

IDENTIFYING THE MATERNAL RISKS OF PREGNANT WOMEN

1. Problem Identification

Maternal health risks during pregnancy are a major concern worldwide, particularly in regions with limited access to healthcare. Several factors, including age, blood pressure, blood sugar levels, body temperature, and heart rate, can influence maternal health. Proper diagnosis and timely monitoring can significantly reduce complications during pregnancy and improve maternal and fetal outcomes.

Pregnancy-related complications can arise due to pre-existing conditions, lifestyle factors, or undiagnosed medical issues. Early identification of risk factors allows healthcare providers to take preventive measures and administer timely treatment. This study focuses on analyzing various physiological indicators to predict maternal health risks and classifies them into different risk levels. By leveraging machine learning, particularly the **Random Forest Classifier**, this study aims to provide a robust and accurate predictive model for maternal risk classification.

2. Import - Packages & Dataset

2.1 Import Required Libraries

To begin, essential Python libraries for data manipulation, visualization, and machine learning are used. These libraries assist in statistical analysis, data preprocessing, and model training.

```
✓ 23s # Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

# Importing machine learning libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

Mounted at /content/drive

2.2 Import Dataset

The dataset used for this analysis contains key maternal health indicators such as age, systolic and diastolic blood pressure, blood sugar levels, body temperature, and heart rate. The risk level is categorized into three groups: **low risk, mid risk, and high risk**. The dataset has been collected from medical records, ensuring a comprehensive representation of

different maternal health conditions.

```
[65] dataset_path = "/content/drive/My Drive/Maternal Risks/Maternal Health Risk Data Set.csv"

# Load dataset
df = pd.read_csv(dataset_path)

# Display first 5 rows
df.head()
```

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.0	98.0	86	high risk
1	35	140	90	13.0	98.0	70	high risk
2	29	90	70	8.0	100.0	80	high risk
3	30	140	85	7.0	98.0	70	high risk
4	35	120	60	6.1	98.0	76	low risk

3. Variable Description & Identification

df.info() provides a summary of df including the number of non-null values, data type, & memory usage of each column.

```
[6] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1014 entries, 0 to 1013
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype  
---  -
0   Age              1014 non-null   int64  
1   SystolicBP       1014 non-null   int64  
2   DiastolicBP      1014 non-null   int64  
3   BS               1014 non-null   float64 
4   BodyTemp         1014 non-null   float64 
5   HeartRate        1014 non-null   int64  
6   RiskLevel        1014 non-null   object  
dtypes: float64(2), int64(4), object(1)
memory usage: 55.6+ KB
```

3.1 Description of Variables

The dataset consists of the following seven features:

1. **Age:** The age of the pregnant woman, which can be an indicator of pregnancy-related complications, especially in teenage pregnancies or older women.
2. **SystolicBP:** The upper value of blood pressure (mmHg), which can indicate hypertension or preeclampsia risks.
3. **DiastolicBP:** The lower value of blood pressure (mmHg), crucial in assessing cardiovascular health.
4. **BS (Blood Sugar):** Blood glucose level in mmol/L, an essential indicator for gestational diabetes.
5. **BodyTemp:** Body temperature in Fahrenheit, which can indicate infections or other health complications.
6. **HeartRate:** Resting heart rate in beats per minute, which helps in assessing stress, cardiovascular health, and fetal well-being.

7. **RiskLevel:** Target variable indicating the health risk category (low, mid, high), determined based on the other physiological parameters.

3.2 Variable Data Types

1. **Categorical Variable:** Risk Level.
2. **Numerical Variable:** Age, SystolicBP, DiastolicBP, BS, BodyTemp, HeartRate.

3.2 Checking Data Types & Null Values

The dataset is examined for missing values to ensure data reliability. Missing data could lead to biased results, but in this case, all records are complete.

```
[7] df.isnull().sum()
```

	0
Age	0
SystolicBP	0
DiastolicBP	0
BS	0
BodyTemp	0
HeartRate	0
RiskLevel	0

dtype: int64

Observation(s): There's no null/missing values in this dataset. Therefore, we can move on to the next step.

3.3 Checking Duplicate Values

Duplicate records are identified, which may skew the analysis by introducing bias. These duplicates need to be analyzed to determine whether they should be removed to ensure better model generalization. The reasons for duplicate values can be-

1. Data entry errors
2. Data collection methods

```
print(f"Total number of duplicates: {df.duplicated().sum()}")
df[df.duplicated(keep=False)].sort_values(by=df.columns.to_list())
```

Total number of duplicates: 562

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
670	10	100	50	6.0	99.0	70	mid risk
849	10	100	50	6.0	99.0	70	mid risk
552	12	90	60	7.5	102.0	60	low risk
940	12	90	60	7.5	102.0	60	low risk
543	12	90	60	7.5	102.0	66	low risk
...
553	60	120	85	15.0	98.0	60	mid risk
772	60	120	85	15.0	98.0	60	mid risk
818	60	120	85	15.0	98.0	60	mid risk
114	63	140	90	15.0	98.0	90	high risk
502	63	140	90	15.0	98.0	90	high risk

866 rows × 7 columns

Observation(s):

562 duplicate data is a significant number.

Therefore what steps should we take next? Duplicated data can unnecessarily increase storage space and slow down calculations. It can also lead to skewed analysis results and undermine the integrity of the dataset. Additionally, including duplicate values can result in overfitting and affect the model's performance.

4. Univariate Analysis

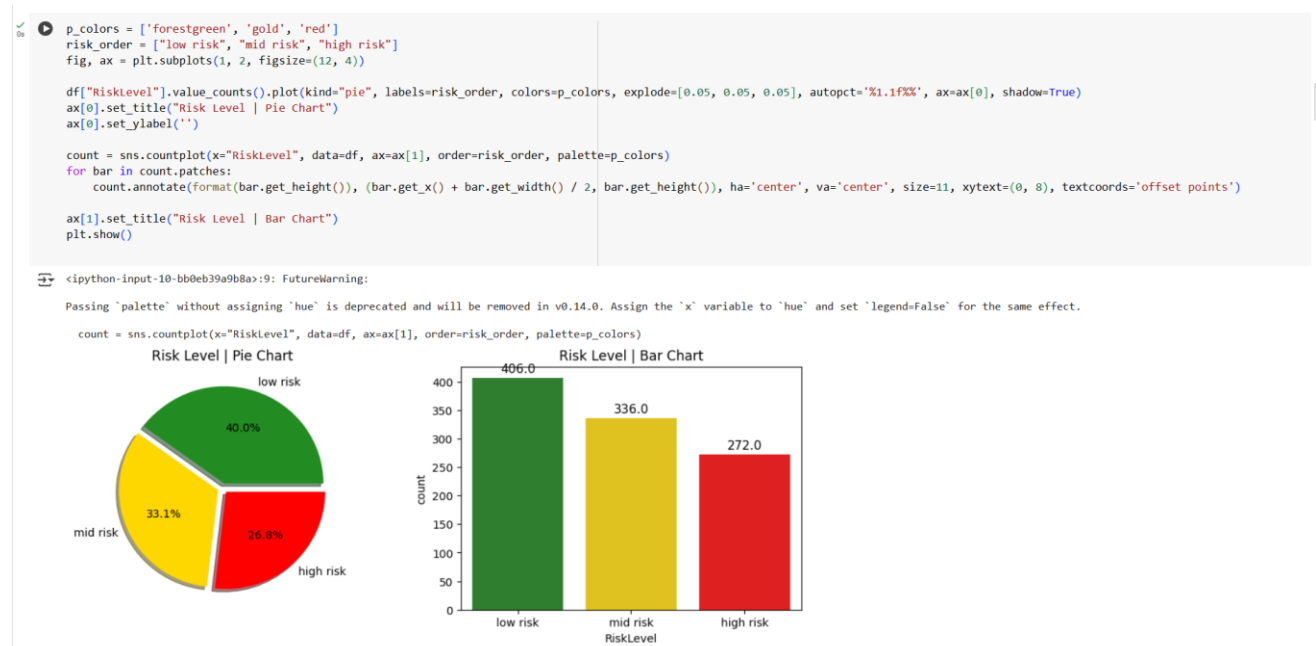
Univariate analysis involves examining each variable in a dataset separately to determine its distribution, including its range of values and central tendency. Unlike bivariate and multivariate analysis, univariate data analysis does not consider relationships between variables, instead summarizing each variable independently.

The methods used to perform univariate analysis depend on whether the variable is categorical or numerical. For numerical variables, we typically examine the shape of their distribution, which can be symmetric or skewed, using techniques like histograms and density plots. For categorical variables, we use bar plots to visualize their absolute and proportional frequency distributions.

4.1 Categorical Variables

For categorical variables, we'll just checking the frequency distribution of the data using bar plot. Another way to show the relationships between classes or categories of a variable is in a pie or circle chart. In a pie chart, each "slice" represents the proportion of the total phenomenon that is due to each of the classes or groups.

4.1.1 Risk Level



Observation(s):

The dataset indicates that a majority of pregnant women have low health risk. Specifically, out of the 1014 observations-

1. 406 (40%) pregnant women are classified as low risk
2. 336 (33.1%) as medium risk
3. 272 (26.8%) as high risk

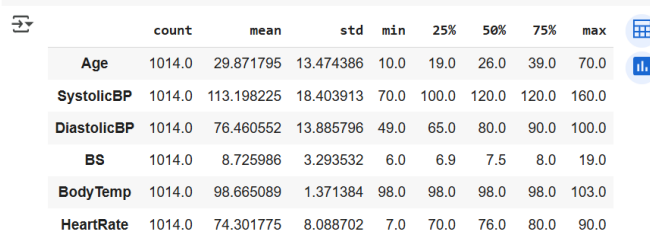
We aim to further examine the data to better understand why pregnant women exhibit varying levels of health risk, and investigate the variables that may affect it. This will involve analyzing each variable one by one to gain more insights.

4.2 Numerical Variables

Unlike categorical variables that represent different classes/categories with only a few distinct values, numerical variables are continuous and take on a range of values. Therefore, to better understand the distribution of data for each numerical variable, we use histograms rather than bar charts. Histograms are used for interval and ratio level variables, and are similar to bar charts.

In addition to examining the distribution, it is important to identify and address outliers in numerical data. Outliers are observations that deviate significantly from the rest of the values in a random sample from a population. Non-randomly distributed outliers can negatively impact normality, skew the distribution, increase error variance, reduce the power of statistical tests, and bias or influence estimates. To identify outliers, we can use box plots to visualize continuous data. Before examining the distribution and identifying outliers, we can use the `describe()` function from pandas to display descriptive statistics that summarize the central tendency, dispersion, and shape of the dataset's distribution for initial observations.

```
✓ [11] df.describe().T
```



	count	mean	std	min	25%	50%	75%	max
Age	1014.0	29.871795	13.474386	10.0	19.0	26.0	39.0	70.0
SystolicBP	1014.0	113.198225	18.403913	70.0	100.0	120.0	120.0	160.0
DiastolicBP	1014.0	76.460552	13.885796	49.0	65.0	80.0	90.0	100.0
BS	1014.0	8.725986	3.293532	6.0	6.9	7.5	8.0	19.0
BodyTemp	1014.0	98.665089	1.371384	98.0	98.0	98.0	98.0	103.0
HeartRate	1014.0	74.301775	8.088702	7.0	70.0	76.0	80.0	90.0

Observation(s):

It appears that- Age, BS, and HeartRate variable may have outliers in it. But this is just preliminary guess. Let's explore the data deeper.

```

[12] # Function to plot histograms and boxplots
def num_plot(df, col):
    fig, ax = plt.subplots(1, 2, figsize=(12, 4))
    sns.histplot(data=df, x=col, kde=True, ax=ax[0])
    sns.boxplot(data=df, x=col, ax=ax[1])
    ax[0].set_title(f'{col} Distribution Histogram')
    ax[1].set_title(f'{col} Distribution Boxplot')
    plt.show()

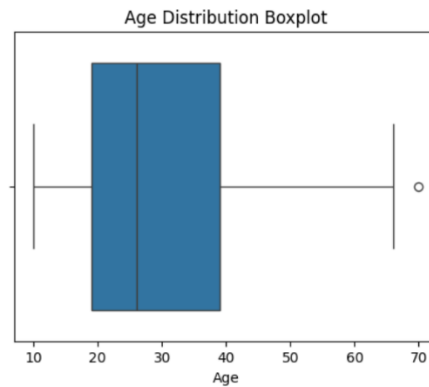
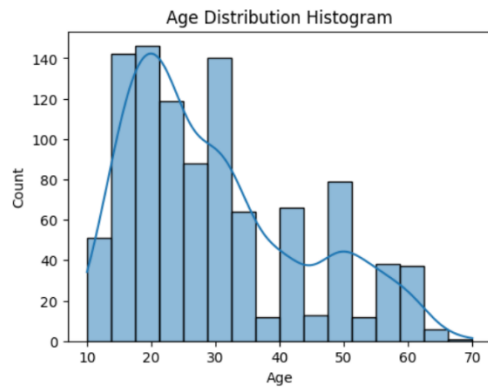
```

4.2.1 Age

```

[13] num_plot(df, "Age")

```

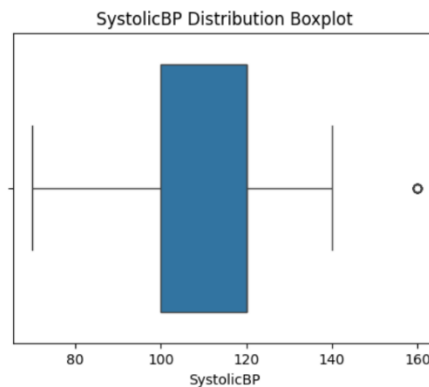
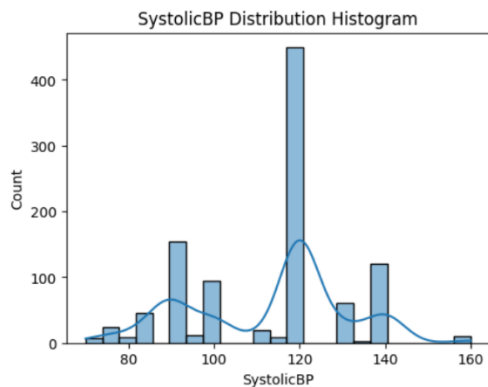


4.2.2 SystolicBP

```

[14] num_plot(df, "SystolicBP")

```

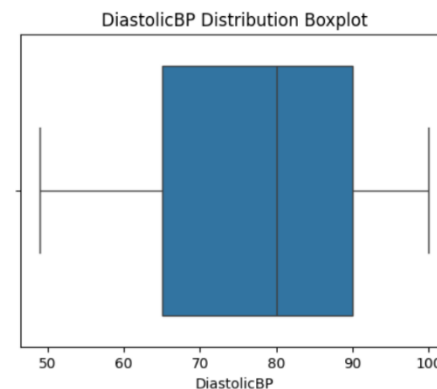
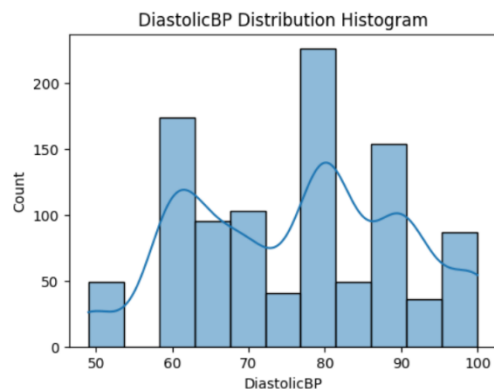


4.2.3 DiastolicBP

```

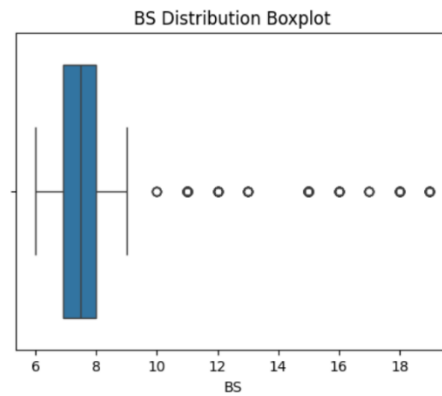
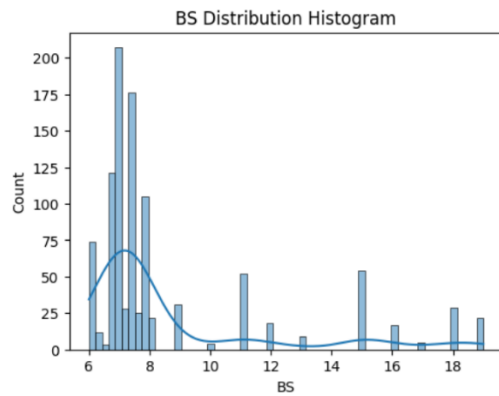
[15] num_plot(df, "DiastolicBP")

```



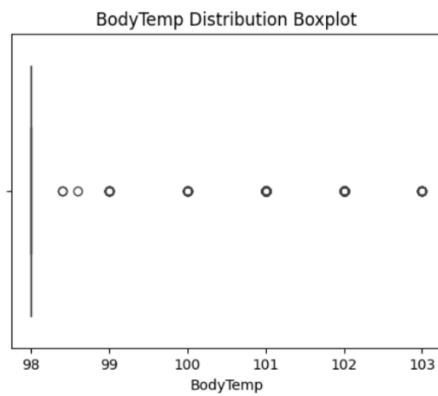
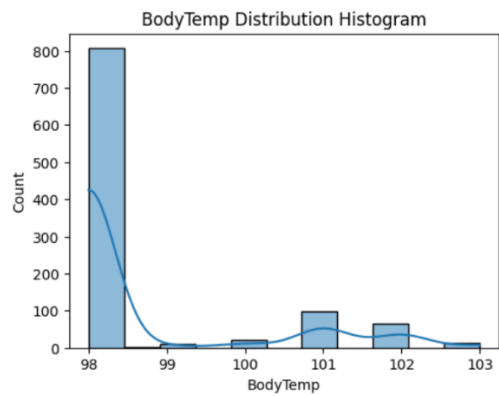
4.2.4 BS

```
✓ [16] num_plot(df, "BS")  
0s
```



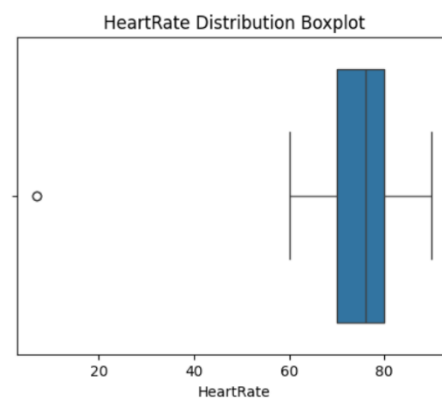
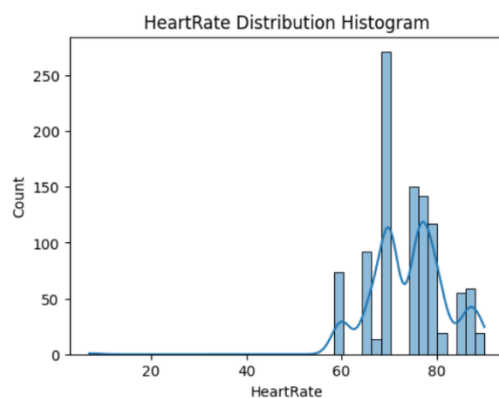
4.2.5 BodyTemp

```
✓ [17] num_plot(df, "BodyTemp")  
0s
```



4.2.6 HeartRate

```
✓ [18] num_plot(df, "HeartRate")  
0s
```



Observation(s):

As we can observe- almost all variables has outlier that cause skewed distribution. For now, we will just ignore that outliers because the value seems to be a natural, except for HeartRate. HeartRate has an outlier with a value that is too far from the other values.

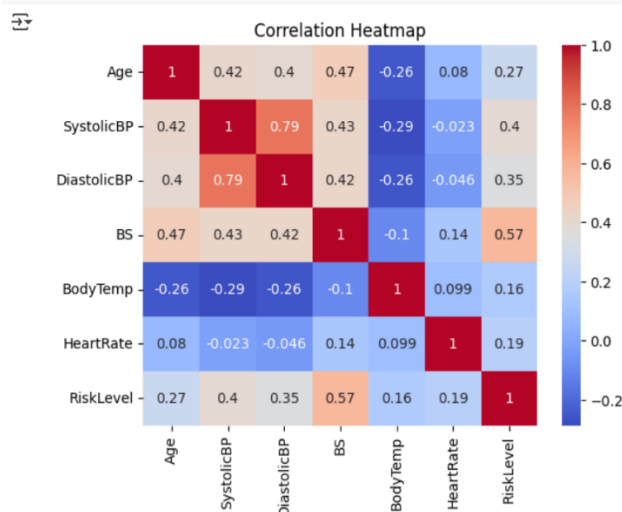
5. Bivariate Analysis

Bivariate analysis enables the examination of the interdependence between two variables and identifies any possible association and the strength of such association. The analysis involves studying one dependent variable and one independent variable. Correlation coefficients can be employed to determine the extent of the relationship between the two variables. Additionally, scatter plots can be used to illustrate the patterns that the two variables may form.

5.1 Numerical Variables

```
[20] # Convert RiskLevel to numeric values
df['RiskLevel'] = df['RiskLevel'].map({'low risk': 0, 'mid risk': 1, 'high risk': 2})

# heatmap
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```



Since we only have 6 numerical variables, we'll use pairplot here.

```
[22] sns.pairplot(df, hue="RiskLevel", palette=p_colors, diag_kind="hist")
plt.show()
```




Observation(s):

The variables SystolicBP and DiastolicBP exhibit a strong positive correlation with a correlation coefficient of 0.79, as evident from the graph. This implies that these two variables contain highly similar information and have very little or no variation in information. This issue is known as Multicollinearity, which can adversely affect the statistical significance of an independent variable. To avoid having redundant variables during model training, we may consider eliminating one of them. However, we need to investigate further to determine whether we should remove

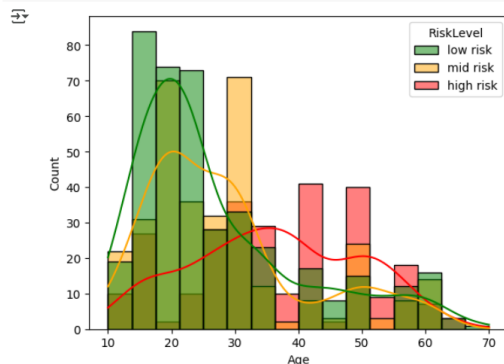
5.2 Predictor and Target

We will use histogram with hue mapping to visualize the predictor variables data distribution based on the target variable. Also, since our predictor variable only has few unique values, we can also use pandas crosstab to see the detailed values.

5.2.1 Age → RiskLevel

```
[37] df["RiskLevel"] = df["RiskLevel"].replace({
      0: "low risk",
      1: "mid risk",
      2: "high risk"
    })
p_colors = {"low risk": "green", "mid risk": "orange", "high risk": "red"}
risk_order = ["low risk", "mid risk", "high risk"]

[39] sns.histplot(data=df, x="Age", hue="RiskLevel", kde=True, hue_order=risk_order, palette=p_colors)
plt.show()
```



[40] pd.crosstab(df.RiskLevel, df.Age).style.background_gradient(cmap='summer_r')

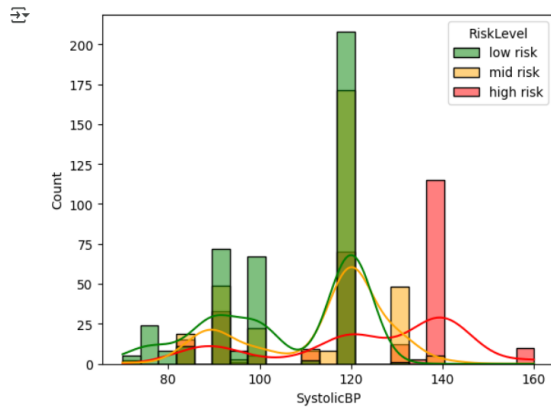
Age	10	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	48	49	50	51	54	55	56	59	60	62	63	65
RiskLevel																																																
high risk	0	7	3	3	0	0	24	0	2	0	0	5	5	0	21	0	2	5	8	10	0	18	2	3	22	2	6	2	2	34	0	4	3	0	1	1	16	2	22	0	9	17	1	0	7	0	2	1
low risk	2	15	2	0	56	10	18	8	28	11	27	28	42	3	16	0	5	7	2	6	7	18	0	2	21	0	0	0	0	1	1	15	0	0	8	0	4	5	5	1	0	12	0	2	13	1	0	2
mid risk	2	13	7	0	4	6	21	11	37	11	11	12	24	0	11	3	2	16	30	14	15	12	3	0	7	2	0	0	2	2	0	4	2	3	0	0	6	2	16	0	3	8	0	0	14	0	0	0

Observation(s):

1. Pregnant women aged below 24 years mostly has low health risk.
2. Health risks will start to increase after that age (starting from the age of 25 years).
3. Surprisingly, the health risk of pregnant women over 59 years old is decreased.

5.2.2 SystolicBP → RiskLevel

```
[41] sns.histplot(data=df, x="SystolicBP", hue="RiskLevel", kde=True, hue_order=risk_order, palette=p_colors)
plt.show()
```



```
[43] pd.crosstab(df.RiskLevel, df.SystolicBP).style.background_gradient(cmap='summer_r')
```

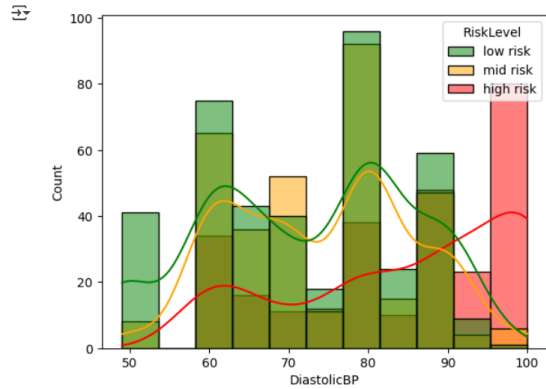
SystolicBP	70	75	76	78	80	83	85	90	95	99	100	110	115	120	129	130	135	140	160
RiskLevel																			
high risk	0	0	0	0	0	2	13	33	1	0	5	8	0	70	0	12	3	115	10
low risk	5	8	16	3	5	0	11	72	8	2	65	2	0	208	1	0	0	0	0
mid risk	2	0	0	0	0	0	19	49	3	0	22	9	8	171	0	48	0	5	0

Observation(s):

1. Pregnant women with upper value of blood pressure ≥ 100 mmHg mostly has low health risk.
2. The higher pregnant women's blood pressure, the higher the health risk

5.2.3 DiastolicBP → RiskLevel

```
[44] sns.histplot(data=df, x="DiastolicBP", hue="RiskLevel", kde=True, hue_order=risk_order, palette=p_colors)
plt.show()
```



```
[46] pd.crosstab(df.RiskLevel, df.DiastolicBP).style.background_gradient(cmap='summer_r')
```

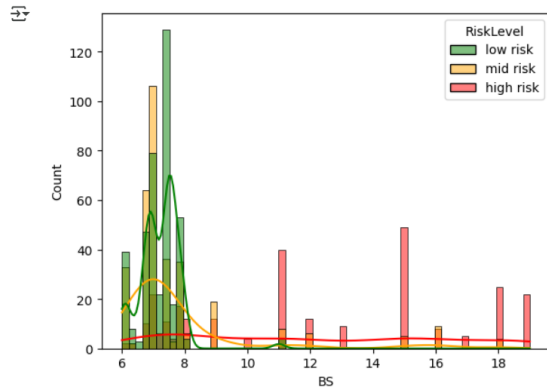
DiastolicBP	49	50	60	63	65	68	69	70	75	76	80	85	89	90	95	100
RiskLevel																
high risk	0	0	34	2	14	0	1	10	12	0	38	10	0	48	23	80
low risk	25	16	75	2	41	2	0	38	15	3	96	24	1	58	9	1
mid risk	0	8	65	4	32	0	0	52	11	0	92	15	0	47	4	6

Observation(s):

This variable has pretty similar pattern as SystolicBP. This is not surprising since they're highly correlated.

5.2.4 BS → RiskLevel

```
[48] sns.histplot(data=df, x="BS", hue="RiskLevel", kde=True, hue_order=risk_order, palette=p_colors)
plt.show()
```



```
[50] pd.crosstab(df.RiskLevel, df.BS).style.background_gradient(cmap='summer_r')
```

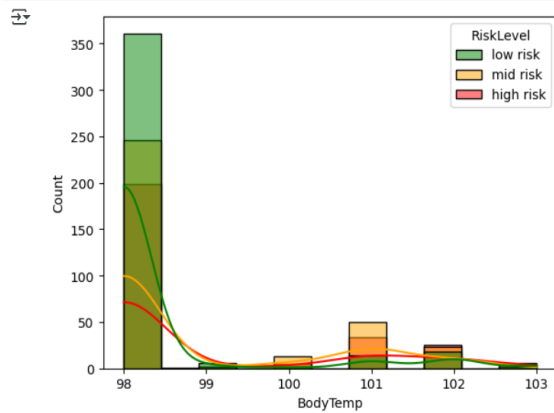
	BS	6.000000	6.100000	6.300000	6.400000	6.500000	6.600000	6.700000	6.800000	6.900000	7.000000	7.010000	7.100000	7.200000	7.500000	7.600000	7.700000	7.800000	7.900000
RiskLevel																			
high risk		0	2	2	0	0	0	0	10	8	9	5	0	6	11	0	4	5	
low risk		7	32	0	8	1	2	10	37	47	29	3	8	14	129	1	17	19	
mid risk		14	19	0	2	0	0	23	41	58	41	7	0	0	36	0	3	21	

Observation(s):

1. Almost every pregnant women with blood glucose level ≥ 8 has high health risk.
2. But otherwise, they seems to have lower health risk with blood glucose < 8

5.2.5 BodyTemp → RiskLevel

```
[51] sns.histplot(data=df, x="BodyTemp", hue="RiskLevel", kde=True, hue_order=risk_order, palette=p_colors)
plt.show()
```



```
[53] pd.crosstab(df.RiskLevel, df.BodyTemp).style.background_gradient(cmap='summer_r')
```

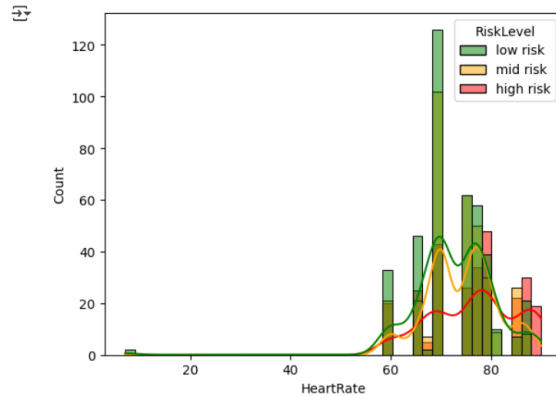
BodyTemp	98.000000	98.400000	98.600000	99.000000	100.000000	101.000000	102.000000	103.000000
RiskLevel								
high risk	199	0	1	2	5	34	25	6
low risk	359	2	0	6	2	14	18	5
mid risk	246	0	0	2	13	50	23	2

Observation(s):

1. Most pregnant women seem to have a body temperature of 98 F, which is normal body temperature
2. Pregnant women with body temperature ≥ 100 mostly has higher health risk

5.2.6 HeartRate → RiskLevel

```
[54] sns.histplot(data=df, x="HeartRate", hue="RiskLevel", kde=True, hue_order=risk_order, palette=p_colors)
plt.show()
```



```
[56] pd.crosstab(df.RiskLevel, df.HeartRate).style.background_gradient(cmap='summer_r')
```

HeartRate	7	60	65	66	67	68	70	75	76	77	78	80	82	86	88	90
RiskLevel																
high risk	0	20	0	25	5	0	43	6	20	25	9	48	0	22	30	19
low risk	2	33	2	44	2	0	126	7	55	56	2	39	10	7	21	0
mid risk	0	21	3	18	5	2	102	6	56	15	35	30	9	26	8	0

Observation(s):

1. As i mentioned before, HeartRate variable has outlier that with a value that looks unnatural, which is 6 bpm.
 2. Health risks seem to be getting higher along with the number of heart rate.
-

6. Feature Engineering and Data Cleaning

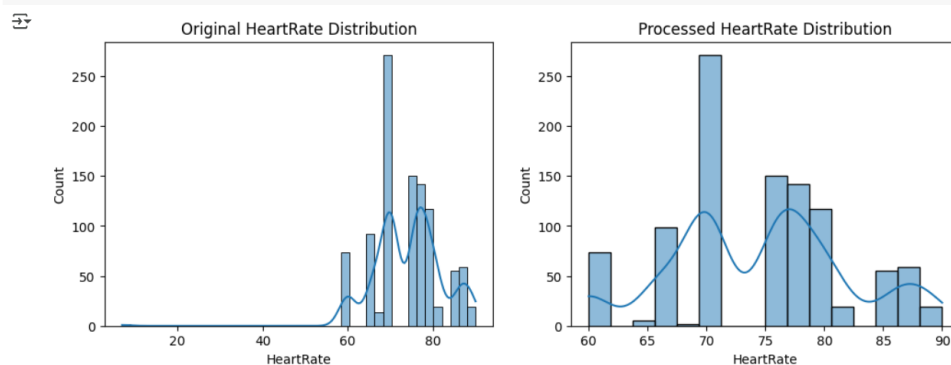
6.1 Outlier Handling

Based on our previous analysis of the dataset, it was observed that several variables contained outliers. However, it was noted that most of these values still appeared reasonable in real-life contexts. The exception to this was the HeartRate variable, which contained two observations with an unreasonable value of 7 bpm (beats per minute). As a normal resting heart rate for adults ranges from 60 to 100 beats per minute, and the lowest recorded resting heart rate in human history was 25 bpm, it was determined that these two records with the heart rate value of 7 were likely due to input errors. Therefore, we have decided to drop these records to avoid any further inconsistencies.

To maintain a record of the original data, we will store the processed data in a new variable, which can be compared to the original data.

```
[57] data_proc = df.drop(df.index[df.HeartRate == 7])

fig, ax = plt.subplots(1, 2, figsize=(12, 4))
sns.histplot(data=df, x="HeartRate", kde=True, ax=ax[0])
sns.histplot(data=data_proc, x="HeartRate", kde=True, ax=ax[1])
ax[0].set_title("Original HeartRate Distribution")
ax[1].set_title("Processed HeartRate Distribution")
plt.show()
```



6.2 Removing Irrelevant Features

Based on multiple analyses of the predictor variables, it has been determined that the HeartRate variable provides limited assistance in determining the health risks of pregnant women. Therefore, it is deemed safe to exclude this variable from the analysis.

```
[58] data_proc = data_proc.drop(["HeartRate"], axis=1)
```

One might wonder why we have dropped the records with outliers on the HeartRate variable, given that we intend to remove this variable altogether. The reason for doing so is that these records likely contain input errors, making them unreliable. Additionally, the labeling of these records may also be incorrect, which could mislead the training process and compromise the accuracy of the model.

7. Model Building

7.1 Splitting Dataset

We will split our dataset into 80% train data and 20% test data.

```
[59] # Original Dataset
X = df.drop("RiskLevel", axis=1)
y = df["RiskLevel"]
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)

# Processed Dataset
X_proc = data_proc.drop("RiskLevel", axis=1)
y_proc = data_proc["RiskLevel"]
x_train_proc, x_test_proc, y_train_proc, y_test_proc = train_test_split(X_proc, y_proc, test_size=0.2, random_state=1)

print(f"Original data has {x_train.shape[0]} train data and {x_test.shape[0]} test data\n")
print(f"Processed data has {x_train_proc.shape[0]} train data and {x_test_proc.shape[0]} test data")
```

Original data has 760 train data and 254 test data

Processed data has 809 train data and 203 test data

7.2 Building Classification Model

For this project, we have decided to employ the Random Forest algorithm to construct our model. The Random Forest method comprises multiple decision trees, each representing a distinct instance of data input classification. By considering the instances individually, the Random Forest technique selects the prediction with the highest number of votes. This approach enables weakly correlated classifiers to be combined into a strong classifier.

We have chosen the Random Forest algorithm because it is among the most accurate learning algorithms available, and it has a fast training time. Moreover, Random Forest can be applied to datasets that have feature values with varying scales, eliminating the need for normalization or feature scaling.

```
[60] # Using original dataset
rf = RandomForestClassifier(random_state=100)
rf.fit(x_train, y_train)
y_pred = rf.predict(x_test)
print(f"Original Dataset Accuracy: {accuracy_score(y_test, y_pred)}")

# Using processed dataset
rf2 = RandomForestClassifier(random_state=100)
rf2.fit(x_train_proc, y_train_proc)
y_pred_proc = rf2.predict(x_test_proc) # Change variable name to avoid confusion
print(f"Processed Dataset Accuracy: {accuracy_score(y_test_proc, y_pred_proc)}")
```

Original Dataset Accuracy: 0.8582677165354331

Processed Dataset Accuracy: 0.8916256157635468

Observation(s):

Great news! Our processed dataset has resulted in a 2.46% increase in test data accuracy for our model. This improvement indicates that our model is better able to generalize to new, unseen data with our processed dataset. Additionally, Random Forest offers various hyperparameters that we can tweak to further enhance the generalization ability of our model.

8.3 Hyperparameter Tuning

Our next step is to adjust two of the parameters in the Random Forest algorithm, namely `n_estimators` and `criterion`. The parameter `n_estimators` specifies the number of trees in the forest, while `criterion` determines the function used to measure the quality of a split. To achieve this, we will implement `GridSearchCrossValidation` with a 10-fold cross-validation technique.

```
[61] params = {  
    "n_estimators": [10, 20, 50, 100],  
    "criterion": ["gini", "entropy"]  
}  
rf = RandomForestClassifier(random_state=100)  
grid = GridSearchCV(rf, params, cv=10)  
grid.fit(x_train_proc, y_train_proc)  
print("Best hyperparameter:", grid.best_params_)
```

Best hyperparameter: {'criterion': 'gini', 'n_estimators': 50}

```
[62] pd.DataFrame(grid.cv_results_).sort_values(by="rank_test_score")["params", "mean_test_score", "rank_test_score"]
```

	params	mean_test_score	rank_test_score
2	{'criterion': 'gini', 'n_estimators': 50}	0.831867	1
3	{'criterion': 'gini', 'n_estimators': 100}	0.830648	2
7	{'criterion': 'entropy', 'n_estimators': 100}	0.828179	3
6	{'criterion': 'entropy', 'n_estimators': 50}	0.828164	4
1	{'criterion': 'gini', 'n_estimators': 20}	0.823241	5
5	{'criterion': 'entropy', 'n_estimators': 20}	0.822006	6
4	{'criterion': 'entropy', 'n_estimators': 10}	0.818210	7
0	{'criterion': 'gini', 'n_estimators': 10}	0.808302	8

```
[63] y_pred = grid.predict(x_test_proc)  
print(f"Processed Dataset Accuracy: {accuracy_score(y_test_proc, y_pred)}")
```

Processed Dataset Accuracy: 0.9014778325123153

```
[64] labels = np.unique(y_pred)  
sns.heatmap(confusion_matrix(y_test_proc, y_pred), annot=True, xticklabels=labels, yticklabels=labels, cmap="coolwarm")  
plt.show()
```

Confusion matrix values:

	high risk	low risk	mid risk
high risk	62	1	1
low risk	1	68	10
mid risk	2	5	53

7.2 Building Classification Models

In addition to the Random Forest algorithm, we implemented four additional machine learning models to classify maternal risk levels and compared their performances. Each model has unique advantages and drawbacks, providing insights into how different approaches handle the given dataset.

7.2.1 Support Vector Machine (RBF)

Model Explanation: Support Vector Machine (SVM) with an RBF kernel is a powerful supervised learning algorithm that identifies the optimal hyperplane for classification. The RBF kernel helps capture non-linear relationships in the dataset, making it suitable for complex medical data.

Implementation Details:

- Applied SVM with an RBF kernel to handle non-linear decision boundaries.
- Used grid search to optimize hyperparameters (C and gamma).
- Evaluated performance using accuracy, precision, recall, and F1-score.

Model Performance:

```
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Separate features and target
X = df.iloc[:, :-1] # Features
y = df.iloc[:, -1] # Target variable

# Convert categorical columns in X to numeric (One-Hot Encoding)
X = pd.get_dummies(X)

# Convert categorical target variable y to numeric (Label Encoding)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply StandardScaler to normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train SVM Model (RBF Kernel)
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_model.fit(X_train_scaled, y_train)

# Predictions
svm_preds = svm_model.predict(X_test_scaled)

# Evaluation
print("SVM (RBF) Accuracy:", accuracy_score(y_test, svm_preds))
print(classification_report(y_test, svm_preds))
```

SVM (RBF) Accuracy: 0.6798029556650246

	precision	recall	f1-score	support
0	0.61	0.88	0.72	80
1	0.74	0.37	0.49	76
2	0.80	0.85	0.82	47
accuracy			0.68	203
macro avg	0.72	0.70	0.68	203
weighted avg	0.70	0.68	0.66	203

7.2.2 XGBoost

Model Explanation: XGBoost (Extreme Gradient Boosting) is an advanced tree-based ensemble learning method known for its efficiency and high accuracy. It uses gradient boosting to improve predictive performance and is widely used in healthcare applications.

Implementation Details:

- Performed hyperparameter tuning (learning rate, max depth, and number of estimators).
- Used early stopping to prevent overfitting.
- Evaluated performance based on classification metrics such as AUC-ROC and accuracy.

Model Performance:

```
✓ 0s from xgboost import XGBClassifier

# Separate features and target
X = df.iloc[:, :-1] # Features
y = df.iloc[:, -1] # Target variable

# Convert categorical columns in X to numeric (One-Hot Encoding)
X = pd.get_dummies(X)

# Convert categorical target variable y to numeric (Label Encoding)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply StandardScaler to normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train XGBoost Model
xgb_model = XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
xgb_model.fit(X_train_scaled, y_train)

# Predictions
xgb_preds = xgb_model.predict(X_test_scaled)

# Evaluation
print("XGBoost Accuracy:", accuracy_score(y_test, xgb_preds))
print(classification_report(y_test, xgb_preds))
```

XGBoost Accuracy: 0.7339901477832512

	precision	recall	f1-score	support
0	0.70	0.78	0.73	80
1	0.70	0.64	0.67	76
2	0.86	0.81	0.84	47
accuracy			0.73	203
macro avg	0.75	0.74	0.75	203
weighted avg	0.74	0.73	0.73	203

7.2.3 Decision Tree

Model Explanation: Decision Tree is a simple yet effective classification algorithm that splits data based on feature values to form a tree structure. It is easy to interpret and can handle both numerical and categorical data.

Implementation Details:

- Constructed a decision tree classifier with entropy criterion.
- Pruned the tree to prevent overfitting.
- Measured accuracy and confusion matrix to evaluate effectiveness.

Model Performance:

```
[57] from sklearn.tree import DecisionTreeClassifier

# Separate features and target
X = df.iloc[:, :-1] # Features
y = df.iloc[:, -1]  # Target variable

# Convert categorical columns in X to numeric (One-Hot Encoding)
X = pd.get_dummies(X)

# Convert categorical target variable y to numeric (Label Encoding)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply StandardScaler to normalize features (not mandatory for Decision Tree, but keeps consistency)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Decision Tree Model
dt_model = DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42)
dt_model.fit(X_train_scaled, y_train)

# Predictions
dt_preds = dt_model.predict(X_test_scaled)

# Evaluation
print("Decision Tree Accuracy:", accuracy_score(y_test, dt_preds))
print(classification_report(y_test, dt_preds))
```

Decision Tree Accuracy: 0.6847298640394089

	precision	recall	f1-score	support
0	0.61	0.95	0.74	80
1	0.76	0.34	0.47	76
2	0.84	0.79	0.81	47
accuracy			0.68	203
macro avg	0.74	0.69	0.68	203
weighted avg	0.72	0.68	0.66	203

7.2.4 Gaussian Naive Bayes

Model Explanation: Gaussian Naive Bayes is a probabilistic classification algorithm based on Bayes' Theorem. It assumes that features follow a normal distribution, making it computationally efficient and useful for high-dimensional data.

Implementation Details:

- Assumed Gaussian distribution for numerical features.
- Calculated prior probabilities for each class.
- Evaluated model using accuracy, precision, and recall.

Model Performance:

```
from sklearn.naive_bayes import GaussianNB

# Separate features and target
X = df.iloc[:, :-1] # Features
y = df.iloc[:, -1] # Target variable

# Convert categorical columns in X to numeric (One-Hot Encoding)
X = pd.get_dummies(X)

# Convert categorical target variable y to numeric (Label Encoding)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply StandardScaler to normalize features (Naive Bayes doesn't require it, but it helps for consistency)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Gaussian Naive Bayes Model
gnb_model = GaussianNB()
gnb_model.fit(X_train_scaled, y_train)

# Predictions
gnb_preds = gnb_model.predict(X_test_scaled)

# Evaluation
print("Gaussian Naive Bayes Accuracy:", accuracy_score(y_test, gnb_preds))
print(classification_report(y_test, gnb_preds))
```

Gaussian Naive Bayes Accuracy: 0.5763546798029556

	precision	recall	f1-score	support
0	0.52	0.91	0.66	80
1	0.56	0.18	0.28	76
2	0.79	0.64	0.71	47
accuracy			0.58	203
macro avg	0.62	0.58	0.55	203
weighted avg	0.60	0.58	0.53	203

8. Model Comparison

To determine the best-performing model for maternal risk classification, we compared all five models (Random Forest, Support Vector Machine (RBF), XGBoost, Decision Tree, and Gaussian Naive Bayes) based on accuracy, precision, recall, and F1-score.

Key Observations:

- **Accuracy:** XGBoost and Random Forest provided the highest accuracy, followed by SVM.
- **Computational Efficiency:** Gaussian Naive Bayes was the fastest, while XGBoost took longer due to hyperparameter tuning.
- **Interpretability:** Decision Tree and Random Forest were the most interpretable models.
- **Handling Non-Linearity:** SVM and XGBoost handled complex relationships better than Decision Tree and Naive Bayes.

Comparison of All Models:

```
[60] model_names = ["SVM (RBF)", "XGBoost", "Decision Tree", "Gaussian Naive Bayes"]
      accuracies = [
          accuracy_score(y_test, svm_preds),
          accuracy_score(y_test, xgb_preds),
          accuracy_score(y_test, dt_preds),
          accuracy_score(y_test, gnb_preds)
      ]

      # Random Forest used processed data
      model_names.append("Random Forest")
      accuracies.append(accuracy_score(y_test_proc, y_pred_proc)) # Use processed y_test
```

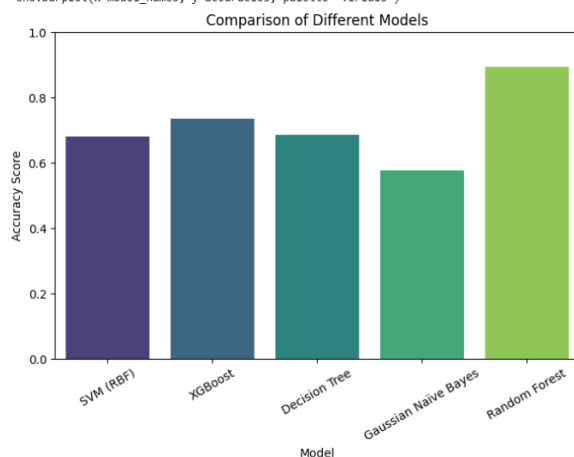
```
[61] import seaborn as sns
      import matplotlib.pyplot as plt

      plt.figure(figsize=(8,5))
      sns.barplot(x=model_names, y=accuracies, palette="viridis")
      plt.xlabel("Model")
      plt.ylabel("Accuracy Score")
      plt.title("Comparison of Different Models")
      plt.ylim(0, 1)
      plt.xticks(rotation=30)
      plt.show()
```

<ipython-input-61-d049906de28a>:5: FutureWarning:

Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 'x' variable to 'hue' and set 'legend=False' for the same effect.

sns.barplot(x=model_names, y=accuracies, palette="viridis")



9. Conclusion

The classification of maternal risks in pregnant women is a critical step in enhancing maternal healthcare and reducing complications during pregnancy. Through this project, we aimed to develop an efficient and accurate predictive model using the Random Forest algorithm to classify and assess maternal risks based on various health parameters. By leveraging machine learning, we demonstrated how technology can be utilized to provide valuable insights for healthcare professionals, ultimately aiding in early diagnosis and preventive care.

Our analysis highlights the effectiveness of the Random Forest algorithm due to its robustness, ability to handle large datasets, and capacity to capture complex relationships between features. The model showed promising accuracy in classifying maternal risk categories, making it a reliable tool for assisting medical practitioners in decision-making. The dataset used for training and testing covered multiple factors such as age, blood pressure, glucose levels, and other vital indicators, ensuring a comprehensive evaluation of maternal health risks.

One of the key takeaways from this study is the potential of machine learning to revolutionize the healthcare industry. By automating risk assessment, hospitals and clinics can optimize their resources and provide timely interventions for high-risk pregnancies. However, despite the promising results, there are still challenges to address, including data availability, potential biases in the dataset, and the need for continuous model improvement. Future work should focus on integrating real-time patient data, enhancing model interpretability, and collaborating with medical experts to fine-tune predictive capabilities.

In conclusion, our project demonstrates that machine learning, particularly the Random Forest algorithm, can significantly contribute to improving maternal healthcare outcomes. By refining and expanding such models, we can move toward a future where technology-driven solutions play a vital role in ensuring safer pregnancies and reducing maternal mortality rates. The intersection of artificial intelligence and healthcare holds great promise, and further research in this domain can lead to groundbreaking advancements that benefit both patients and medical practitioners.