS(mg/dl)  Weight gain(Y/N)  hair growth(Y/N)  Skin darkening
(Y/N)  \
0        92.0                 0                 0
0
1        92.0                 0                 0
0
2        84.0                 0                 0
0
3        76.0                 0                 0
0
4        84.0                 0                 0
0

   Hair loss(Y/N)  Pimples(Y/N)  Fast food (Y/N)  Reg.Exercise(Y/N)  \
0               0             0              1.0                  0
1               0             0              0.0                  0
2               1             1              1.0                  0
3               0             0              0.0                  0
4               1             0              0.0                  0

   BP _Systolic (mmHg)  BP _Diastolic (mmHg)  Follicle No. (L)  \
0                  110                    80                 3
1                  120                    70                 3
2                  120                    80                13
3                  120                    70                 2
4                  120                    80                 3

   Follicle No. (R)  Avg. F size (L) (mm)  Avg. F size (R) (mm)  \
0                 3                  18.0                  18.0
1                 5                  15.0                  14.0
2                15                  18.0                  20.0
3                 2                  15.0                  14.0
4                 4                  16.0                  14.0

   Endometrium (mm)
0               8.5
1               3.7
2              10.0
3               7.5
4               7.0
```

```
Number of Rows: 541

Number of Columns: 44

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541 entries, 0 to 540
Data columns (total 44 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Sl. No                   541 non-null    int64  
 1   Patient File No.         541 non-null    int64  
 2   PCOS (Y/N)               541 non-null    int64  
 3    Age (yrs)               541 non-null    int64  
 4   Weight (Kg)              541 non-null    float64
 5   Height(Cm)               541 non-null    float64
 6   BMI                      541 non-null    float64
 7   Blood Group              541 non-null    int64  
 8   Pulse rate(bpm)          541 non-null    int64  
 9   RR (breaths/min)         541 non-null    int64  
 10  Hb(g/dl)                 541 non-null    float64
 11  Cycle(R/I)               541 non-null    int64  
 12  Cycle length(days)       541 non-null    int64  
 13  Marraige Status (Yrs)    540 non-null    float64
 14  Pregnant(Y/N)            541 non-null    int64  
 15  No. of aborptions        541 non-null    int64  
 16    I   beta-HCG(mIU/mL)   541 non-null    float64
 17  II    beta-HCG(mIU/mL)   541 non-null    float64
 18  FSH(mIU/mL)              541 non-null    float64
 19  LH(mIU/mL)               541 non-null    float64
 20  FSH/LH                   541 non-null    float64
 21  Hip(inch)                541 non-null    int64  
 22  Waist(inch)              541 non-null    int64  
 23  Waist:Hip Ratio          541 non-null    float64
 24  TSH (mIU/L)              541 non-null    float64
 25  AMH(ng/mL)               541 non-null    float64
 26  PRL(ng/mL)               541 non-null    float64
 27  Vit D3 (ng/mL)           541 non-null    float64
 28  PRG(ng/mL)               541 non-null    float64
 29  RBS(mg/dl)               541 non-null    float64
 30  Weight gain(Y/N)         541 non-null    int64  
 31  hair growth(Y/N)         541 non-null    int64  
 32  Skin darkening (Y/N)     541 non-null    int64  
 33  Hair loss(Y/N)           541 non-null    int64  
 34  Pimples(Y/N)             541 non-null    int64  
 35  Fast food (Y/N)          540 non-null    float64
 36  Reg.Exercise(Y/N)        541 non-null    int64  
 37  BP _Systolic (mmHg)      541 non-null    int64  
 38  BP _Diastolic (mmHg)     541 non-null    int64  
 39  Follicle No. (L)         541 non-null    int64
```

```
 40  Follicle No. (R)         541 non-null    int64
 41  Avg. F size (L) (mm)     541 non-null    float64
 42  Avg. F size (R) (mm)     541 non-null    float64
 43  Endometrium (mm)         541 non-null    float64
dtypes: float64(21), int64(23)
memory usage: 186.1 KB
```

Data Preparation

Remove unnecessary spaces from column names

Binary, Qualittaitve and Other Fetures

```
data=original_data.copy()

# Remove leading and trailing spaces from column names
data.columns = data.columns.str.strip()

# Verify the changes
print("Updated column names:", data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541 entries, 0 to 540
Data columns (total 44 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Sl. No                   541 non-null    int64
 1   Patient File No.         541 non-null    int64
 2   PCOS (Y/N)               541 non-null    int64
 3   Age (yrs)                541 non-null    int64
 4   Weight (Kg)              541 non-null    float64
 5   Height(Cm)               541 non-null    float64
 6   BMI                      541 non-null    float64
 7   Blood Group              541 non-null    int64
 8   Pulse rate(bpm)          541 non-null    int64
 9   RR (breaths/min)         541 non-null    int64
 10  Hb(g/dl)                 541 non-null    float64
 11  Cycle(R/I)               541 non-null    int64
 12  Cycle length(days)       541 non-null    int64
 13  Marraige Status (Yrs)    540 non-null    float64
 14  Pregnant(Y/N)            541 non-null    int64
 15  No. of aborptions        541 non-null    int64
 16  I   beta-HCG(mIU/mL)     541 non-null    float64
 17  II    beta-HCG(mIU/mL)   541 non-null    float64
 18  FSH(mIU/mL)              541 non-null    float64
 19  LH(mIU/mL)               541 non-null    float64
 20  FSH/LH                   541 non-null    float64
 21  Hip(inch)                541 non-null    int64
 22  Waist(inch)              541 non-null    int64
 23  Waist:Hip Ratio          541 non-null    float64
```

```
 24  TSH (mIU/L)              541 non-null    float64
 25  AMH(ng/mL)               541 non-null    float64
 26  PRL(ng/mL)               541 non-null    float64
 27  Vit D3 (ng/mL)           541 non-null    float64
 28  PRG(ng/mL)               541 non-null    float64
 29  RBS(mg/dl)               541 non-null    float64
 30  Weight gain(Y/N)         541 non-null    int64
 31  hair growth(Y/N)         541 non-null    int64
 32  Skin darkening (Y/N)     541 non-null    int64
 33  Hair loss(Y/N)           541 non-null    int64
 34  Pimples(Y/N)             541 non-null    int64
 35  Fast food (Y/N)          540 non-null    float64
 36  Reg.Exercise(Y/N)        541 non-null    int64
 37  BP _Systolic (mmHg)      541 non-null    int64
 38  BP _Diastolic (mmHg)     541 non-null    int64
 39  Follicle No. (L)         541 non-null    int64
 40  Follicle No. (R)         541 non-null    int64
 41  Avg. F size (L) (mm)     541 non-null    float64
 42  Avg. F size (R) (mm)     541 non-null    float64
 43  Endometrium (mm)         541 non-null    float64
dtypes: float64(21), int64(23)
memory usage: 186.1 KB
Updated column names: None
```

```
# List of columns in the dataset
columns = data.columns.tolist()

# Identify binary features
binary_columns = [col for col in columns if data[col].nunique() == 2]

print("Binary Features:")
print(binary_columns)
print("\nNumber of Binary Features:", len(binary_columns))


# Identify non-binary features
non_binary_features = [col for col in data.columns if col not in
binary_columns]

# Check the data types of each column
data_types = data.dtypes

# Identify qualitative features based on data type (object type in
pandas usually indicates strings/characters)
qualitative_features = data_types[data_types ==
'object'].index.tolist()

print("\nQualitative Features:")
print(qualitative_features)
print("\nNumber of Qualitative Features:", len(qualitative_features))
```

```
# Identify other non-binary features which are not character/quality features
non_binary_columns = [col for col in non_binary_features if col not in qualitative_features]

print("\nOther Non-Binary Features:")
print(non_binary_columns )
print("\nNumber of Other Features:", len(non_binary_columns ))
```

```
Binary Features:
['PCOS (Y/N)', 'Pregnant(Y/N)', 'Weight gain(Y/N)', 'hair growth(Y/N)', 'Skin darkening (Y/N)', 'Hair loss(Y/N)', 'Pimples(Y/N)', 'Fast food (Y/N)', 'Reg.Exercise(Y/N)']

Number of Binary Features: 9

Qualitative Features:
[]

Number of Qualitative Features: 0

Other Non-Binary Features:
['Sl. No', 'Patient File No.', 'Age (yrs)', 'Weight (Kg)', 'Height(Cm)', 'BMI', 'Blood Group', 'Pulse rate(bpm)', 'RR (breaths/min)', 'Hb(g/dl)', 'Cycle(R/I)', 'Cycle length(days)', 'Marraige Status (Yrs)', 'No. of aborptions', 'I   beta-HCG(mIU/mL)', 'II    beta-HCG(mIU/mL)', 'FSH(mIU/mL)', 'LH(mIU/mL)', 'FSH/LH', 'Hip(inch)', 'Waist(inch)', 'Waist:Hip Ratio', 'TSH (mIU/L)', 'AMH(ng/mL)', 'PRL(ng/mL)', 'Vit D3 (ng/mL)', 'PRG(ng/mL)', 'RBS(mg/dl)', 'BP _Systolic (mmHg)', 'BP _Diastolic (mmHg)', 'Follicle No. (L)', 'Follicle No. (R)', 'Avg. F size (L) (mm)', 'Avg. F size (R) (mm)', 'Endometrium (mm)']

Number of Other Features: 35
```

Data Cleaning

Find missing values

```
# Task 1: Remove 'Unnamed' columns (if any)
data = data.loc[:, ~data.columns.str.contains('^Unnamed')]

# Task 2: Find missing values
missing_values = data.isnull().sum()

# Get columns with missing values
missing_columns = missing_values[missing_values > 0]

# Get rows with missing values
```

```
missing_rows = data.isnull().sum(axis=1)
missing_rows = missing_rows[missing_rows > 0]  # Filter only rows with
missing values

# Print summary
print(f"Total number of rows with missing values:
{len(missing_rows)}")
print(f"Total number of columns with missing values:
{len(missing_columns)}")
print(f"Total missing values in dataset: {data.isnull().sum().sum()}\
n")

# Print missing values per row
print("Total missing values in each row with missing data:\n",
missing_rows, "\n")

# Print missing values per column
print("Total missing values in each column with missing data:\n",
missing_columns, "\n")
```

```
Total number of rows with missing values: 2
Total number of columns with missing values: 2
Total missing values in dataset: 2

Total missing values in each row with missing data:
 156    1
458    1
dtype: int64

Total missing values in each column with missing data:
 Marraige Status (Yrs)    1
Fast food (Y/N)           1
dtype: int64
```

Impute missing values using regression model

```
data_clean = data.copy()
imputed_values_list = []  # Store imputed values

def impute_all_missing_values(df, target_column):
    """
    Imputes missing values in the target_column using a linear
regression model
    based on the relationship between 'Age (yrs)' and the
target_column.

    If the target column is binary, predictions are rounded to 0 or 1.
    If the target column is an integer, predictions are rounded to the
```

```
nearest integer.
    If the target column is a float, predictions are rounded to match
the maximum decimal places
    used in the original dataset.

    Parameters:
    df (pd.DataFrame): The dataset
    target_column (str): The column with missing values to be
predicted

    Returns:
    pd.DataFrame: Updated dataframe with missing values imputed
    """

    global imputed_values_list  # Use the global list to store imputed
values

    # Make a copy to avoid modifying the original dataset
    df_copy = df.copy()

    # Remove rows where 'Age (yrs)' or the target column have missing
values
    df_clean = df_copy.dropna(subset=['Age (yrs)', target_column])

    # Ensure there is enough data to train the model
    if df_clean.shape[0] == 0:
        print(f"Skipping {target_column}: Not enough data to train the
model.")
        return df_copy

    # Extract 'Age' as independent variable and target column as
dependent variable
    X = df_clean[['Age (yrs)']]
    y = df_clean[target_column]

    # Train a simple linear regression model
    model = LinearRegression()
    model.fit(X, y)

    # Predict missing values
    missing_rows = df_copy[target_column].isnull()
    missing_indices = df_copy[missing_rows].index

    if len(missing_indices) == 0:
        print(f"No missing values found for {target_column}.
Skipping...")
        return df_copy

    predicted_values = model.predict(df_copy.loc[missing_indices,
["Age (yrs)"]])
```

```
    # **1. If the column is binary, round predictions to 0 or 1**
    if df_clean[target_column].nunique() == 2:
        predicted_values = np.round(predicted_values).astype(int)

    # **2. If the column is an integer, round predictions to the
nearest integer**
    elif pd.api.types.is_integer_dtype(df_clean[target_column]):
        predicted_values = np.round(predicted_values).astype(int)

    # **3. If the column is a float, determine the max decimal places
in original data**
    elif pd.api.types.is_float_dtype(df_clean[target_column]):
        # Find the max decimal places in the original (non-null) data
        def count_decimals(num):
            num_str = str(num)
            if "." in num_str:
                return len(num_str.split(".")[1])
            return 0

        max_decimals =
df_clean[target_column].dropna().apply(count_decimals).max()
        predicted_values = np.round(predicted_values, max_decimals)

    # Replace missing values and track changes
    for idx, pred in zip(missing_indices, predicted_values):
        df_copy.at[idx, target_column] = pred
        imputed_values_list.append((idx, target_column, pred))  #
Store index, column, and value

    # Print equation of the regression model
    m = model.coef_[0]  # Slope
    b = model.intercept_  # Intercept
    equation = f"{target_column} = {m:.4f} * Age + {b:.4f}"
    print(f"Equation for {target_column}: {equation}")

    return df_copy


# **Usage: Impute missing values for all columns with nulls**
for col, count in missing_values.items():
    if count > 0:
        print(f"Imputing missing values for {col}...")
        data_clean = impute_all_missing_values(data_clean, col)

# **Final Check: Ensure all missing values are handled**
null_count = data_clean.isnull().sum().sum()
if null_count > 0:
    print(f"Operation unsuccessful. {null_count} missing values
remain.")
```

```
else:
    print("\nAll missing values successfully imputed!")

    # Display the imputed values in requested format
    print("\nImputed Values:")
    for idx, col_name, value in imputed_values_list:
        print(f"Index = {idx}, Column = {col_name}, Imputed Value =
{value}")
```

```
Imputing missing values for Marraige Status (Yrs)...
Equation for Marraige Status (Yrs): Marraige Status (Yrs) = 0.5876 *
Age + -10.7838
Imputing missing values for Fast food (Y/N)...
Equation for Fast food (Y/N): Fast food (Y/N) = -0.0035 * Age + 0.6243

All missing values successfully imputed!

Imputed Values:
Index = 458, Column = Marraige Status (Yrs), Imputed Value = 10.4
Index = 156, Column = Fast food (Y/N), Imputed Value = 1
```

Check duplicate rows

```
# Check duplicate rows
duplicate_rows = data_clean[data_clean.duplicated()]

# Print duplicate rows details
if len(duplicate_rows>0):
    print("Number of Duplicate Rows:",len(duplicate_rows))
    print('\n')

    # Remove duplicate rows
    data_short = data_clean.drop_duplicates()

    # Print the number of rows before and after removing duplicates
    print(f"Number of rows before removing duplicates:
{len(data_clean)}")
    print(f"Number of rows after removing duplicates:
{len(data_short)}")

else:
    data_short = data_clean
    print("No Duplicate data\n")

print("\n")
```

```
No Duplicate data
```

Saving Created Clean files

```
# Define the file path where you want to save the CSV file
file_path = 'Data_modified.csv'

# Save the DataFrame to CSV
data_short.to_csv(file_path, index=False)  # Set index=False to
exclude row numbers (index) from the saved file

print("Modified data file is saved as 'Data_modified.csv'")
```

```
Modified data file is saved as 'Data_modified.csv'
```

EDA

```
# Load the dataset
data_short = pd.read_csv('Data_modified.csv')

# Display the first few rows
display(data_short.head())
```

```
   Sl. No  Patient File No.  PCOS (Y/N)  Age (yrs)  Weight (Kg)
Height(Cm)  \
0       1                 1           0         28         44.6
152.0
1       2                 2           0         36         65.0
161.5
2       3                 3           1         33         68.8
165.0
3       4                 4           0         37         65.0
148.0
4       5                 5           0         25         52.0
161.0

     BMI  Blood Group  Pulse rate(bpm)  RR (breaths/min)  Hb(g/dl)  \
0  19.30           15               78                22     10.48
1  24.92           15               74                20     11.70
2  25.27           11               72                18     11.80
3  29.67           13               72                20     12.00
4  20.06           11               72                18     10.00

   Cycle(R/I)  Cycle length(days)  Marraige Status (Yrs)
Pregnant(Y/N)  \
0           2                   5                   7.0
0
1           2                   5                  11.0
1
2           2                   5                  10.0
1
3           2                   5                   4.0
0
```

```
4           2                   5                    1.0
1

   No. of aborptions  I   beta-HCG(mIU/mL)  II    beta-HCG(mIU/mL)  \
0                  0                1.99                    1.99
1                  0               60.80                    1.99
2                  0              494.08                  494.08
3                  0                1.99                    1.99
4                  0              801.45                  801.45

   FSH(mIU/mL)  LH(mIU/mL)  FSH/LH  Hip(inch)  Waist(inch)  Waist:Hip
Ratio  \
0         7.95        3.68    2.16         36           30
0.83
1         6.73        1.09    6.17         38           32
0.84
2         5.54        0.88    6.30         40           36
0.90
3         8.06        2.36    3.42         42           36
0.86
4         3.98        0.90    4.42         37           30
0.81

   TSH (mIU/L)  AMH(ng/mL)  PRL(ng/mL)  Vit D3 (ng/mL)  PRG(ng/mL)  \
0         0.68        2.07       45.16            17.1        0.57
1         3.16        1.53       20.09            61.3        0.97
2         2.54        6.63       10.52            49.7        0.36
3        16.41        1.22       36.90            33.4        0.36
4         3.57        2.26       30.09            43.8        0.38

   RBS(mg/dl)  Weight gain(Y/N)  hair growth(Y/N)  Skin darkening
(Y/N)  \
0        92.0                 0                 0
0
1        92.0                 0                 0
0
2        84.0                 0                 0
0
3        76.0                 0                 0
0
4        84.0                 0                 0
0

   Hair loss(Y/N)  Pimples(Y/N)  Fast food (Y/N)  Reg.Exercise(Y/N)  \
0               0             0              1.0                  0
1               0             0              0.0                  0
2               1             1              1.0                  0
3               0             0              0.0                  0
4               1             0              0.0                  0
```

```
   BP _Systolic (mmHg)  BP _Diastolic (mmHg)  Follicle No. (L)  \
0                  110                    80                 3
1                  120                    70                 3
2                  120                    80                13
3                  120                    70                 2
4                  120                    80                 3

   Follicle No. (R)  Avg. F size (L) (mm)  Avg. F size (R) (mm)  \
0                 3                  18.0                  18.0
1                 5                  15.0                  14.0
2                15                  18.0                  20.0
3                 2                  15.0                  14.0
4                 4                  16.0                  14.0

   Endometrium (mm)
0               8.5
1               3.7
2              10.0
3               7.5
4               7.0
```

Drop 'PCOS (Y/N)' column as This is indicator not feature

```
# Drop the 'PCOS (Y/N)' column
data_features = data_short.drop(columns=['PCOS (Y/N)'])

# Display the first few rows of the updated dataset to confirm the
column has been dropped
display(data_features.head())
```

```
   Sl. No  Patient File No.  Age (yrs)  Weight (Kg)  Height(Cm)    BMI
\
0       1                 1         28         44.6       152.0  19.30

1       2                 2         36         65.0       161.5  24.92

2       3                 3         33         68.8       165.0  25.27

3       4                 4         37         65.0       148.0  29.67

4       5                 5         25         52.0       161.0  20.06


   Blood Group  Pulse rate(bpm)  RR (breaths/min)  Hb(g/dl)
Cycle(R/I)  \
0           15               78                22     10.48
2
1           15               74                20     11.70
2
2           11               72                18     11.80
2
```

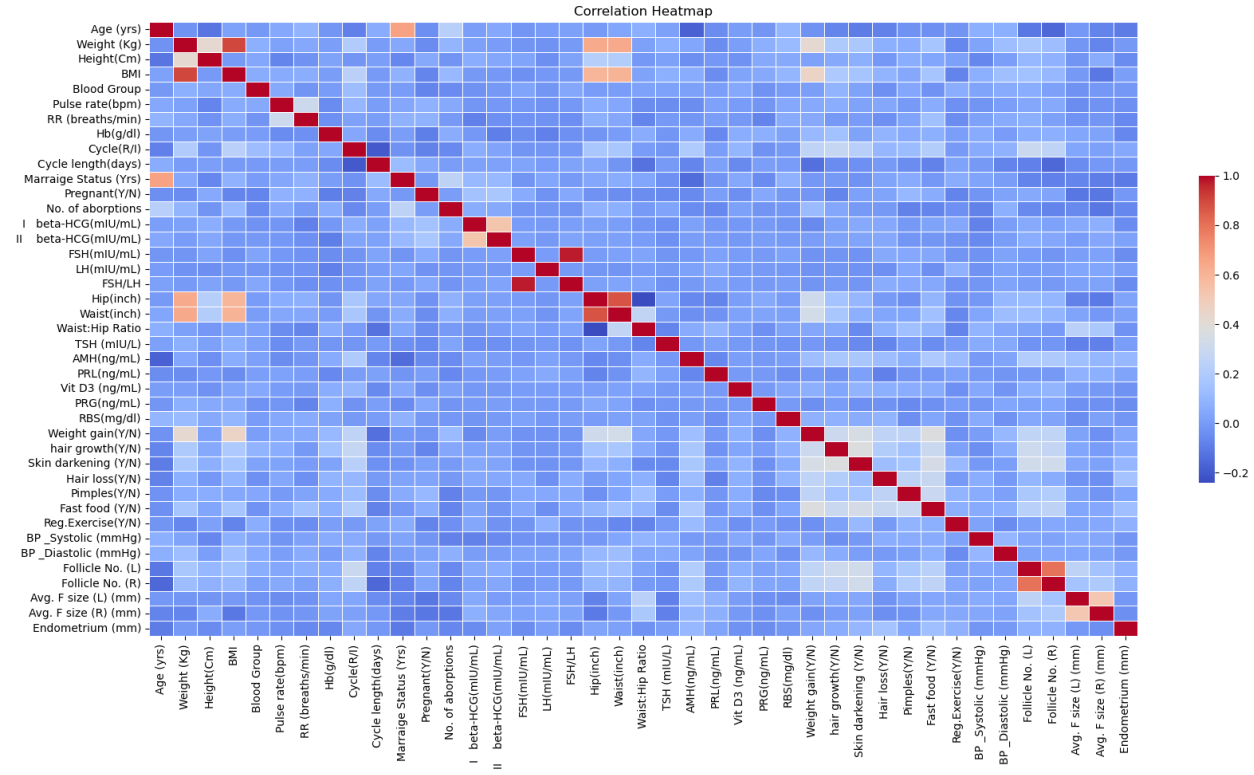
```
3           13               72                20     12.00
2
4           11               72                18     10.00
2

   Cycle length(days)  Marraige Status (Yrs)  Pregnant(Y/N)  \
0                   5                    7.0              0
1                   5                   11.0              1
2                   5                   10.0              1
3                   5                    4.0              0
4                   5                    1.0              1

   No. of aborptions  I   beta-HCG(mIU/mL)  II    beta-HCG(mIU/mL)  \
0                  0                 1.99                    1.99
1                  0                60.80                    1.99
2                  0               494.08                  494.08
3                  0                 1.99                    1.99
4                  0               801.45                  801.45

   FSH(mIU/mL)  LH(mIU/mL)  FSH/LH  Hip(inch)  Waist(inch)  Waist:Hip
Ratio  \
0         7.95        3.68    2.16         36           30
0.83
1         6.73        1.09    6.17         38           32
0.84
2         5.54        0.88    6.30         40           36
0.90
3         8.06        2.36    3.42         42           36
0.86
4         3.98        0.90    4.42         37           30
0.81

   TSH (mIU/L)  AMH(ng/mL)  PRL(ng/mL)  Vit D3 (ng/mL)  PRG(ng/mL)  \
0         0.68        2.07       45.16            17.1        0.57
1         3.16        1.53       20.09            61.3        0.97
2         2.54        6.63       10.52            49.7        0.36
3        16.41        1.22       36.90            33.4        0.36
4         3.57        2.26       30.09            43.8        0.38

   RBS(mg/dl)  Weight gain(Y/N)  hair growth(Y/N)  Skin darkening
(Y/N)  \
0        92.0                 0                 0
0
1        92.0                 0                 0
0
2        84.0                 0                 0
0
3        76.0                 0                 0
0
4        84.0                 0                 0
```

```
0

   Hair loss(Y/N)  Pimples(Y/N)  Fast food (Y/N)  Reg.Exercise(Y/N)  \
0               0             0              1.0                  0
1               0             0              0.0                  0
2               1             1              1.0                  0
3               0             0              0.0                  0
4               1             0              0.0                  0

   BP _Systolic (mmHg)  BP _Diastolic (mmHg)  Follicle No. (L)  \
0                  110                    80                 3
1                  120                    70                 3
2                  120                    80                13
3                  120                    70                 2
4                  120                    80                 3

   Follicle No. (R)  Avg. F size (L) (mm)  Avg. F size (R) (mm)  \
0                 3                  18.0                  18.0
1                 5                  15.0                  14.0
2                15                  18.0                  20.0
3                 2                  15.0                  14.0
4                 4                  16.0                  14.0

   Endometrium (mm)
0               8.5
1               3.7
2              10.0
3               7.5
4               7.0
```

Check correlation through Heatmap

```
data1=data_features.copy()

# Remove the "Sl No" column if it exists
if 'Sl. No' in data1.columns:
    data1.drop(columns=['Sl. No'], inplace=True)

# Remove Identification column "Patient File No."
data1.drop(columns=['Patient File No.'], inplace=True)

# Compute the correlation matrix
corr_matrix = data1.corr()

# Set up the matplotlib figure
plt.figure(figsize=(20, 10))

# Draw the heatmap without masking and without annotations
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm', linewidths=.5,
cbar_kws={"shrink": .5})
```

```
# Add title
plt.title('Correlation Heatmap')
plt.show()
```

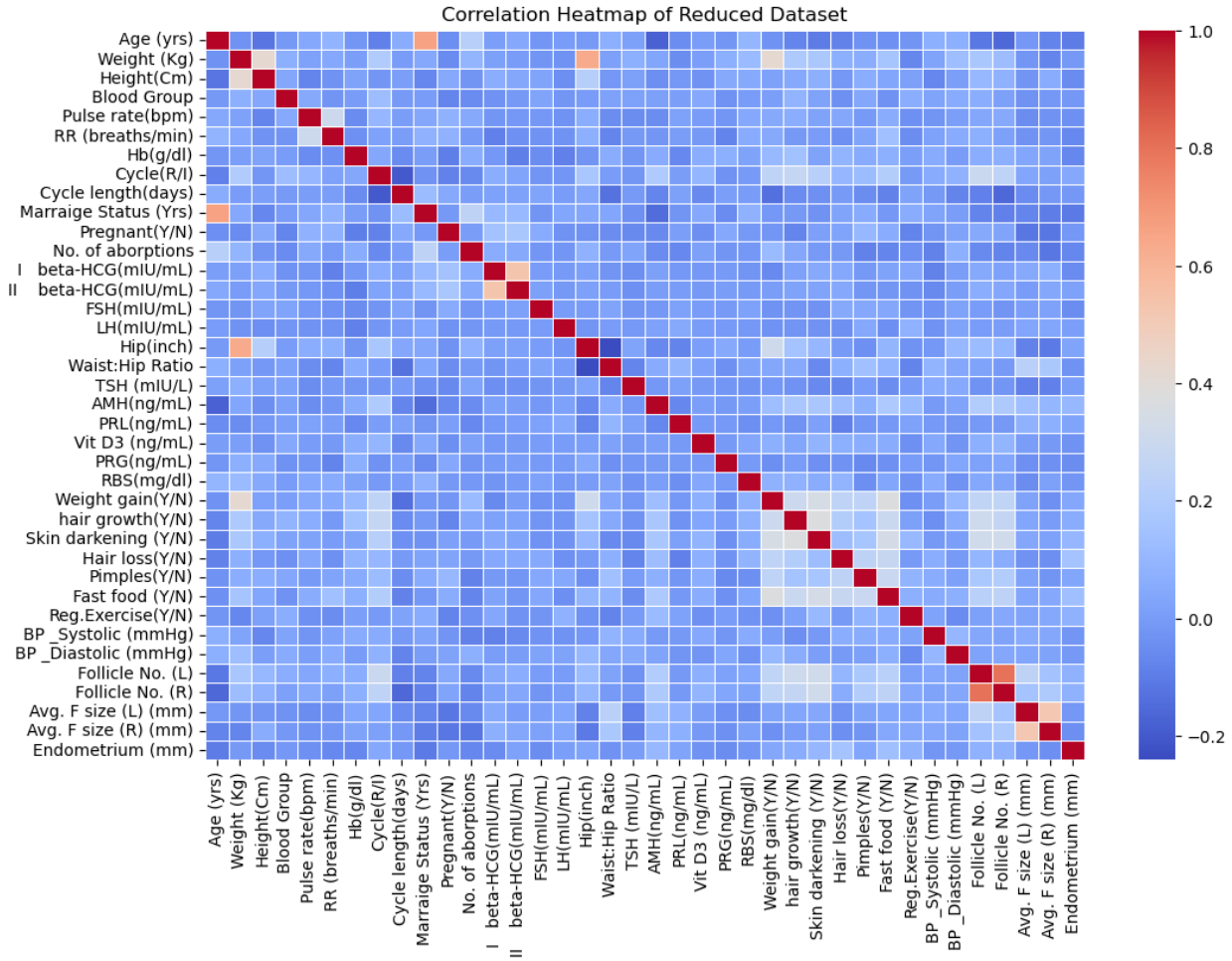



```
# Set the correlation threshold
threshold = 0.8

# Find feature pairs with correlation >= 80%
high_corr_pairs = []
for i in range(len(corr_matrix.columns)):
    for j in range(i + 1, len(corr_matrix.columns)):  # Avoid self-
correlation
        if corr_matrix.iloc[i, j] >= threshold:
            high_corr_pairs.append((corr_matrix.columns[i],
corr_matrix.columns[j], corr_matrix.iloc[i, j]))

# Print the highly correlated feature pairs
if high_corr_pairs:
    print("Feature pairs with correlation >= 80%:\n")
    for pair in high_corr_pairs:
        print(f"{pair[0]} ↔ {pair[1]}  | Correlation: {pair[2]:.2f}")
else:
    print("No highly correlated feature pairs found.")

# Decide which features to remove (keeping only one from each pair)
```

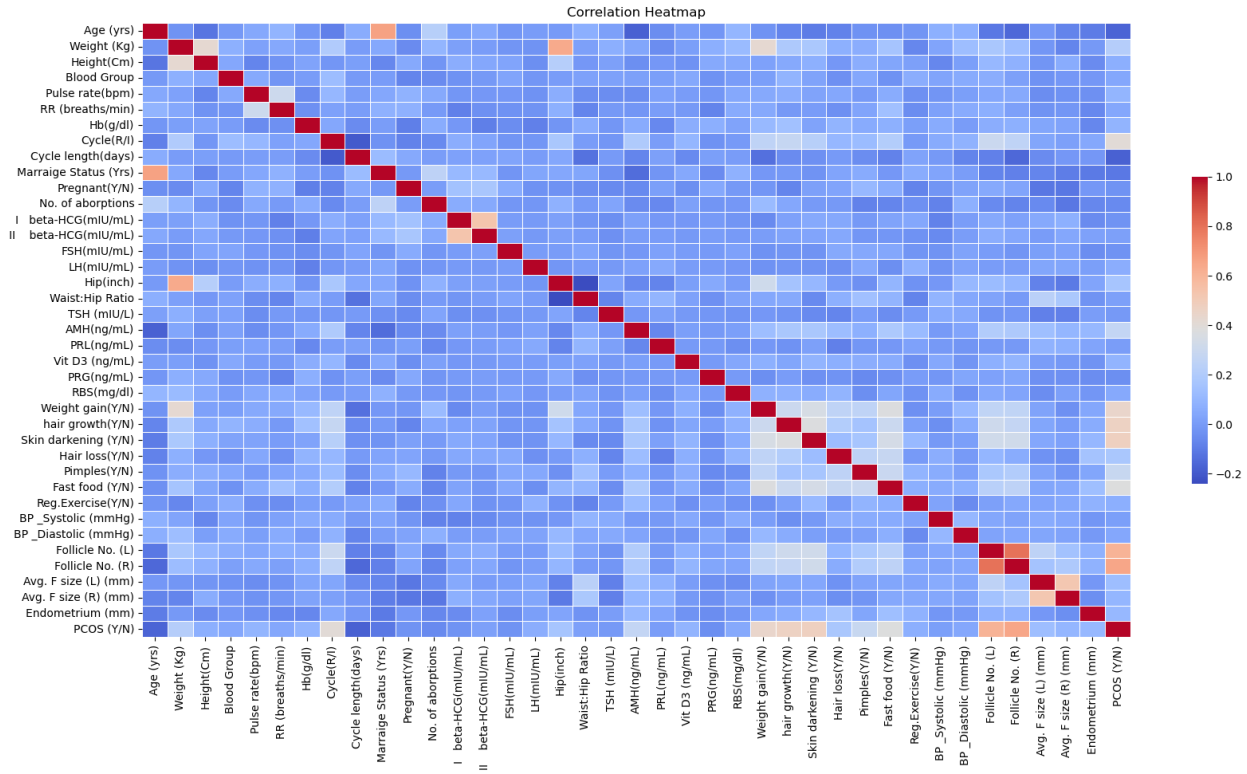
```
features_to_remove = set()
for col1, col2, _ in high_corr_pairs:
    features_to_remove.add(col2)  # Arbitrarily remove the second
feature in each pair

# Print the removed features
print("\nFeatures removed due to high correlation:")
print(", ".join(features_to_remove))

# Save total number of columns before removal
total_columns_before = data1.shape[1]

# Remove duplicate feature columns
data_reduced = data1.drop(columns=features_to_remove)

# Save total number of columns after removal
total_columns_after = data_reduced.shape[1]

# Save the cleaned dataset
data_reduced.to_csv('N_R_data.csv', index=False)

# Print summary
print(f"\nTotal columns before removal: {total_columns_before}")
print(f"Total columns after removal: {total_columns_after}")
print("Final dataset saved as 'N_R_data.csv' after removing redundant
features.")
```

```
Feature pairs with correlation >= 80%:

Weight (Kg) ↔ BMI  | Correlation: 0.90
FSH(mIU/mL) ↔ FSH/LH  | Correlation: 0.97
Hip(inch) ↔ Waist(inch)  | Correlation: 0.87

Features removed due to high correlation:
BMI, FSH/LH, Waist(inch)

Total columns before removal: 41
Total columns after removal: 38
Final dataset saved as 'N_R_data.csv' after removing redundant
features.
```

```
# **Plot heatmap for remaining features**
plt.figure(figsize=(12, 8))
sns.heatmap(data_reduced.corr(), annot=False, fmt=".2f",
cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Heatmap of Reduced Dataset")
plt.show()
```




```
# Add 'PCOS (Y/N)' column from 'data' to 'data2'

data_reduced['PCOS (Y/N)'] = original_data['PCOS (Y/N)']

# Compute the correlation matrix
corr_matrix = data_reduced.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(20, 10))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm', linewidths=.5,
cbar_kws={"shrink": .5})


plt.title('Correlation Heatmap')
plt.show()
```



Arrange features in decending correlation value

```
# Initialize a dictionary to store the features with their correlation values
correlated_features = {}

# Iterate through each column in the correlation matrix
for feature in corr_matrix.columns:
    if feature != 'PCOS (Y/N)':  # Skip the target feature itself
        corr_value = corr_matrix.loc[feature, 'PCOS (Y/N)']
        correlated_features[feature] = round(corr_value, 2)

# Sort the dictionary by the correlation values in descending order
sorted_correlated_features = dict(sorted(correlated_features.items(), key=lambda item: item[1], reverse=True))

# Print the sorted dictionary containing correlated features and their values
if sorted_correlated_features:
    print("Features sorted by correlation with 'PCOS (Y/N)':")
    for feature, corr_value in sorted_correlated_features.items():
        print(f"{feature}: {corr_value}")
else:
    print("No features found to correlate with 'PCOS (Y/N)'.")
```

```
Features sorted by correlation with 'PCOS (Y/N)':
Follicle No. (R): 0.65
```

```
Follicle No. (L): 0.6
Skin darkening (Y/N): 0.48
hair growth(Y/N): 0.46
Weight gain(Y/N): 0.44
Cycle(R/I): 0.4
Fast food (Y/N): 0.38
Pimples(Y/N): 0.29
AMH(ng/mL): 0.26
Weight (Kg): 0.21
Hair loss(Y/N): 0.17
Hip(inch): 0.16
Avg. F size (L) (mm): 0.13
Endometrium (mm): 0.11
Avg. F size (R) (mm): 0.1
Pulse rate(bpm): 0.09
Hb(g/dl): 0.09
Vit D3 (ng/mL): 0.09
Height(Cm): 0.07
Reg.Exercise(Y/N): 0.07
LH(mIU/mL): 0.06
RBS(mg/dl): 0.05
Blood Group: 0.04
RR (breaths/min): 0.04
BP _Diastolic (mmHg): 0.04
II    beta-HCG(mIU/mL): 0.01
Waist:Hip Ratio: 0.01
PRL(ng/mL): 0.01
BP _Systolic (mmHg): 0.01
TSH (mIU/L): -0.01
Pregnant(Y/N): -0.03
I   beta-HCG(mIU/mL): -0.03
FSH(mIU/mL): -0.03
PRG(ng/mL): -0.04
No. of aborptions: -0.06
Marraige Status (Yrs): -0.11
Age (yrs): -0.17
Cycle length(days): -0.18
```

Feature	Measurable by Wearables	Noninvasive but Not Measurable by Wearables	Invasive (Blood Tests, Vaginal Ultrasound, etc.)	Others (Manual Entry)	Correlation with PCOS
Pulse rate (bpm)	Smartwatches	-	-	-	0.09
RR (breaths/min)	Smartwatches	-	-	-	0.04
BP Systolic	Some	-	-	-	0.01

Feature	Measurable by Wearables	Noninvasive but Not Measurable by Wearables	Invasive (Blood Tests, Vaginal Ultrasound, etc.)	Others (Manual Entry)	Correlation with PCOS
(mmHg)	smartwatches				
BP Diastolic (mmHg)	Some smartwatches	-	-	-	0.04
Height (Cm)	Smart scale estimate	-	-	-	0.07
Weight (Kg)	Smart scale	-	-	-	0.21
Waist:Hip Ratio	Smart measuring belts	-	-	-	0.01
Reg. Exercise (Y/N)	Activity sensors	-	-	-	0.07
Fast food (Y/N)	AI-based tracking	-	-	-	0.38
Skin darkening (Y/N)	-	Can be detected via imaging	-	-	0.48
Hair loss (Y/N)	-	Can be detected via imaging	-	-	0.17
Hair growth (Y/N)	-	Can be detected via imaging	-	-	0.46
Weight gain (Y/N)	-	Can be inferred from trends	-	-	0.44
Pimples (Y/N)	-	Can be detected via imaging	-	-	0.29
Follicle No. (R)	-	Ultrasound (Noninvasive)	-	-	0.65
Follicle No. (L)	-	Ultrasound (Noninvasive)	-	-	0.6
Avg. F size (R) (mm)	-	Ultrasound (Noninvasive)	-	-	0.1
Avg. F size (L) (mm)	-	Ultrasound (Noninvasive)	-	-	0.13
Endometrium (mm)	-	Ultrasound (Noninvasive)	-	-	0.11
AMH (ng/mL)	-	-	Blood test	-	0.26

Feature	Measurable by Wearables	Noninvasive but Not Measurable by Wearables	Invasive (Blood Tests, Vaginal Ultrasound, etc.)	Others (Manual Entry)	Correlation with PCOS
Hb (g/dl)	-	-	Blood test	-	0.09
Vit D3 (ng/mL)	-	-	Blood test	-	0.09
RBS (mg/dl)	-	-	Blood test	-	0.05
LH (mIU/mL)	-	-	Blood test	-	0.06
PRL (ng/mL)	-	-	Blood test	-	0.01
FSH (mIU/mL)	-	-	Blood test	-	-0.03
PRG (ng/mL)	-	-	Blood test	-	-0.04
beta-HCG I (mIU/mL)	-	-	Blood test	-	-0.03
beta-HCG II (mIU/mL)	-	-	Blood test	-	0.01
TSH (mIU/L)	-	-	Blood test	-	-0.01
II beta-HCG (mIU/mL)	-	-	Blood test	-	0.01
Pregnant (Y/N)	-	-	-	Medical records	-0.03
No. of abortions	-	-	-	Medical records	-0.06
Marriage Status (Yrs)	-	-	-	Self-reported	-0.11
Age (yrs)	-	-	-	Self-reported	-0.17
Blood Group	-	-	-	Medical records	0.04
Hip (inch)	-	-	-	Measured manually	0.16
Cycle (R/I)	-	-	-	Self-reported	0.4
Cycle length (days)	-	-	-	Self-reported	-0.18

Measurable Features with Measuring Devices

#	Features*	Impact**	Devices Measuring This Parameter
1	Pulse Rate	1	Apple Watch, Fitbit, Empatica E4, Biovotion,

#	Features*	Impact**	Devices Measuring This Parameter
			GENEActiv, FreeStyle Libre, Dexcom G6, Muse EEG headband
2	HRV	-1	Apple Watch, Fitbit, Empatica E4, Ava Bracelet, Muse EEG headband
3	RR (Respiratory Rate)	1	Fitbit, Samsung Charm, Withings ScanWatch, WristOx2, Sense-Wear armband, Health Tags
4	BP Systolic	1	HeartGuide, Smart Wear
5	BP Diastolic	1	HeartGuide, Smart Wear
6	Glucose	1	FreeStyle Libre, Dexcom G6, iGLU 2.0
7	Skin Temp	1	Ava Bracelet, Oura Ring
8	Sleep Quality	-1	Fitbit, Withings ScanWatch, Muse EEG headband, Re-timer light device
9	Body Temp	1	Ava Bracelet, Oura Ring
10	Weight	1	Fitbit, Samsung Charm
11	Waist:Hip Ratio	1	Not explicitly mentioned in the article
12	Activity Levels	-1	Fitbit, Garmin Vivofit 2, Jawbone Wearable, Samsung Charm, Empatica E4, Biovotion, GENEActiv, McRoberts and DynaPort Hybrid System, Opal Inertial Sensor
13	Calorie Intake	1	Garmin Vivofit 2, Fitbit, Jawbone Wearable, Noom app
14	EDA (Electrodermal Activity)	1	Empatica E4

* The indicative symptoms for PCOS ** Explanation of Impact on PCOS:

1 → Higher value of the parameter increases the risk or contributes to the disease. -1 → Lower value of the parameter increases the risk or contributes to the disease.

Externally Measurable Features which have impact on PCOS which are available in our dataset

Sl No	Features	Correlation with PCOS	Wearable Devices Available
1	Pulse rate (bpm)	0.09	Apple Watch, Fitbit, Empatica E4, Biovotion, GENEActiv, FreeStyle Libre, Dexcom G6, Muse EEG headband
2	RR (breaths/min)	0.04	Fitbit, Samsung Charm, Withings ScanWatch, WristOx2, Sense-Wear armband, Health Tags
3	BP _Systolic (mmHg)	0.01	HeartGuide, Smart Wear
4	BP _Diastolic (mmHg)	0.04	HeartGuide, Smart Wear
5	Waist:Hip Ratio	0.01	Not explicitly mentioned

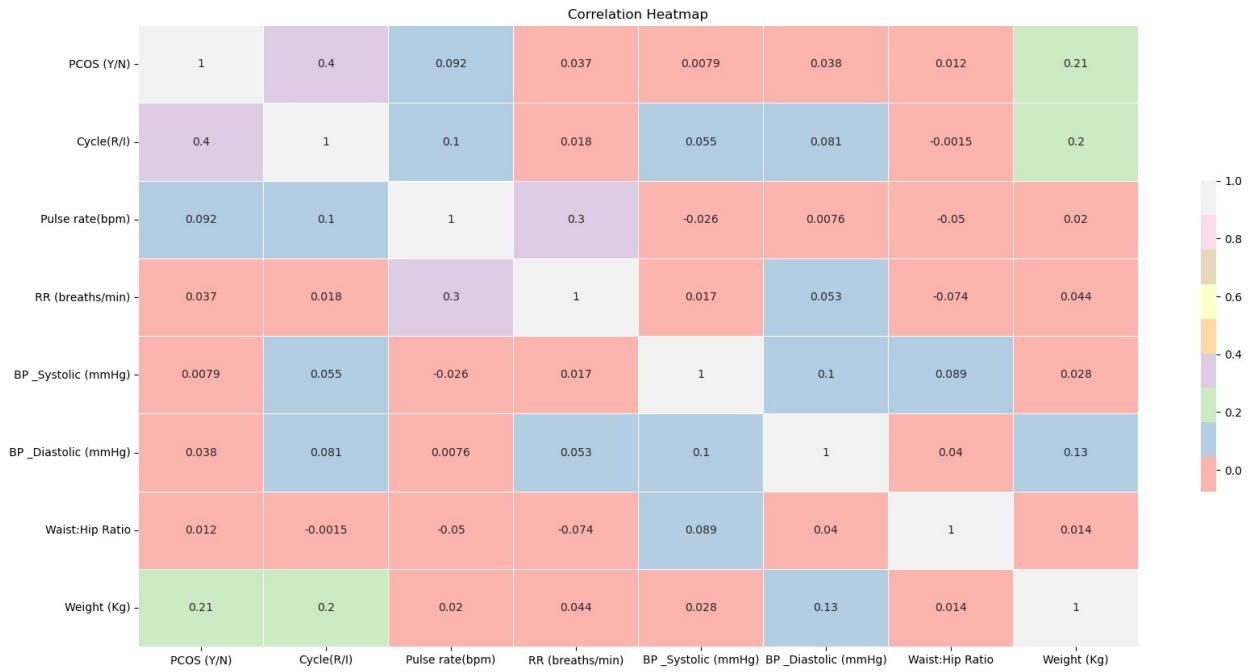
Sl No	Features	Correlation with PCOS	Wearable Devices Available
6	Weight (Kg)	0.21	Fitbit, Samsung Charm
7	Age (yrs)	-0.17	Self-reported

We have selected only the above `Six` Features and One $Manual$ input Feature `Cycle(R/I)` to prepare a final dataset for Model designing

```
file_path = 'PCOS_Likelihood.csv'
original_data = pd.read_csv(file_path)


# Import necessary library
import pandas as pd

# Load the dataset (assuming it's a CSV file)
data_reduced = pd.read_csv('N_R_data.csv')
data_reduced['PCOS (Y/N)'] = original_data['PCOS (Y/N)']

# Selecting required columns
selected_columns = ['PCOS (Y/N)', 'Cycle(R/I)', 'Pulse rate(bpm)', 'RR
(breaths/min)',
                    'BP _Systolic (mmHg)', 'BP _Diastolic (mmHg)',
                    'Waist:Hip Ratio', 'Weight (Kg)']


# Extracting the selected columns
data_filtered = data_reduced[selected_columns]


# Display the first few rows
display(data_filtered.head())

# Display summary statistics
data_filtered.describe()
```

```
   PCOS (Y/N)  Cycle(R/I)  Pulse rate(bpm)  RR (breaths/min)  \
0           0           2               78                22
1           0           2               74                20
2           1           2               72                18
3           0           2               72                20
4           0           2               72                18

   BP _Systolic (mmHg)  BP _Diastolic (mmHg)  Waist:Hip Ratio  Weight
(Kg)
0                  110                    80             0.83
44.6
1                  120                    70             0.84
65.0
2                  120                    80             0.90
68.8
```

```
3                   120                    70              0.86
65.0
4                   120                    80              0.81
52.0

       PCOS (Y/N)  Cycle(R/I)  Pulse rate(bpm)  RR (breaths/min)  \
count  541.000000  541.000000       541.000000        541.000000
mean     0.327172    2.560074        73.247689         19.243993
std      0.469615    0.901950         4.430285          1.688629
min      0.000000    2.000000        13.000000         16.000000
25%      0.000000    2.000000        72.000000         18.000000
50%      0.000000    2.000000        72.000000         18.000000
75%      1.000000    4.000000        74.000000         20.000000
max      1.000000    5.000000        82.000000         28.000000

       BP _Systolic (mmHg)  BP _Diastolic (mmHg)  Waist:Hip Ratio
Weight (Kg)
count           541.000000            541.000000       541.000000
541.000000
mean            114.661738             76.927911         0.891627
59.637153
std               7.384556              5.574112         0.046135
11.028287
min              12.000000              8.000000         0.760000
31.000000
25%             110.000000             70.000000         0.860000
52.000000
50%             110.000000             80.000000         0.890000
59.000000
75%             120.000000             80.000000         0.930000
65.000000
max             140.000000            100.000000         0.980000
108.000000

# Compute the correlation matrix
corr_matrix = data_filtered.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(20, 10))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_matrix, annot=True, cmap='Pastel1', linewidths=.5,
cbar_kws={"shrink": .5})


plt.title('Correlation Heatmap')
plt.show()
```



Measurable:

'Pulse rate(bpm)',

'RR (breaths/min)',

'BP _Systolic (mmHg)',

'BP _Diastolic (mmHg)',

'Waist:Hip Ratio',

'Weight (Kg)',

'Weight gain(Y/N)'

Self-reported:

'Skin darkening (Y/N)',

'hair growth(Y/N)',

'Cycle(R/I)',

'Age (yrs)',

'Cycle length (days)'

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import f_oneway

from sklearn.metrics import balanced_accuracy_score
from sklearn.decomposition import PCA
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import VarianceThreshold


from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, classification_report

from scipy.stats.mstats import winsorize
```

---------- Read and Process Data ----------

```
# Read main file
file_path = 'PCOS_Likelihood.csv'
original_data = pd.read_csv(file_path)

# Print all column names
print("=== Column Names in Dataset ===")
print(original_data.columns.tolist())  # Print column names as a list
```

```
=== Column Names in Dataset ===
['Sl. No', 'Patient File No.', 'PCOS (Y/N)', ' Age (yrs)', 'Weight
(Kg)', 'Height(Cm) ', 'BMI', 'Blood Group', 'Pulse rate(bpm) ', 'RR
(breaths/min)', 'Hb(g/dl)', 'Cycle(R/I)', 'Cycle length(days)',
'Marraige Status (Yrs)', 'Pregnant(Y/N)', 'No. of aborptions', '  I
beta-HCG(mIU/mL)', 'II    beta-HCG(mIU/mL)', 'FSH(mIU/mL)',
'LH(mIU/mL)', 'FSH/LH', 'Hip(inch)', 'Waist(inch)', 'Waist:Hip Ratio',
'TSH (mIU/L)', 'AMH(ng/mL)', 'PRL(ng/mL)', 'Vit D3 (ng/mL)',
'PRG(ng/mL)', 'RBS(mg/dl)', 'Weight gain(Y/N)', 'hair growth(Y/N)',
'Skin darkening (Y/N)', 'Hair loss(Y/N)', 'Pimples(Y/N)', 'Fast food
(Y/N)', 'Reg.Exercise(Y/N)', 'BP _Systolic (mmHg)', 'BP _Diastolic
(mmHg)', 'Follicle No. (L)', 'Follicle No. (R)', 'Avg. F size (L)
(mm)', 'Avg. F size (R) (mm)', 'Endometrium (mm)']
```

```
# Load the preprocessed dataset
data_reduced = pd.read_csv('N_R_data.csv')


# Print all column names
print("=== Column Names in Dataset ===")
print(data_reduced.columns.tolist())  # Print column names as a list
```

```
=== Column Names in Dataset ===
['Age (yrs)', 'Weight (Kg)', 'Height(Cm)', 'Blood Group', 'Pulse
rate(bpm)', 'RR (breaths/min)', 'Hb(g/dl)', 'Cycle(R/I)', 'Cycle
length(days)', 'Marraige Status (Yrs)', 'Pregnant(Y/N)', 'No. of
aborptions', 'I   beta-HCG(mIU/mL)', 'II    beta-HCG(mIU/mL)',
'FSH(mIU/mL)', 'LH(mIU/mL)', 'Hip(inch)', 'Waist:Hip Ratio', 'TSH
(mIU/L)', 'AMH(ng/mL)', 'PRL(ng/mL)', 'Vit D3 (ng/mL)', 'PRG(ng/mL)',
'RBS(mg/dl)', 'Weight gain(Y/N)', 'hair growth(Y/N)', 'Skin darkening
(Y/N)', 'Hair loss(Y/N)', 'Pimples(Y/N)', 'Fast food (Y/N)',
'Reg.Exercise(Y/N)', 'BP _Systolic (mmHg)', 'BP _Diastolic (mmHg)',
'Follicle No. (L)', 'Follicle No. (R)', 'Avg. F size (L) (mm)', 'Avg.
F size (R) (mm)', 'Endometrium (mm)']



# Load the preprocessed dataset
data_reduced['PCOS (Y/N)'] = original_data['PCOS (Y/N)']

# Selecting required columns
selected_columns = ['PCOS (Y/N)', 'Pulse rate(bpm)', 'RR
(breaths/min)',
                    'BP _Systolic (mmHg)', 'BP _Diastolic (mmHg)',
                    'Waist:Hip Ratio', 'Weight (Kg)', 'Weight
gain(Y/N)',
                    'Skin darkening (Y/N)', 'hair growth(Y/N)',
'Cycle(R/I)',
                    'Age (yrs)', 'Cycle length(days)']


# Extracting the selected columns
data_filtered = data_reduced[selected_columns]


# Display the first few rows
display(data_filtered.head())

# Display summary statistics
data_filtered.describe()
```

	PCOS (Y/N)	Pulse rate(bpm)	RR (breaths/min)	BP _Systolic (mmHg)
0	0	78	22	110
1	0	74	20	120
2	1	72	18	120
3	0	72	20	120
4	0	72	18	120

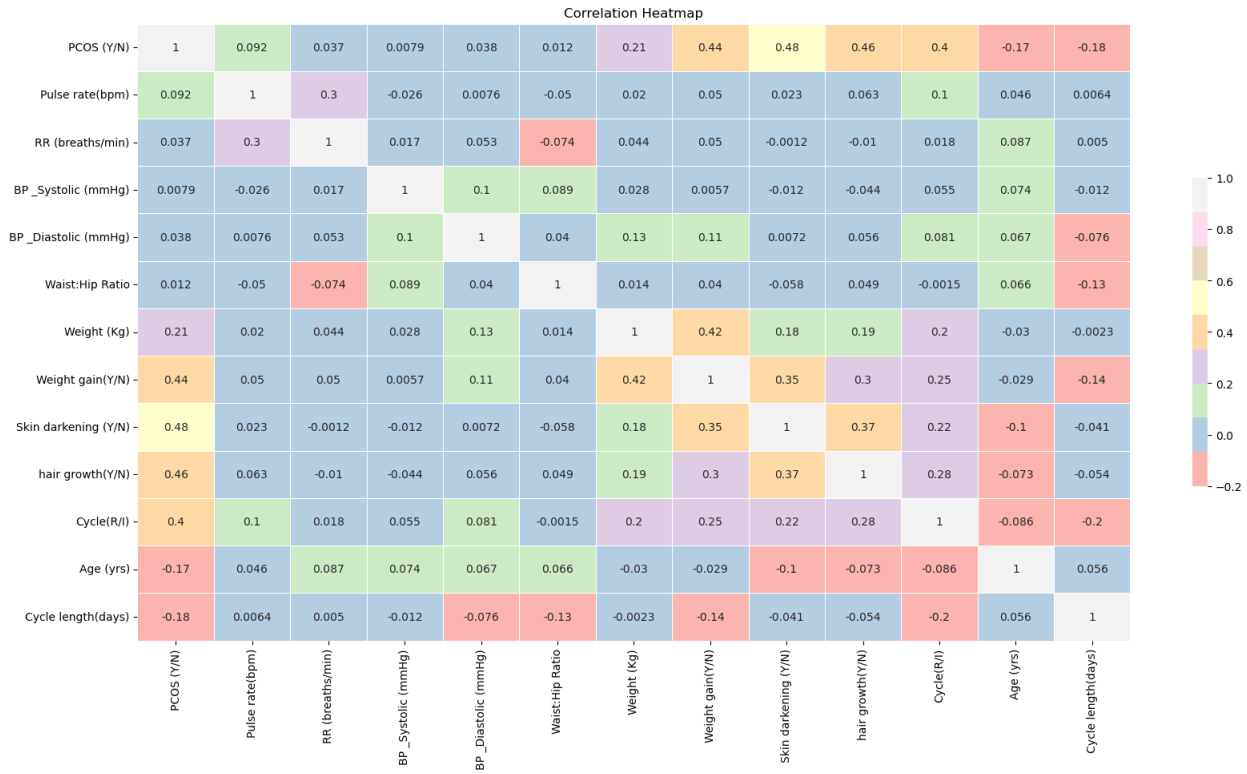
```
   BP _Diastolic (mmHg)  Waist:Hip Ratio  Weight (Kg)  Weight
gain(Y/N)  \
0                     80             0.83         44.6
0
1                     70             0.84         65.0
0
2                     80             0.90         68.8
0
3                     70             0.86         65.0
0
4                     80             0.81         52.0
0

   Skin darkening (Y/N)  hair growth(Y/N)  Cycle(R/I)  Age (yrs)  \
0                     0                 0           2         28
1                     0                 0           2         36
2                     0                 0           2         33
3                     0                 0           2         37
4                     0                 0           2         25

   Cycle length(days)
0                   5
1                   5
2                   5
3                   5
4                   5

       PCOS (Y/N)  Pulse rate(bpm)  RR (breaths/min)  BP _Systolic
(mmHg)  \
count  541.000000       541.000000        541.000000
541.000000
mean     0.327172        73.247689         19.243993
114.661738
std      0.469615         4.430285          1.688629
7.384556
min      0.000000        13.000000         16.000000
12.000000
25%      0.000000        72.000000         18.000000
110.000000
50%      0.000000        72.000000         18.000000
110.000000
75%      1.000000        74.000000         20.000000
120.000000
max      1.000000        82.000000         28.000000
140.000000

       BP _Diastolic (mmHg)  Waist:Hip Ratio  Weight (Kg)  Weight
gain(Y/N)  \
count            541.000000       541.000000   541.000000
```

```
541.000000
mean               76.927911          0.891627    59.637153
0.377079
std                 5.574112          0.046135    11.028287
0.485104
min                 8.000000          0.760000    31.000000
0.000000
25%                70.000000          0.860000    52.000000
0.000000
50%                80.000000          0.890000    59.000000
0.000000
75%                80.000000          0.930000    65.000000
1.000000
max               100.000000          0.980000   108.000000
1.000000

       Skin darkening (Y/N)  hair growth(Y/N)  Cycle(R/I)   Age (yrs)
\
count             541.000000        541.000000  541.000000  541.000000

mean                0.306839          0.273567    2.560074   31.430684

std                 0.461609          0.446202    0.901950    5.411006

min                 0.000000          0.000000    2.000000   20.000000

25%                 0.000000          0.000000    2.000000   28.000000

50%                 0.000000          0.000000    2.000000   31.000000

75%                 1.000000          1.000000    4.000000   35.000000

max                 1.000000          1.000000    5.000000   48.000000


       Cycle length(days)
count           541.00000
mean              4.94085
std               1.49202
min               0.00000
25%               4.00000
50%               5.00000
75%               5.00000
max              12.00000
```

---------- Check Correlation ----------

```
# Compute the correlation matrix
corr_matrix = data_filtered.corr()
```

```
# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(20, 10))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_matrix, annot=True, cmap="Pastel1", linewidths=.5,
cbar_kws={"shrink": .5})   # cmap='coolwarm'


plt.title('Correlation Heatmap')
plt.show()
```



---------- Outliers Detection and Removal ----------

```
data = data_filtered.copy()

# Dictionary to store outliers
outliers_quantile = {}

# Detect outliers using IQR (Interquartile Range)
for col in data.columns:
        # Compute IQR
        Q1 = data[col].quantile(0.25)
```

```
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1

        # Identify outliers
        outlier_indices_quantile = data[
            (data[col] < (Q1 - 1.5 * IQR)) | (data[col] > (Q3 + 1.5 *
IQR))
        ].index

        if len(outlier_indices_quantile) > 0:
            outliers_quantile[col] = outlier_indices_quantile

# Display the results
features_with_outliers = {col: len(indices) for col, indices in
outliers_quantile.items()}
print("\nFeatures with Outliers (as per quantile):",
features_with_outliers)
```

```
Features with Outliers (as per quantile): {'Pulse rate(bpm)': 94, 'RR
(breaths/min)': 14, 'BP _Systolic (mmHg)': 3, 'BP _Diastolic (mmHg)':
2, 'Weight (Kg)': 18, 'Age (yrs)': 5, 'Cycle length(days)': 77}
```

```
# Total number of rows in the dataset
total_rows = len(data)

# Dictionary to store the percentage of outliers
outliers_percentage = {
    col: (len(indices) / total_rows) * 100 for col, indices in
outliers_quantile.items()
}

# Display the percentage of outliers
print("\nPercentage of Outliers per Feature:")
for feature, percentage in outliers_percentage.items():
    print(f"{feature}: {percentage:.2f}%")
```

```
Percentage of Outliers per Feature:
Pulse rate(bpm): 17.38%
RR (breaths/min): 2.59%
BP _Systolic (mmHg): 0.55%
BP _Diastolic (mmHg): 0.37%
Weight (Kg): 3.33%
Age (yrs): 0.92%
Cycle length(days): 14.23%
```

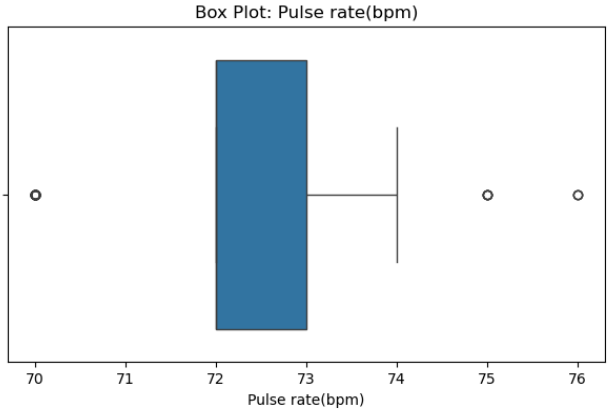
```
# List of features to remove outliers from
features_with_outliers = outliers_quantile.keys()

# Function to remove outliers using IQR
```

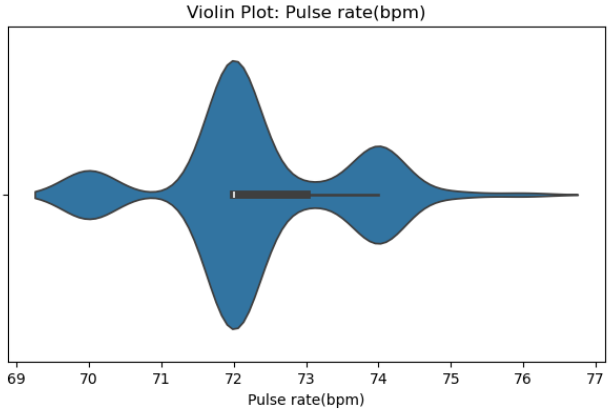
```
def remove_outliers(df, feature):
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Remove rows where feature value is outside bounds
    filtered_df = df[(df[feature] >= lower_bound) & (df[feature] <=
upper_bound)]

    print(f"Removed {len(df) - len(filtered_df)} outliers from
{feature}")
    return filtered_df

# Apply outlier removal to all selected features
for feature in features_with_outliers:
    data = remove_outliers(data, feature)

# Display remaining data shape after outlier removal
print(f"\nFinal dataset shape: {data.shape}")
```

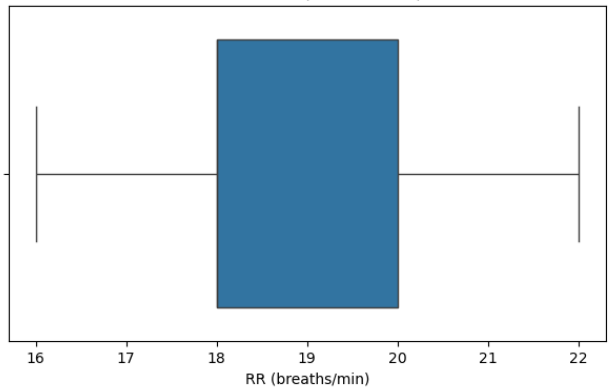
```
Removed 94 outliers from Pulse rate(bpm)
Removed 6 outliers from RR (breaths/min)
Removed 2 outliers from BP _Systolic (mmHg)
Removed 1 outliers from BP _Diastolic (mmHg)
Removed 14 outliers from Weight (Kg)
Removed 1 outliers from Age (yrs)
Removed 50 outliers from Cycle length(days)

Final dataset shape: (373, 13)
```

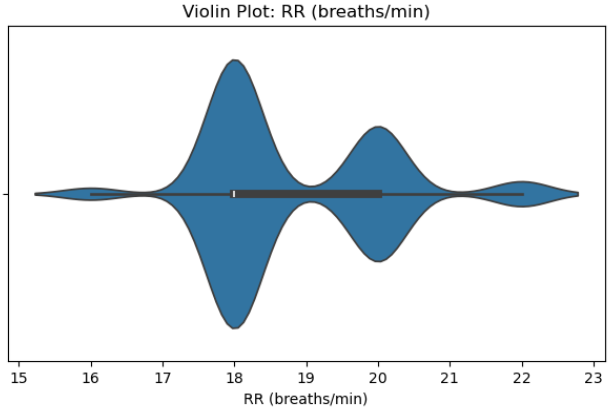
```
# Number of rows for subplots (one row per feature)
num_rows = len(features_with_outliers)

# Creating subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(12, num_rows * 4))
axes = axes.reshape(num_rows, 2)  # Ensure it's 2D for indexing

for i, feature in enumerate(features_with_outliers):
    # Box Plot
    sns.boxplot(data=data, x=feature, ax=axes[i, 0])
    axes[i, 0].set_title(f'Box Plot: {feature}')

    # Violin Plot
    sns.violinplot(data=data, x=feature, ax=axes[i, 1])
    axes[i, 1].set_title(f'Violin Plot: {feature}')

# Adjust layout
```

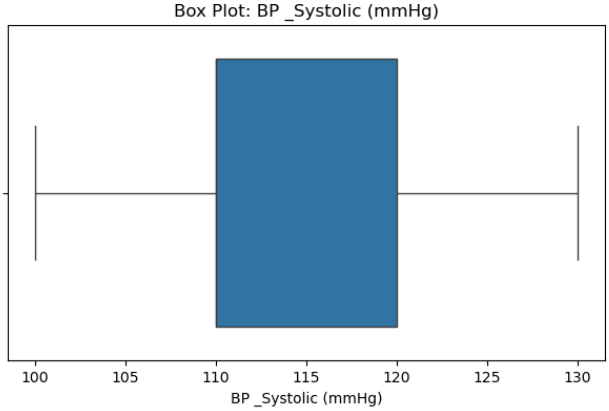
```
plt.tight_layout()
plt.show()
```

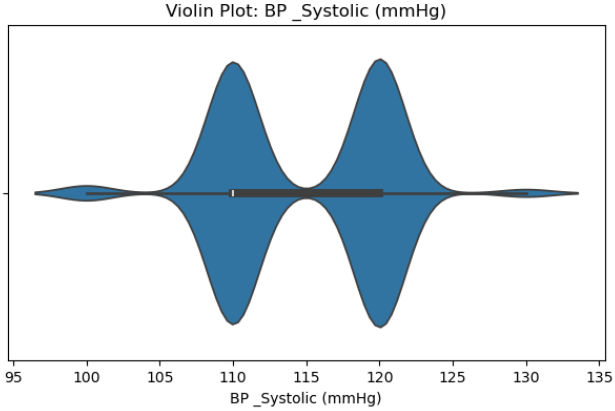

Box Plot: Pulse rate(bpm)
70
71
72
73
74
75
76
Pulse rate(bpm)

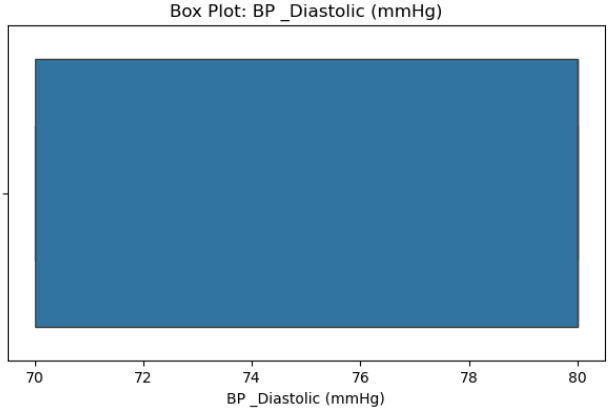

Violin Plot: Pulse rate(bpm)
69
70
71
72
73
74
75
76
77
Pulse rate(bpm)

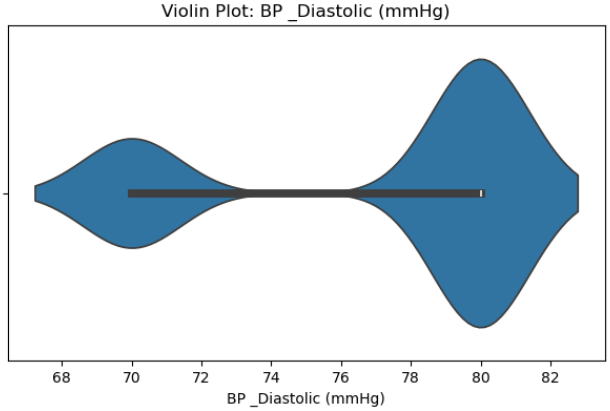

Box Plot: RR (breaths/min)
16
17
18
19
20
21
22
RR (breaths/min)

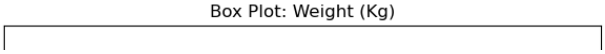

Violin Plot: RR (breaths/min)
15
16
17
18
19
20
21
22
23
RR (breaths/min)

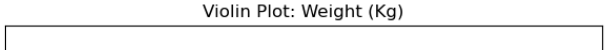

Box Plot: BP _Systolic (mmHg)
100
105
110
115
120
125
130
BP _Systolic (mmHg)


Violin Plot: BP _Systolic (mmHg)
95
100
105
110
115
120
125
130
135
BP _Systolic (mmHg)


Box Plot: BP _Diastolic (mmHg)
70
72
74
76
78
80
BP _Diastolic (mmHg)


Violin Plot: BP _Diastolic (mmHg)
68
70
72
74
76
78
80
82
BP _Diastolic (mmHg)


Box Plot: Weight (Kg)


Violin Plot: Weight (Kg)

---------- Check Modified Data Head ----------

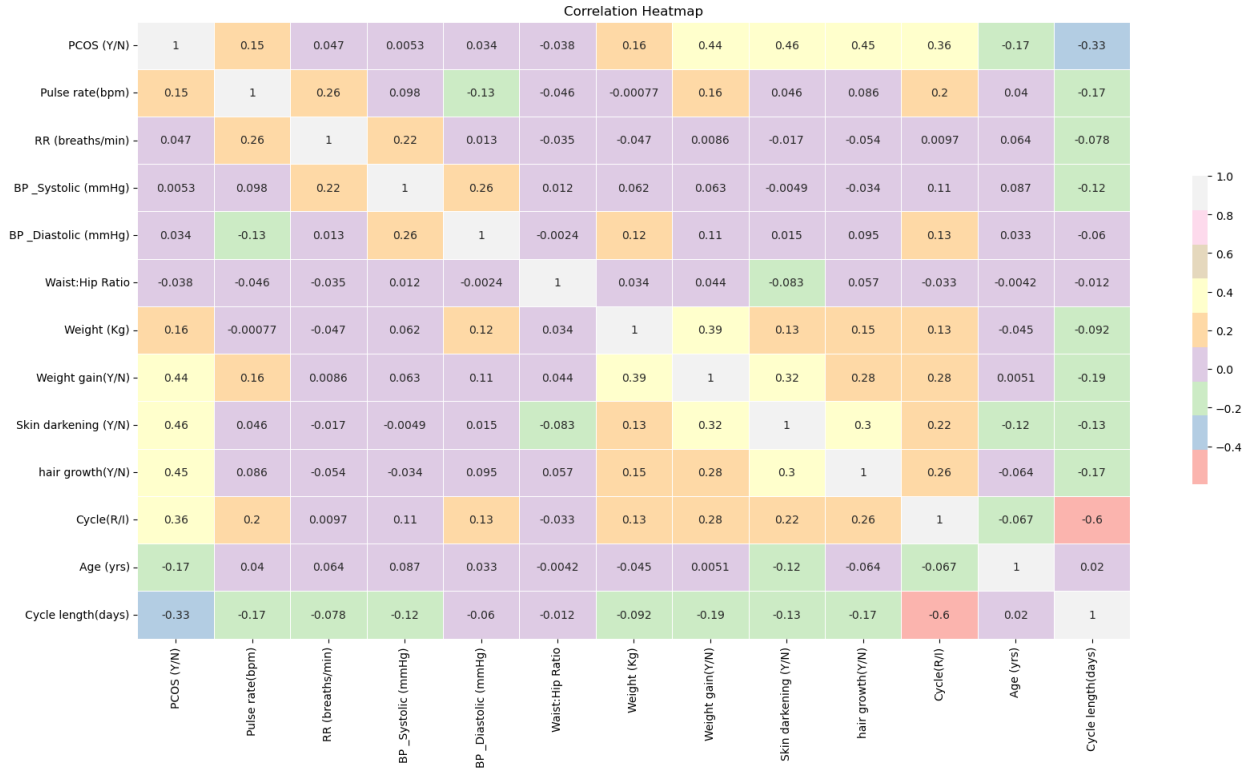
```
display(data.head())
data.describe()
data.to_csv('mix_12_featires_outlier_removed_data.csv', index= False)
```

```
   PCOS (Y/N)  Pulse rate(bpm)  RR (breaths/min)  BP _Systolic (mmHg)
\
1           0               74                20                  120

2           1               72                18                  120

3           0               72                20                  120

4           0               72                18                  120

6           0               72                18                  120


   BP _Diastolic (mmHg)  Waist:Hip Ratio  Weight (Kg)  Weight
gain(Y/N)  \
1                    70             0.84         65.0
0
2                    80             0.90         68.8
0
3                    70             0.86         65.0
0
4                    80             0.81         52.0
0
6                    80             0.85         64.0
0

   Skin darkening (Y/N)  hair growth(Y/N)  Cycle(R/I)  Age (yrs)  \
1                     0                 0           2         36
2                     0                 0           2         33
3                     0                 0           2         37
4                     0                 0           2         25
6                     0                 0           2         34

   Cycle length(days)
1                   5
2                   5
3                   5
4                   5
6                   5
```

---------- Check Correlation ----------

```
data = pd.read_csv('mix_12_featires_outlier_removed_data.csv')

# Compute the correlation matrix
```

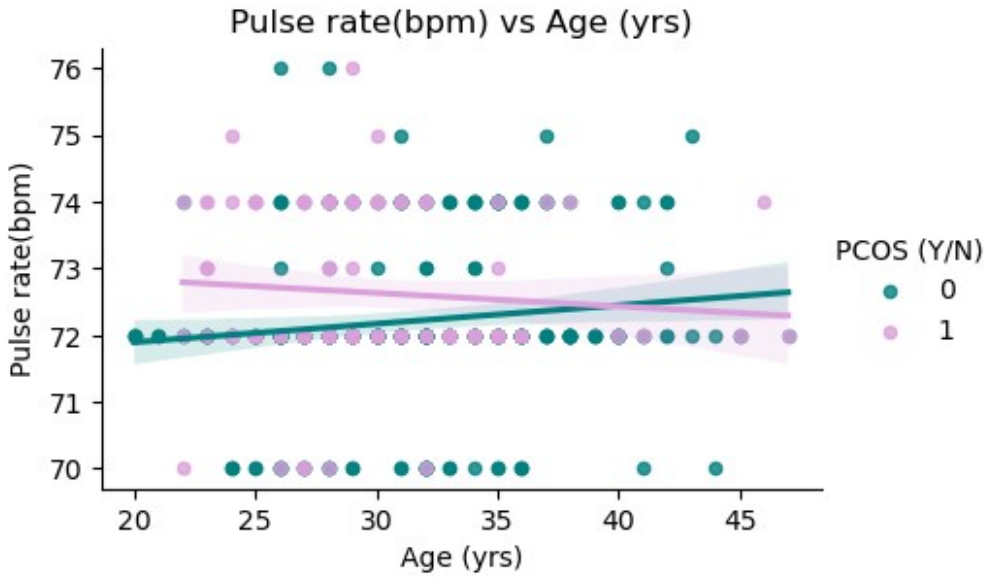
```
corr_matrix = data.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(20, 10))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_matrix, annot=True, cmap="Pastel1", linewidths=.5,
cbar_kws={"shrink": .5})   # cmap='coolwarm'


plt.title('Correlation Heatmap')
plt.show()
```

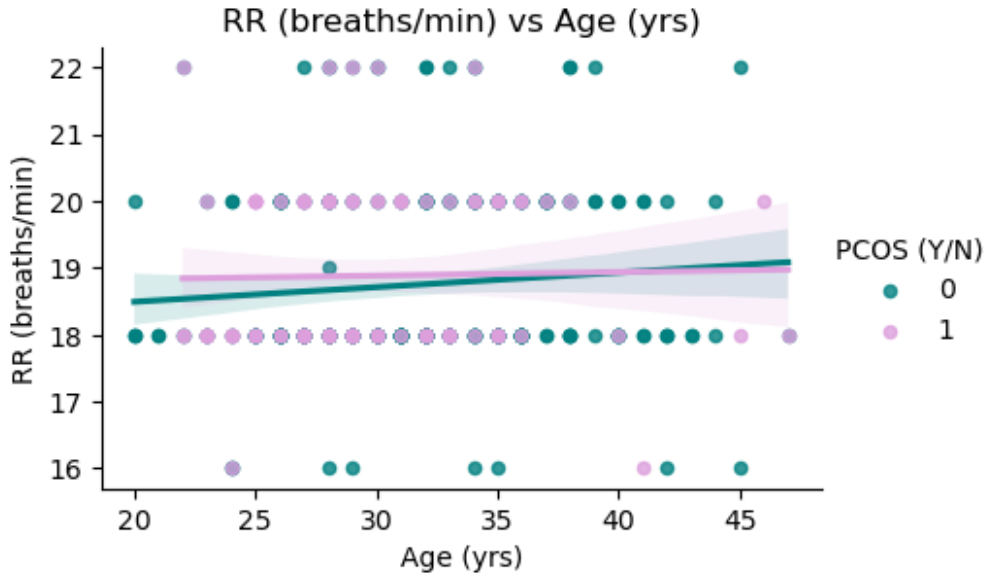


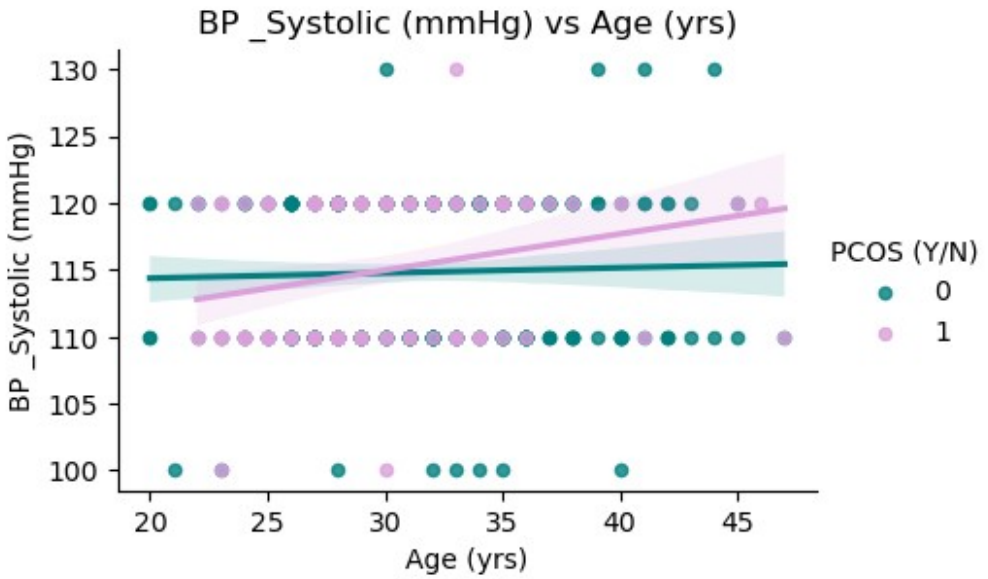
---------- Patterns Of Feature Distribution ----------

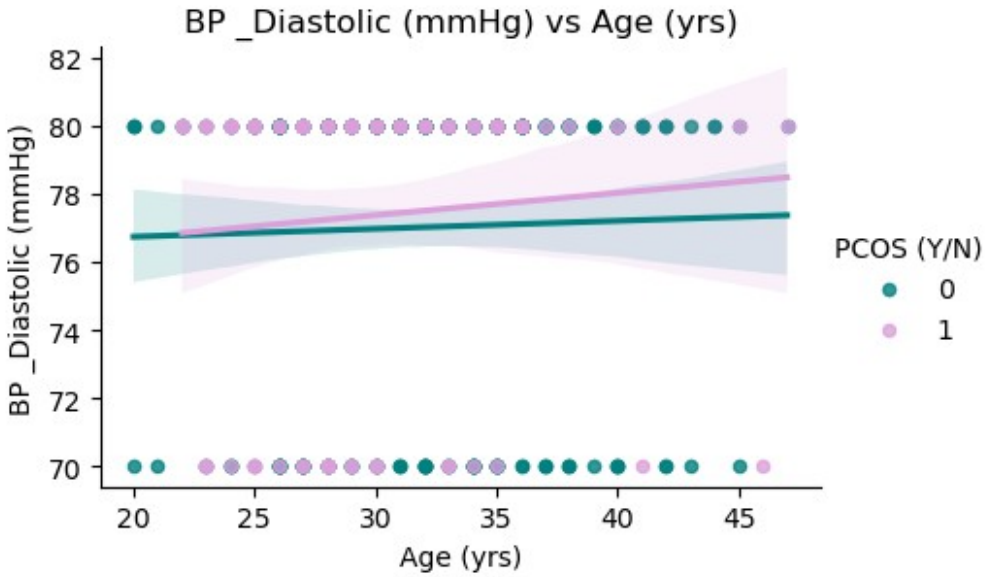
```
import seaborn as sns
import matplotlib.pyplot as plt

color = ["teal", "plum"]
count_column = data.columns.drop("PCOS (Y/N)", errors='ignore')

# Loop through each feature and create separate lmplots
for feature in count_column:
```

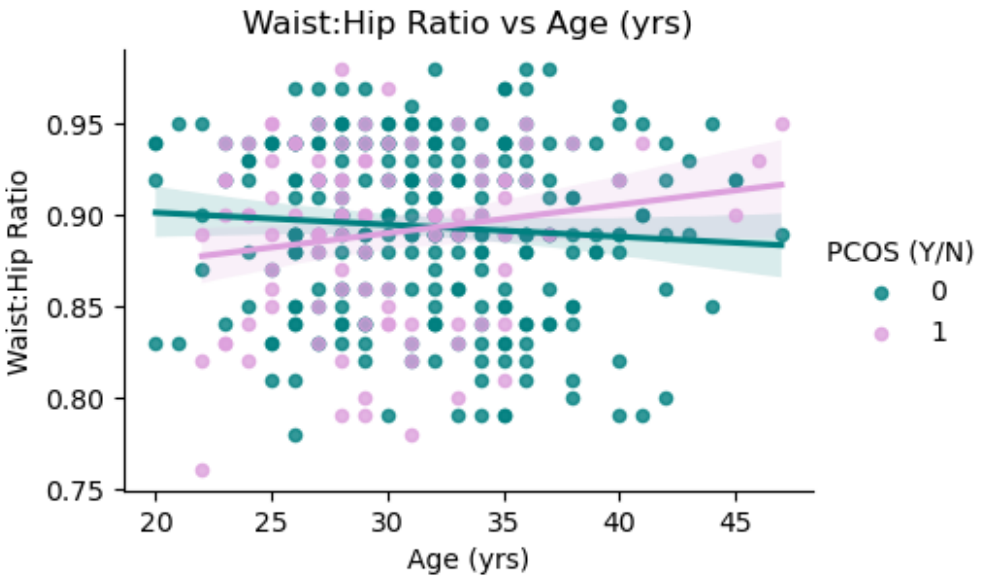
```
    if feature != "Age (yrs)":
        if feature != "PCOS (Y/N)":
            g = sns.lmplot(data=data,
                   x="Age (yrs)", y=feature, hue="PCOS (Y/N)",
                   palette=color, height=3, aspect=1.5,
                   scatter_kws={"s": 20}
                  )
            g.set(title=f"{feature} vs Age (yrs)")
            plt.show()  # Show each plot separately
```

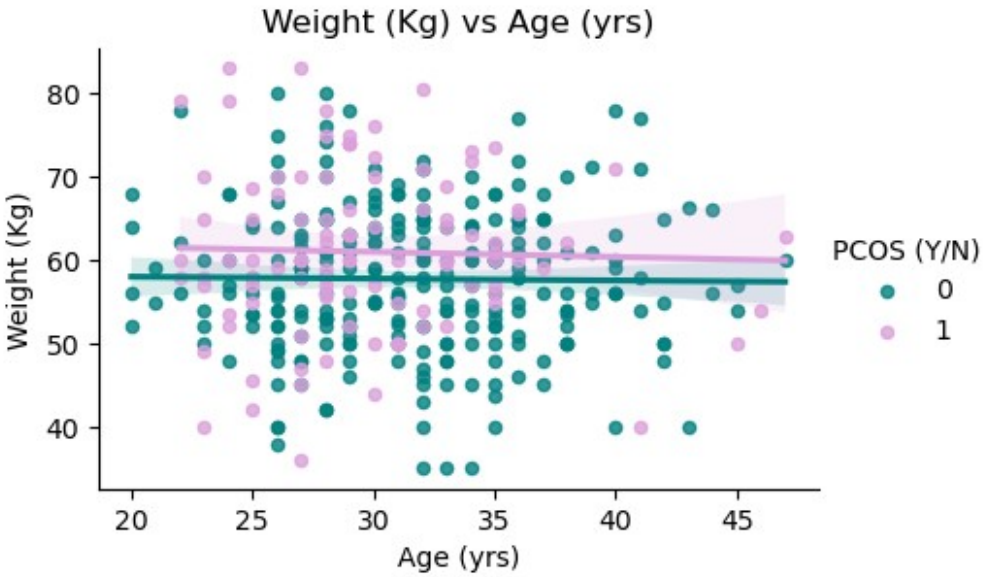


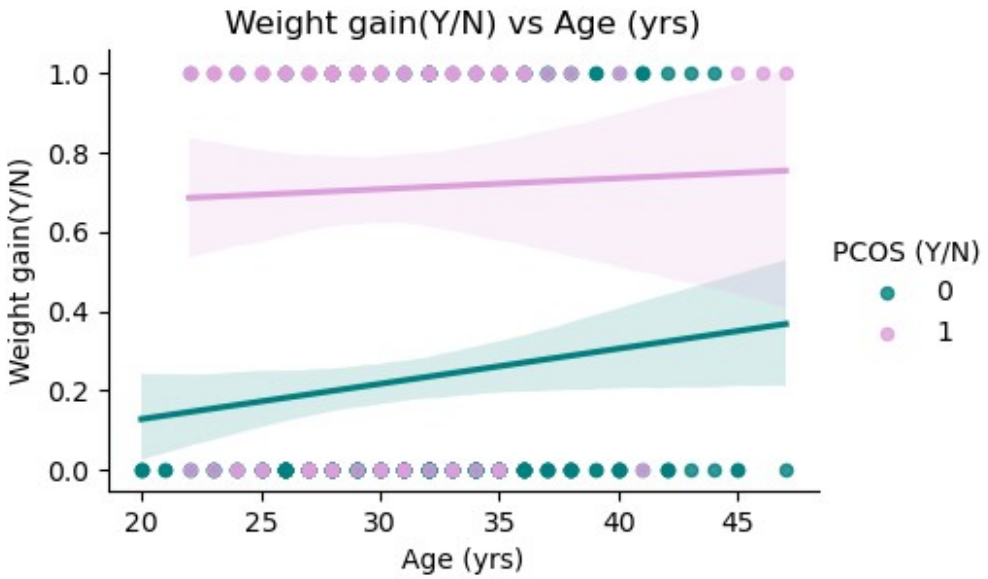


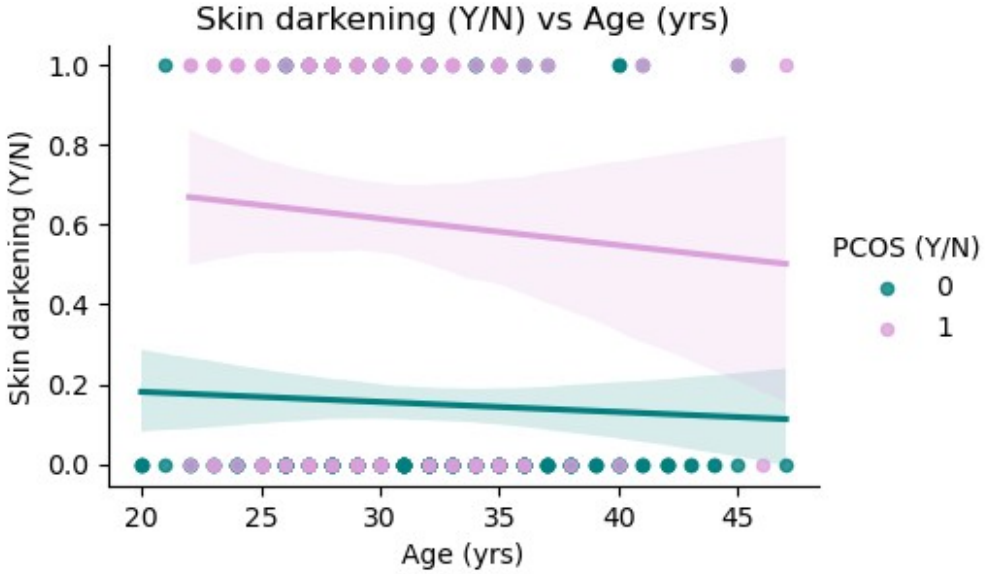

BP _Systolic (mmHg) vs Age (yrs)
BP _Systolic (mmHg)
130
125
120
115
110
105
100
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

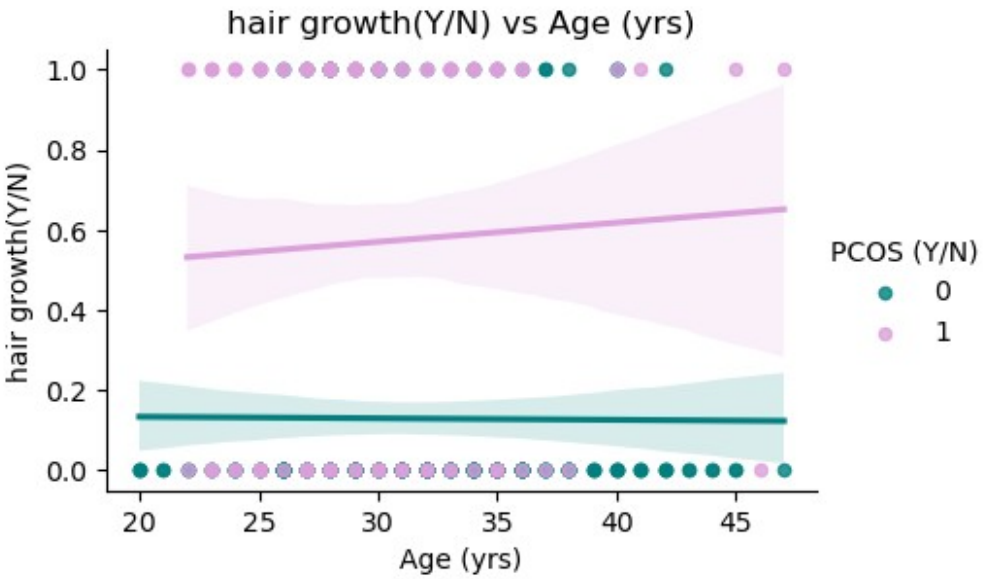

BP _Diastolic (mmHg) vs Age (yrs)
BP _Diastolic (mmHg)
82
80
78
76
74
72
70
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

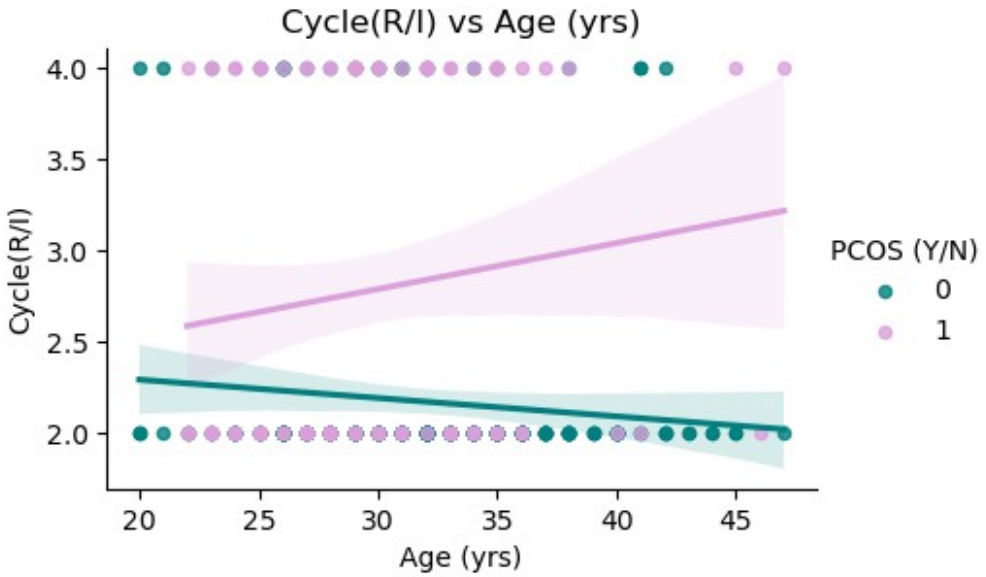

Waist:Hip Ratio vs Age (yrs)
Waist:Hip Ratio
0.95
0.90
0.85
0.80
0.75
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

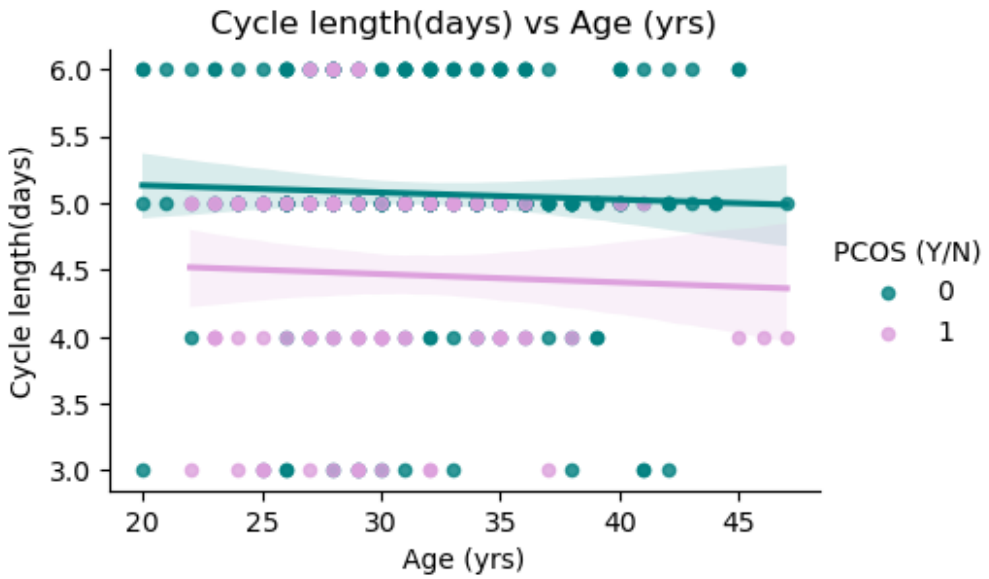

Weight (Kg) vs Age (yrs)
Weight (Kg)
80
70
60
50
40
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1


Weight gain(Y/N) vs Age (yrs)
Weight gain(Y/N)
1.0
0.8
0.6
0.4
0.2
0.0
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1


Skin darkening (Y/N) vs Age (yrs)
Skin darkening (Y/N)
1.0
0.8
0.6
0.4
0.2
0.0
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

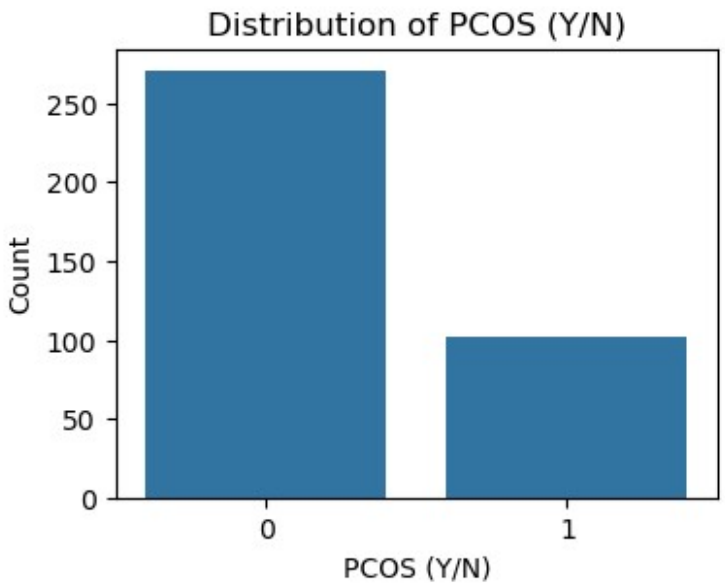

hair growth(Y/N) vs Age (yrs)
hair growth(Y/N)
1.0
0.8
0.6
0.4
0.2
0.0
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1


Cycle(R/I) vs Age (yrs)
Cycle(R/I)
4.0
3.5
3.0
2.5
2.0
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1



---------- Check Data Balance ----------

```
# Check the distribution of the target variable (e.g., 'PCOS (Y/N)' or
disease stage column)
target_column = 'PCOS (Y/N)'

# Check the counts of each class
class_counts = data[target_column].value_counts()

# Display the class counts
print(f"\nClass Distribution in {target_column}:\n", class_counts)

# Plotting the class distribution
plt.figure(figsize=(4, 3))
sns.countplot(x=target_column, data=data)
plt.title(f'Distribution of {target_column}')
plt.xlabel(target_column)
plt.ylabel('Count')
plt.show()


Class Distribution in PCOS (Y/N):
 PCOS (Y/N)
0    271
1    102
Name: count, dtype: int64
```



---------- Model Building ----------

```
# Define features (X) and target (y)
X = data.drop(columns=['PCOS (Y/N)'])
y = data['PCOS (Y/N)']

# Split into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=5)


from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
import pandas as pd
from collections import Counter

# Standardizing the dataset
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train),
columns=X_train.columns)
X_test = pd.DataFrame(scaler.transform(X_test),
columns=X_test.columns)

# Handling Imbalance in dataset
smote = SMOTE(random_state=5)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)

# Check class distribution after SMOTE
print("Before SMOTE:", Counter(y_train))
print("After SMOTE:", Counter(y_train_balanced))
```

```
Before SMOTE: Counter({0: 212, 1: 86})
After SMOTE: Counter({1: 212, 0: 212})

X_train = X_train_balanced
y_train = y_train_balanced
```

Logistic Regression

```
# Train Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Predictions
y_pred_logreg = logreg.predict(X_test)

# Evaluation
print("Logistic Regression Results:")
print(accuracy_score(y_test, y_pred_logreg))
print(classification_report(y_test, y_pred_logreg))
```

```
Logistic Regression Results:
0.84
              precision    recall  f1-score   support

           0       0.93      0.86      0.89        59
           1       0.60      0.75      0.67        16

    accuracy                           0.84        75
   macro avg       0.76      0.81      0.78        75
weighted avg       0.86      0.84      0.85        75
```

Random Forest Classifier

```
# Train Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predictions
y_pred_rf = rf.predict(X_test)

# Evaluation
print("Random Forest Results:")
print(accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Results:
0.8
              precision    recall  f1-score   support

           0       0.88      0.86      0.87        59
```

```
           1       0.53      0.56      0.55        16

    accuracy                           0.80        75
   macro avg       0.70      0.71      0.71        75
weighted avg       0.80      0.80      0.80        75
```

Compare Models & Select the Best One

```python
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score

# Function to evaluate the model
def evaluate_model(model_name, y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    print(f"\n===== {model_name} =====")
    print(f"Confusion Matrix:\n{cm}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")

    return {'model': model_name, 'accuracy': accuracy, 'precision':
precision, 'recall': recall, 'f1': f1}

# Evaluate Logistic Regression
logreg_results = evaluate_model("Logistic Regression", y_test,
y_pred_logreg)

# Evaluate Random Forest
rf_results = evaluate_model("Random Forest", y_test, y_pred_rf)

# Compare and select the best model based on F1-score
best_model = max([logreg_results, rf_results], key=lambda x: x['f1'])

print(f"\nBest Model: {best_model['model']} with F1-score:
{best_model['f1']:.4f}")
```

```
===== Logistic Regression =====
Confusion Matrix:
[[51  8]
 [ 4 12]]
Accuracy: 0.8400
Precision: 0.6000
```

```
Recall: 0.7500
F1-score: 0.6667

===== Random Forest =====
Confusion Matrix:
[[51  8]
 [ 7  9]]
Accuracy: 0.8000
Precision: 0.5294
Recall: 0.5625
F1-score: 0.5455

Best Model: Logistic Regression with F1-score: 0.6667
```

Try to improve Logistic Regression using Hyperparameter Tuning!

Why Hyperparameter Tuning?

 Better Regularization → Prevents overfitting or underfitting
 Optimized Solver → Faster convergence & better performance
 Higher Accuracy & F1-Score → Improves classification results

Hyperparameters to Tune for Logistic Regression:

- **C (Regularization strength)** → Controls the penalty (default: 1.0)
- **Solver** → Optimization algorithm (`liblinear`, `lbfgs`, `saga`)
- **Penalty** → `l1`, `l2`, `elasticnet` (L1 is for feature selection)
- **Max Iterations** → Increase if the model is not converging

Hyperparameter Tuning using GridSearchCV

```
import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Suppress warnings
warnings.filterwarnings("ignore")

# Define the parameter grid
param_grid = {
    'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],  # Regularization
strength
    'solver': ['liblinear', 'lbfgs', 'saga'],  # Valid solvers
    'penalty': ['l2'],  # L1 is removed because 'lbfgs' doesn't
support it
    'class_weight': [None, 'balanced']
}

# Initialize Logistic Regression
logreg = LogisticRegression(max_iter=50)
```

```python
# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='f1',
n_jobs=-1, error_score='raise')
grid_search.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)

# Train the best model
best_logreg = grid_search.best_estimator_
y_pred_best_logreg = best_logreg.predict(X_test)

# Evaluate the optimized model
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

print("===== Tuned Logistic Regression =====")

print("Accuracy:", accuracy_score(y_test, y_pred_best_logreg))
print("Classification Report:\n", classification_report(y_test,
y_pred_best_logreg))
```

```
Best Parameters: {'C': 1, 'class_weight': None, 'penalty': 'l2',
'solver': 'lbfgs'}
===== Tuned Logistic Regression =====
Accuracy: 0.84
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.86      0.89        59
           1       0.60      0.75      0.67        16

    accuracy                           0.84        75
   macro avg       0.76      0.81      0.78        75
weighted avg       0.86      0.84      0.85        75
```

hyperparameter tuning using `GridSearchCV` for **Random Forest**:

Steps Covered:

1. Define a **parameter grid** with key hyperparameters for Random Forest.
2. Perform **Grid Search with Cross-Validation** (`cv=5`) using **F1-score** as the evaluation metric.
3. Train the **best model** and evaluate its performance.

What's Tuned in Random Forest?

- `n_estimators`: Number of trees in the forest
- `max_depth`: Depth of trees to prevent overfitting
- `min_samples_split`: Minimum samples required to split a node
- `min_samples_leaf`: Minimum samples required at a leaf node
- `bootstrap`: Whether to use bootstrap samples
- `class_weight`: To handle imbalanced data

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import warnings

# Suppress warnings
warnings.filterwarnings("ignore")

# Define the parameter grid for Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 200],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],  # Depth of trees
    'min_samples_split': [2, 5, 10],  # Minimum samples required to
split a node
    'min_samples_leaf': [1, 2, 4],  # Minimum samples at a leaf node
    'bootstrap': [True, False],  # Use bootstrap samples
    'class_weight': [None, 'balanced']
}

# Initialize Random Forest Classifier
rf = RandomForestClassifier(random_state=42)

# Perform Grid Search with Cross-Validation
grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='f1',
n_jobs=-1, error_score='raise')
grid_search_rf.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", grid_search_rf.best_params_)

# Train the best model
best_rf = grid_search_rf.best_estimator_
y_pred_best_rf = best_rf.predict(X_test)

# Evaluate the optimized model
```

```
print("===== Tuned Random Forest =====")
print("Accuracy:", accuracy_score(y_test, y_pred_best_rf))
print("Classification Report:\n", classification_report(y_test,
y_pred_best_rf))
```

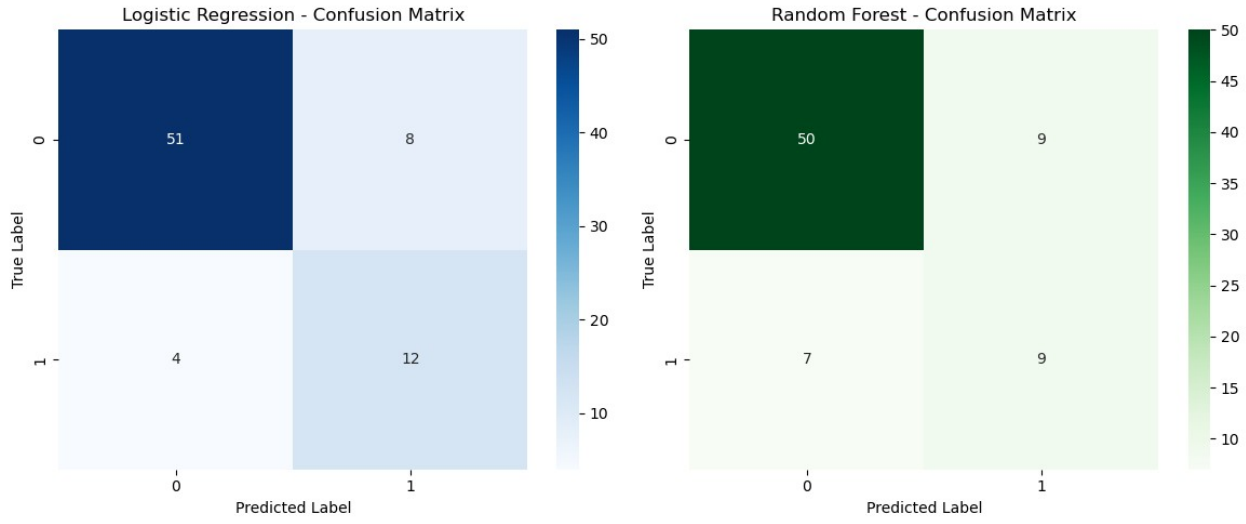
```
Best Parameters: {'bootstrap': True, 'class_weight': 'balanced',
'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 200}
===== Tuned Random Forest =====
Accuracy: 0.7866666666666666
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.85      0.86        59
           1       0.50      0.56      0.53        16

    accuracy                           0.79        75
   macro avg       0.69      0.70      0.70        75
weighted avg       0.80      0.79      0.79        75
```

---------- Compare Tuned Logistic Regression & Random Forest ----------

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score, confusion_matrix

# Compute performance metrics for Logistic Regression
logreg_metrics = {
    "Model": "Logistic Regression",
    "Accuracy": accuracy_score(y_test, y_pred_best_logreg),
    "F1 Score": f1_score(y_test, y_pred_best_logreg),
    "Precision": precision_score(y_test, y_pred_best_logreg),
    "Recall": recall_score(y_test, y_pred_best_logreg)
}

# Compute performance metrics for Random Forest
rf_metrics = {
    "Model": "Random Forest",
    "Accuracy": accuracy_score(y_test, y_pred_best_rf),
    "F1 Score": f1_score(y_test, y_pred_best_rf),
    "Precision": precision_score(y_test, y_pred_best_rf),
    "Recall": recall_score(y_test, y_pred_best_rf)
}

# Convert results to a DataFrame for easy comparison
comparison_df = pd.DataFrame([logreg_metrics, rf_metrics])
```

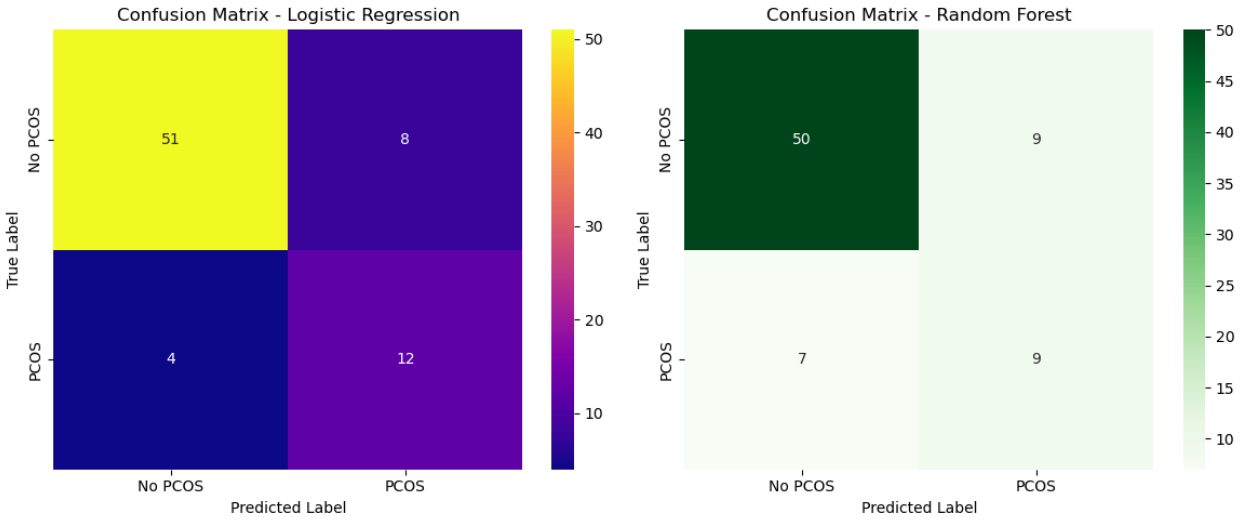
```
# Display performance comparison table
print("===== Model Performance Comparison =====")
print(comparison_df)

# Plot Confusion Matrices
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Logistic Regression Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred_best_logreg), annot=True,
fmt='d', cmap="Blues", ax=axes[0])
axes[0].set_title("Logistic Regression - Confusion Matrix")
axes[0].set_xlabel("Predicted Label")
axes[0].set_ylabel("True Label")

# Random Forest Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred_best_rf), annot=True,
fmt='d', cmap="Greens", ax=axes[1])
axes[1].set_title("Random Forest - Confusion Matrix")
axes[1].set_xlabel("Predicted Label")
axes[1].set_ylabel("True Label")

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

```
===== Model Performance Comparison =====
                 Model  Accuracy  F1 Score  Precision  Recall
0  Logistic Regression  0.840000  0.666667        0.6  0.7500
1        Random Forest  0.786667  0.529412        0.5  0.5625
```




```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

```
# Compute confusion matrices
cm_logreg = confusion_matrix(y_test, y_pred_best_logreg)
cm_rf = confusion_matrix(y_test, y_pred_best_rf)

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Logistic Regression Confusion Matrix
sns.heatmap(cm_logreg, annot=True, fmt="d", cmap="plasma",
xticklabels=["No PCOS", "PCOS"], yticklabels=["No PCOS", "PCOS"],
ax=axes[0])
axes[0].set_title("Confusion Matrix - Logistic Regression")
axes[0].set_xlabel("Predicted Label")
axes[0].set_ylabel("True Label")

# Random Forest Confusion Matrix
sns.heatmap(cm_rf, annot=True, fmt="d", cmap="Greens",
xticklabels=["No PCOS", "PCOS"], yticklabels=["No PCOS", "PCOS"],
ax=axes[1])
axes[1].set_title("Confusion Matrix - Random Forest")
axes[1].set_xlabel("Predicted Label")
axes[1].set_ylabel("True Label")

# Adjust layout
plt.tight_layout()
plt.show()
```



Hyperparameter Tuning using GridSearchCV for SVM

Steps Covered:

- **Define a parameter grid:** Specify key hyperparameters relevant to SVM, such as regularization strength and kernel parameters.

- **Perform Grid Search with Cross-Validation:** Use GridSearchCV with 5-fold cross-validation (cv=5) and F1-score as the evaluation metric to systematically explore the hyperparameter combinations.
- **Train and Evaluate the Best Model:** Fit the SVM model using the optimal hyperparameters found and evaluate its performance on the test set.

What's Tuned in SVM?

- **C:** Regularization parameter that controls the trade-off between achieving a low training error and maintaining a simple decision boundary for better generalization.
- **kernel:** Determines the type of hyperplane used to separate the data (e.g., 'linear', 'rbf', 'poly'). Different kernels can capture different patterns in the data.
- **gamma:** Kernel coefficient for 'rbf' and 'poly' kernels, which defines the influence of individual training examples. A small gamma means a far-reaching influence, while a large gamma implies a close reach.
- **degree:** Degree of the polynomial kernel function (used only when the kernel is set to 'poly').
- **coef0:** Independent term in the kernel function, relevant for 'poly' and 'sigmoid' kernels, which can affect the model's performance.
- **class_weight:** Adjusts weights inversely proportional to class frequencies in the data to help handle imbalanced classes.

Using GridSearchCV for SVM ensures that the model is optimally tuned for the given dataset and task—in this case, predicting PCOS likelihood as 0 (no) or 1 (yes).

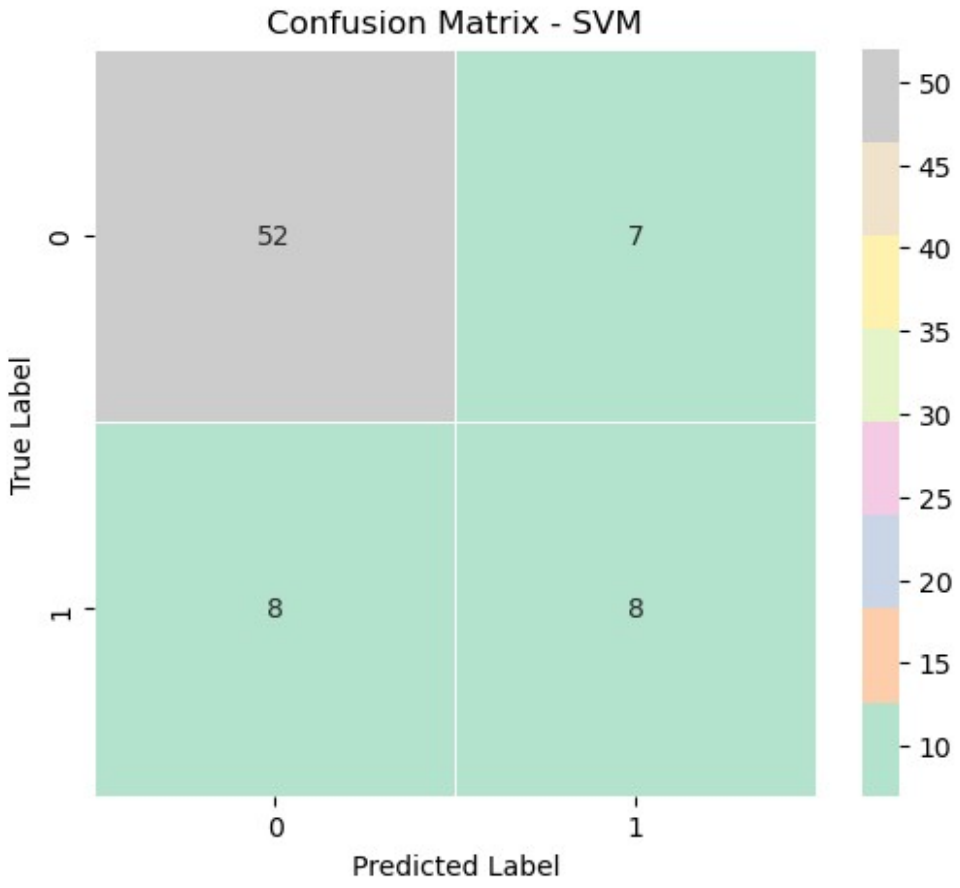
```
import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Suppress warnings
warnings.filterwarnings("ignore")

'''param_grid_svm = {
    'C': [0.01, 0.1, 1, 10, 100, 1000],              # Regularization
strength
    'kernel': ['linear', 'rbf', 'poly'],             # Different
kernel functions to try
    'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1], # Add more gamma
values
    'degree': [2, 3, 4],                            # Only for 'poly'
kernel
    'class_weight': [None, 'balanced']              # To handle class
imbalance
}
'''
# Define the parameter grid for SVM
param_grid_svm = {
```

```
    'C': [0.1, 1, 10, 100],                # Regularization strength
    'kernel': ['linear', 'rbf', 'poly'],    # Different kernel
functions to try
    'gamma': ['scale', 'auto'],             # Kernel coefficient for
'rbf' and 'poly'
    'class_weight': [None, 'balanced']      # To handle class
imbalance
}

# Initialize the SVM classifier
svm_model = SVC()

# Perform Grid Search with 5-fold Cross-Validation using f1 score as
the metric
grid_search_svm = GridSearchCV(svm_model, param_grid_svm, cv=10,
scoring='f1', n_jobs=-1, error_score='raise')
grid_search_svm.fit(X_train, y_train)

# Display best parameters
print("Best Parameters (SVM):", grid_search_svm.best_params_)

# Train the best model
best_svm = grid_search_svm.best_estimator_
y_pred_best_svm = best_svm.predict(X_test)

# Evaluate the optimized SVM model
print("===== Tuned SVM =====")
print("Accuracy:", accuracy_score(y_test, y_pred_best_svm))
print("Classification Report:\n", classification_report(y_test,
y_pred_best_svm))
```

```
Best Parameters (SVM): {'C': 10, 'class_weight': None, 'gamma':
'scale', 'kernel': 'rbf'}
===== Tuned SVM =====
Accuracy: 0.8
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.88      0.87        59
           1       0.53      0.50      0.52        16

    accuracy                           0.80        75
   macro avg       0.70      0.69      0.70        75
weighted avg       0.80      0.80      0.80        75
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Compute confusion matrix
```

```
cm = confusion_matrix(y_test, y_pred_best_svm)

# Plot the heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Pastel2', linewidths=0.5)

# Labels and title
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - SVM")

# Show plot
plt.show()
```



---------- Final Note ----------

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

# Compute performance metrics for both models
metrics = {
    "Model": ["Logistic Regression", "Random Forest"],
```

```
    "Accuracy": [accuracy_score(y_test, y_pred_best_logreg),
accuracy_score(y_test, y_pred_best_rf)],
    "Precision": [precision_score(y_test, y_pred_best_logreg),
precision_score(y_test, y_pred_best_rf)],
    "Recall": [recall_score(y_test, y_pred_best_logreg),
recall_score(y_test, y_pred_best_rf)],
    "F1-Score": [f1_score(y_test, y_pred_best_logreg),
f1_score(y_test, y_pred_best_rf)]
}

# Convert to a DataFrame for better visualization
import pandas as pd
df_metrics = pd.DataFrame(metrics)
print("=== Model Performance Comparison ===")
print(df_metrics)

# Identify the best model based on F1-score
best_model_name = df_metrics.iloc[df_metrics['F1-Score'].idxmax()]
['Model']
best_model = best_logreg if best_model_name == "Logistic Regression"
else best_rf

print("\n===== Best Model Identified =====")
print(f"Best Model: {best_model_name}")

# Print model structure and hyperparameters
print("\n=== Model Structure & Hyperparameters ===")
print(best_model)

# Conclusion
print("\n===== Conclusion =====")
if best_model_name == "Random Forest":
    print("Random Forest performed better with higher F1-score, making
it the best choice for PCOS prediction.")
else:
    print("Logistic Regression performed better, making it the
preferred model for PCOS prediction.")
```

```
=== Model Performance Comparison ===
                 Model  Accuracy  Precision  Recall  F1-Score
0  Logistic Regression  0.840000        0.6  0.7500  0.666667
1        Random Forest  0.786667        0.5  0.5625  0.529412

===== Best Model Identified =====
Best Model: Logistic Regression

=== Model Structure & Hyperparameters ===
LogisticRegression(C=1, max_iter=50)

===== Conclusion =====
```

```
Logistic Regression performed better, making it the preferred model
for PCOS prediction.
```

Measurable:

'Pulse rate(bpm)',

'RR (breaths/min)',

'BP _Systolic (mmHg)',

'BP _Diastolic (mmHg)',

'Waist:Hip Ratio',

'Weight (Kg)',

'Weight gain(Y/N)'

Self-reported:

'Skin darkening (Y/N)',

'hair growth(Y/N)',

'Cycle(R/I)',

'Age (yrs)',

'Cycle length (days)'

Pathological:

'Follicle No. (R)',

'Follicle No. (L)'

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import f_oneway

from sklearn.metrics import balanced_accuracy_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import VarianceThreshold


from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, classification_report

from scipy.stats.mstats import winsorize
```

---------- Read and Process Data ----------

```
# Read main file
file_path = 'PCOS_Likelihood.csv'
original_data = pd.read_csv(file_path)

# Print all column names
print("=== Column Names in Dataset ===")
print(original_data.columns.tolist())  # Print column names as a list
```

```
=== Column Names in Dataset ===
['Sl. No', 'Patient File No.', 'PCOS (Y/N)', ' Age (yrs)', 'Weight
(Kg)', 'Height(Cm) ', 'BMI', 'Blood Group', 'Pulse rate(bpm) ', 'RR
(breaths/min)', 'Hb(g/dl)', 'Cycle(R/I)', 'Cycle length(days)',
'Marraige Status (Yrs)', 'Pregnant(Y/N)', 'No. of aborptions', '  I
beta-HCG(mIU/mL)', 'II    beta-HCG(mIU/mL)', 'FSH(mIU/mL)',
'LH(mIU/mL)', 'FSH/LH', 'Hip(inch)', 'Waist(inch)', 'Waist:Hip Ratio',
'TSH (mIU/L)', 'AMH(ng/mL)', 'PRL(ng/mL)', 'Vit D3 (ng/mL)',
'PRG(ng/mL)', 'RBS(mg/dl)', 'Weight gain(Y/N)', 'hair growth(Y/N)',
'Skin darkening (Y/N)', 'Hair loss(Y/N)', 'Pimples(Y/N)', 'Fast food
(Y/N)', 'Reg.Exercise(Y/N)', 'BP _Systolic (mmHg)', 'BP _Diastolic
(mmHg)', 'Follicle No. (L)', 'Follicle No. (R)', 'Avg. F size (L)
(mm)', 'Avg. F size (R) (mm)', 'Endometrium (mm)']
```

```
# Load the preprocessed dataset
data_reduced = pd.read_csv('N_R_data.csv')


# Print all column names
print("=== Column Names in Dataset ===")
print(data_reduced.columns.tolist())  # Print column names as a list
```

```
=== Column Names in Dataset ===
['Age (yrs)', 'Weight (Kg)', 'Height(Cm)', 'Blood Group', 'Pulse
rate(bpm)', 'RR (breaths/min)', 'Hb(g/dl)', 'Cycle(R/I)', 'Cycle
length(days)', 'Marraige Status (Yrs)', 'Pregnant(Y/N)', 'No. of
aborptions', 'I   beta-HCG(mIU/mL)', 'II    beta-HCG(mIU/mL)',
'FSH(mIU/mL)', 'LH(mIU/mL)', 'Hip(inch)', 'Waist:Hip Ratio', 'TSH
(mIU/L)', 'AMH(ng/mL)', 'PRL(ng/mL)', 'Vit D3 (ng/mL)', 'PRG(ng/mL)',
'RBS(mg/dl)', 'Weight gain(Y/N)', 'hair growth(Y/N)', 'Skin darkening
(Y/N)', 'Hair loss(Y/N)', 'Pimples(Y/N)', 'Fast food (Y/N)',
'Reg.Exercise(Y/N)', 'BP _Systolic (mmHg)', 'BP _Diastolic (mmHg)',
'Follicle No. (L)', 'Follicle No. (R)', 'Avg. F size (L) (mm)', 'Avg.
F size (R) (mm)', 'Endometrium (mm)']
```

```
# Load the preprocessed dataset
data_reduced['PCOS (Y/N)'] = original_data['PCOS (Y/N)']

# Selecting required columns
selected_columns = ['PCOS (Y/N)', 'Pulse rate(bpm)', 'RR
(breaths/min)',
                    'BP _Systolic (mmHg)', 'BP _Diastolic (mmHg)',
                    'Waist:Hip Ratio', 'Weight (Kg)', 'Weight
gain(Y/N)',
                    'Skin darkening (Y/N)', 'hair growth(Y/N)',
'Cycle(R/I)',
                    'Age (yrs)', 'Cycle length(days)',
                    'Follicle No. (R)', 'Follicle No. (L)']


# Extracting the selected columns
data_filtered = data_reduced[selected_columns]


# Display the first few rows
display(data_filtered.head())

# Display summary statistics
data_filtered.describe()
```

```
   PCOS (Y/N)  Pulse rate(bpm)  RR (breaths/min)  BP _Systolic (mmHg)
\
0           0               78                22                  110

1           0               74                20                  120

2           1               72                18                  120

3           0               72                20                  120

4           0               72                18                  120


   BP _Diastolic (mmHg)  Waist:Hip Ratio  Weight (Kg)  Weight
gain(Y/N)  \
0                    80             0.83         44.6
0
1                    70             0.84         65.0
0
2                    80             0.90         68.8
0
3                    70             0.86         65.0
0
4                    80             0.81         52.0
```

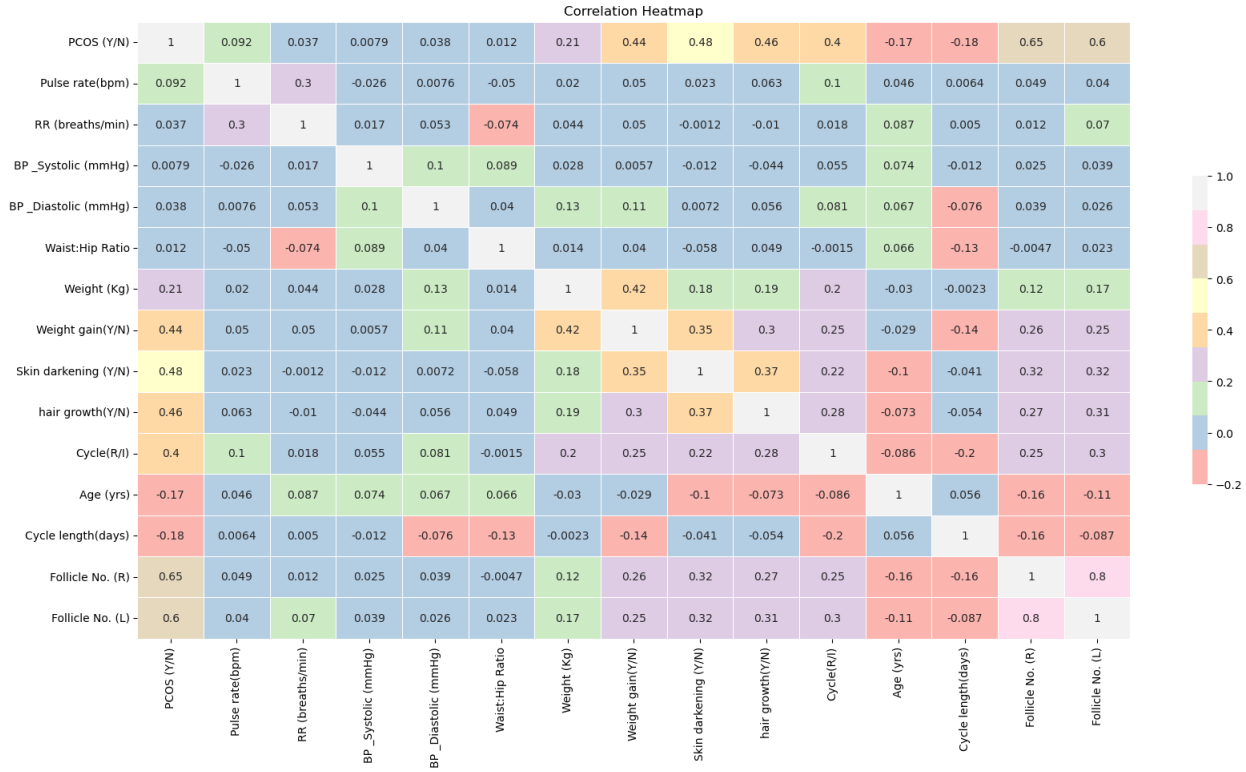
```
   Skin darkening (Y/N)  hair growth(Y/N)  Cycle(R/I)  Age (yrs)  \
0                     0                 0           2         28
1                     0                 0           2         36
2                     0                 0           2         33
3                     0                 0           2         37
4                     0                 0           2         25

   Cycle length(days)  Follicle No. (R)  Follicle No. (L)
0                   5                 3                 3
1                   5                 5                 3
2                   5                15                13
3                   5                 2                 2
4                   5                 4                 3

       PCOS (Y/N)  Pulse rate(bpm)  RR (breaths/min)  BP _Systolic
(mmHg)  \
count  541.000000       541.000000        541.000000
541.000000
mean     0.327172        73.247689         19.243993
114.661738
std      0.469615         4.430285          1.688629
7.384556
min      0.000000        13.000000         16.000000
12.000000
25%      0.000000        72.000000         18.000000
110.000000
50%      0.000000        72.000000         18.000000
110.000000
75%      1.000000        74.000000         20.000000
120.000000
max      1.000000        82.000000         28.000000
140.000000

       BP _Diastolic (mmHg)  Waist:Hip Ratio  Weight (Kg)  Weight
gain(Y/N)  \
count             541.000000       541.000000   541.000000
541.000000
mean               76.927911         0.891627    59.637153
0.377079
std                 5.574112         0.046135    11.028287
0.485104
min                 8.000000         0.760000    31.000000
0.000000
25%                70.000000         0.860000    52.000000
0.000000
50%                80.000000         0.890000    59.000000
0.000000
75%                80.000000         0.930000    65.000000
```

```
1.000000
max               100.000000          0.980000   108.000000
1.000000

       Skin darkening (Y/N)  hair growth(Y/N)  Cycle(R/I)   Age (yrs)
\
count            541.000000        541.000000  541.000000  541.000000

mean               0.306839          0.273567    2.560074   31.430684

std                0.461609          0.446202    0.901950    5.411006

min                0.000000          0.000000    2.000000   20.000000

25%                0.000000          0.000000    2.000000   28.000000

50%                0.000000          0.000000    2.000000   31.000000

75%                1.000000          1.000000    4.000000   35.000000

max                1.000000          1.000000    5.000000   48.000000


       Cycle length(days)  Follicle No. (R)  Follicle No. (L)
count           541.00000        541.000000        541.000000
mean              4.94085          6.641405          6.129390
std               1.49202          4.436889          4.229294
min               0.00000          0.000000          0.000000
25%               4.00000          3.000000          3.000000
50%               5.00000          6.000000          5.000000
75%               5.00000         10.000000          9.000000
max              12.00000         20.000000         22.000000
```

---------- Check Correlation ----------

```
# Compute the correlation matrix
corr_matrix = data_filtered.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(20, 10))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_matrix, annot=True, cmap="Pastel1", linewidths=.5,
cbar_kws={"shrink": .5})   # cmap='coolwarm'
```

```
plt.title('Correlation Heatmap')
plt.show()
```



---------- Outliers Detection and Removal ----------

```
data = data_filtered.copy()

# Dictionary to store outliers
outliers_quantile = {}

# Detect outliers using IQR (Interquartile Range)
for col in data.columns:
        # Compute IQR
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1

        # Identify outliers
        outlier_indices_quantile = data[
            (data[col] < (Q1 - 1.5 * IQR)) | (data[col] > (Q3 + 1.5 *
IQR))
        ].index

        if len(outlier_indices_quantile) > 0:
            outliers_quantile[col] = outlier_indices_quantile
```

```
# Display the results
features_with_outliers = {col: len(indices) for col, indices in
outliers_quantile.items()}
print("\nFeatures with Outliers (as per quantile):",
features_with_outliers)
```

```
Features with Outliers (as per quantile): {'Pulse rate(bpm)': 94, 'RR
(breaths/min)': 14, 'BP _Systolic (mmHg)': 3, 'BP _Diastolic (mmHg)':
2, 'Weight (Kg)': 18, 'Age (yrs)': 5, 'Cycle length(days)': 77,
'Follicle No. (L)': 6}
```

```
# Total number of rows in the dataset
total_rows = len(data)

# Dictionary to store the percentage of outliers
outliers_percentage = {
    col: (len(indices) / total_rows) * 100 for col, indices in
outliers_quantile.items()
}

# Display the percentage of outliers
print("\nPercentage of Outliers per Feature:")
for feature, percentage in outliers_percentage.items():
    print(f"{feature}: {percentage:.2f}%")
```

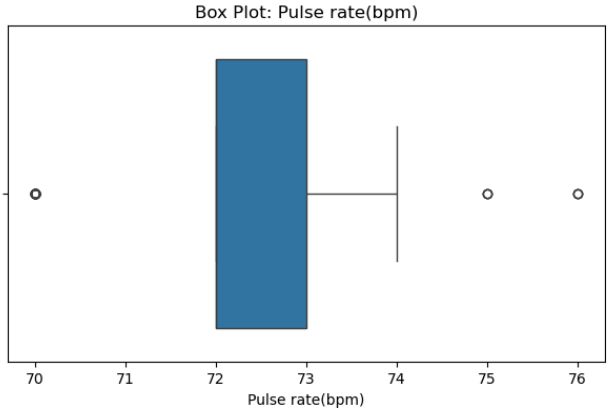
```
Percentage of Outliers per Feature:
Pulse rate(bpm): 17.38%
RR (breaths/min): 2.59%
BP _Systolic (mmHg): 0.55%
BP _Diastolic (mmHg): 0.37%
Weight (Kg): 3.33%
Age (yrs): 0.92%
Cycle length(days): 14.23%
Follicle No. (L): 1.11%
```

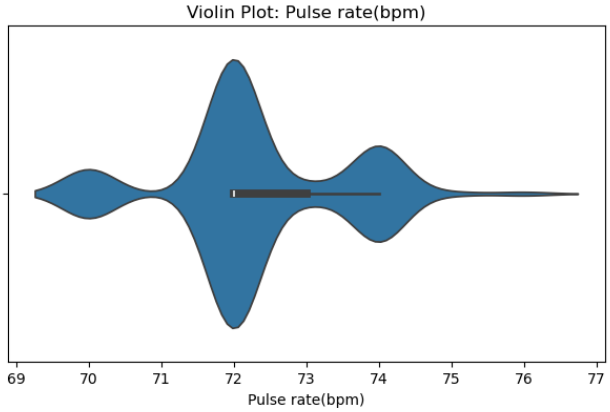
```
# List of features to remove outliers from
features_with_outliers = outliers_quantile.keys()

# Function to remove outliers using IQR
def remove_outliers(df, feature):
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Remove rows where feature value is outside bounds
    filtered_df = df[(df[feature] >= lower_bound) & (df[feature] <=
```

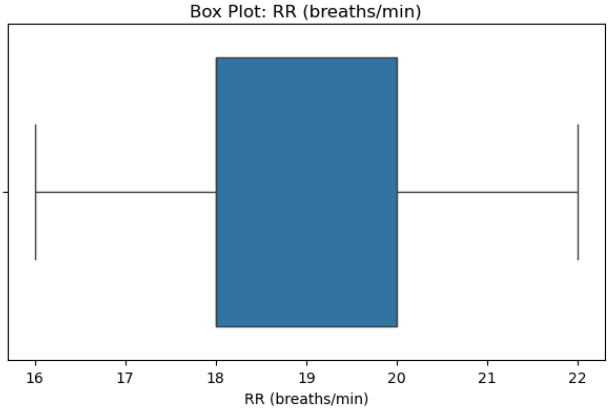
```
upper_bound)]

    print(f"Removed {len(df) - len(filtered_df)} outliers from
{feature}")
    return filtered_df

# Apply outlier removal to all selected features
for feature in features_with_outliers:
    data = remove_outliers(data, feature)

# Display remaining data shape after outlier removal
print(f"\nFinal dataset shape: {data.shape}")
```

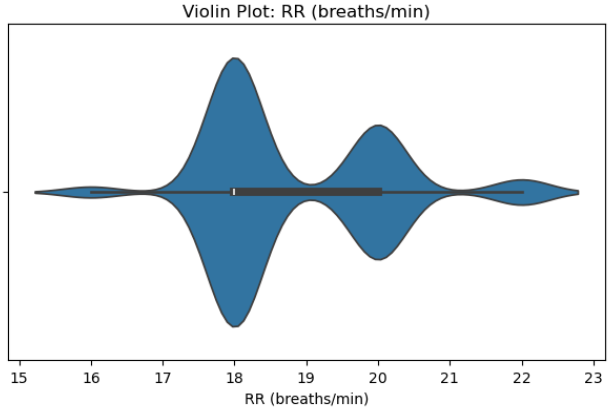
```
Removed 94 outliers from Pulse rate(bpm)
Removed 6 outliers from RR (breaths/min)
Removed 2 outliers from BP _Systolic (mmHg)
Removed 1 outliers from BP _Diastolic (mmHg)
Removed 14 outliers from Weight (Kg)
Removed 1 outliers from Age (yrs)
Removed 50 outliers from Cycle length(days)
Removed 6 outliers from Follicle No. (L)

Final dataset shape: (367, 15)
```

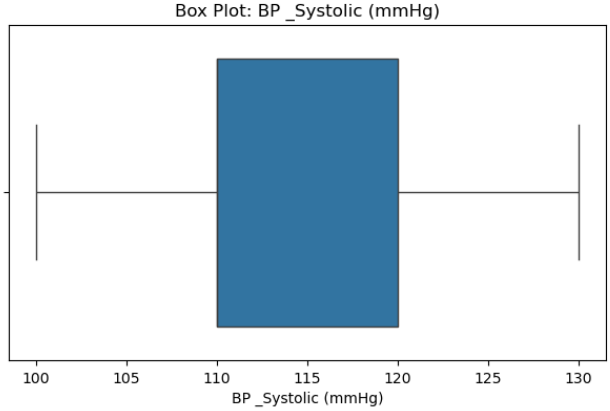
```
# Number of rows for subplots (one row per feature)
num_rows = len(features_with_outliers)

# Creating subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(12, num_rows * 4))
axes = axes.reshape(num_rows, 2)  # Ensure it's 2D for indexing

for i, feature in enumerate(features_with_outliers):
    # Box Plot
    sns.boxplot(data=data, x=feature, ax=axes[i, 0])
    axes[i, 0].set_title(f'Box Plot: {feature}')

    # Violin Plot
    sns.violinplot(data=data, x=feature, ax=axes[i, 1])
    axes[i, 1].set_title(f'Violin Plot: {feature}')

# Adjust layout
plt.tight_layout()
plt.show()
```

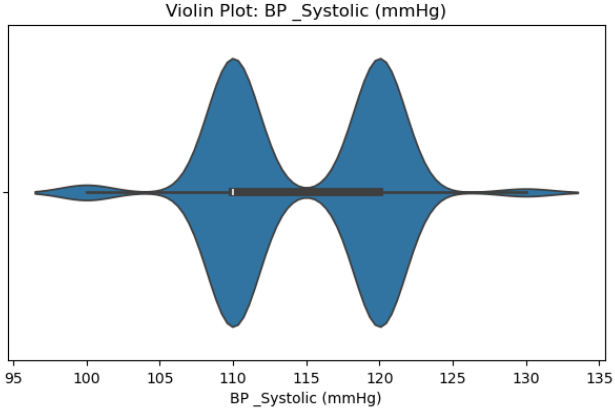

Box Plot: Pulse rate(bpm)
70
71
72
73
74
75
76
Pulse rate(bpm)

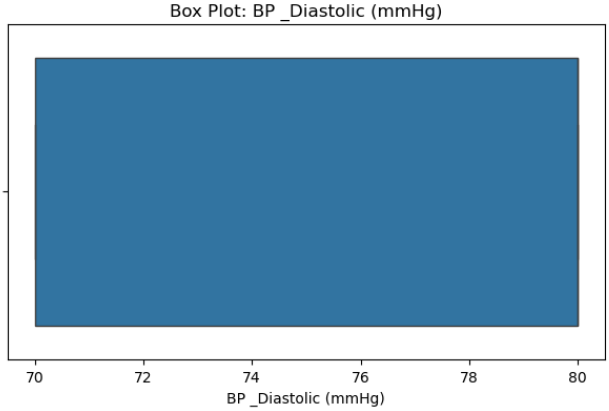

Violin Plot: Pulse rate(bpm)
69
70
71
72
73
74
75
76
77
Pulse rate(bpm)

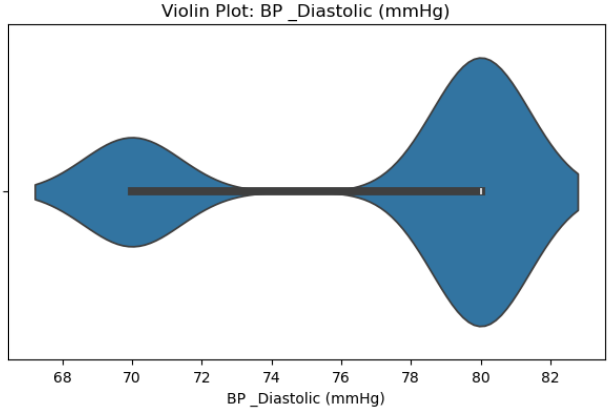

Box Plot: RR (breaths/min)
16
17
18
19
20
21
22
RR (breaths/min)

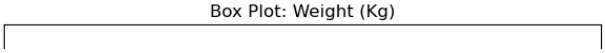

Violin Plot: RR (breaths/min)
15
16
17
18
19
20
21
22
23
RR (breaths/min)

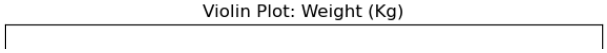

Box Plot: BP _Systolic (mmHg)
100
105
110
115
120
125
130
BP _Systolic (mmHg)


Violin Plot: BP _Systolic (mmHg)
95
100
105
110
115
120
125
130
135
BP _Systolic (mmHg)


Box Plot: BP _Diastolic (mmHg)
70
72
74
76
78
80
BP _Diastolic (mmHg)


Violin Plot: BP _Diastolic (mmHg)
68
70
72
74
76
78
80
82
BP _Diastolic (mmHg)


Box Plot: Weight (Kg)


Violin Plot: Weight (Kg)

---------- Check Modified Data Head ----------

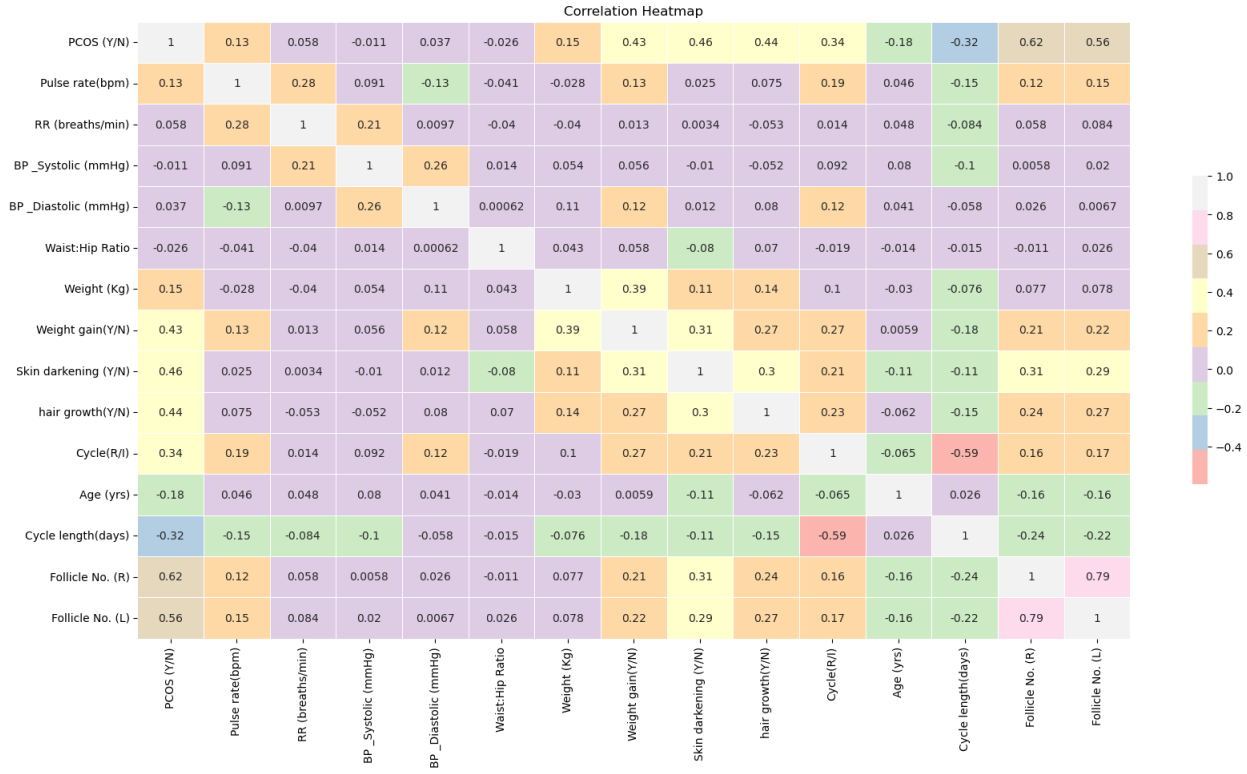
```
display(data.head())
data.describe()
data.to_csv('mix_14_featires_outlier_removed_data.csv', index=False)
```

```
   PCOS (Y/N)  Pulse rate(bpm)  RR (breaths/min)  BP _Systolic (mmHg)
\
1           0               74                20                  120

2           1               72                18                  120

3           0               72                20                  120

4           0               72                18                  120

6           0               72                18                  120


   BP _Diastolic (mmHg)  Waist:Hip Ratio  Weight (Kg)  Weight
gain(Y/N)  \
1                    70             0.84         65.0
0
2                    80             0.90         68.8
0
3                    70             0.86         65.0
0
4                    80             0.81         52.0
0
6                    80             0.85         64.0
0

   Skin darkening (Y/N)  hair growth(Y/N)  Cycle(R/I)  Age (yrs)  \
1                     0                 0           2         36
2                     0                 0           2         33
3                     0                 0           2         37
4                     0                 0           2         25
6                     0                 0           2         34

   Cycle length(days)  Follicle No. (R)  Follicle No. (L)
1                   5                 5                 3
2                   5                15                13
3                   5                 2                 2
4                   5                 4                 3
6                   5                 6                 6
```

---------- Check Correlation ----------

```
data = pd.read_csv('mix_14_featires_outlier_removed_data.csv')

# Compute the correlation matrix
```

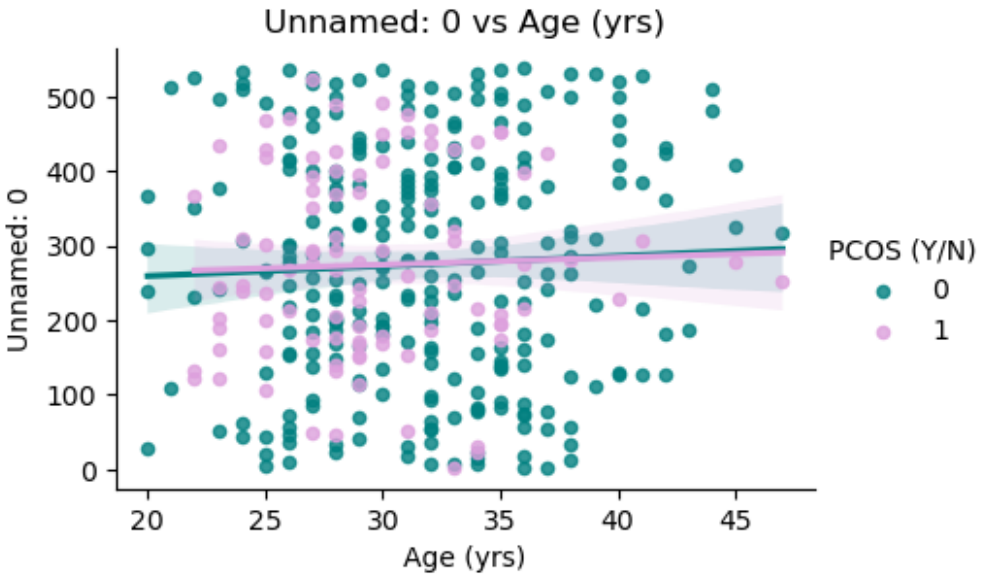
```
corr_matrix = data.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(20, 10))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_matrix, annot=True, cmap="Pastel1", linewidths=.5,
cbar_kws={"shrink": .5})   # cmap='coolwarm'


plt.title('Correlation Heatmap')
plt.show()
```

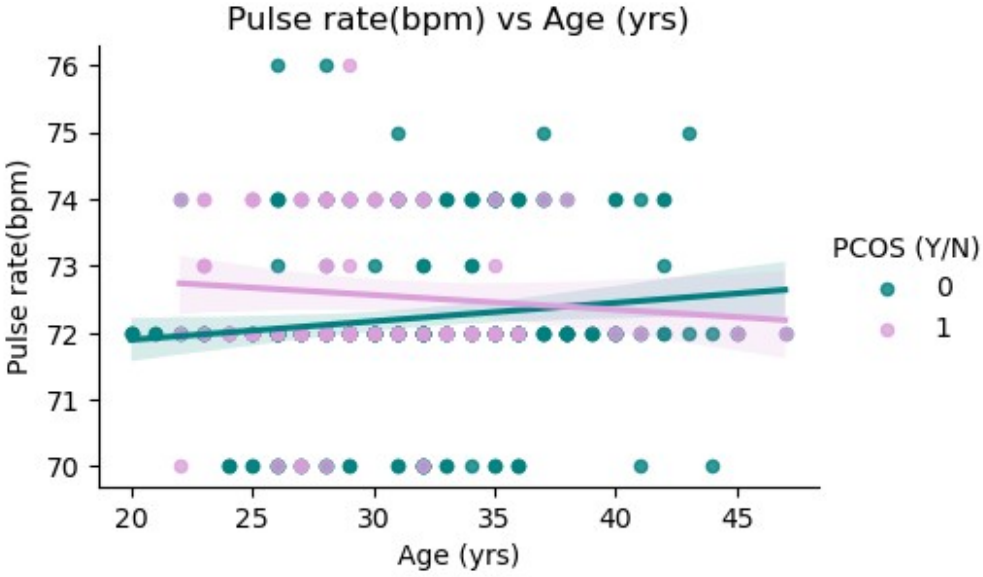


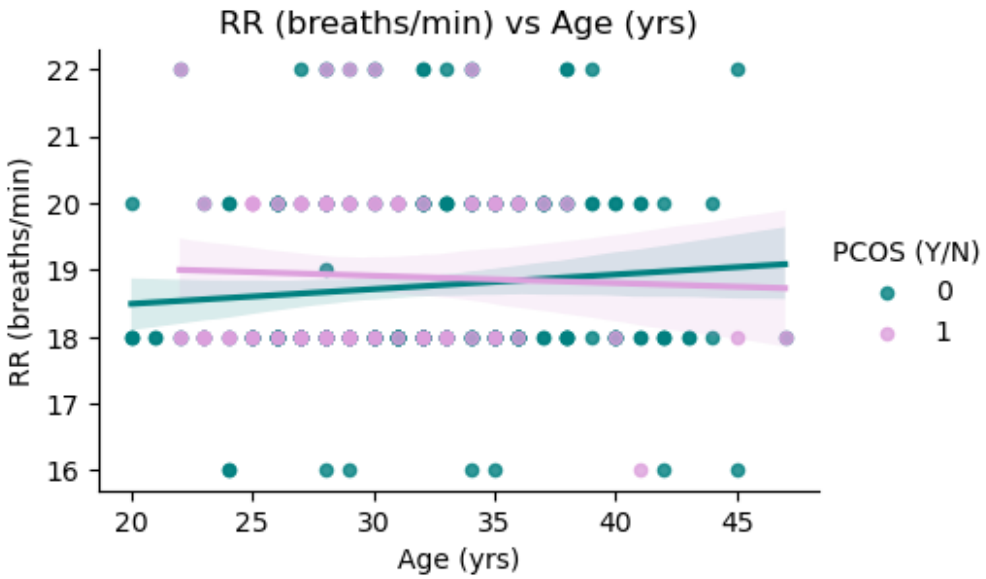
---------- Patterns Of Feature Distribution ----------

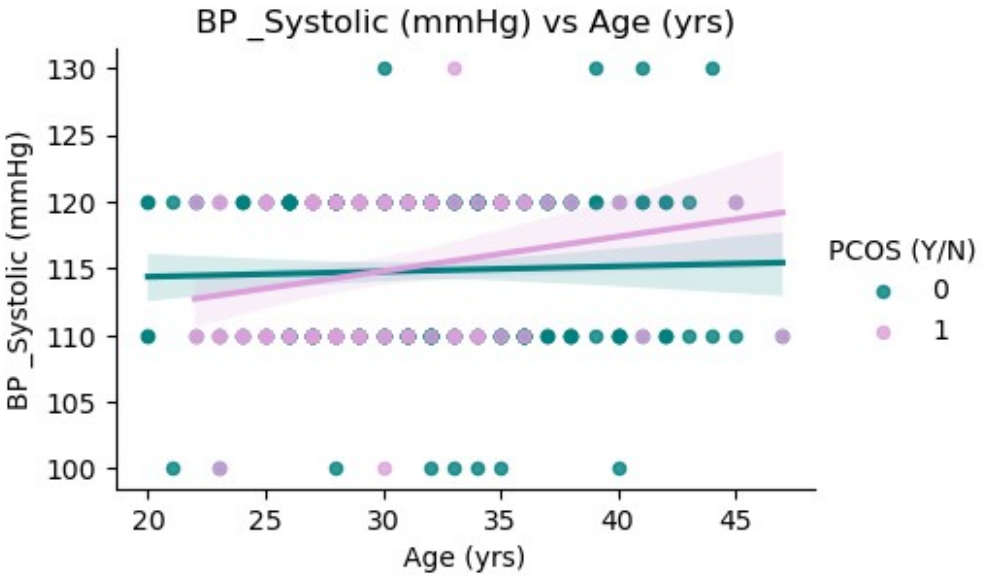
```
import seaborn as sns
import matplotlib.pyplot as plt

color = ["teal", "plum"]
count_column = data.columns.drop("PCOS (Y/N)", errors='ignore')

# Loop through each feature and create separate lmplots
for feature in count_column:
```

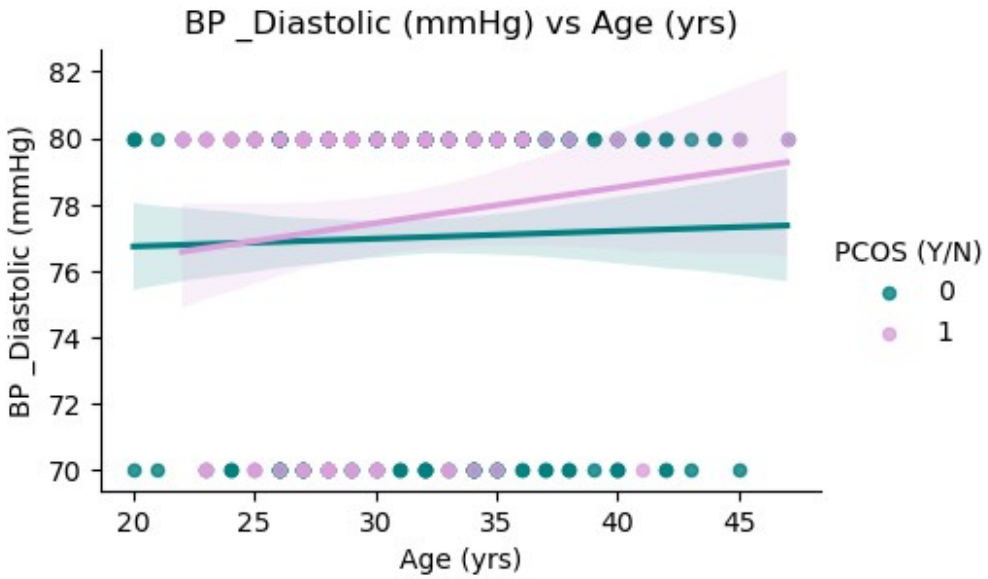
```
    if feature != "Age (yrs)":
        if feature != "PCOS (Y/N)":
            g = sns.lmplot(data=data,
                   x="Age (yrs)", y=feature, hue="PCOS (Y/N)",
                   palette=color, height=3, aspect=1.5,
                   scatter_kws={"s": 20}
                  )
            g.set(title=f"{feature} vs Age (yrs)")
            plt.show()  # Show each plot separately
```

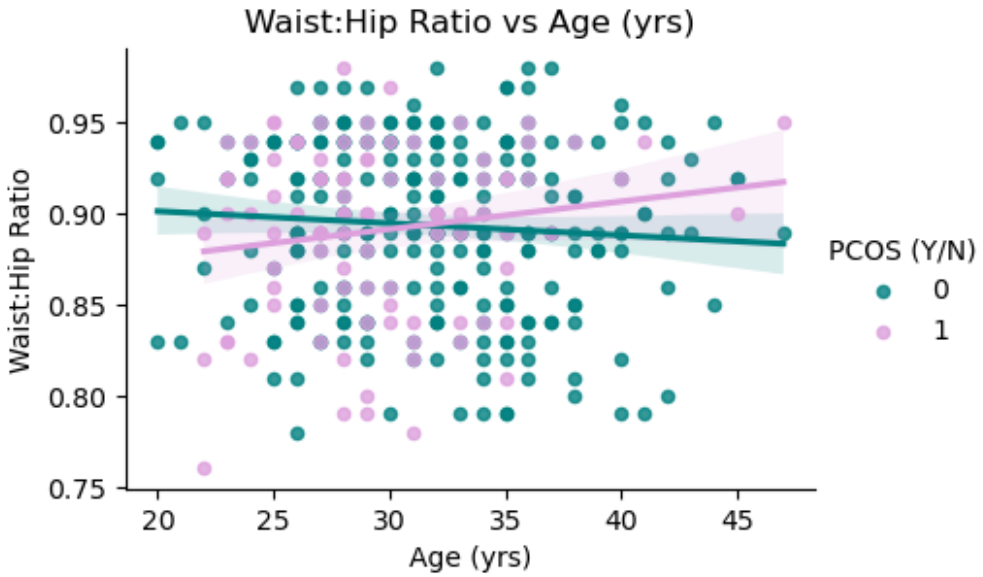


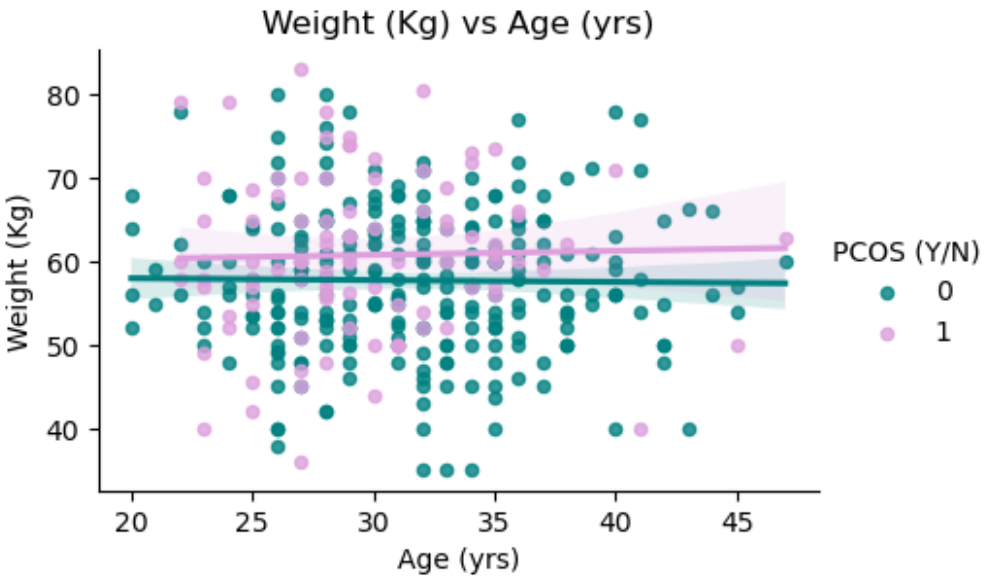


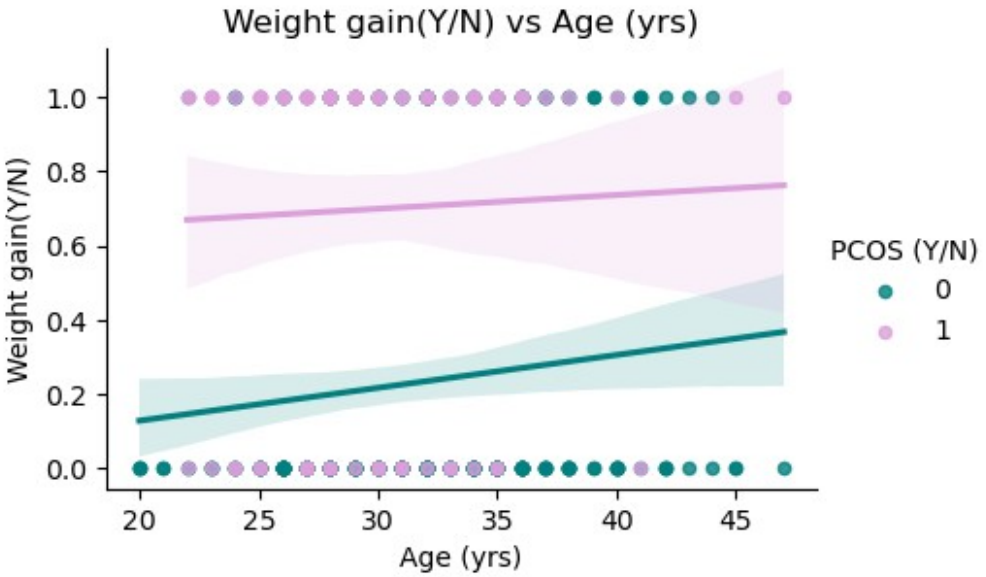

RR (breaths/min) vs Age (yrs)
RR (breaths/min)
22
21
20
19
18
17
16
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

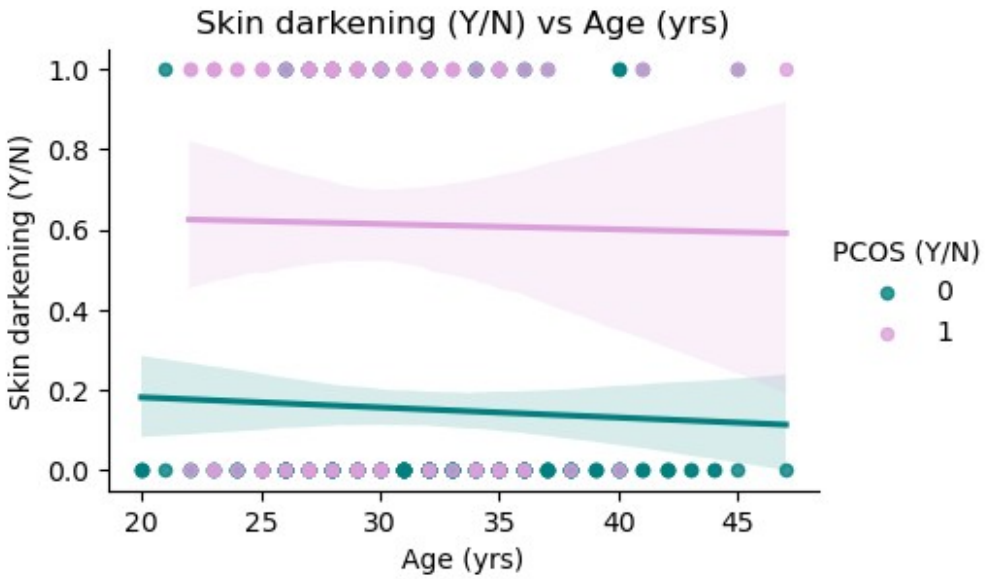

BP _Systolic (mmHg) vs Age (yrs)
BP _Systolic (mmHg)
130
125
120
115
110
105
100
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

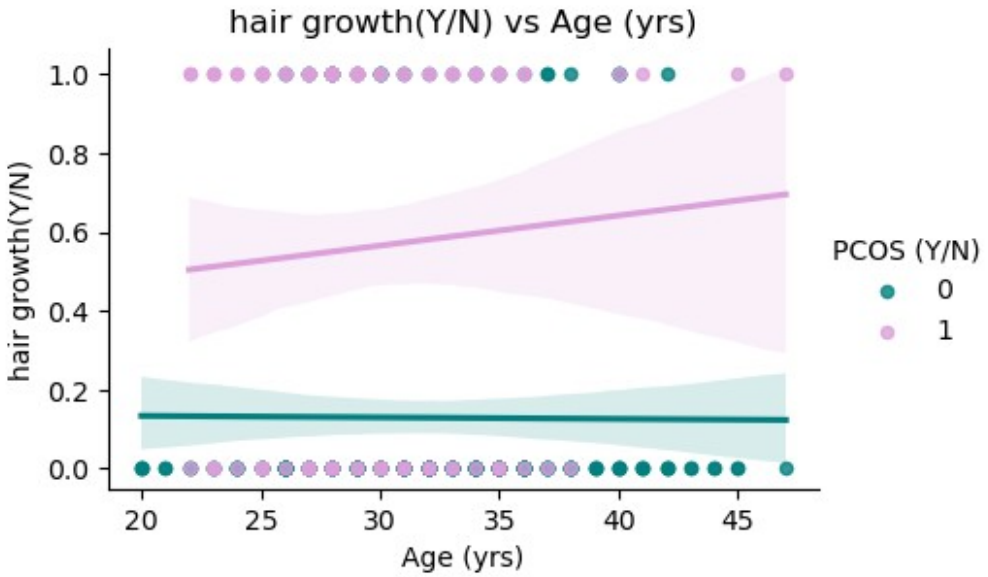

BP _Diastolic (mmHg) vs Age (yrs)
BP _Diastolic (mmHg)
82
80
78
76
74
72
70
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

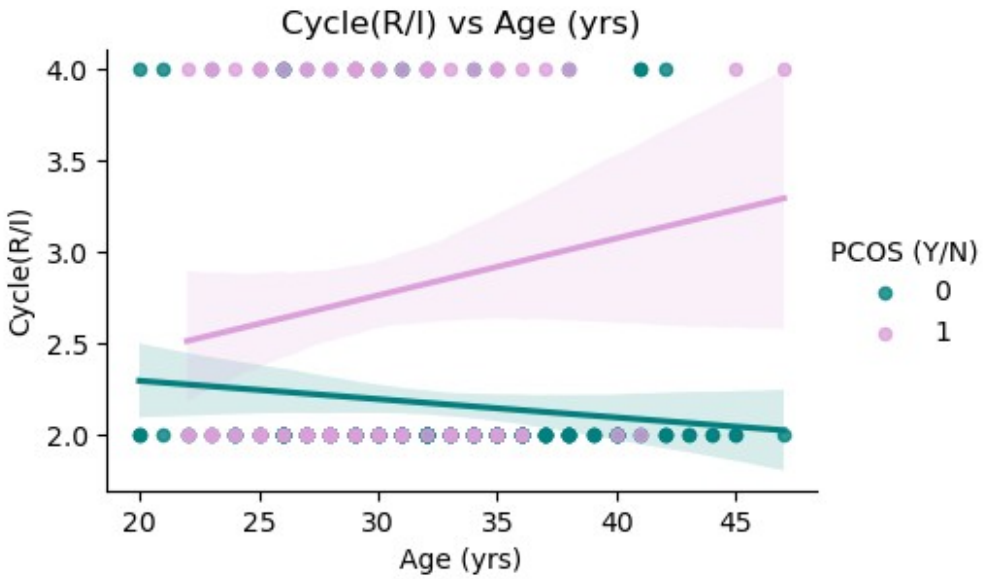

Waist:Hip Ratio vs Age (yrs)
Waist:Hip Ratio
0.95
0.90
0.85
0.80
0.75
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

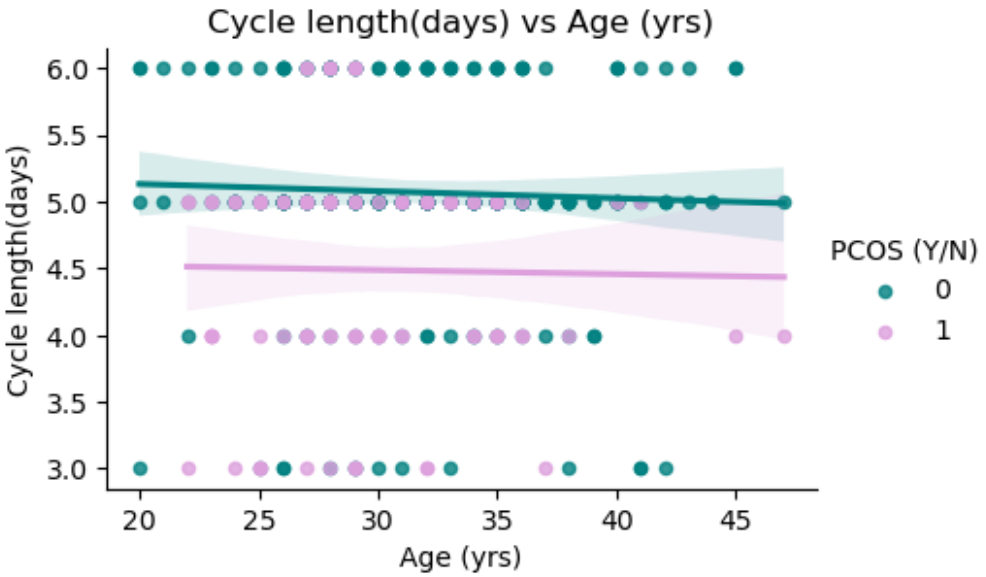

Weight (Kg) vs Age (yrs)
Weight (Kg)
80
70
60
50
40
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

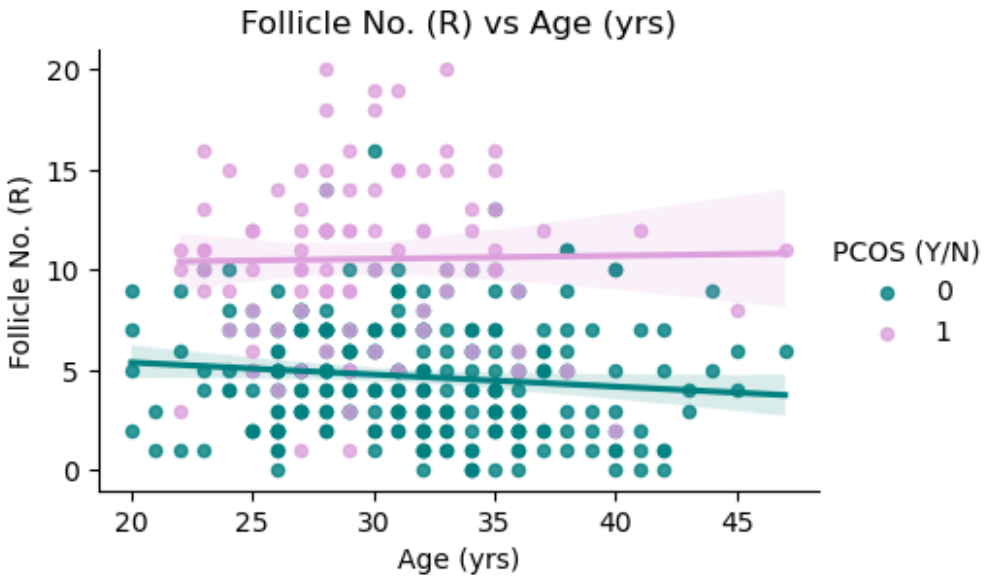

Weight gain(Y/N) vs Age (yrs)
Weight gain(Y/N)
1.0
0.8
0.6
0.4
0.2
0.0
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

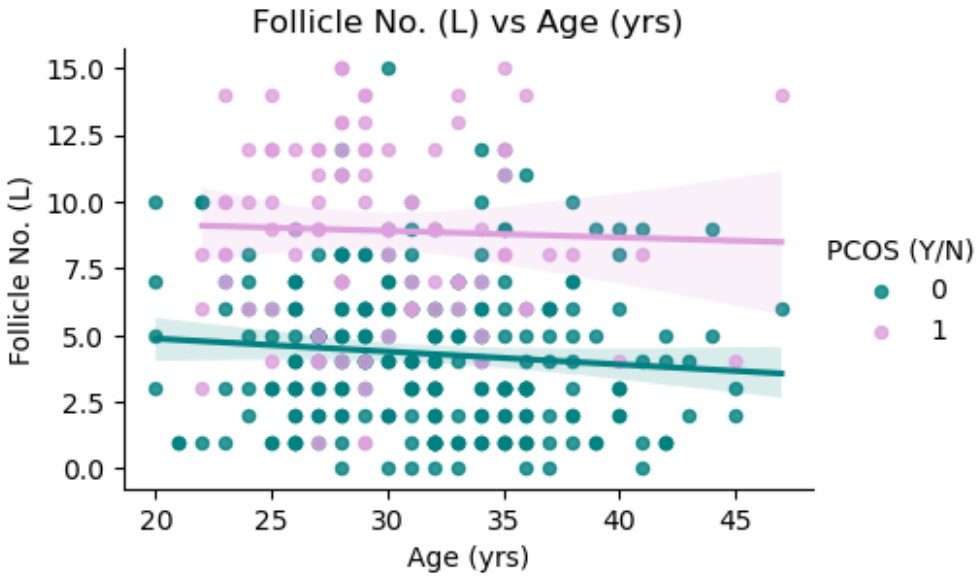

Skin darkening (Y/N) vs Age (yrs)
Skin darkening (Y/N)
1.0
0.8
0.6
0.4
0.2
0.0
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1


hair growth(Y/N) vs Age (yrs)
hair growth(Y/N)
1.0
0.8
0.6
0.4
0.2
0.0
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1


Cycle(R/I) vs Age (yrs)
Cycle(R/I)
4.0
3.5
3.0
2.5
2.0
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1


Cycle length(days) vs Age (yrs)
Cycle length(days)
6.0
5.5
5.0
4.5
4.0
3.5
3.0
20
25
30
35
40
45
Age (yrs)
PCOS (Y/N)
0
1

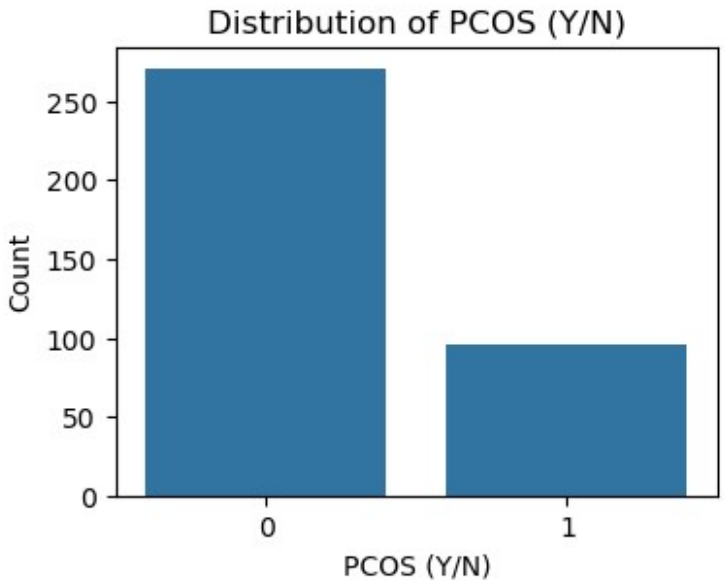




---------- Check Data Balance ----------

```
# Check the distribution of the target variable (e.g., 'PCOS (Y/N)' or
disease stage column)
target_column = 'PCOS (Y/N)'

# Check the counts of each class
class_counts = data[target_column].value_counts()

# Display the class counts
print(f"\nClass Distribution in {target_column}:\n", class_counts)

# Plotting the class distribution
```

```
plt.figure(figsize=(4, 3))
sns.countplot(x=target_column, data=data)
plt.title(f'Distribution of {target_column}')
plt.xlabel(target_column)
plt.ylabel('Count')
plt.show()


Class Distribution in PCOS (Y/N):
 PCOS (Y/N)
0    271
1     96
Name: count, dtype: int64
```



---------- Model Building ----------

```
# Define features (X) and target (y)
X = data.drop(columns=['PCOS (Y/N)'])
y = data['PCOS (Y/N)']

# Split into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=5)


from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
import pandas as pd
from collections import Counter

# Standardizing the dataset
```

```
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train),
columns=X_train.columns)
X_test = pd.DataFrame(scaler.transform(X_test),
columns=X_test.columns)

# Handling Imbalance in dataset
smote = SMOTE(random_state=5)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)

# Check class distribution after SMOTE
print("Before SMOTE:", Counter(y_train))
print("After SMOTE:", Counter(y_train_balanced))
```

```
Before SMOTE: Counter({0: 211, 1: 82})
After SMOTE: Counter({0: 211, 1: 211})
```

```
X_train = X_train_balanced
y_train = y_train_balanced
```

Logistic Regression

```
# Train Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Predictions
y_pred_logreg = logreg.predict(X_test)

# Evaluation
print("Logistic Regression Results:")
print(accuracy_score(y_test, y_pred_logreg))
print(classification_report(y_test, y_pred_logreg))
```

```
Logistic Regression Results:
0.9054054054054054
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        60
           1       0.71      0.86      0.77        14

    accuracy                           0.91        74
   macro avg       0.84      0.89      0.86        74
weighted avg       0.92      0.91      0.91        74
```

Random Forest Classifier

```
# Train Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
rf.fit(X_train, y_train)

# Predictions
y_pred_rf = rf.predict(X_test)

# Evaluation
print("Random Forest Results:")
print(accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Results:
0.9054054054054054
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        60
           1       0.73      0.79      0.76        14

    accuracy                           0.91        74
   macro avg       0.84      0.86      0.85        74
weighted avg       0.91      0.91      0.91        74
```

Compare Models & Select the Best One

```
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score

# Function to evaluate the model
def evaluate_model(model_name, y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    print(f"\n===== {model_name} =====")
    print(f"Confusion Matrix:\n{cm}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")

    return {'model': model_name, 'accuracy': accuracy, 'precision':
precision, 'recall': recall, 'f1': f1}

# Evaluate Logistic Regression
logreg_results = evaluate_model("Logistic Regression", y_test,
y_pred_logreg)

# Evaluate Random Forest
```

```
rf_results = evaluate_model("Random Forest", y_test, y_pred_rf)

# Compare and select the best model based on F1-score
best_model = max([logreg_results, rf_results], key=lambda x: x['f1'])

print(f"\nBest Model: {best_model['model']} with F1-score:
{best_model['f1']:.4f}")
```

```
===== Logistic Regression =====
Confusion Matrix:
[[55  5]
 [ 2 12]]
Accuracy: 0.9054
Precision: 0.7059
Recall: 0.8571
F1-score: 0.7742

===== Random Forest =====
Confusion Matrix:
[[56  4]
 [ 3 11]]
Accuracy: 0.9054
Precision: 0.7333
Recall: 0.7857
F1-score: 0.7586

Best Model: Logistic Regression with F1-score: 0.7742
```

Try to improve Logistic Regression using Hyperparameter Tuning!

Why Hyperparameter Tuning?

 Better Regularization → Prevents overfitting or underfitting
 Optimized Solver → Faster convergence & better performance
 Higher Accuracy & F1-Score → Improves classification results

Hyperparameters to Tune for Logistic Regression:

- **C (Regularization strength)** → Controls the penalty (default: 1.0)
- **Solver** → Optimization algorithm (`liblinear`, `lbfgs`, `saga`)
- **Penalty** → `l1`, `l2`, `elasticnet` (L1 is for feature selection)
- **Max Iterations** → Increase if the model is not converging

Hyperparameter Tuning using GridSearchCV

```
import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Suppress warnings
```

```
warnings.filterwarnings("ignore")

# Define the parameter grid
param_grid = {
    'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],  # Regularization
strength
    'solver': ['liblinear', 'lbfgs', 'saga'],  # Valid solvers
    'penalty': ['l2'],  # L1 is removed because 'lbfgs' doesn't
support it
    'class_weight': [None, 'balanced']
}

# Initialize Logistic Regression
logreg = LogisticRegression(max_iter=50)

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='f1',
n_jobs=-1, error_score='raise')
grid_search.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)

# Train the best model
best_logreg = grid_search.best_estimator_
y_pred_best_logreg = best_logreg.predict(X_test)

# Evaluate the optimized model
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

print("===== Tuned Logistic Regression =====")

print("Accuracy:", accuracy_score(y_test, y_pred_best_logreg))
print("Classification Report:\n", classification_report(y_test,
y_pred_best_logreg))
```

```
Best Parameters: {'C': 1, 'class_weight': None, 'penalty': 'l2',
'solver': 'lbfgs'}
===== Tuned Logistic Regression =====
Accuracy: 0.9054054054054054
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        60
           1       0.71      0.86      0.77        14

    accuracy                           0.91        74
   macro avg       0.84      0.89      0.86        74
weighted avg       0.92      0.91      0.91        74
```

hyperparameter tuning using `GridSearchCV` for **Random Forest**:

Steps Covered:

1. Define a **parameter grid** with key hyperparameters for Random Forest.
2. Perform **Grid Search with Cross-Validation** (`cv=5`) using **F1-score** as the evaluation metric.
3. Train the **best model** and evaluate its performance.

What's Tuned in Random Forest?

- `n_estimators`: Number of trees in the forest
- `max_depth`: Depth of trees to prevent overfitting
- `min_samples_split`: Minimum samples required to split a node
- `min_samples_leaf`: Minimum samples required at a leaf node
- `bootstrap`: Whether to use bootstrap samples
- `class_weight`: To handle imbalanced data

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import warnings

# Suppress warnings
warnings.filterwarnings("ignore")

# Define the parameter grid for Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 200],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],  # Depth of trees
    'min_samples_split': [2, 5, 10],  # Minimum samples required to
split a node
    'min_samples_leaf': [1, 2, 4],  # Minimum samples at a leaf node
    'bootstrap': [True, False],  # Use bootstrap samples
    'class_weight': [None, 'balanced']
}

# Initialize Random Forest Classifier
rf = RandomForestClassifier(random_state=42)

# Perform Grid Search with Cross-Validation
grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='f1',
```

```
n_jobs=-1, error_score='raise')
grid_search_rf.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", grid_search_rf.best_params_)

# Train the best model
best_rf = grid_search_rf.best_estimator_
y_pred_best_rf = best_rf.predict(X_test)

# Evaluate the optimized model
print("===== Tuned Random Forest =====")
print("Accuracy:", accuracy_score(y_test, y_pred_best_rf))
print("Classification Report:\n", classification_report(y_test,
y_pred_best_rf))
```

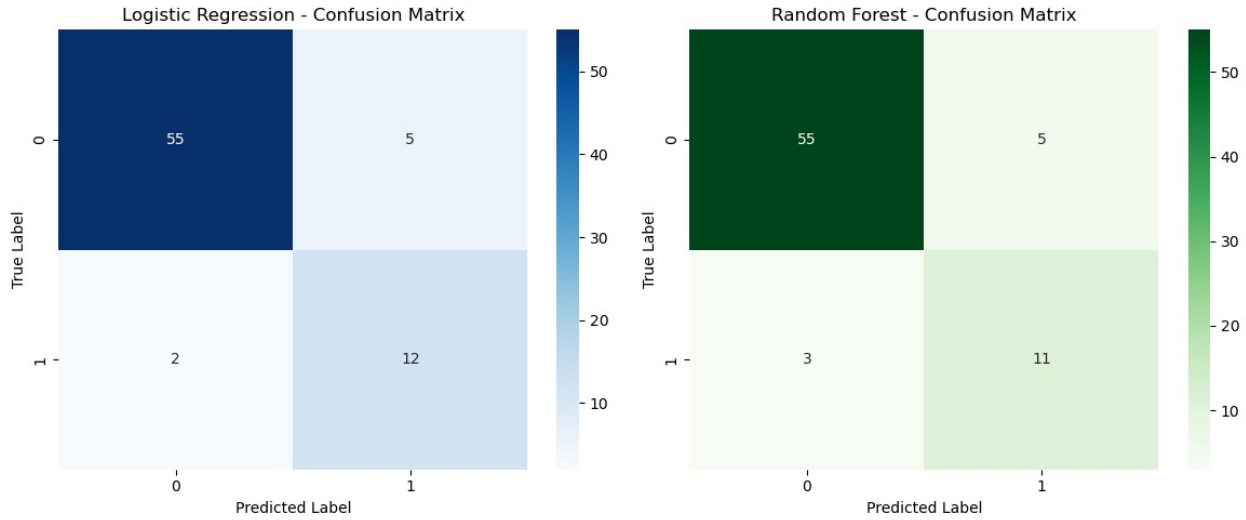
```
Best Parameters: {'bootstrap': False, 'class_weight': 'balanced',
'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 200}
===== Tuned Random Forest =====
Accuracy: 0.8918918918918919
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.92      0.93        60
           1       0.69      0.79      0.73        14

    accuracy                           0.89        74
   macro avg       0.82      0.85      0.83        74
weighted avg       0.90      0.89      0.89        74
```

---------- Compare Tuned Logistic Regression & Random Forest ----------

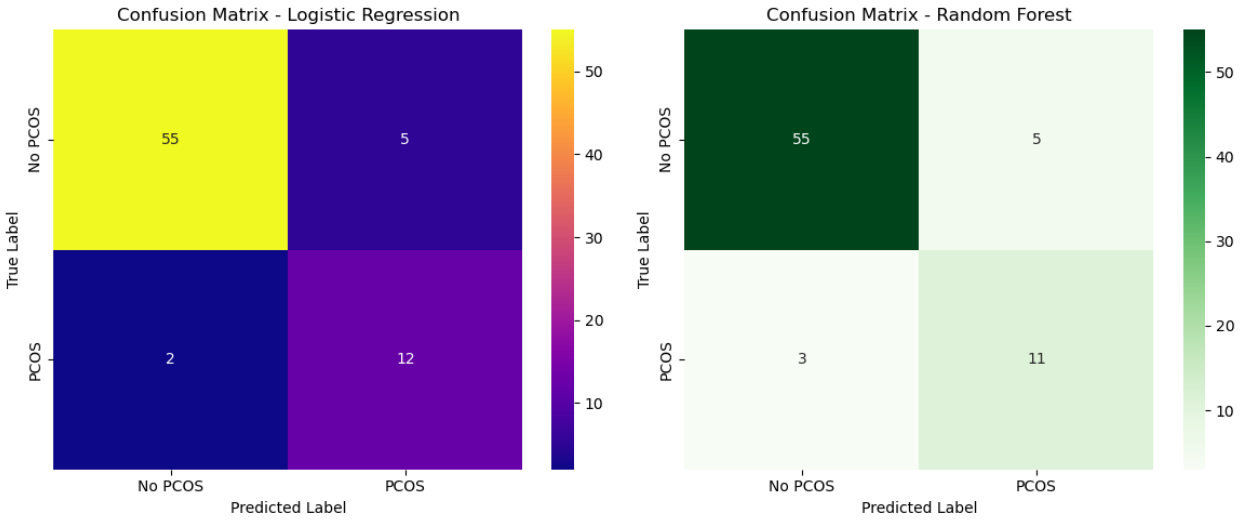
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score, confusion_matrix

# Compute performance metrics for Logistic Regression
logreg_metrics = {
    "Model": "Logistic Regression",
    "Accuracy": accuracy_score(y_test, y_pred_best_logreg),
    "F1 Score": f1_score(y_test, y_pred_best_logreg),
    "Precision": precision_score(y_test, y_pred_best_logreg),
    "Recall": recall_score(y_test, y_pred_best_logreg)
}
```

```python
# Compute performance metrics for Random Forest
rf_metrics = {
    "Model": "Random Forest",
    "Accuracy": accuracy_score(y_test, y_pred_best_rf),
    "F1 Score": f1_score(y_test, y_pred_best_rf),
    "Precision": precision_score(y_test, y_pred_best_rf),
    "Recall": recall_score(y_test, y_pred_best_rf)
}

# Convert results to a DataFrame for easy comparison
comparison_df = pd.DataFrame([logreg_metrics, rf_metrics])

# Display performance comparison table
print("===== Model Performance Comparison =====")
print(comparison_df)

# Plot Confusion Matrices
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Logistic Regression Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred_best_logreg), annot=True,
fmt='d', cmap="Blues", ax=axes[0])
axes[0].set_title("Logistic Regression - Confusion Matrix")
axes[0].set_xlabel("Predicted Label")
axes[0].set_ylabel("True Label")

# Random Forest Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred_best_rf), annot=True,
fmt='d', cmap="Greens", ax=axes[1])
axes[1].set_title("Random Forest - Confusion Matrix")
axes[1].set_xlabel("Predicted Label")
axes[1].set_ylabel("True Label")

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

```
===== Model Performance Comparison =====
                 Model  Accuracy  F1 Score  Precision    Recall
0  Logistic Regression  0.905405  0.774194   0.705882  0.857143
1        Random Forest  0.891892  0.733333   0.687500  0.785714
```




```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Compute confusion matrices
cm_logreg = confusion_matrix(y_test, y_pred_best_logreg)
cm_rf = confusion_matrix(y_test, y_pred_best_rf)

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Logistic Regression Confusion Matrix
sns.heatmap(cm_logreg, annot=True, fmt="d", cmap="plasma",
xticklabels=["No PCOS", "PCOS"], yticklabels=["No PCOS", "PCOS"],
ax=axes[0])
axes[0].set_title("Confusion Matrix - Logistic Regression")
axes[0].set_xlabel("Predicted Label")
axes[0].set_ylabel("True Label")

# Random Forest Confusion Matrix
sns.heatmap(cm_rf, annot=True, fmt="d", cmap="Greens",
xticklabels=["No PCOS", "PCOS"], yticklabels=["No PCOS", "PCOS"],
ax=axes[1])
axes[1].set_title("Confusion Matrix - Random Forest")
axes[1].set_xlabel("Predicted Label")
axes[1].set_ylabel("True Label")

# Adjust layout
plt.tight_layout()
plt.show()
```



---------- Final Note ----------

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

# Compute performance metrics for both models
metrics = {
    "Model": ["Logistic Regression", "Random Forest"],
    "Accuracy": [accuracy_score(y_test, y_pred_best_logreg),
accuracy_score(y_test, y_pred_best_rf)],
    "Precision": [precision_score(y_test, y_pred_best_logreg),
precision_score(y_test, y_pred_best_rf)],
    "Recall": [recall_score(y_test, y_pred_best_logreg),
recall_score(y_test, y_pred_best_rf)],
    "F1-Score": [f1_score(y_test, y_pred_best_logreg),
f1_score(y_test, y_pred_best_rf)]
}

# Convert to a DataFrame for better visualization
import pandas as pd
df_metrics = pd.DataFrame(metrics)
print("=== Model Performance Comparison ===")
print(df_metrics)

# Identify the best model based on F1-score
best_model_name = df_metrics.iloc[df_metrics['F1-Score'].idxmax()]
['Model']
best_model = best_logreg if best_model_name == "Logistic Regression"
else best_rf

print("\n===== Best Model Identified =====")
print(f"Best Model: {best_model_name}")

# Print model structure and hyperparameters
```

```
print("\n=== Model Structure & Hyperparameters ===")
print(best_model)

# Conclusion
print("\n===== Conclusion =====")
if best_model_name == "Random Forest":
    print("Random Forest performed better with higher F1-score, making
it the best choice for PCOS prediction.")
else:
    print("Logistic Regression performed better, making it the
preferred model for PCOS prediction.")
```

```
=== Model Performance Comparison ===
                 Model  Accuracy  Precision    Recall  F1-Score
0  Logistic Regression  0.905405   0.705882  0.857143  0.774194
1        Random Forest  0.891892   0.687500  0.785714  0.733333

===== Best Model Identified =====
Best Model: Logistic Regression

=== Model Structure & Hyperparameters ===
LogisticRegression(C=1, max_iter=50)

===== Conclusion =====
Logistic Regression performed better, making it the preferred model
for PCOS prediction.
```

---------- END ----------

```
!pip install scikeras
```

```
Requirement already satisfied: scikeras in c:\users\soumiz\anaconda3\
lib\site-packages (0.13.0)
Requirement already satisfied: keras>=3.2.0 in c:\users\soumiz\
anaconda3\lib\site-packages (from scikeras) (3.2.0)
Requirement already satisfied: scikit-learn>=1.4.2 in c:\users\soumiz\
anaconda3\lib\site-packages (from scikeras) (1.4.2)
Requirement already satisfied: absl-py in c:\users\soumiz\anaconda3\
lib\site-packages (from keras>=3.2.0->scikeras) (2.1.0)
Requirement already satisfied: numpy in c:\users\soumiz\anaconda3\lib\
site-packages (from keras>=3.2.0->scikeras) (1.26.4)
Requirement already satisfied: rich in c:\users\soumiz\anaconda3\lib\
site-packages (from keras>=3.2.0->scikeras) (13.3.5)
Requirement already satisfied: namex in c:\users\soumiz\anaconda3\lib\
site-packages (from keras>=3.2.0->scikeras) (0.0.8)
```

```
Requirement already satisfied: h5py in c:\users\soumiz\anaconda3\lib\
site-packages (from keras>=3.2.0->scikeras) (3.11.0)
Requirement already satisfied: optree in c:\users\soumiz\anaconda3\
lib\site-packages (from keras>=3.2.0->scikeras) (0.12.1)
Requirement already satisfied: ml-dtypes in c:\users\soumiz\anaconda3\
lib\site-packages (from keras>=3.2.0->scikeras) (0.4.0)
Requirement already satisfied: scipy>=1.6.0 in c:\users\soumiz\
anaconda3\lib\site-packages (from scikit-learn>=1.4.2->scikeras)
(1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\soumiz\
anaconda3\lib\site-packages (from scikit-learn>=1.4.2->scikeras)
(1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\
soumiz\anaconda3\lib\site-packages (from scikit-learn>=1.4.2-
>scikeras) (2.2.0)
Requirement already satisfied: typing-extensions>=4.5.0 in c:\users\
soumiz\anaconda3\lib\site-packages (from optree->keras>=3.2.0-
>scikeras) (4.11.0)
Requirement already satisfied: markdown-it-py<3.0.0,>=2.2.0 in c:\
users\soumiz\anaconda3\lib\site-packages (from rich->keras>=3.2.0-
>scikeras) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\
soumiz\anaconda3\lib\site-packages (from rich->keras>=3.2.0->scikeras)
(2.15.1)
Requirement already satisfied: mdurl~=0.1 in c:\users\soumiz\
anaconda3\lib\site-packages (from markdown-it-py<3.0.0,>=2.2.0->rich-
>keras>=3.2.0->scikeras) (0.1.0)
```

Method 1: Classification with Logistic Regression and Random Forest

We will use **Logistic Regression and Random Forest?**

For **binary classification task (PCOS: 0 or 1)**, these two modelarere selected because they offer a balance between **simplicity, interpretability, and predictive power**.

---

1. Logistic Regression

Why?

- It is a **baseline model** for binary classification.
- Provides **probabilistic outputs**, which can be useful for medical predictions.
- Easy to interpret the effect of features (weights of coefficients).

- Works well when features are **independent and linearly separable**.
- Computationally efficient, especially for small to medium datasets.

 When it struggles?

- If data has **complex, non-linear relationships**, Logistic Regression may underperform.

---

2 Random Forest

 Why?

- Handles **non-linearity** well, unlike Logistic Regression.
- Less sensitive to outliers and feature correlations.
- **Feature importance analysis** helps in understanding key predictors.
- **Better generalization** due to ensemble learning (multiple decision trees).

 When it struggles?

- Can be **computationally expensive** for very large datasets.
- Requires **hyperparameter tuning** (e.g., number of trees, depth).

Why Not Other Methods?

Model	Why Not Used?
SVM	Works well for binary classification, but tuning kernels is computationally expensive.
KNN	Sensitive to feature scaling and less interpretable.
XGBoost	Powerful, but needs more hyperparameter tuning and may be overkill for small datasets.
Neural Networks	Needs large data; can be complex and hard to interpret.

Final Thoughts

- **Logistic Regression** gives a **simple, interpretable baseline**.
- **Random Forest** provides a **stronger, flexible model** for complex relationships.
- If **Random Forest performs significantly better**, it suggests **non-linearity** in the data.
- If both perform similarly, **Logistic Regression seLet's goe to try** XGBoost or SVM for further comparison?** o try **XGBoost or SVM for further comparison?**

```
data = data_filtered.copy()

# Define features (X) and target (y)
X = data.drop(columns=['PCOS (Y/N)'])
y = data['PCOS (Y/N)']

# Split into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardizing the dataset for better performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Logistic Regression

```
# Train Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Predictions
y_pred_logreg = logreg.predict(X_test)

# Evaluation
print("Logistic Regression Results:")
print(accuracy_score(y_test, y_pred_logreg))
print(classification_report(y_test, y_pred_logreg))
```

```
Logistic Regression Results:
0.7522935779816514
              precision    recall  f1-score   support

           0       0.81      0.84      0.83        77
           1       0.59      0.53      0.56        32

    accuracy                           0.75       109
   macro avg       0.70      0.69      0.69       109
weighted avg       0.75      0.75      0.75       109
```

Random Forest Classifier

```
# Train Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predictions
y_pred_rf = rf.predict(X_test)

# Evaluation
```

```
print("Random Forest Results:")
print(accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Results:
0.7706422018348624
              precision    recall  f1-score   support

           0       0.82      0.86      0.84        77
           1       0.62      0.56      0.59        32

    accuracy                           0.77       109
   macro avg       0.72      0.71      0.72       109
weighted avg       0.77      0.77      0.77       109
```

Compare Models & Select the Best One

```
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score

# Function to evaluate the model
def evaluate_model(model_name, y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    print(f"\n===== {model_name} =====")
    print(f"Confusion Matrix:\n{cm}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")

    return {'model': model_name, 'accuracy': accuracy, 'precision':
precision, 'recall': recall, 'f1': f1}

# Evaluate Logistic Regression
logreg_results = evaluate_model("Logistic Regression", y_test,
y_pred_logreg)

# Evaluate Random Forest
rf_results = evaluate_model("Random Forest", y_test, y_pred_rf)

# Compare and select the best model based on F1-score
best_model = max([logreg_results, rf_results], key=lambda x: x['f1'])
```

```
print(f"\nBest Model: {best_model['model']} with F1-score:
{best_model['f1']:.4f}")
```

```
===== Logistic Regression =====
Confusion Matrix:
[[65 12]
 [15 17]]
Accuracy: 0.7523
Precision: 0.5862
Recall: 0.5312
F1-score: 0.5574

===== Random Forest =====
Confusion Matrix:
[[66 11]
 [14 18]]
Accuracy: 0.7706
Precision: 0.6207
Recall: 0.5625
F1-score: 0.5902

Best Model: Random Forest with F1-score: 0.5902
```

The F1 score is low In above process, we have avoided few features having better correlation with PCOS as those can not be measured by smart gadgets. We have added those features to create a new dataset and tried the classification again

```
# Additional features to be added
new_features = ['Follicle No. (R)', 'Follicle No. (L)', 'Skin
darkening (Y/N)',
                'hair growth(Y/N)', 'Weight gain(Y/N)']

# Ensure data_filtered is a copy to avoid SettingWithCopyWarning
data_filtered_short = data_filtered.copy()

# Copy these features from data_reduced to data_filtered
for feature in new_features:
    data_filtered[feature] = data_reduced[feature]

# Display first few rows to verify
display(data_filtered.head())

# Display summary statistics
data_filtered.describe()

# save this file
data_filtered.to_csv('Data_Filtered.csv', index = False)
```

```
C:\Users\Soumiz\AppData\Local\Temp\ipykernel_19620\3918952636.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data_filtered[feature] = data_reduced[feature]
C:\Users\Soumiz\AppData\Local\Temp\ipykernel_19620\3918952636.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data_filtered[feature] = data_reduced[feature]
C:\Users\Soumiz\AppData\Local\Temp\ipykernel_19620\3918952636.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data_filtered[feature] = data_reduced[feature]
C:\Users\Soumiz\AppData\Local\Temp\ipykernel_19620\3918952636.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data_filtered[feature] = data_reduced[feature]
C:\Users\Soumiz\AppData\Local\Temp\ipykernel_19620\3918952636.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data_filtered[feature] = data_reduced[feature]
```

	PCOS (Y/N)	Cycle(R/I)	Pulse rate(bpm)	RR (breaths/min) \
0	0	2	78	22
1	0	2	74	20
2	1	2	72	18

```
3           0           2                72                  20
4           0           2                72                  18

   BP _Systolic (mmHg)  BP _Diastolic (mmHg)  Waist:Hip Ratio  Weight
(Kg)  \
0                   110                    80             0.83
44.6
1                   120                    70             0.84
65.0
2                   120                    80             0.90
68.8
3                   120                    70             0.86
65.0
4                   120                    80             0.81
52.0

   Follicle No. (R)  Follicle No. (L)  Skin darkening (Y/N)  hair
growth(Y/N)  \
0                 3                 3                     0
0
1                 5                 3                     0
0
2                15                13                     0
0
3                 2                 2                     0
0
4                 4                 3                     0
0

   Weight gain(Y/N)
0                 0
1                 0
2                 0
3                 0
4                 0
```

```
data = data_filtered.copy()

# Define features (X) and target (y)
X = data.drop(columns=['PCOS (Y/N)'])
y = data['PCOS (Y/N)']

# Split into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardizing the dataset for better performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Logistic Regression

```
# Train Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Predictions
y_pred_logreg = logreg.predict(X_test)

# Evaluation
print("Logistic Regression Results:")
print(accuracy_score(y_test, y_pred_logreg))
print(classification_report(y_test, y_pred_logreg))
```

```
Logistic Regression Results:
0.8807339449541285
              precision    recall  f1-score   support

           0       0.89      0.95      0.92        77
           1       0.85      0.72      0.78        32

    accuracy                           0.88       109
   macro avg       0.87      0.83      0.85       109
weighted avg       0.88      0.88      0.88       109
```

Random Forest Classifier

```
# Train Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predictions
y_pred_rf = rf.predict(X_test)

# Evaluation
print("Random Forest Results:")
print(accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Results:
0.8623853211009175
              precision    recall  f1-score   support

           0       0.88      0.94      0.91        77
           1       0.81      0.69      0.75        32

    accuracy                           0.86       109
   macro avg       0.85      0.81      0.83       109
weighted avg       0.86      0.86      0.86       109
```

Compare Models & Select the Best One

```
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score

# Function to evaluate the model
def evaluate_model(model_name, y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    print(f"\n===== {model_name} =====")
    print(f"Confusion Matrix:\n{cm}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")

    return {'model': model_name, 'accuracy': accuracy, 'precision':
precision, 'recall': recall, 'f1': f1}

# Evaluate Logistic Regression
logreg_results = evaluate_model("Logistic Regression", y_test,
y_pred_logreg)

# Evaluate Random Forest
rf_results = evaluate_model("Random Forest", y_test, y_pred_rf)

# Compare and select the best model based on F1-score
best_model = max([logreg_results, rf_results], key=lambda x: x['f1'])

print(f"\nBest Model: {best_model['model']} with F1-score:
{best_model['f1']:.4f}")


===== Logistic Regression =====
Confusion Matrix:
[[73  4]
 [ 9 23]]
Accuracy: 0.8807
Precision: 0.8519
Recall: 0.7188
F1-score: 0.7797

===== Random Forest =====
Confusion Matrix:
[[72  5]
 [10 22]]
Accuracy: 0.8624
```

```
Precision: 0.8148
Recall: 0.6875
F1-score: 0.7458

Best Model: Logistic Regression with F1-score: 0.7797
```

Try to improve Logistic Regression using Hyperparameter Tuning!

Why Hyperparameter Tuning?

 Better Regularization → Prevents overfitting or underfitting
 Optimized Solver → Faster convergence & better performance
 Higher Accuracy & F1-Score → Improves classification results

Hyperparameters to Tune for Logistic Regression:

- **C (Regularization strength)** → Controls the penalty (default: 1.0)
- **Solver** → Optimization algorithm (`liblinear`, `lbfgs`, `saga`)
- **Penalty** → `l1`, `l2`, `elasticnet` (L1 is for feature selection)
- **Max Iterations** → Increase if the model is not converging

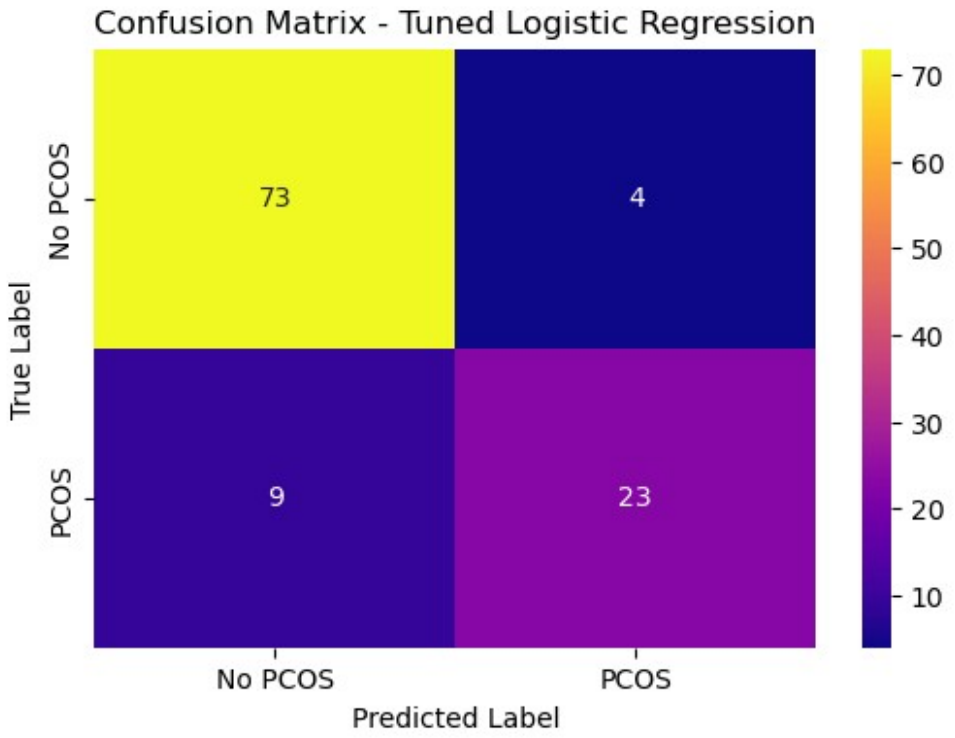
Hyperparameter Tuning using GridSearchCV

```
import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Suppress warnings
warnings.filterwarnings("ignore")

# Define the parameter grid
param_grid = {
    'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],  # Regularization
strength
    'solver': ['liblinear', 'lbfgs', 'saga'],  # Valid solvers
    'penalty': ['l2'],  # L1 is removed because 'lbfgs' doesn't
support it
    'class_weight': [None, 'balanced']
}

# Initialize Logistic Regression
logreg = LogisticRegression(max_iter=50)

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='f1',
n_jobs=-1, error_score='raise')
grid_search.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)
```

```python
# Train the best model
best_logreg = grid_search.best_estimator_
y_pred_best_logreg = best_logreg.predict(X_test)

# Evaluate the optimized model
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

print("===== Tuned Logistic Regression =====")

print("Accuracy:", accuracy_score(y_test, y_pred_best_logreg))
print("Classification Report:\n", classification_report(y_test,
y_pred_best_logreg))
```

```
Best Parameters: {'C': 1, 'class_weight': None, 'penalty': 'l2',
'solver': 'lbfgs'}
===== Tuned Logistic Regression =====
Accuracy: 0.8807339449541285
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.95      0.92        77
           1       0.85      0.72      0.78        32

    accuracy                           0.88       109
   macro avg       0.87      0.83      0.85       109
weighted avg       0.88      0.88      0.88       109
```

```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_best_logreg)

# Plot the heatmap
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="plasma", xticklabels=["No
PCOS", "PCOS"], yticklabels=["No PCOS", "PCOS"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Tuned Logistic Regression")
plt.show()
```

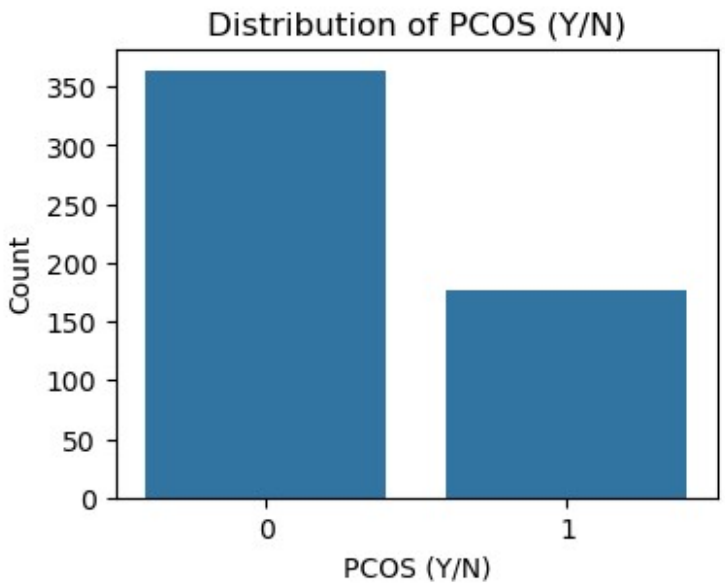


Imbalance Indentification

Check Balance in Disease Stages

```
# Check the distribution of the target variable (e.g., 'PCOS (Y/N)' or
disease stage column)
target_column = 'PCOS (Y/N)'

# Check the counts of each class
class_counts = data[target_column].value_counts()

# Display the class counts
print(f"\nClass Distribution in {target_column}:\n", class_counts)

# Plotting the class distribution
plt.figure(figsize=(4, 3))
sns.countplot(x=target_column, data=data)
plt.title(f'Distribution of {target_column}')
plt.xlabel(target_column)
plt.ylabel('Count')
plt.show()
```

```
Class Distribution in PCOS (Y/N):
 PCOS (Y/N)
0    364
1    177
Name: count, dtype: int64
```



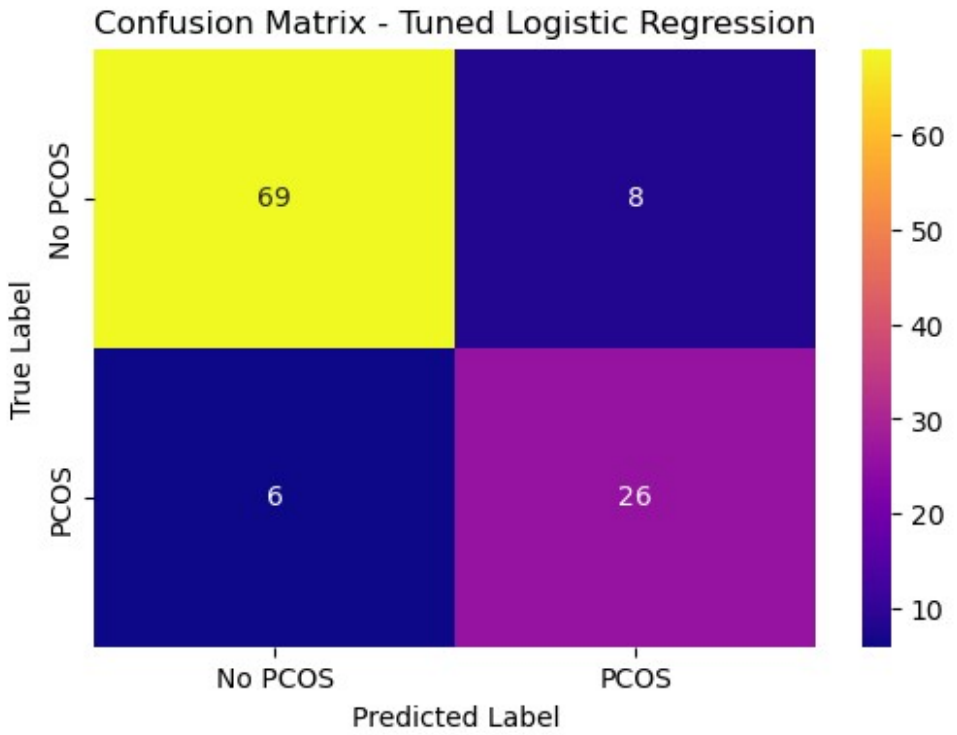
Solution is the use of SMOTE (Synthetic Minority Oversampling Technique)

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)

logreg = LogisticRegression(max_iter=500)
logreg.fit(X_train_balanced, y_train_balanced)
y_pred = logreg.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.90      0.91        77
           1       0.76      0.81      0.79        32

    accuracy                           0.87       109
   macro avg       0.84      0.85      0.85       109
weighted avg       0.87      0.87      0.87       109
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the heatmap
```

```
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="plasma", xticklabels=["No
PCOS", "PCOS"], yticklabels=["No PCOS", "PCOS"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Tuned Logistic Regression")
plt.show()
```



Method 2: Outliers Adjustment and Model Training

```
data = pd. read_csv('data_filtered.csv')


# Dictionary to store outliers
outliers_quantile = {}

# Detect outliers using IQR (Interquartile Range)
for col in data.columns:
        # Compute IQR
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1

        # Identify outliers
        outlier_indices_quantile = data[
            (data[col] < (Q1 - 1.5 * IQR)) | (data[col] > (Q3 + 1.5 *
IQR))
```

```
        ].index

        if len(outlier_indices_quantile) > 0:
            outliers_quantile[col] = outlier_indices_quantile

# Display the results
features_with_outliers = {col: len(indices) for col, indices in
outliers_quantile.items()}
print("\nFeatures with Outliers (as per quantile):",
features_with_outliers)
```

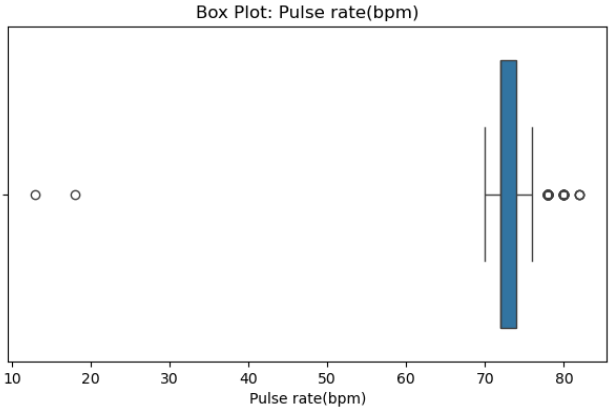
```
Features with Outliers (as per quantile): {'Pulse rate(bpm)': 94, 'RR
(breaths/min)': 14, 'BP _Systolic (mmHg)': 3, 'BP _Diastolic (mmHg)':
2, 'Weight (Kg)': 18, 'Follicle No. (L)': 6}
```

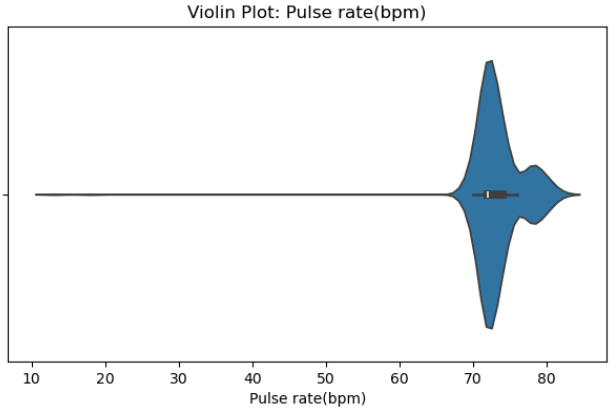
```
# Total number of rows in the dataset
total_rows = len(data)

# Dictionary to store the percentage of outliers
outliers_percentage = {
    col: (len(indices) / total_rows) * 100 for col, indices in
outliers_quantile.items()
}

# Display the percentage of outliers
print("\nPercentage of Outliers per Feature:")
for feature, percentage in outliers_percentage.items():
    print(f"{feature}: {percentage:.2f}%")
```

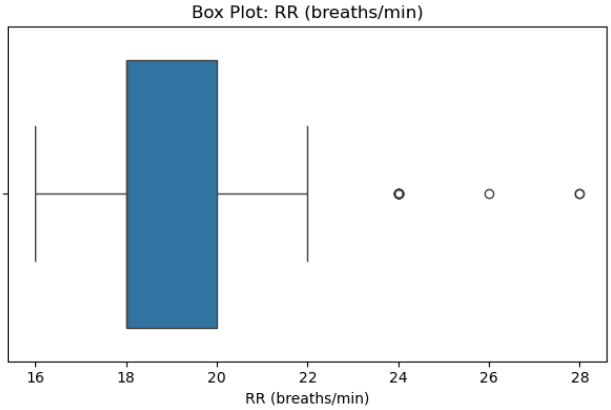
```
Percentage of Outliers per Feature:
Pulse rate(bpm): 17.38%
RR (breaths/min): 2.59%
BP _Systolic (mmHg): 0.55%
BP _Diastolic (mmHg): 0.37%
Weight (Kg): 3.33%
Follicle No. (L): 1.11%
```

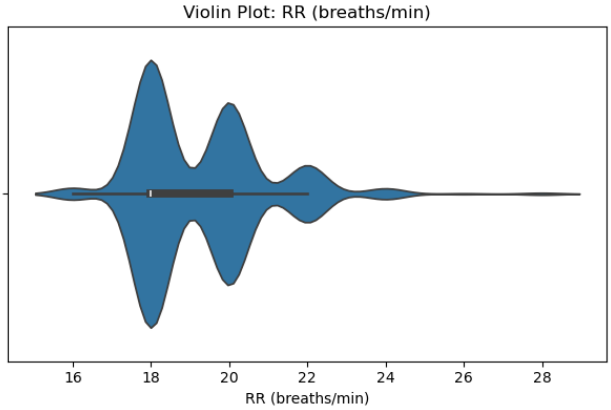
```
import matplotlib.pyplot as plt
import seaborn as sns

# Target column
target_column = 'PCOS (Y/N)'

# Extract all feature columns except the target
outliers_all = [col for col in outliers_quantile.keys() if col !=
target_column]

# Number of rows for subplots (one row per feature)
num_rows = len(outliers_all)
```

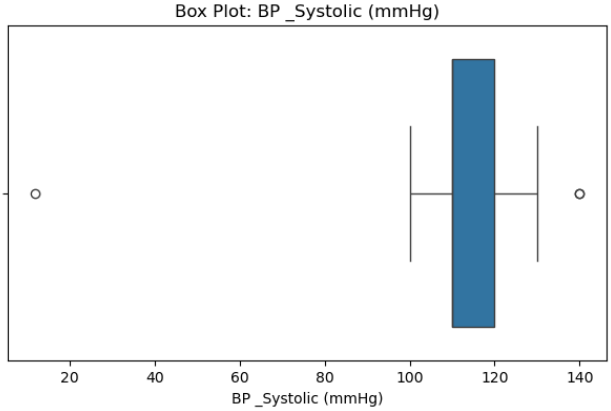
```
# Creating subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(12, num_rows * 4))
axes = axes.reshape(num_rows, 2)  # Ensure it's 2D for indexing

for i, feature in enumerate(outliers_all):
    # Box Plot
    sns.boxplot(data=data, x=feature, ax=axes[i, 0])
    axes[i, 0].set_title(f'Box Plot: {feature}')

    # Violin Plot
    sns.violinplot(data=data, x=feature, ax=axes[i, 1])
    axes[i, 1].set_title(f'Violin Plot: {feature}')

# Adjust layout
plt.tight_layout()
plt.show()
```

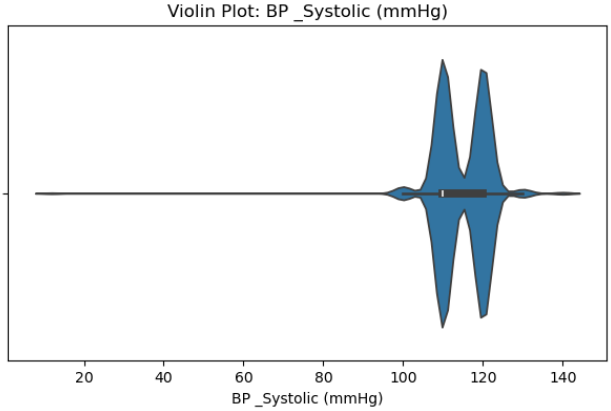

Box Plot: Pulse rate(bpm)
10
20
30
40
50
60
70
80
Pulse rate(bpm)

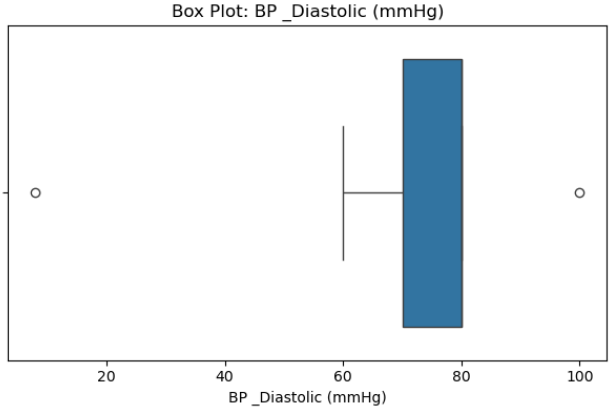

Violin Plot: Pulse rate(bpm)
10
20
30
40
50
60
70
80
Pulse rate(bpm)

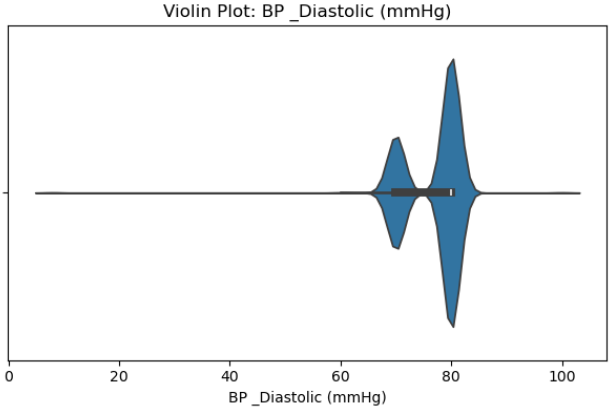

Box Plot: RR (breaths/min)
16
18
20
22
24
26
28
RR (breaths/min)

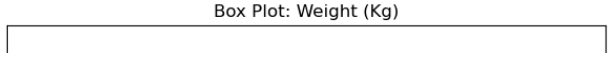

Violin Plot: RR (breaths/min)
16
18
20
22
24
26
28
RR (breaths/min)

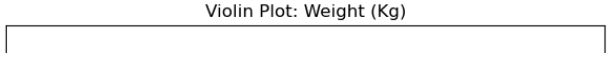

Box Plot: BP _Systolic (mmHg)
20
40
60
80
100
120
140
BP _Systolic (mmHg)


Violin Plot: BP _Systolic (mmHg)
20
40
60
80
100
120
140
BP _Systolic (mmHg)


Box Plot: BP _Diastolic (mmHg)
20
40
60
80
100
BP _Diastolic (mmHg)


Violin Plot: BP _Diastolic (mmHg)
0
20
40
60
80
100
BP _Diastolic (mmHg)


Box Plot: Weight (Kg)


Violin Plot: Weight (Kg)

Proposal for Outlier Handling in PCOS Classification Dataset

During the analysis of our dataset, we identified outliers in multiple features using the Interquartile Range (IQR) method. Below are the features along with the percentage of data points classified as outliers:

- **Pulse rate (bpm)**: 17.38%
- **RR (breaths/min)**: 2.59%
- **BP Systolic (mmHg)**: 0.55%
- **BP Diastolic (mmHg)**: 0.37%
- **Weight (Kg)**: 3.33%
- **Follicle No. (L)**: 1.11%

Since different features have varying degrees of outliers, I will implement specific handling methods based on their impact on the model and the percentage of outliers.

Outlier Handling Methods & Implementation

1. **Pulse rate (bpm) (17.38% outliers) – Winsorization (Capping method)**
   - Since the percentage of outliers is relatively high, directly removing them may lead to a significant data loss.
   - Instead, I will apply **Winsorization**, capping extreme values at the 1st and 99th percentiles.
   - This ensures that extreme values do not skew the model while preserving the majority of the data.
2. **RR (breaths/min) (2.59% outliers) – Mean/Median Imputation**
   - The number of outliers is relatively low, so complete removal is not necessary.
   - I will replace extreme values using **median imputation** since the data may have skewness, making the median a more robust measure.
3. **BP Systolic & Diastolic (0.55% and 0.37% outliers) – No Change**
   - The number of outliers is minimal and does not significantly impact the dataset.
   - Removing or modifying these values may introduce unnecessary bias, so I will **retain these values as they are**.
4. **Weight (Kg) (3.33% outliers) – Winsorization**
   - Since weight is a crucial feature and removing data may lead to loss of information, I will apply **Winsorization (capping at 1st and 99th percentile)** to handle extreme values.

- This ensures that the feature remains within a reasonable range without distorting data distribution.

5. **Follicle No. (L) (1.11% outliers) – No Change**
   - Given the low percentage of outliers and its clinical relevance, I will **retain these values as they are** to maintain the original data distribution.

Justification for Selected Methods

- **Winsorization** is used for features where outliers are significant but retaining information is essential (Pulse rate, Weight).
- **Median imputation** is chosen for RR (breaths/min) since it is less affected by skewness.
- **No modification** is applied to features with very few outliers (BP Systolic/Diastolic and Follicle No. (L)), as their removal or modification may introduce bias.

This approach balances outlier management without excessive data loss or distortion, ensuring that the dataset remains reliable for PCOS classification.

Further Inspection of "Pulse rate(bpm)", "Weight (Kg)" for better implementation of *Winsorization*

```
# Dictionary to store outlier percentages separately for lower and upper bounds
outlier_tails = {}

# Compute percentage of outliers separately for upper and lower tails
for feature in ["Pulse rate(bpm)", "Weight (Kg)"]:
    Q1 = data[feature].quantile(0.25)
    Q3 = data[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    lower_outliers = (data[feature] < lower_bound).sum()
    upper_outliers = (data[feature] > upper_bound).sum()
    total_outliers = lower_outliers + upper_outliers
    total_rows = len(data)

    outlier_tails[feature] = {
        "lower_outliers": lower_outliers,
        "upper_outliers": upper_outliers,
        "lower_percentage": (lower_outliers / total_rows) * 100,
        "upper_percentage": (upper_outliers / total_rows) * 100
    }

# Display results
for feature, stats in outlier_tails.items():
    print(f"\nFeature: {feature}")
    print(f"   Lower Outliers: {stats['lower_outliers']}
```

```
({stats['lower_percentage']:.2f}%)")
    print(f"   Upper Outliers: {stats['upper_outliers']}
({stats['upper_percentage']:.2f}%)")


Feature: Pulse rate(bpm)
   Lower Outliers: 2 (0.37%)
   Upper Outliers: 92 (17.01%)

Feature: Weight (Kg)
   Lower Outliers: 3 (0.55%)
   Upper Outliers: 15 (2.77%)
```

Since the outlier distribution is **asymmetric**, we should apply **custom Winsorization** with different limits for the **lower and upper tails** of each feature.

Custom Handling Strategy:

- **Pulse rate (bpm):**
  - **Lower outliers:** 0.37% → **Ignore** (minimal impact).
  - **Upper outliers:** 17.01% → **Winsorize aggressively** (cap at **99th percentile**).
- **Weight (Kg):**
  - **Lower outliers:** 0.55% → **Mild capping** (cap at **1st percentile**).
  - **Upper outliers:** 2.77% → **Moderate capping** (cap at **98th percentile**).

Custom Winsorization Code

```
# Custom function to apply asymmetric Winsorization
def custom_winsorize(feature, lower_percentile, upper_percentile):
    lower_bound = np.percentile(data[feature], lower_percentile)
    upper_bound = np.percentile(data[feature], upper_percentile)

    # Apply capping
    data[feature] = np.where(data[feature] < lower_bound, lower_bound,
data[feature])
    data[feature] = np.where(data[feature] > upper_bound, upper_bound,
data[feature])

    print(f"{feature} - Winsorization Applied: [{lower_percentile}th,
{upper_percentile}th percentile]")

# Apply custom Winsorization
custom_winsorize("Pulse rate(bpm)", lower_percentile=1,
```

```
upper_percentile=92)   # Strong capping at upper
custom_winsorize("Weight (Kg)", lower_percentile=1,
upper_percentile=96)       # Mild at lower, moderate at upper
```

```
Pulse rate(bpm) - Winsorization Applied: [1th, 92th percentile]
Weight (Kg) - Winsorization Applied: [1th, 96th percentile]
```

Justification for This Approach:

- **Preserves Data Integrity** – Lower outliers in both features are minimal, so they don't require strong adjustments.
- **Prevents Skewed Model Training** – Upper outliers in Pulse rate (bpm) are high (17.01%), so strong Winsorization is needed.
- **Maintains Clinical Relevance** – Weight is a **key factor** in PCOS classification, so handling needs to be balanced.

This method ensures that extreme values do not distort the analysis while keeping meaningful variation in the dataset.

Median Imputation for RR (breaths/min)

```
# Median Imputation for RR (breaths/min)
median_rr = data["RR (breaths/min)"].median()
Q1_rr = data["RR (breaths/min)"].quantile(0.25)
Q3_rr = data["RR (breaths/min)"].quantile(0.75)
IQR_rr = Q3_rr - Q1_rr
lower_bound_rr = Q1_rr - 1.5 * IQR_rr
upper_bound_rr = Q3_rr + 1.5 * IQR_rr
data.loc[(data["RR (breaths/min)"] < lower_bound_rr) | (data["RR
(breaths/min)"] > upper_bound_rr), "RR (breaths/min)"] = median_rr

print('Median Imputation for RR (breaths/min)')
```

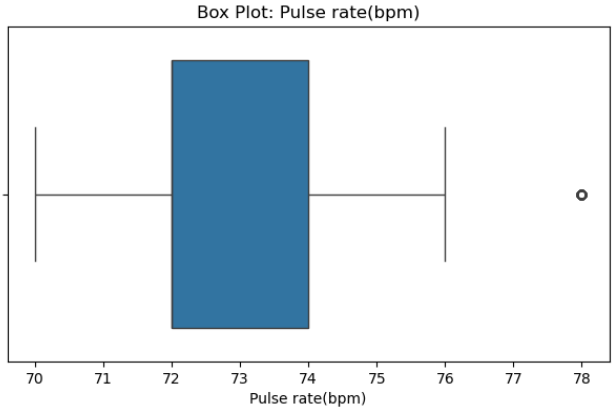
```
Median Imputation for RR (breaths/min)
```

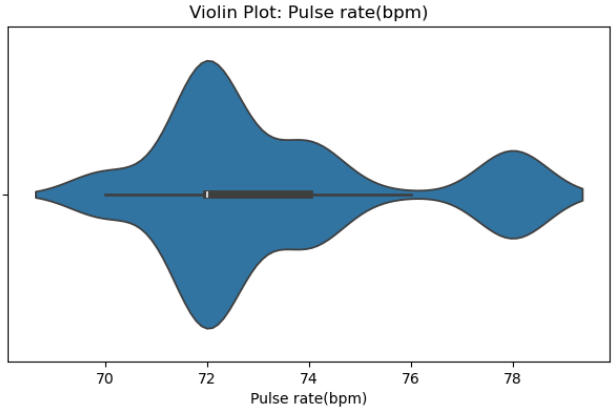
```
print("Outliers handled as per the proposed methodology.\n\
No Change for BP Systolic, BP Diastolic, and Follicle No. (L)\n\
Median Imputation for RR (breaths/min)\n\
Winsorization for Pulse rate(bpm), Weight (Kg)")
```

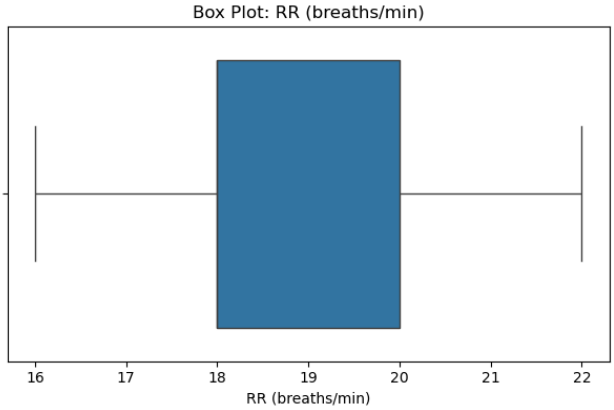
```
Outliers handled as per the proposed methodology.
No Change for BP Systolic, BP Diastolic, and Follicle No. (L)
Median Imputation for RR (breaths/min)
Winsorization for Pulse rate(bpm), Weight (Kg)
```

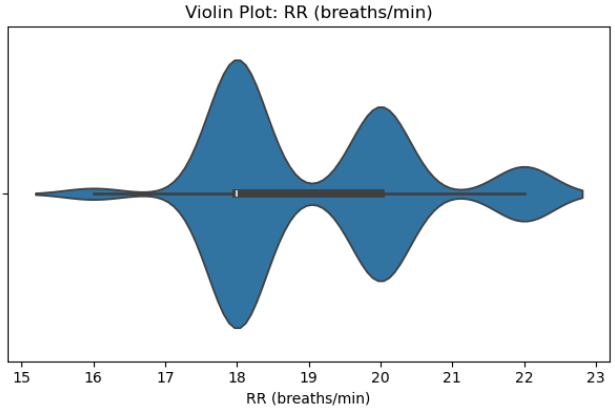
```
# Creating subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(12, num_rows * 4))
```

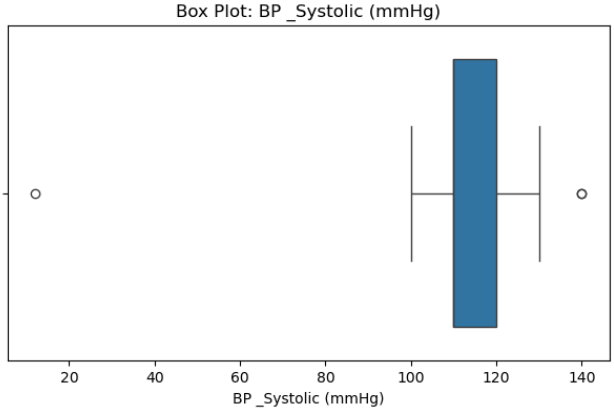
```
axes = axes.reshape(num_rows, 2)  # Ensure it's 2D for indexing

for i, feature in enumerate(outliers_all):
    # Box Plot
    sns.boxplot(data=data, x=feature, ax=axes[i, 0])
    axes[i, 0].set_title(f'Box Plot: {feature}')

    # Violin Plot
    sns.violinplot(data=data, x=feature, ax=axes[i, 1])
    axes[i, 1].set_title(f'Violin Plot: {feature}')

# Adjust layout
plt.tight_layout()
plt.show()
```

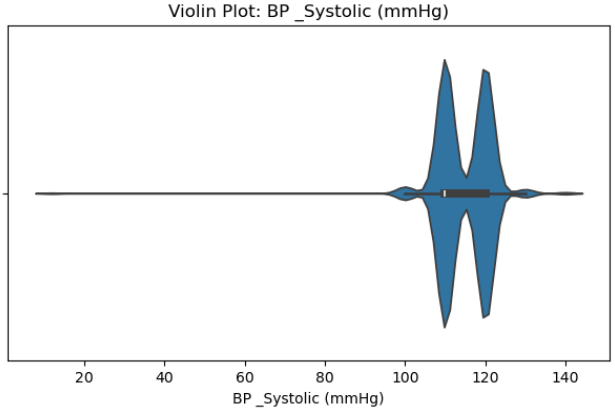

Box Plot: Pulse rate(bpm)
70
71
72
73
74
75
76
77
78
Pulse rate(bpm)

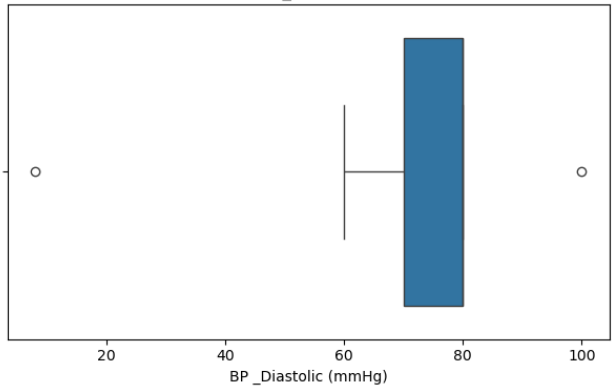

Violin Plot: Pulse rate(bpm)
70
72
74
76
78
Pulse rate(bpm)

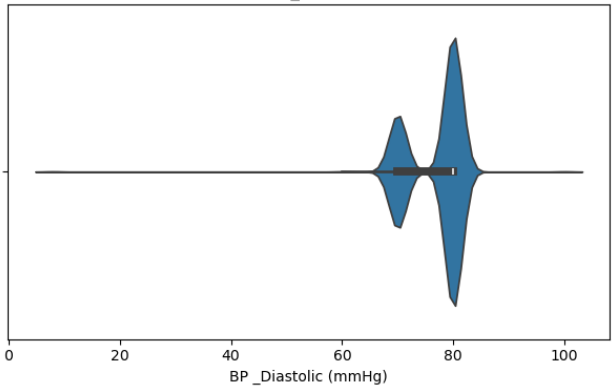

Box Plot: RR (breaths/min)
16
17
18
19
20
21
22
RR (breaths/min)

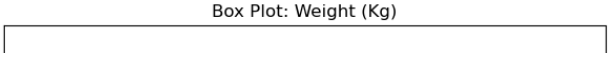

Violin Plot: RR (breaths/min)
15
16
17
18
19
20
21
22
23
RR (breaths/min)

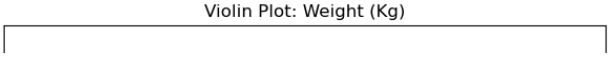

Box Plot: BP _Systolic (mmHg)
20
40
60
80
100
120
140
BP _Systolic (mmHg)


Violin Plot: BP _Systolic (mmHg)
20
40
60
80
100
120
140
BP _Systolic (mmHg)


Box Plot: BP _Diastolic (mmHg)
20
40
60
80
100
BP _Diastolic (mmHg)


Violin Plot: BP _Diastolic (mmHg)
0
20
40
60
80
100
BP _Diastolic (mmHg)


Box Plot: Weight (Kg)


Violin Plot: Weight (Kg)

Model Creation

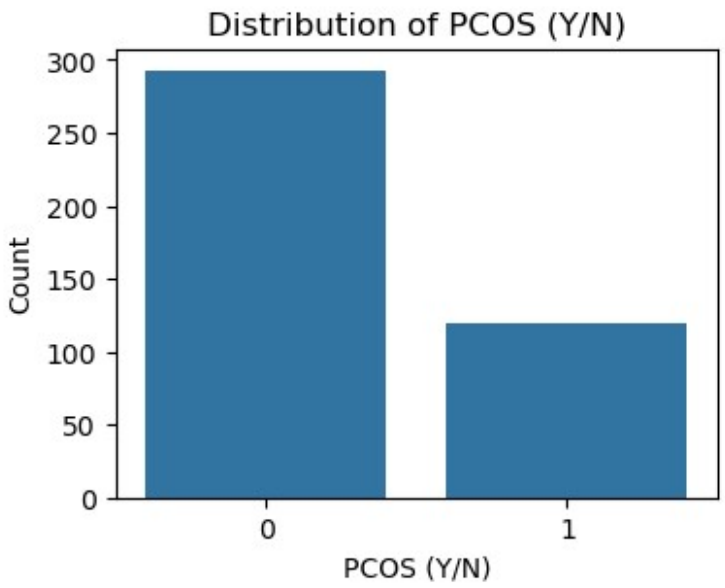
```
# Define features (X) and target (y)
X = data.drop(columns=['PCOS (Y/N)'])
y = data['PCOS (Y/N)']

# Split into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Imbalance Indentification

Check Balance in Disease Stages

```
# Check the distribution of the target variable (e.g., 'PCOS (Y/N)' or
disease stage column)
target_column = 'PCOS (Y/N)'

# Check the counts of each class
class_counts = data[target_column].value_counts()

# Display the class counts
print(f"\nClass Distribution in {target_column}:\n", class_counts)

# Plotting the class distribution
plt.figure(figsize=(4, 3))
sns.countplot(x=target_column, data=data)
plt.title(f'Distribution of {target_column}')
plt.xlabel(target_column)
plt.ylabel('Count')
plt.show()
```

```
Class Distribution in PCOS (Y/N):
 PCOS (Y/N)
0    293
1    120
Name: count, dtype: int64
```



Solution is the use of SMOTE (Synthetic Minority Oversampling Technique)

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)


logreg = LogisticRegression(max_iter=700)
logreg.fit(X_train_balanced, y_train_balanced)
y_pred = logreg.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.90      0.92        61
           1       0.75      0.82      0.78        22

    accuracy                           0.88        83
   macro avg       0.84      0.86      0.85        83
weighted avg       0.88      0.88      0.88        83


C:\Users\Soumiz\anaconda3\Lib\site-packages\sklearn\linear_model\
_logistic.py:469: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

Hyperparameter Tuning using GridSearchCV

```
# Standardizing the dataset for better performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Suppress warnings
warnings.filterwarnings("ignore")

# Define the parameter grid
param_grid = {
    'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],  # Regularization
strength
    'solver': ['liblinear', 'lbfgs', 'saga'],  # Valid solvers
    'penalty': ['l2'],  # L1 is removed because 'lbfgs' doesn't
support it
    'class_weight': [None, 'balanced']
}

# Initialize Logistic Regression
logreg = LogisticRegression(max_iter=50)

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='f1',
n_jobs=-1, error_score='raise')
grid_search.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)

# Train the best model
best_logreg = grid_search.best_estimator_
y_pred_best_logreg = best_logreg.predict(X_test)

# Evaluate the optimized model
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

print("===== Tuned Logistic Regression =====")
```

```
print("Accuracy:", accuracy_score(y_test, y_pred_best_logreg))
print("Classification Report:\n", classification_report(y_test,
y_pred_best_logreg))
```

```
Best Parameters: {'C': 1, 'class_weight': None, 'penalty': 'l2',
'solver': 'lbfgs'}
===== Tuned Logistic Regression =====
Accuracy: 0.891566265060241
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        61
           1       0.88      0.68      0.77        22

    accuracy                           0.89        83
   macro avg       0.89      0.82      0.85        83
weighted avg       0.89      0.89      0.89        83
```

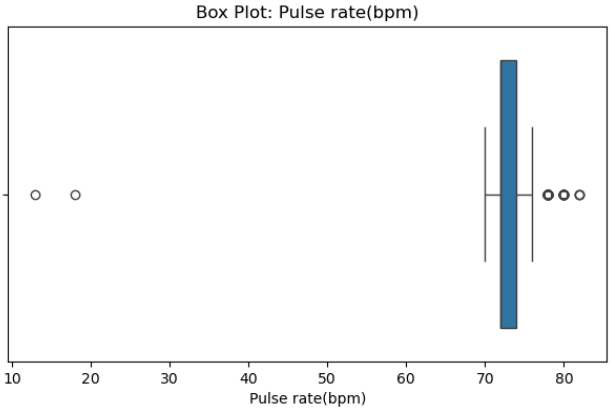
Method 3: Outlier Removal and Model training

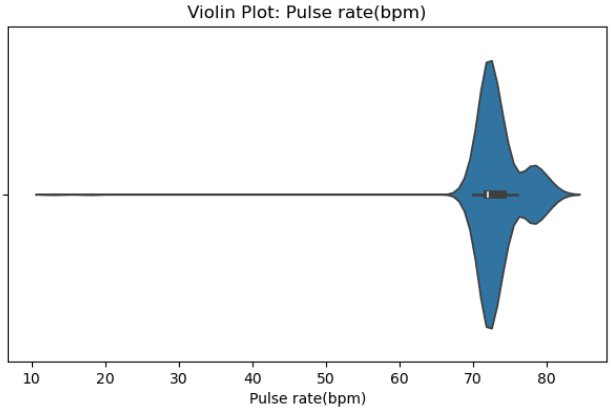
```
data = pd. read_csv('data_filtered.csv')
data.describe()
```

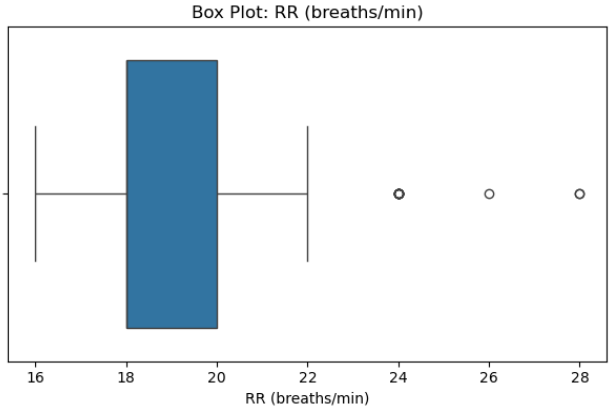
```
       PCOS (Y/N)  Cycle(R/I)  Pulse rate(bpm)  RR (breaths/min)  \
count  541.000000  541.000000       541.000000        541.000000
mean     0.327172    2.560074        73.247689         19.243993
std      0.469615    0.901950         4.430285          1.688629
min      0.000000    2.000000        13.000000         16.000000
25%      0.000000    2.000000        72.000000         18.000000
50%      0.000000    2.000000        72.000000         18.000000
75%      1.000000    4.000000        74.000000         20.000000
max      1.000000    5.000000        82.000000         28.000000

       BP _Systolic (mmHg)  BP _Diastolic (mmHg)  Waist:Hip Ratio  \
count           541.000000            541.000000       541.000000
mean            114.661738             76.927911         0.891627
std               7.384556              5.574112         0.046135
min              12.000000              8.000000         0.760000
25%             110.000000             70.000000         0.860000
50%             110.000000             80.000000         0.890000
75%             120.000000             80.000000         0.930000
max             140.000000            100.000000         0.980000

       Weight (Kg)  Follicle No. (R)  Follicle No. (L)  Skin darkening
(Y/N)  \
count   541.000000        541.000000        541.000000
541.000000
mean     59.637153          6.641405          6.129390
```

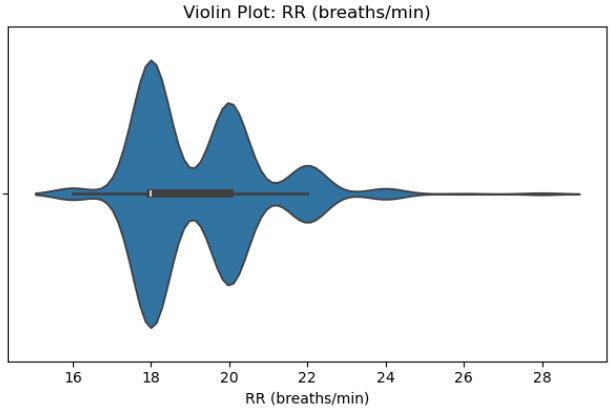
```
0.306839
std      11.028287           4.436889           4.229294
0.461609
min      31.000000           0.000000           0.000000
0.000000
25%      52.000000           3.000000           3.000000
0.000000
50%      59.000000           6.000000           5.000000
0.000000
75%      65.000000          10.000000           9.000000
1.000000
max     108.000000          20.000000          22.000000
1.000000

       hair growth(Y/N)  Weight gain(Y/N)
count         541.000000        541.000000
mean            0.273567          0.377079
std             0.446202          0.485104
min             0.000000          0.000000
25%             0.000000          0.000000
50%             0.000000          0.000000
75%             1.000000          1.000000
max             1.000000          1.000000


# Dictionary to store outliers
outliers_quantile = {}

# Detect outliers using IQR (Interquartile Range)
for col in data.columns:
        # Compute IQR
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1

        # Identify outliers
        outlier_indices_quantile = data[
            (data[col] < (Q1 - 1.5 * IQR)) | (data[col] > (Q3 + 1.5 *
IQR))
        ].index

        if len(outlier_indices_quantile) > 0:
            outliers_quantile[col] = outlier_indices_quantile

# Display the results
features_with_outliers = {col: len(indices) for col, indices in
outliers_quantile.items()}
print("\nFeatures with Outliers (as per quantile):",
features_with_outliers)
```

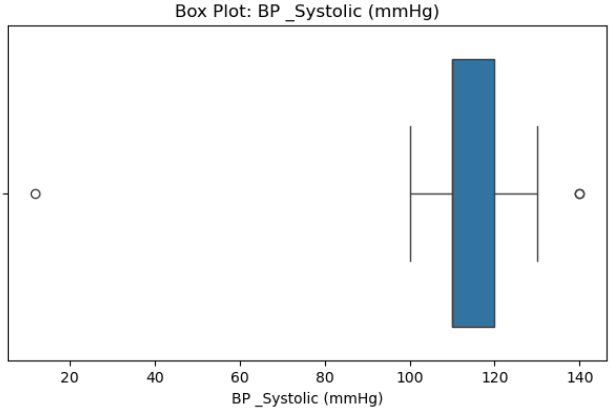
```
Features with Outliers (as per quantile): {'Pulse rate(bpm)': 94, 'RR (breaths/min)': 14, 'BP _Systolic (mmHg)': 3, 'BP _Diastolic (mmHg)': 2, 'Weight (Kg)': 18, 'Follicle No. (L)': 6}

# Create a list of Fetaures having outliers
outliers_all = outliers_quantile.keys()

# Number of rows for subplots (one row per feature)
num_rows = len(outliers_all)

# Creating subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(12, num_rows * 4))
axes = axes.reshape(num_rows, 2)  # Ensure it's 2D for indexing

for i, feature in enumerate(outliers_all):
    # Box Plot
    sns.boxplot(data=data, x=feature, ax=axes[i, 0])
    axes[i, 0].set_title(f'Box Plot: {feature}')

    # Violin Plot
    sns.violinplot(data=data, x=feature, ax=axes[i, 1])
    axes[i, 1].set_title(f'Violin Plot: {feature}')

# Adjust layout
plt.tight_layout()
plt.show()
```

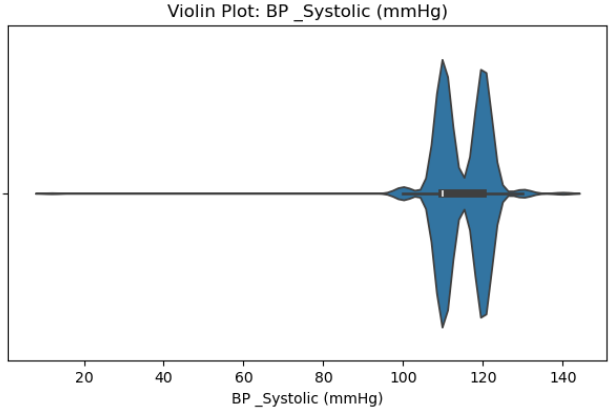

Box Plot: Pulse rate(bpm)
10
20
30
40
50
60
70
80
Pulse rate(bpm)

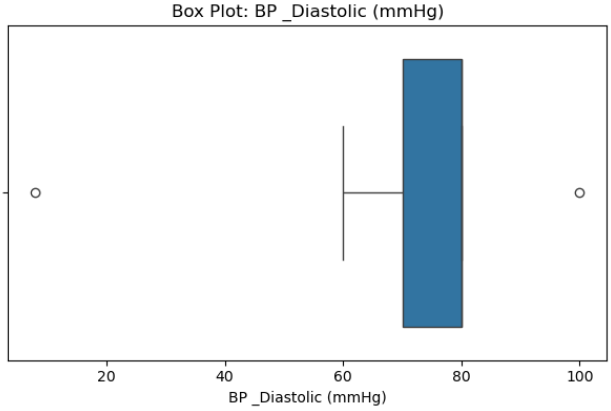

Violin Plot: Pulse rate(bpm)
10
20
30
40
50
60
70
80
Pulse rate(bpm)

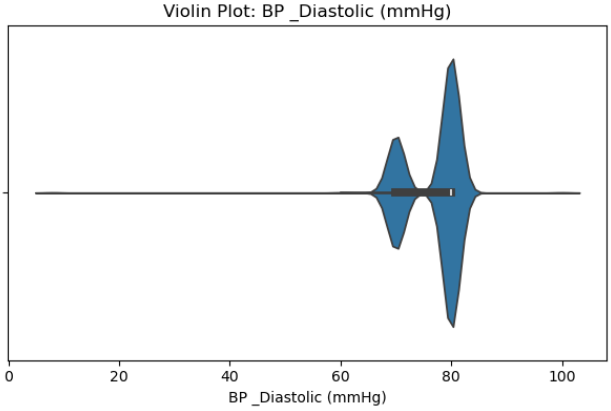

Box Plot: RR (breaths/min)
16
18
20
22
24
26
28
RR (breaths/min)

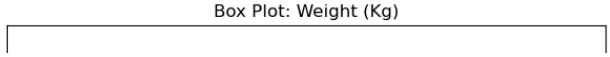

Violin Plot: RR (breaths/min)
16
18
20
22
24
26
28
RR (breaths/min)

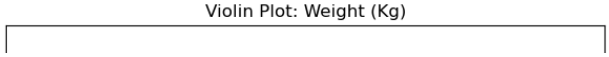

Box Plot: BP _Systolic (mmHg)
20
40
60
80
100
120
140
BP _Systolic (mmHg)


Violin Plot: BP _Systolic (mmHg)
20
40
60
80
100
120
140
BP _Systolic (mmHg)


Box Plot: BP _Diastolic (mmHg)
20
40
60
80
100
BP _Diastolic (mmHg)


Violin Plot: BP _Diastolic (mmHg)
0
20
40
60
80
100
BP _Diastolic (mmHg)


Box Plot: Weight (Kg)


Violin Plot: Weight (Kg)

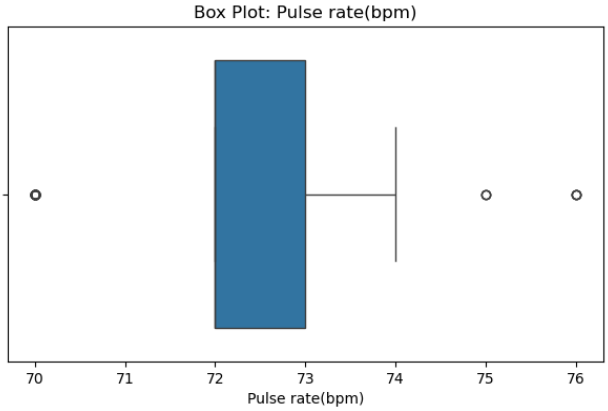
Remove outliers

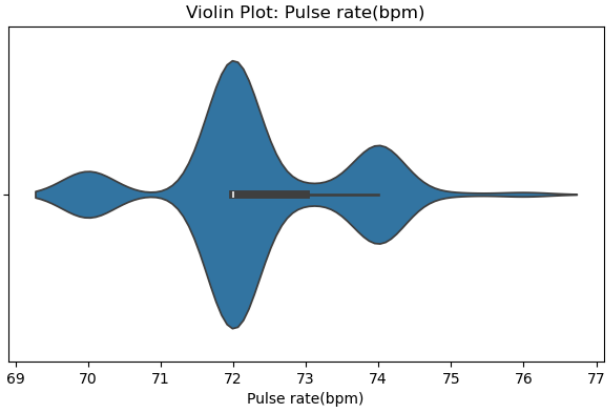
```
# List of features to remove outliers from
features_with_outliers = outliers_quantile.keys()

# Function to remove outliers using IQR
def remove_outliers(df, feature):
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Remove rows where feature value is outside bounds
    filtered_df = df[(df[feature] >= lower_bound) & (df[feature] <=
upper_bound)]

    print(f"Removed {len(df) - len(filtered_df)} outliers from
{feature}")
    return filtered_df

# Apply outlier removal to all selected features
for feature in features_with_outliers:
    data = remove_outliers(data, feature)

# Display remaining data shape after outlier removal
print(f"\nFinal dataset shape: {data.shape}")
```

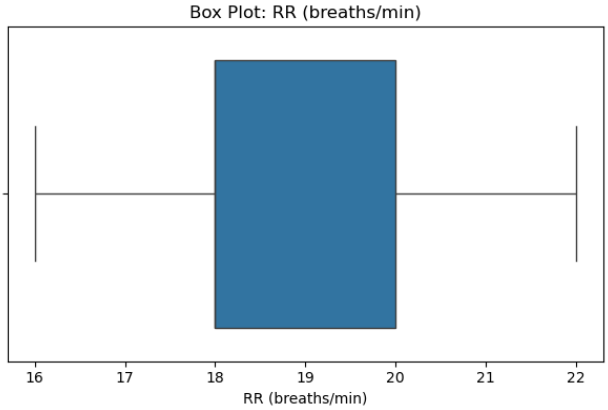
```
Removed 94 outliers from Pulse rate(bpm)
Removed 6 outliers from RR (breaths/min)
Removed 2 outliers from BP _Systolic (mmHg)
Removed 1 outliers from BP _Diastolic (mmHg)
Removed 14 outliers from Weight (Kg)
Removed 11 outliers from Follicle No. (L)

Final dataset shape: (413, 13)
```

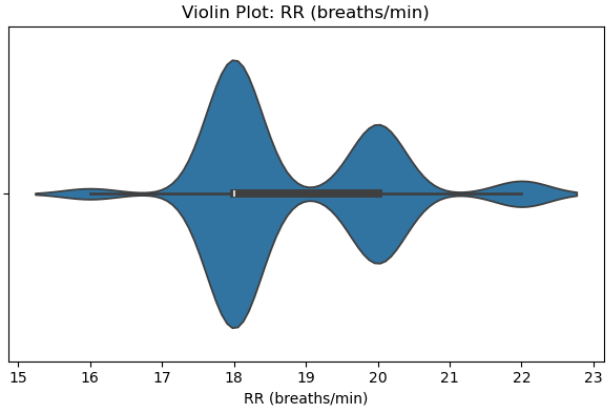
```
# Creating subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(12, num_rows * 4))
axes = axes.reshape(num_rows, 2)  # Ensure it's 2D for indexing

for i, feature in enumerate(outliers_all):
    # Box Plot
    sns.boxplot(data=data, x=feature, ax=axes[i, 0])
    axes[i, 0].set_title(f'Box Plot: {feature}')

    # Violin Plot
    sns.violinplot(data=data, x=feature, ax=axes[i, 1])
    axes[i, 1].set_title(f'Violin Plot: {feature}')
```

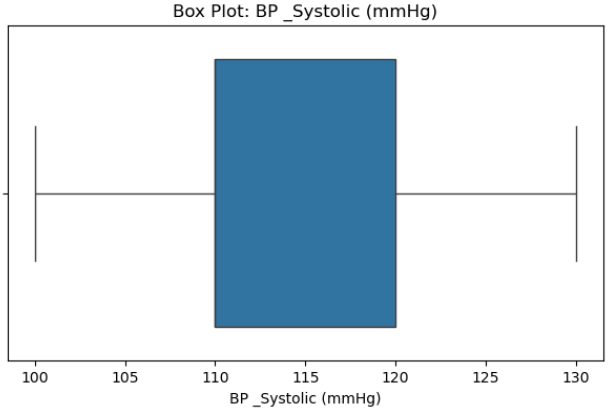
```
# Adjust layout
plt.tight_layout()
plt.show()
```

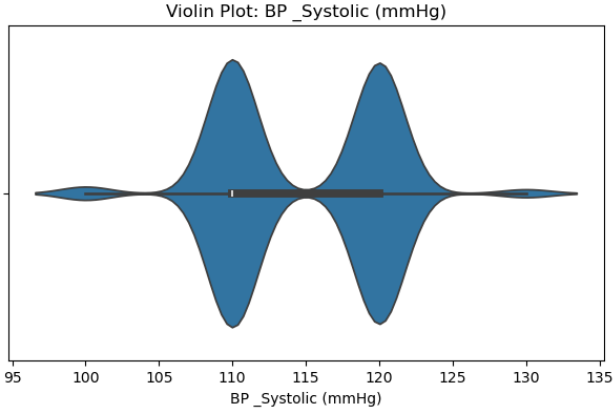

Box Plot: Pulse rate(bpm)
70
71
72
73
74
75
76
Pulse rate(bpm)

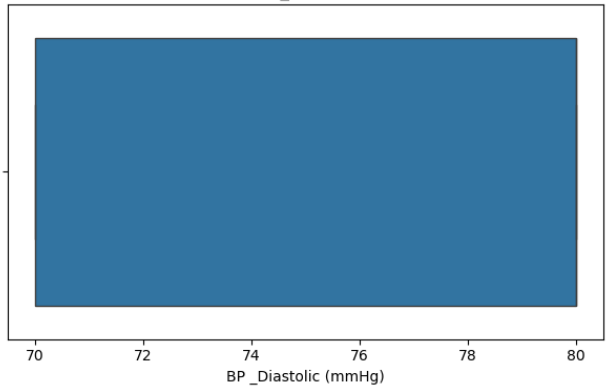

Violin Plot: Pulse rate(bpm)
69
70
71
72
73
74
75
76
77
Pulse rate(bpm)

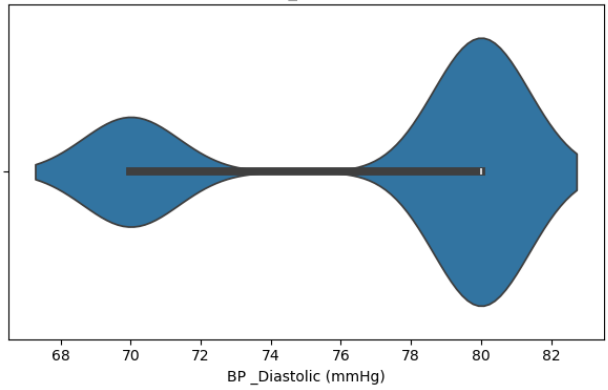

Box Plot: RR (breaths/min)
16
17
18
19
20
21
22
RR (breaths/min)

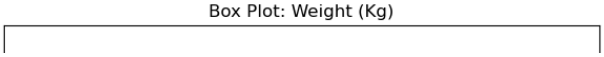

Violin Plot: RR (breaths/min)
15
16
17
18
19
20
21
22
23
RR (breaths/min)

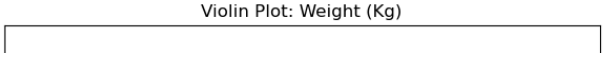

Box Plot: BP _Systolic (mmHg)
100
105
110
115
120
125
130
BP _Systolic (mmHg)


Violin Plot: BP _Systolic (mmHg)
95
100
105
110
115
120
125
130
135
BP _Systolic (mmHg)


Box Plot: BP _Diastolic (mmHg)
70
72
74
76
78
80
BP _Diastolic (mmHg)


Violin Plot: BP _Diastolic (mmHg)
68
70
72
74
76
78
80
82
BP _Diastolic (mmHg)


Box Plot: Weight (Kg)


Violin Plot: Weight (Kg)

```
data.to_csv("No Outliers data.csv", index=False)
```

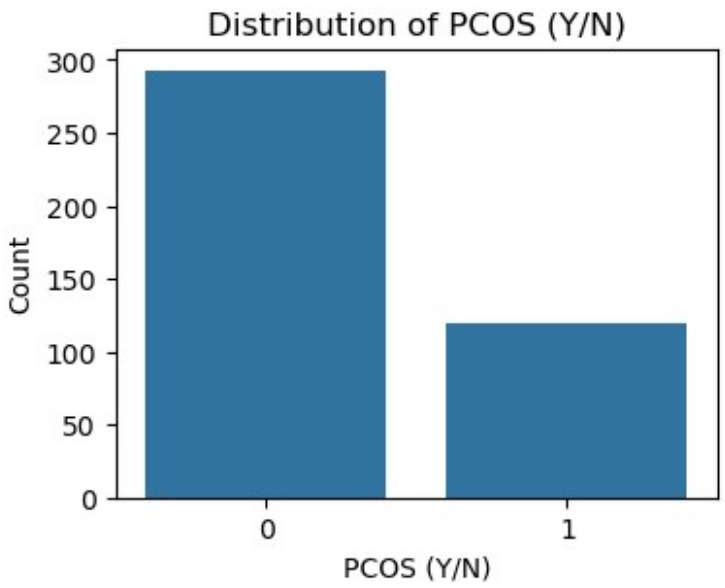
Model Creation

```
# Define features (X) and target (y)
X = data.drop(columns=['PCOS (Y/N)'])
y = data['PCOS (Y/N)']

# Split into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Imbalance Indentification

Check Balance in Disease Stages

```
# Check the distribution of the target variable (e.g., 'PCOS (Y/N)' or
disease stage column)
target_column = 'PCOS (Y/N)'

# Check the counts of each class
class_counts = data[target_column].value_counts()

# Display the class counts
print(f"\nClass Distribution in {target_column}:\n", class_counts)

# Plotting the class distribution
plt.figure(figsize=(4, 3))
sns.countplot(x=target_column, data=data)
plt.title(f'Distribution of {target_column}')
plt.xlabel(target_column)
plt.ylabel('Count')
plt.show()


Class Distribution in PCOS (Y/N):
 PCOS (Y/N)
0    293
1    120
Name: count, dtype: int64
```



Solution is the use of SMOTE (Synthetic Minority Oversampling Technique)

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)


logreg = LogisticRegression(max_iter=700)
logreg.fit(X_train_balanced, y_train_balanced)
y_pred = logreg.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.90      0.92        61
           1       0.75      0.82      0.78        22

    accuracy                           0.88        83
   macro avg       0.84      0.86      0.85        83
weighted avg       0.88      0.88      0.88        83


C:\Users\Soumiz\anaconda3\Lib\site-packages\sklearn\linear_model\
_logistic.py:469: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

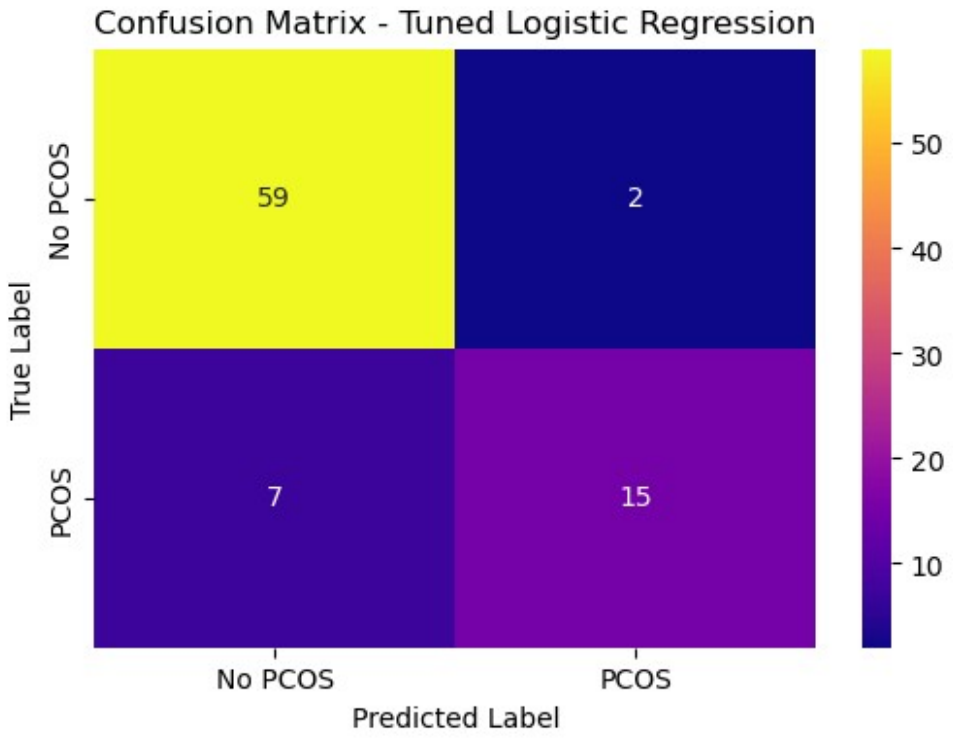
Hyperparameter Tuning using GridSearchCV

```
# Standardizing the dataset for better performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Suppress warnings
warnings.filterwarnings("ignore")

# Define the parameter grid
param_grid = {
    'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],  # Regularization
strength
    'solver': ['liblinear', 'lbfgs', 'saga'],  # Valid solvers
    'penalty': ['l2'],  # L1 is removed because 'lbfgs' doesn't
support it
    'class_weight': [None, 'balanced']
}

# Initialize Logistic Regression
logreg = LogisticRegression(max_iter=50)

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='f1',
n_jobs=-1, error_score='raise')
grid_search.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)

# Train the best model
best_logreg = grid_search.best_estimator_
y_pred_best_logreg = best_logreg.predict(X_test)

# Evaluate the optimized model
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

print("===== Tuned Logistic Regression =====")
```

```
print("Accuracy:", accuracy_score(y_test, y_pred_best_logreg))
print("Classification Report:\n", classification_report(y_test,
y_pred_best_logreg))
```

```
Best Parameters: {'C': 1, 'class_weight': None, 'penalty': 'l2',
'solver': 'lbfgs'}
===== Tuned Logistic Regression =====
Accuracy: 0.891566265060241
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        61
           1       0.88      0.68      0.77        22

    accuracy                           0.89        83
   macro avg       0.89      0.82      0.85        83
weighted avg       0.89      0.89      0.89        83
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_best_logreg)

# Plot the heatmap
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="plasma", xticklabels=["No
PCOS", "PCOS"], yticklabels=["No PCOS", "PCOS"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Tuned Logistic Regression")
plt.show()
```


Confusion Matrix - Tuned Logistic Regression
59
2
7
15
No PCOS
PCOS
True Label
No PCOS
PCOS
Predicted Label
50
40
30
20
10