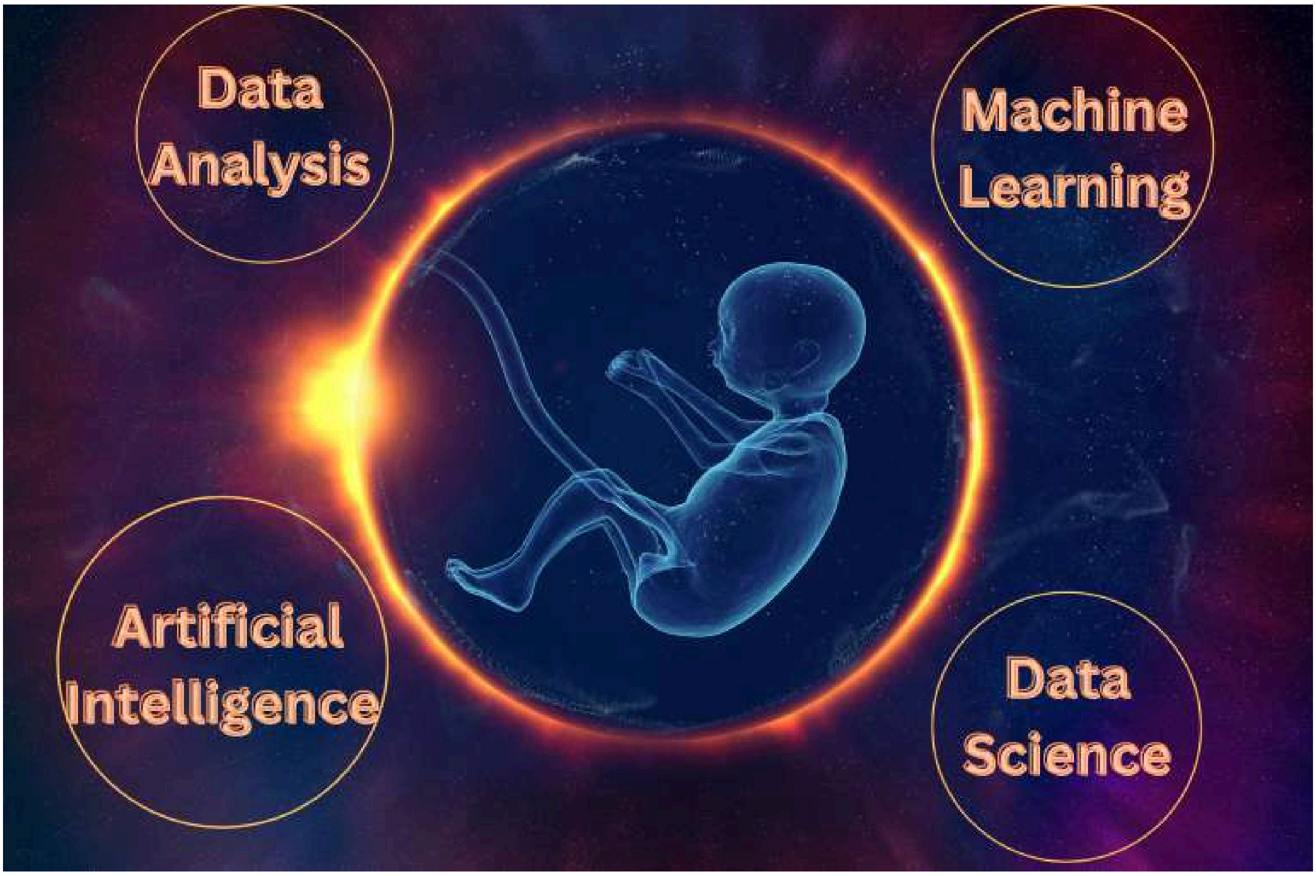
Fetal Health Classification



Fetal Health Classification

The **reduction of child and maternal mortality** is a critical focus for global health and human development. Several of the United Nations' **Sustainable Development Goals** reflect this priority, aiming to **end preventable deaths of newborns and children under 5 by 2030**, while reducing maternal mortality, which claimed over **295,000 lives in 2017**, the vast majority in low-resource settings.

Cardiotocograms (CTGs) provide a simple and cost-effective method to assess fetal health, allowing healthcare professionals to monitor **fetal heart rate, movements, and uterine contractions**. By leveraging CTG data, it is possible to detect potential issues early, preventing complications that could lead to **child or maternal mortality**.

This dataset contains **2126 records** derived from CTG exams, each classified by three expert obstetricians into three categories: **Normal**, **Suspect**, or **Pathological**. The goal of this project is to build a **machine learning model** to accurately predict the **health status of a fetus** based on the CTG data, thereby aiding in early diagnosis and timely medical intervention.

Join Faiz Siddiqui's Network

LinkedIn GitHub Kaggle

Faiz Siddiqui

A student of data science who loves learning and solving problems with machine learning. Dedicated to improving and helping others succeed in data science. Always excited to take on new challenges and share knowledge with the community.

Aim & Objectives

Aim

The aim of this notebook is to develop and evaluate machine learning models for classifying fetal health based on cardiotocogram (CTG) data. The goal is to accurately predict the health status of a fetus as either **Normal**, **Suspect**, or **Pathological**, facilitating early detection of potential issues that could lead to maternal or child mortality.

Objectives

- To perform an exploratory data analysis (EDA) on the fetal health dataset to understand key patterns and relationships within the data, specifically focusing on the **Normal**, **Suspect**, and **Pathological** health categories.
- To preprocess the CTG data, ensuring it is clean, standardized, and suitable for model training.
- To train and evaluate multiple machine learning models, including **XGBoost**, **Random Forest**, **SVM**, **KNN**, **Logistic Regression**, and ensemble techniques, for fetal health classification.
- To compare model performance based on **Accuracy** and **Matthews Correlation Coefficient (MCC)** and identify the best model for fetal health prediction.
- To provide insights and recommendations for healthcare professionals on the potential application of the selected model in clinical settings for early detection of fetal health issues.

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,VotingClassifier,StackingClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,matthews_corrco
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from itertools import cycle
import plotly.graph_objs as go
import plotly.subplots as sp
plt.style.use('seaborn-v0_8-whitegrid')
```

In [3]:
```
df=pd.read_csv('/kaggle/input/fetal-health-classification/fetal_health.csv')
df
```

Out[3]:

	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	severe_decelerations	prolongu
0	120.0	0.000	0.000	0.000	0.000	0.0	
1	132.0	0.006	0.000	0.006	0.003	0.0	
2	133.0	0.003	0.000	0.008	0.003	0.0	
3	134.0	0.003	0.000	0.008	0.003	0.0	
4	132.0	0.007	0.000	0.008	0.000	0.0	
...	...	...	...	...	...	...	
2121	140.0	0.000	0.000	0.007	0.000	0.0	
2122	140.0	0.001	0.000	0.007	0.000	0.0	
2123	140.0	0.001	0.000	0.007	0.000	0.0	
2124	140.0	0.001	0.000	0.006	0.000	0.0	
2125	142.0	0.002	0.002	0.008	0.000	0.0	

2126 rows × 22 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2126 entries, 0 to 2125
Data columns (total 22 columns):
 #   Column                                                  Non-Null Count  Dtype  
---  ------                                                  --------------  -----  
 0   baseline value                                          2126 non-null   float64
 1   accelerations                                           2126 non-null   float64
 2   fetal_movement                                          2126 non-null   float64
 3   uterine_contractions                                    2126 non-null   float64
 4   light_decelerations                                     2126 non-null   float64
 5   severe_decelerations                                    2126 non-null   float64
 6   prolongued_decelerations                                2126 non-null   float64
 7   abnormal_short_term_variability                         2126 non-null   float64
 8   mean_value_of_short_term_variability                    2126 non-null   float64
 9   percentage_of_time_with_abnormal_long_term_variability  2126 non-null   float64
 10  mean_value_of_long_term_variability                     2126 non-null   float64
 11  histogram_width                                         2126 non-null   float64
 12  histogram_min                                           2126 non-null   float64
 13  histogram_max                                           2126 non-null   float64
 14  histogram_number_of_peaks                               2126 non-null   float64
 15  histogram_number_of_zeroes                              2126 non-null   float64
 16  histogram_mode                                          2126 non-null   float64
 17  histogram_mean                                          2126 non-null   float64
 18  histogram_median                                        2126 non-null   float64
 19  histogram_variance                                      2126 non-null   float64
 20  histogram_tendency                                      2126 non-null   float64
 21  fetal_health                                            2126 non-null   float64
dtypes: float64(22)
memory usage: 365.5 KB
```

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
baseline value                                            0
accelerations                                             0
fetal_movement                                            0
uterine_contractions                                      0
light_decelerations                                       0
severe_decelerations                                      0
prolongued_decelerations                                  0
abnormal_short_term_variability                           0
mean_value_of_short_term_variability                      0
percentage_of_time_with_abnormal_long_term_variability    0
mean_value_of_long_term_variability                       0
histogram_width                                           0
histogram_min                                             0
histogram_max                                             0
histogram_number_of_peaks                                 0
histogram_number_of_zeroes                                0
histogram_mode                                            0
histogram_mean                                            0
histogram_median                                          0
histogram_variance                                        0
histogram_tendency                                        0
fetal_health                                              0
dtype: int64
```

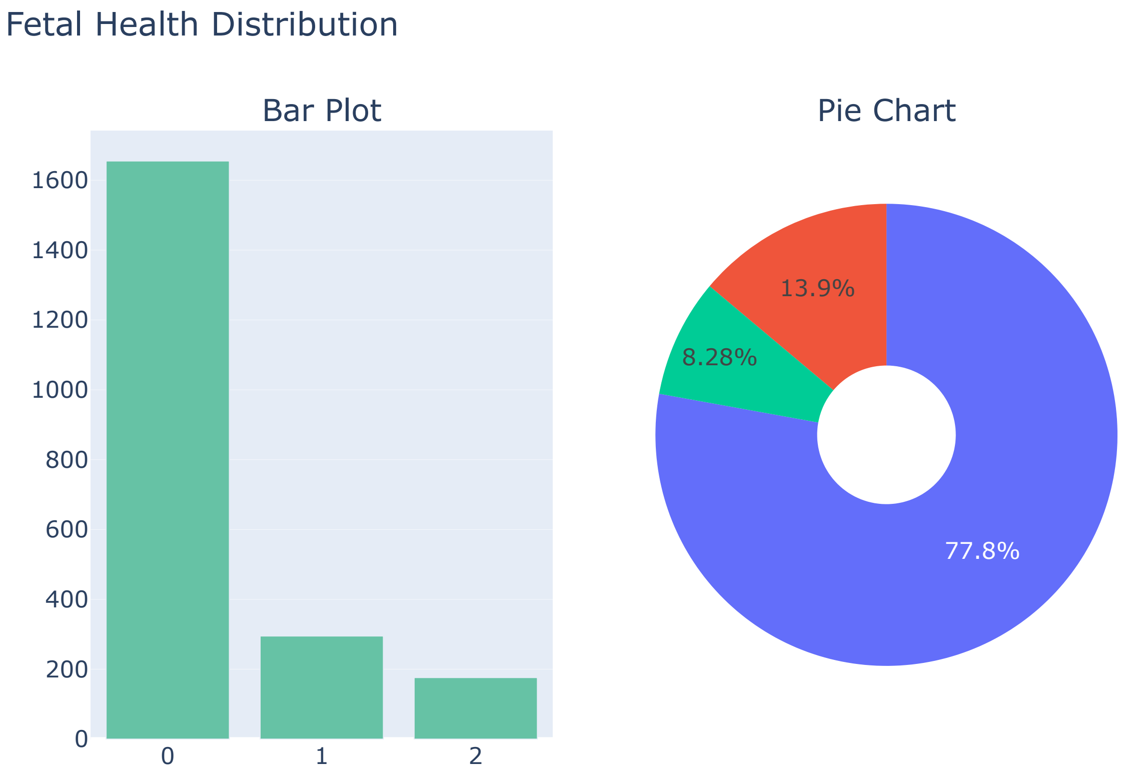
Exploratory Data Analysis (EDA)

• Univariate Analysis

In [6]:

```
# Bar plot for fetal_health value counts

bar_fig = go.Bar(
    x=(df['fetal_health']-1).value_counts().index,
    y=(df['fetal_health']-1).value_counts().values,
    marker=dict(color='#66C2A5')
)
```

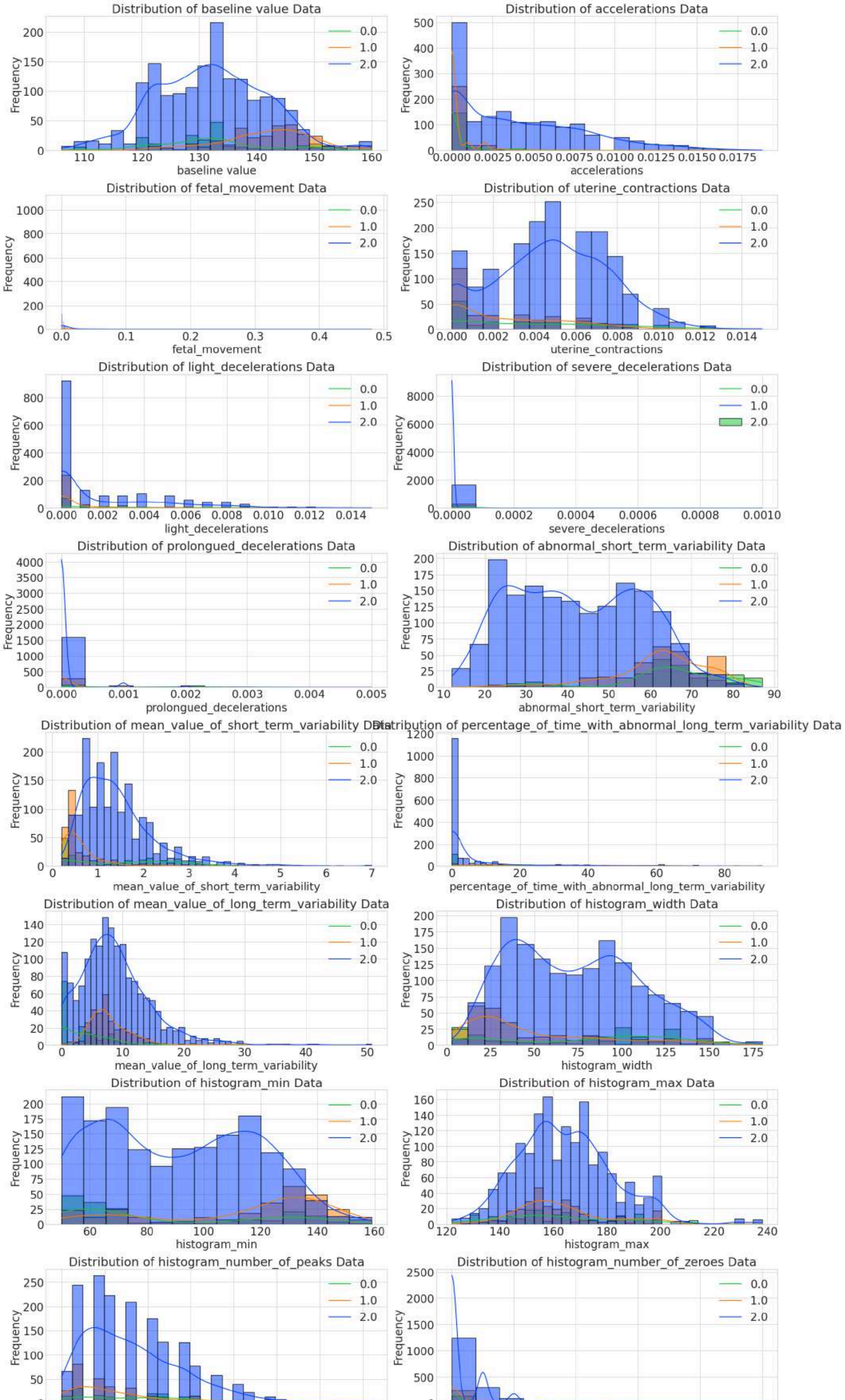
```
# Pie chart for fetal_health value counts
pie_fig = go.Pie(
    labels=(df['fetal_health']-1).value_counts().index,
    values=(df['fetal_health']-1).value_counts().values,
    hole=0.3  # Optional: for a donut-style pie chart
)

# Create subplots with Plotly
fig = sp.make_subplots(
    rows=1, cols=2,
    subplot_titles=("Bar Plot", "Pie Chart"),
    specs=[[{"type": "bar"}, {"type": "pie"}]]
)

# Add the bar plot to the first subplot
fig.add_trace(bar_fig, row=1, col=1)

# Add the pie chart to the second subplot
fig.add_trace(pie_fig, row=1, col=2)

# Update layout for better visualization
fig.update_layout(
    title_text="Fetal Health Distribution",
    showlegend=False,  # Hide legend to avoid redundancy in pie chart
    height=500,
    width=700
)

# Show the figure
fig.show()
```

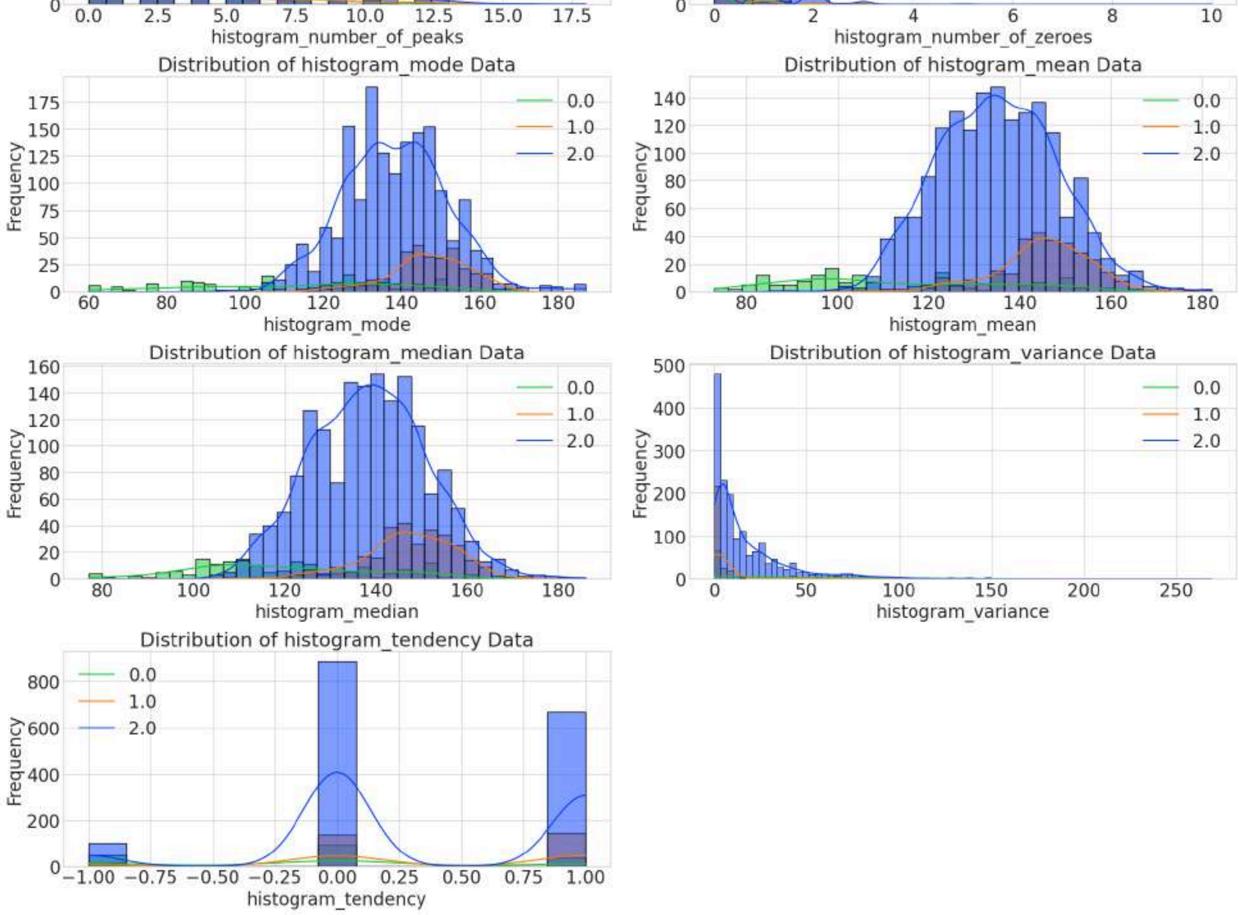


• Visualizing Data Distributions with `histplot`

In [7]:
```
import warnings
import seaborn as sns
```

```python
import matplotlib.pyplot as plt

# Suppress warnings
warnings.filterwarnings('ignore')

# Set seaborn style
plt.style.use('seaborn-v0_8-whitegrid')

# Drop the target variable from the features
X = df.drop('fetal_health', axis=1)

# Create the figure
plt.figure(figsize=(20, 50))
num_row = 1

# Loop through each column to create the subplots
for col in X.columns:
    plt.subplot(11, 2, num_row)

    # Set title with larger font size
    plt.title(f"Distribution of {col} Data", fontsize=22)

    # Plot histogram with KDE and hue
    sns.histplot(x=df[col], kde=True, hue=df['fetal_health'], palette='bright')

    # Set x and y axis labels with larger font size
    plt.xlabel(col, fontsize=20)
    plt.ylabel('Frequency', fontsize=20)

    # Set larger font size for ticks
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)

    # Set larger font size for the legend
    plt.legend(df['fetal_health'].value_counts().index-1,title_fontsize='20', fontsize='20')
    #plt.legend(title='fetal_health',title_fontsize='15', fontsize='15')
    # Adjust layout
    plt.tight_layout()

    num_row += 1

# Show the plots

plt.show()
```

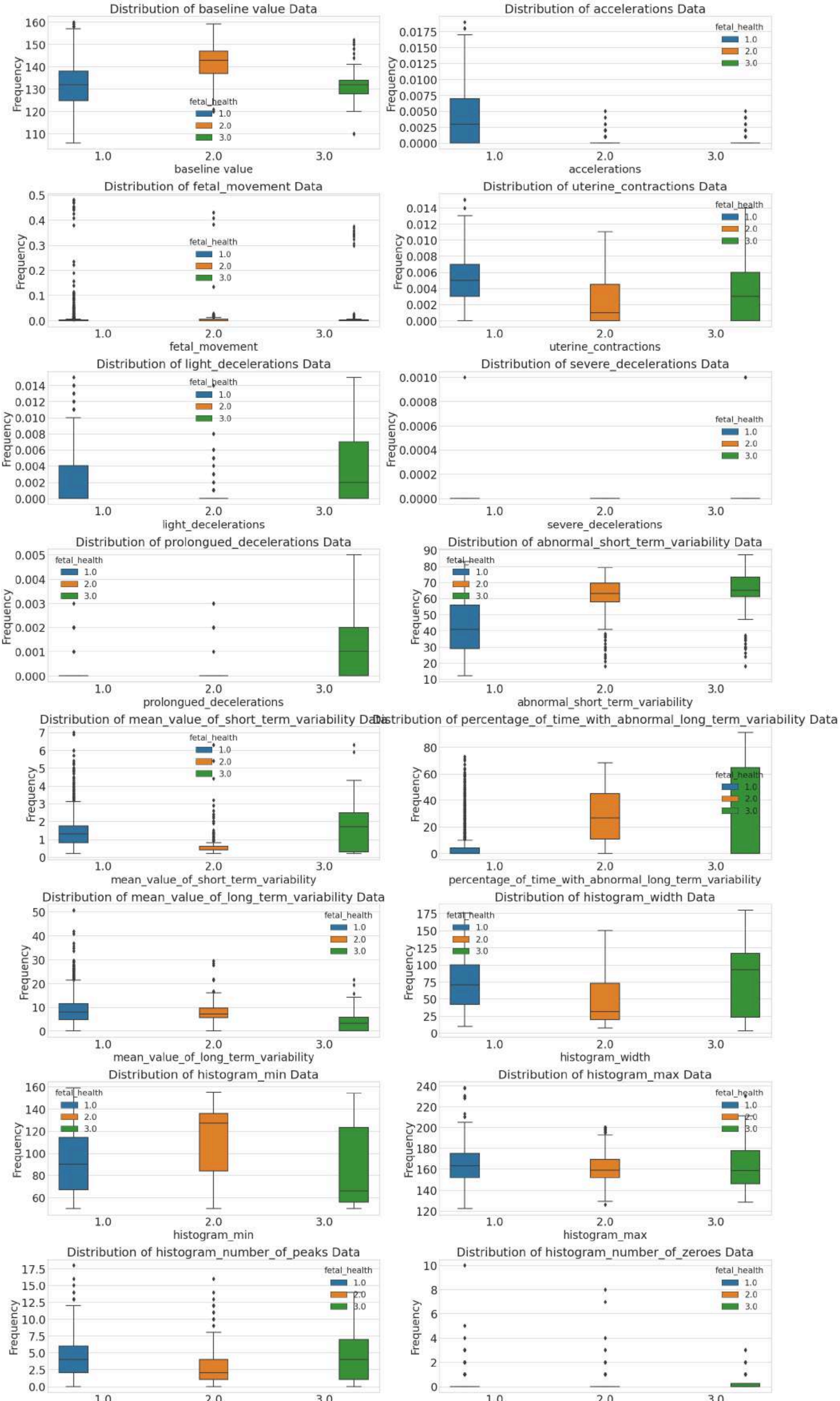

Distribution of baseline value Data
Frequency
0
50
100
150
200
110
120
130
140
150
160
baseline value
0.0
1.0
2.0
Distribution of accelerations Data
0
100
200
300
400
500
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
accelerations
Distribution of fetal_movement Data
0
200
400
600
800
1000
0.0
0.1
0.2
0.3
0.4
0.5
fetal_movement
Distribution of uterine_contractions Data
0
50
100
150
200
250
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
uterine_contractions
Distribution of light_decelerations Data
0
200
400
600
800
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
light_decelerations
Distribution of severe_decelerations Data
0
2000
4000
6000
8000
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
severe_decelerations
Distribution of prolongued_decelerations Data
0
500
1000
1500
2000
2500
3000
3500
4000
0.000
0.001
0.002
0.003
0.004
0.005
prolongued_decelerations
Distribution of abnormal_short_term_variability Data
0
25
50
75
100
125
150
175
200
10
20
30
40
50
60
70
80
90
abnormal_short_term_variability
Distribution of mean_value_of_short_term_variability Data
0
50
100
150
200
0
1
2
3
4
5
6
7
mean_value_of_short_term_variability
Distribution of percentage_of_time_with_abnormal_long_term_variability Data
0
200
400
600
800
1000
1200
0
20
40
60
80
percentage_of_time_with_abnormal_long_term_variability
Distribution of mean_value_of_long_term_variability Data
0
20
40
60
80
100
120
140
0
10
20
30
40
50
mean_value_of_long_term_variability
Distribution of histogram_width Data
0
25
50
75
100
125
150
175
200
0
25
50
75
100
125
150
175
histogram_width
Distribution of histogram_min Data
0
25
50
75
100
125
150
175
200
60
80
100
120
140
160
histogram_min
Distribution of histogram_max Data
0
20
40
60
80
100
120
140
160
120
140
160
180
200
220
240
histogram_max
Distribution of histogram_number_of_peaks Data
0
50
100
150
200
250
Distribution of histogram_number_of_zeroes Data
0
500
1000
1500
2000
2500

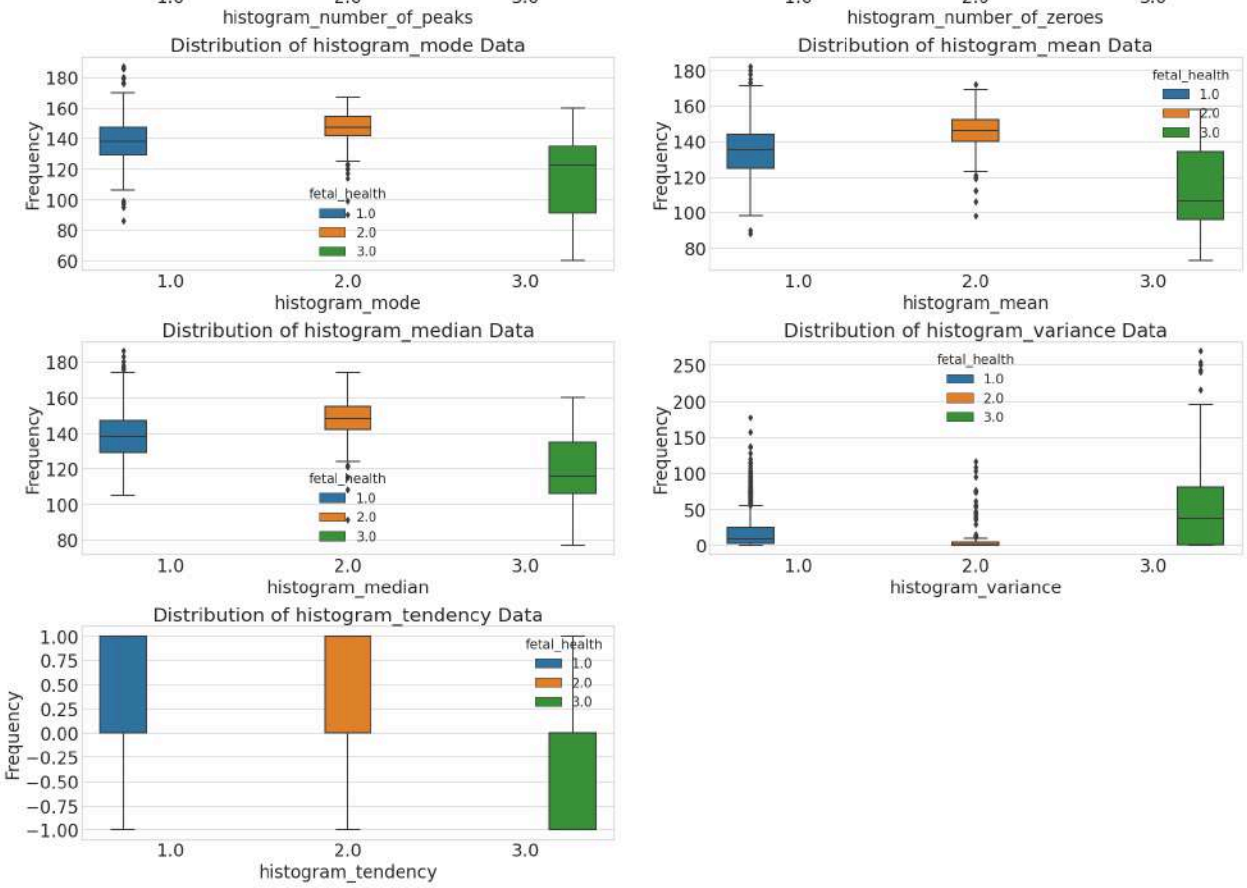


• Visualizing Data Distributions with `Box-Plot`

In [8]:

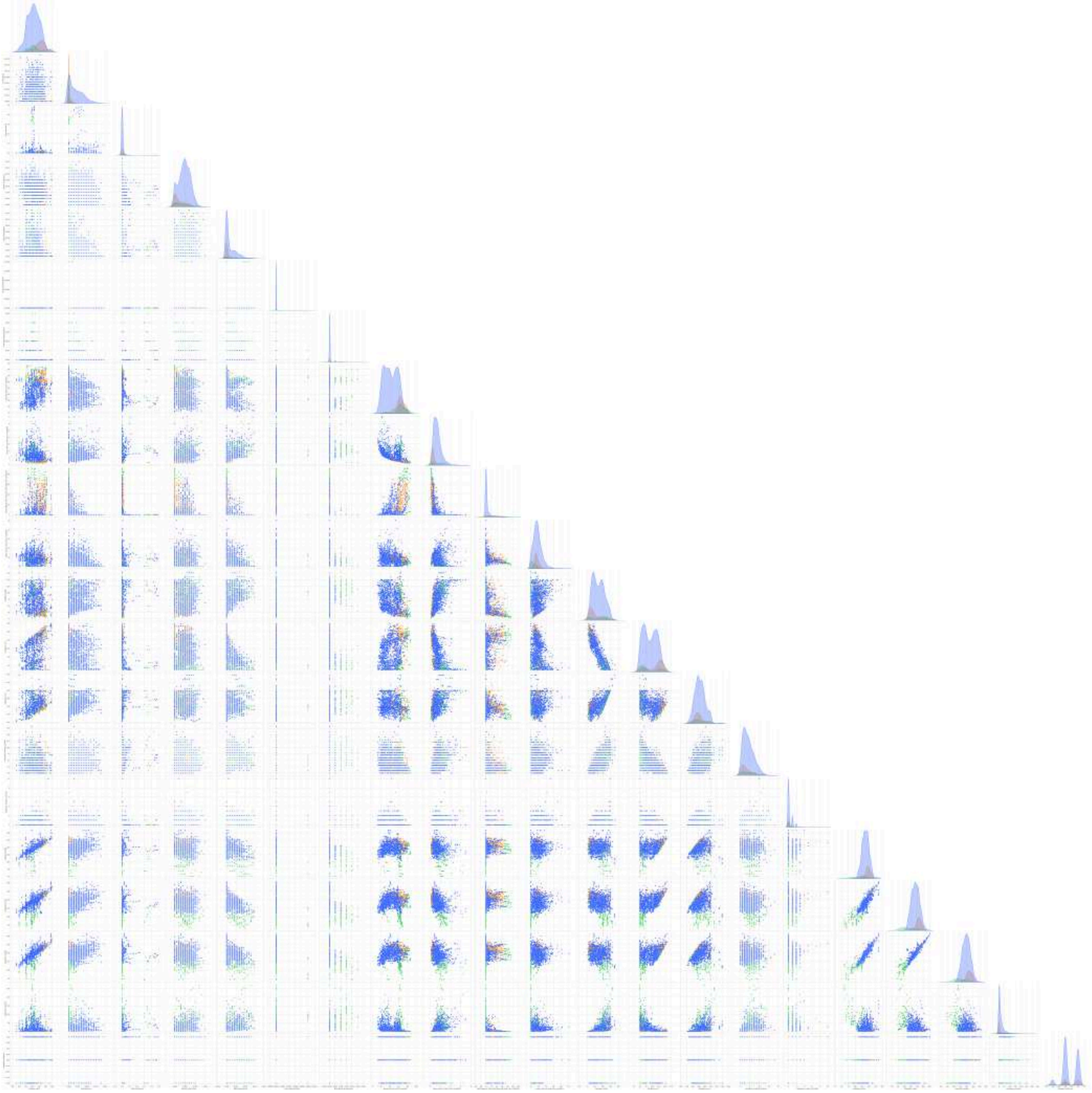
```
import warnings
import seaborn as sns
import matplotlib.pyplot as plt

# Suppress warnings
warnings.filterwarnings('ignore')

# Set seaborn style
plt.style.use('seaborn-v0_8-whitegrid')

# Drop the target variable from the features
X = df.drop('fetal_health', axis=1)

# Create the figure
plt.figure(figsize=(20, 50))
num_row = 1

# Loop through each column to create the subplots
for col in X.columns:
    plt.subplot(11, 2, num_row)

    # Set title with larger font size
    plt.title(f"Distribution of {col} Data", fontsize=22)

    # Plot histogram with KDE and hue
    sns.boxplot(y=df[col], x=df['fetal_health'],hue=df['fetal_health'])

    # Set x and y axis labels with larger font size
    plt.xlabel(col, fontsize=20)
    plt.ylabel('Frequency', fontsize=20)

    # Set larger font size for ticks
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)

    # Set larger font size for the legend
    plt.legend(title='fetal_health',title_fontsize='15', fontsize='15')

    # Adjust layout
    plt.tight_layout()

    num_row += 1

# Show the plots

plt.show()
```


Distribution of baseline value Data
Frequency
baseline value
fetal_health
1.0
2.0
3.0
Distribution of accelerations Data
accelerations
Distribution of fetal_movement Data
fetal_movement
Distribution of uterine_contractions Data
uterine_contractions
Distribution of light_decelerations Data
light_decelerations
Distribution of severe_decelerations Data
severe_decelerations
Distribution of prolongued_decelerations Data
prolongued_decelerations
Distribution of abnormal_short_term_variability Data
abnormal_short_term_variability
Distribution of mean_value_of_short_term_variability Data
mean_value_of_short_term_variability
Distribution of percentage_of_time_with_abnormal_long_term_variability Data
percentage_of_time_with_abnormal_long_term_variability
Distribution of mean_value_of_long_term_variability Data
mean_value_of_long_term_variability
Distribution of histogram_width Data
histogram_width
Distribution of histogram_min Data
histogram_min
Distribution of histogram_max Data
histogram_max
Distribution of histogram_number_of_peaks Data
Distribution of histogram_number_of_zeroes Data



Bivariate Analysis with `pairplot`

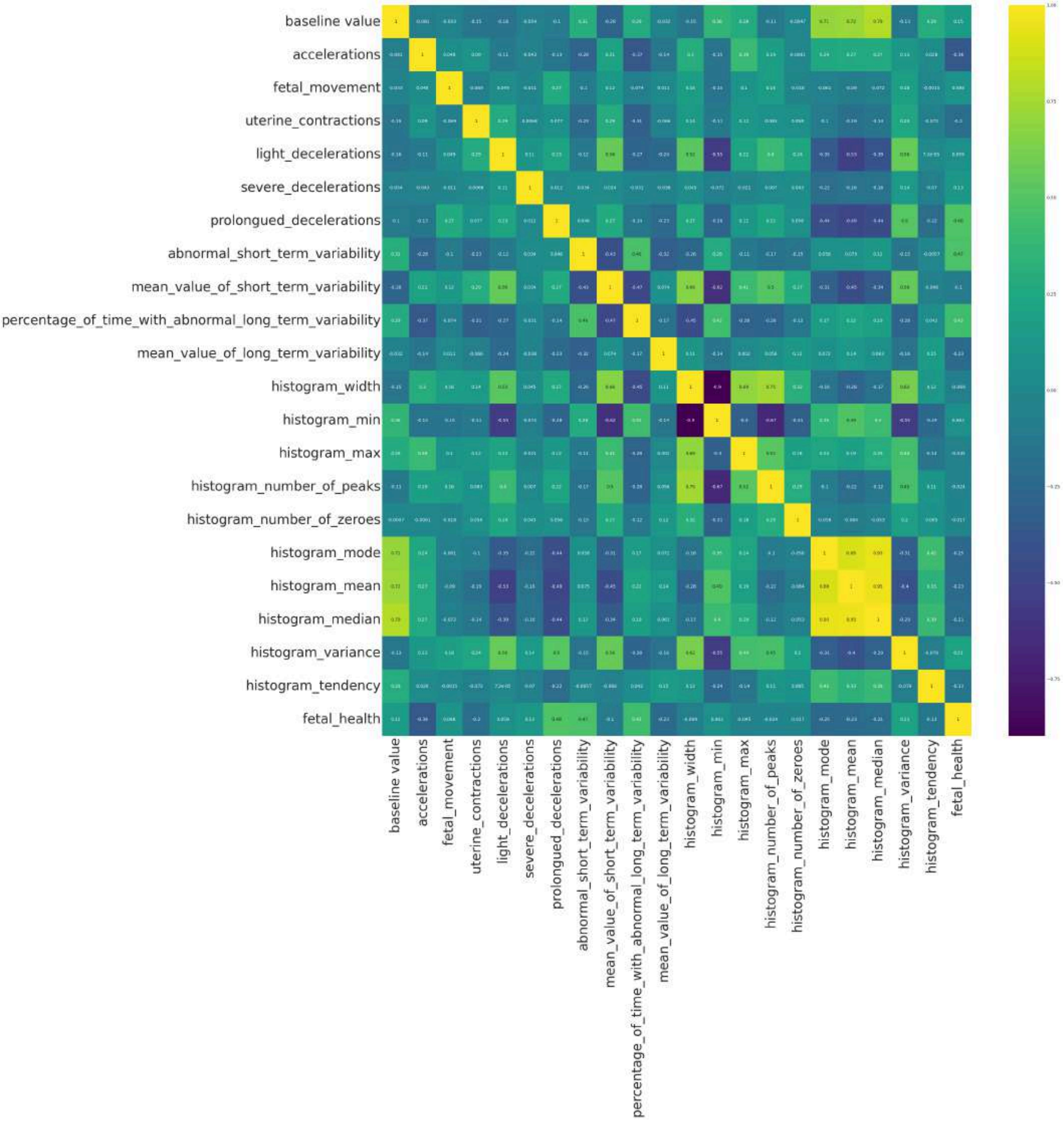
In [9]:
```
sns.pairplot(df, hue='fetal_health', corner=True, palette='bright', height=5)


plt.show()
```



Bivariate Analysis with `Heatmap`

In [10]:

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

# Create the figure
plt.figure(figsize=(30, 30))

# Plot the heatmap
sns.heatmap(df.corr(), annot=True, cmap='viridis')


plt.xticks(fontsize=30)
plt.yticks(fontsize=30)

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



Model Training and Prediction

In [11]:
```
# Splitting data into features and target
X = df.drop(columns=['fetal_health'])
y = df['fetal_health']
y=y-1
# Scaling the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

In [12]:
```
def plot_multi_class_roc(model, X_test, y_test, num_classes=3):
    # Binarize the labels for multi-class ROC calculation
    y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
    y_pred_proba = model.predict_proba(X_test)

    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    # Compute ROC curve and ROC area for each class
    for i in range(num_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plot ROC curve for each class
    plt.figure(figsize=(8, 6))
    colors = cycle(['blue', 'red', 'green'])

    for i, color in zip(range(num_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'Class {i+1} (AUC = {roc_auc[i]:.4f})')

    # Plot the diagonal (random classifier)
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Multi-Class ROC Curve')
    plt.legend(loc="lower right")
    plt.show()
```

- # Logistic Regression

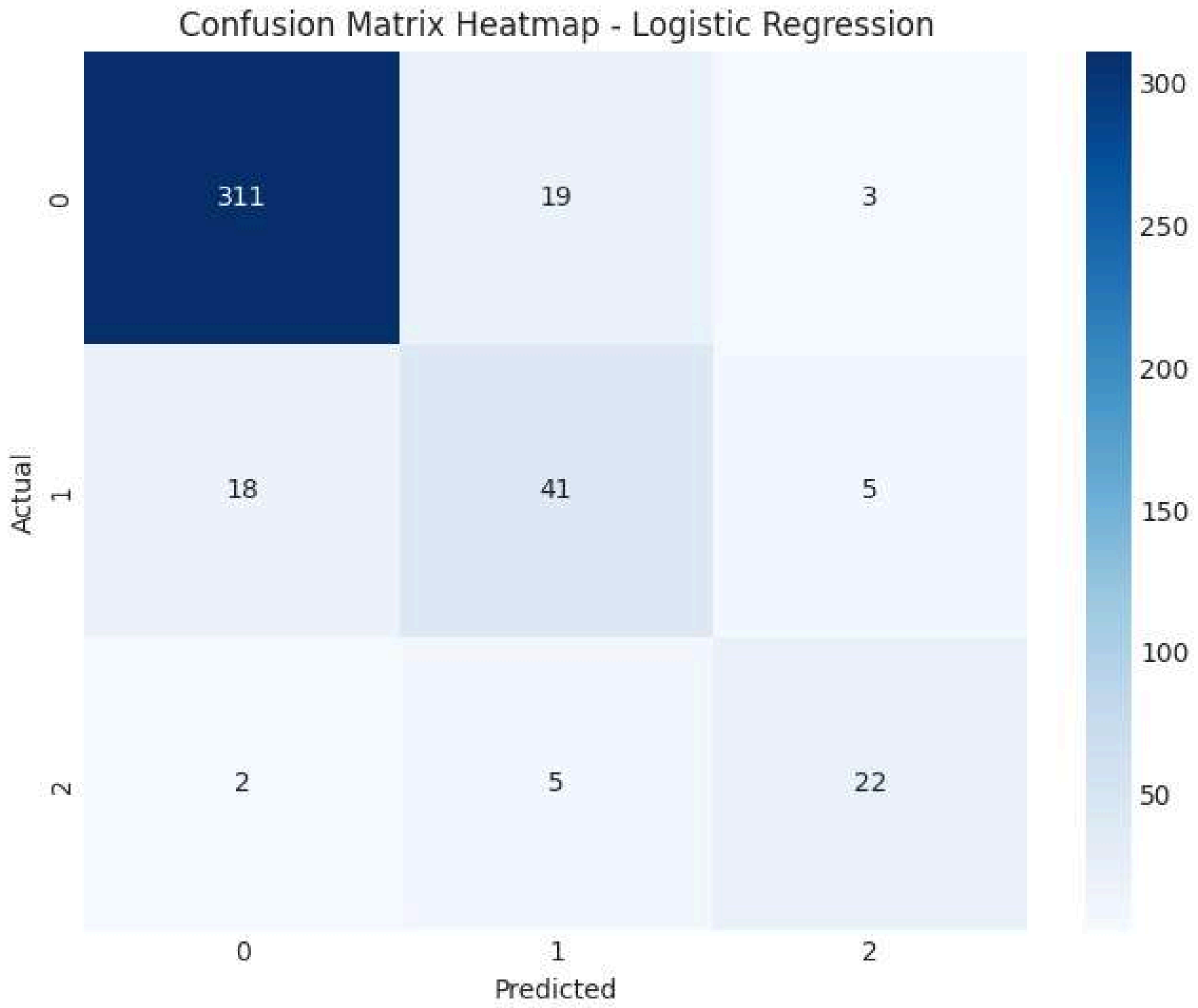
In [13]:

```
# Logistic Regression
log_model = LogisticRegression(random_state=42)
log_model.fit(X_train, y_train)
log_y_pred = log_model.predict(X_test)
log_accuracy = accuracy_score(y_test, log_y_pred)

# Logistic Regression - Classification Report & Confusion Matrix
print("Logistic Regression Classification Report")
print(classification_report(y_test, log_y_pred))
print("Logistic Regression Confusion Matrix")
print(confusion_matrix(y_test, log_y_pred))


# Calculate MCC for the model's predictions
mcc_log = matthews_corrcoef(y_test, log_y_pred)

# Print MCC
print(f"Matthews Correlation Coefficient (MCC): {mcc_log:.4f}")
```

```
Logistic Regression Classification Report
              precision    recall  f1-score   support

         0.0       0.94      0.93      0.94       333
         1.0       0.63      0.64      0.64        64
         2.0       0.73      0.76      0.75        29

    accuracy                           0.88       426
   macro avg       0.77      0.78      0.77       426
weighted avg       0.88      0.88      0.88       426

Logistic Regression Confusion Matrix
[[311  19   3]
 [ 18  41   5]
 [  2   5  22]]
Matthews Correlation Coefficient (MCC): 0.6655
```

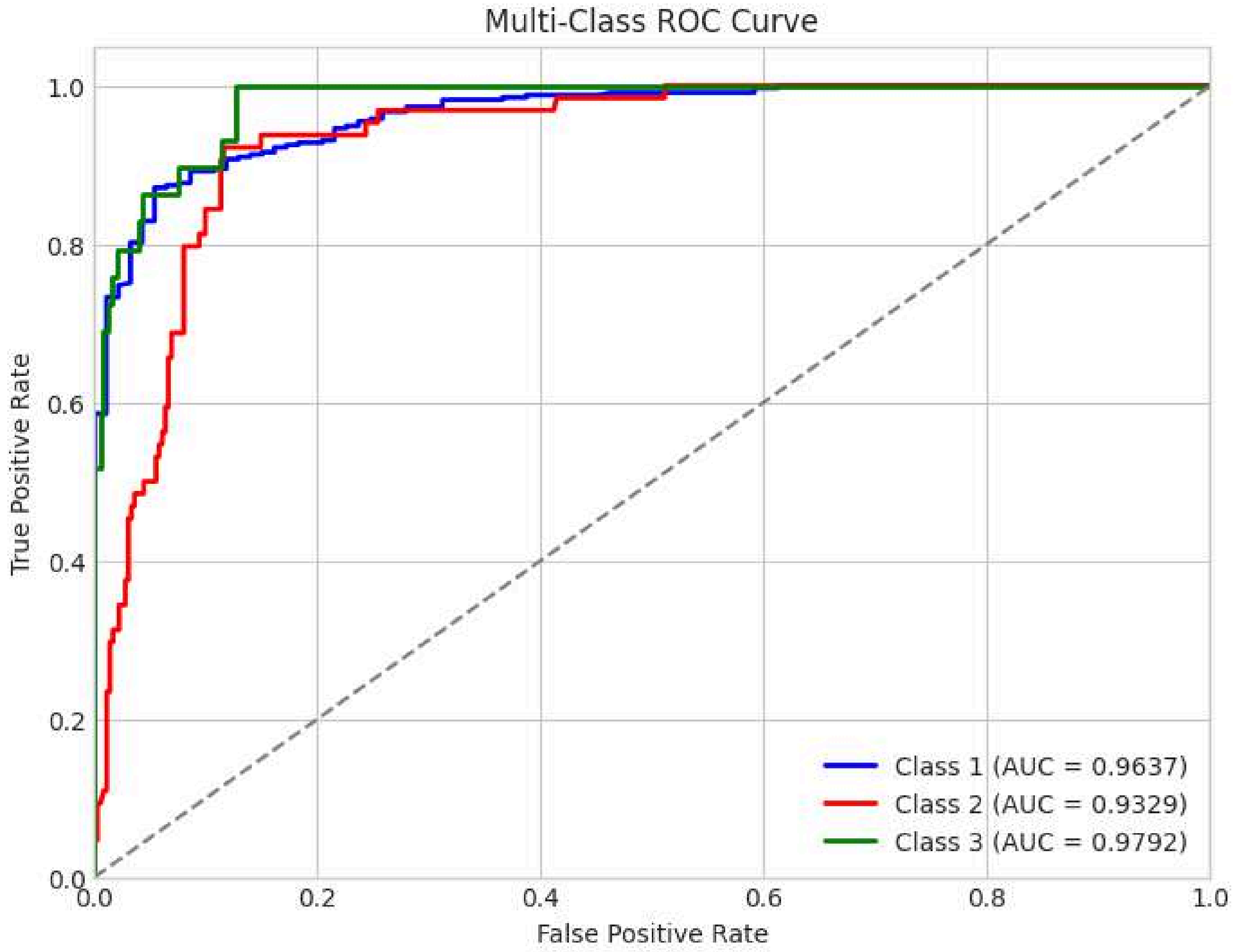
In [14]:

```
def plot_confusion_matrix_heatmap(y_test, y_pred, model_name):
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[0, 1, 2], yticklabels=[0, 1, 2]
    plt.title(f'Confusion Matrix Heatmap - {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
plot_confusion_matrix_heatmap(y_test, log_y_pred, 'Logistic Regression')
```



In [15]:

```
plot_multi_class_roc(log_model, X_test, y_test, num_classes=3)
```



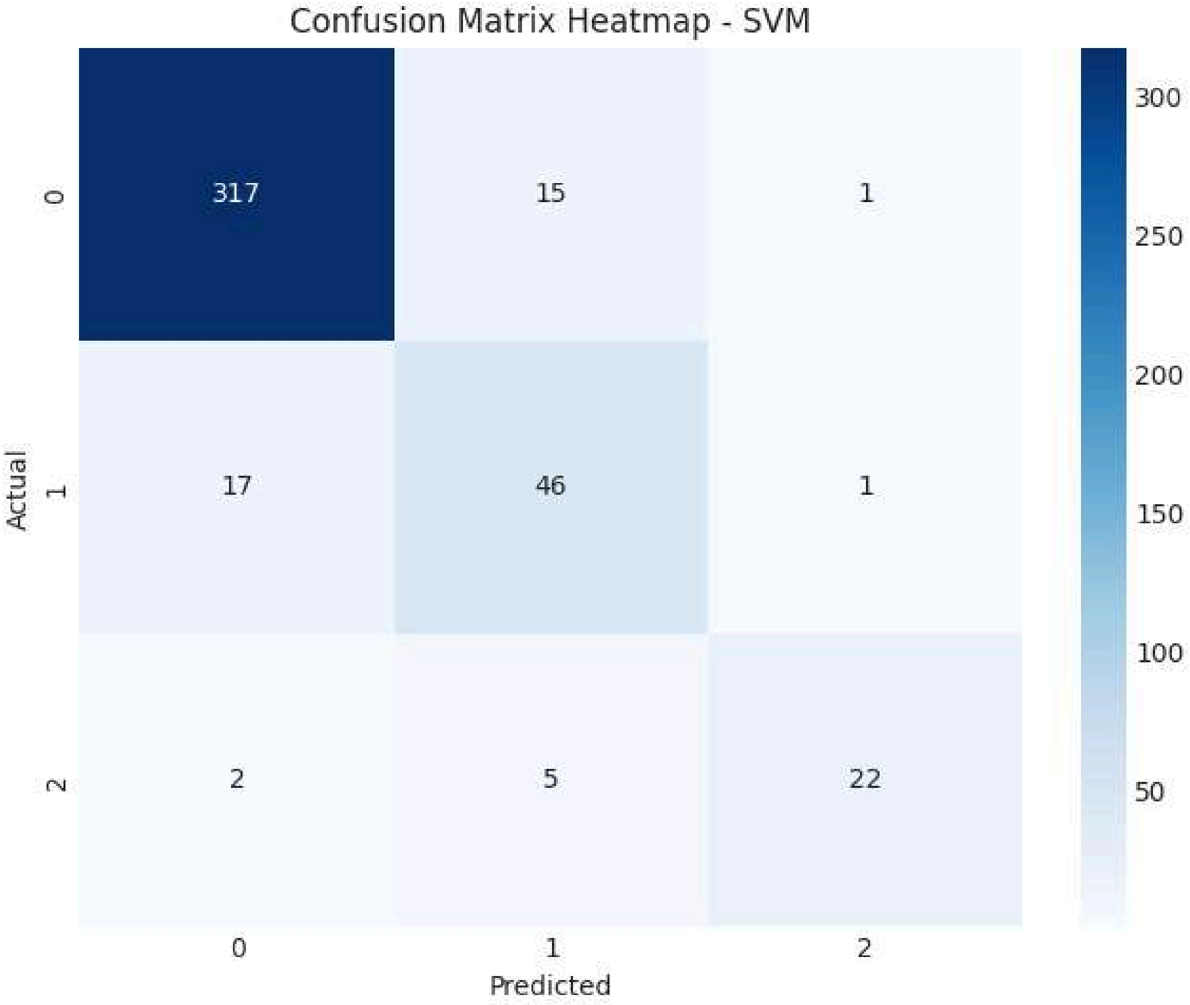
• Support Vector Machine (SVM)

In [16]:

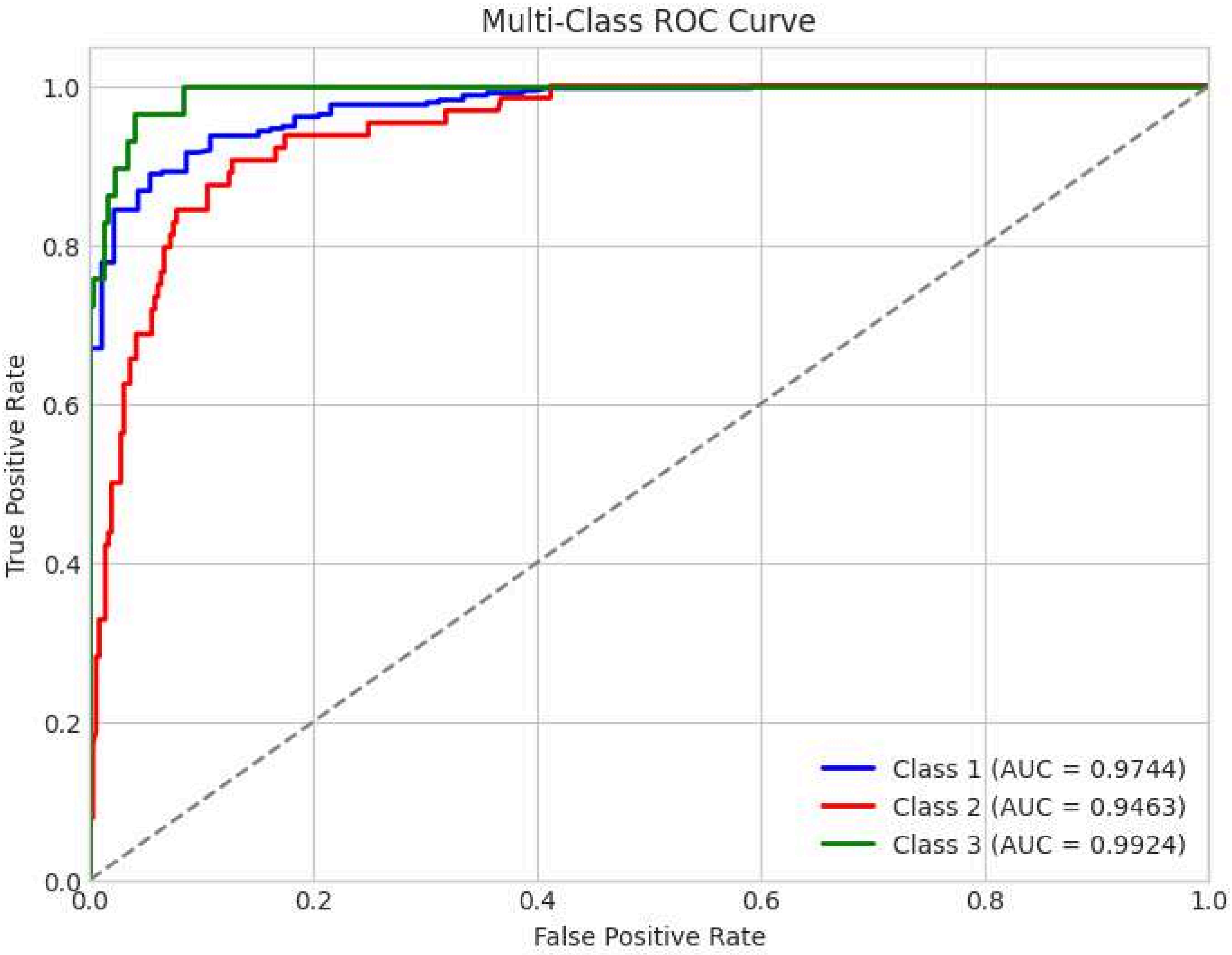
```
# Support Vector Machine (SVM)
svm_model = SVC(random_state=42, probability=True)
svm_model.fit(X_train, y_train)
svm_y_pred = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_y_pred)

# Support Vector Machine (SVM) - Classification Report & Confusion Matrix
print(svm_accuracy)
print("SVM Classification Report")
print(classification_report(y_test, svm_y_pred))
print("SVM Confusion Matrix")
print(confusion_matrix(y_test, svm_y_pred))

# Calculate MCC for the model's predictions
mcc_svm = matthews_corrcoef(y_test,svm_y_pred)

# Print MCC
print(f"Matthews Correlation Coefficient (MCC): {mcc_svm:.4f}")
```

```
0.903755868544601
SVM Classification Report
              precision    recall  f1-score   support

         0.0       0.94      0.95      0.95       333
         1.0       0.70      0.72      0.71        64
         2.0       0.92      0.76      0.83        29

    accuracy                           0.90       426
   macro avg       0.85      0.81      0.83       426
weighted avg       0.90      0.90      0.90       426

SVM Confusion Matrix
[[317  15   1]
 [ 17  46   1]
 [  2   5  22]]
Matthews Correlation Coefficient (MCC): 0.7302
```

In [17]:
```
plot_confusion_matrix_heatmap(y_test, svm_y_pred, 'SVM')
```



In [18]:
```
plot_multi_class_roc(svm_model ,X_test, y_test, num_classes=3)
```



• Random Forest

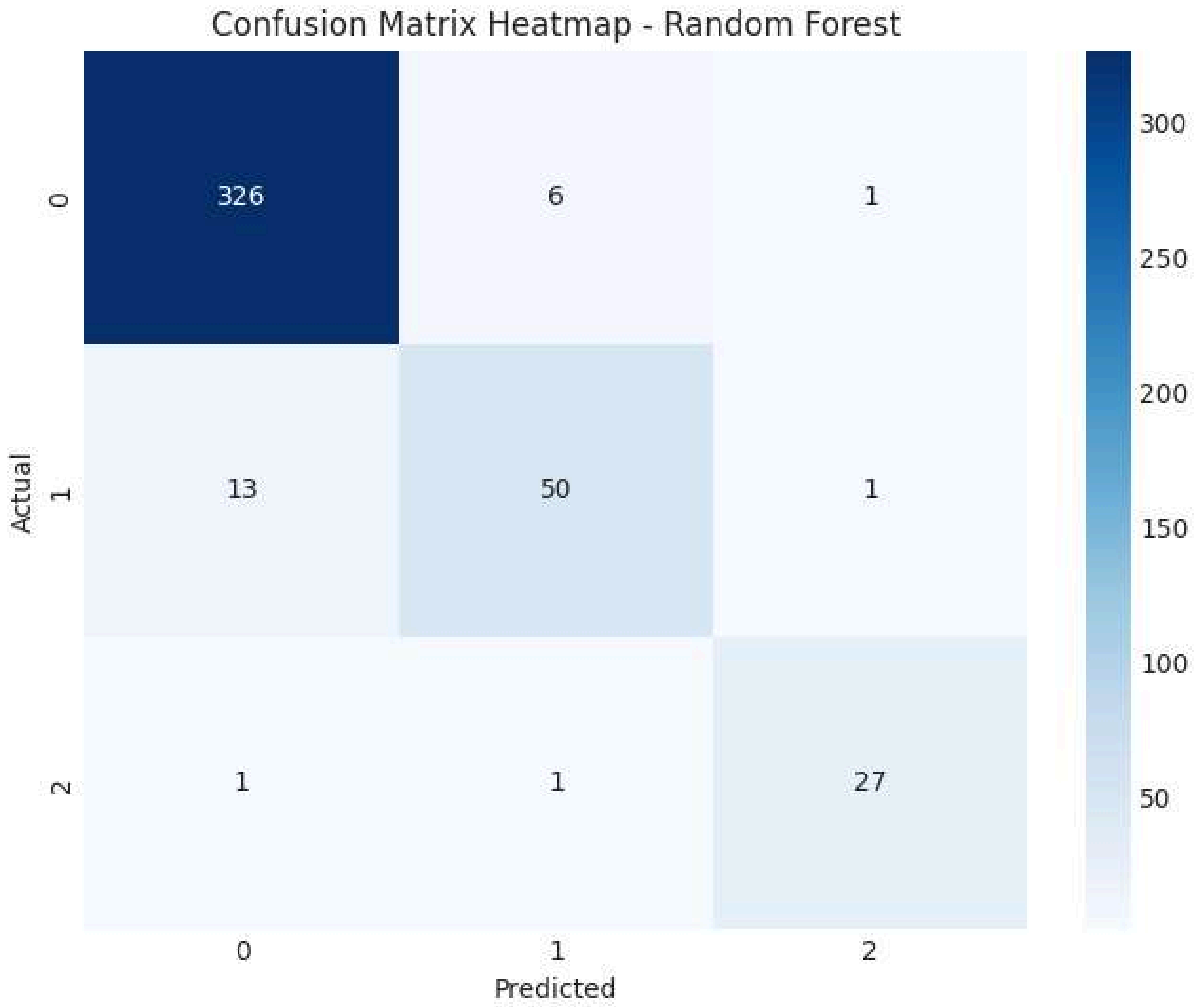
In [19]:

```
# Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_y_pred = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_y_pred)

# Random Forest - Classification Report & Confusion Matrix
print("Random Forest Classification Report")
print(classification_report(y_test, rf_y_pred))
print("Random Forest Confusion Matrix")
print(confusion_matrix(y_test, rf_y_pred))

# Calculate MCC for the model's predictions
mcc_rf = matthews_corrcoef(y_test,rf_y_pred)

# Print MCC
print(f"Matthews Correlation Coefficient (MCC): {mcc_rf:.4f}")
```

```
Random Forest Classification Report
              precision    recall  f1-score   support

         0.0       0.96      0.98      0.97       333
         1.0       0.88      0.78      0.83        64
         2.0       0.93      0.93      0.93        29

    accuracy                           0.95       426
   macro avg       0.92      0.90      0.91       426
weighted avg       0.94      0.95      0.94       426

Random Forest Confusion Matrix
[[326   6   1]
 [ 13  50   1]
 [  1   1  27]]
Matthews Correlation Coefficient (MCC): 0.8474
```

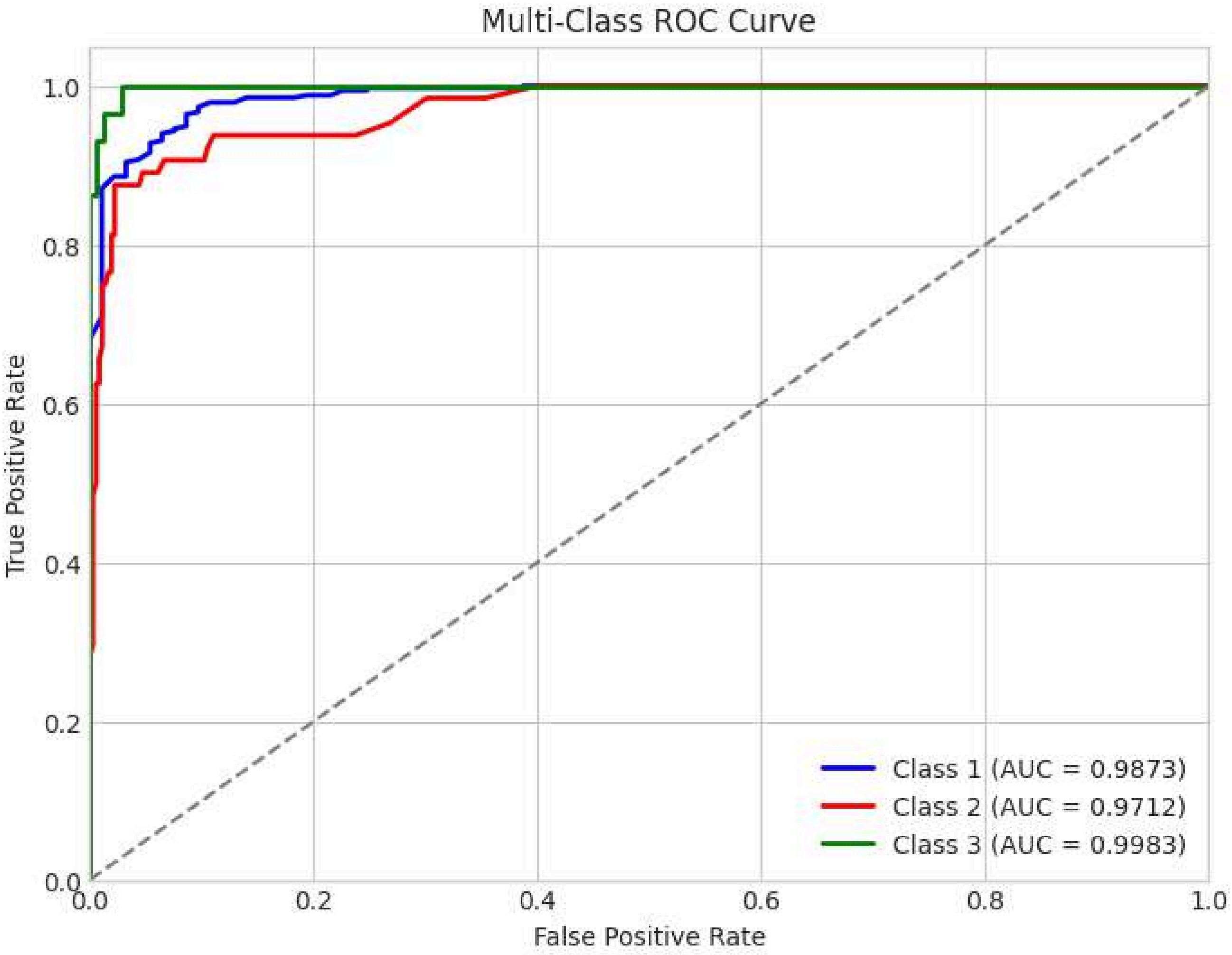
In [20]:

```
plot_confusion_matrix_heatmap(y_test, rf_y_pred, 'Random Forest')
```



In [21]:

```
plot_multi_class_roc(rf_model ,X_test, y_test, num_classes=3)
```



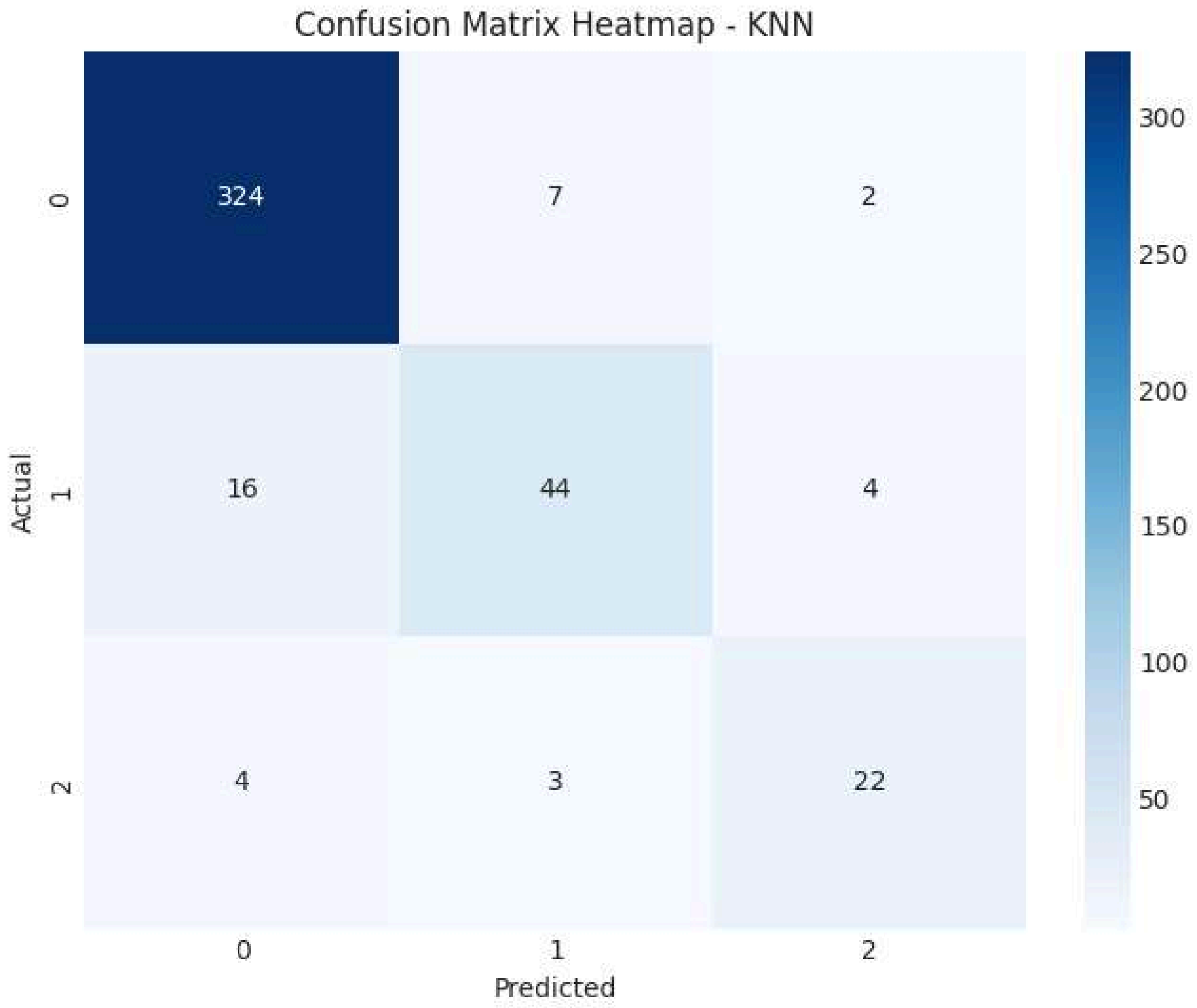
• K-Nearest Neighbors (KNN)

In [22]:

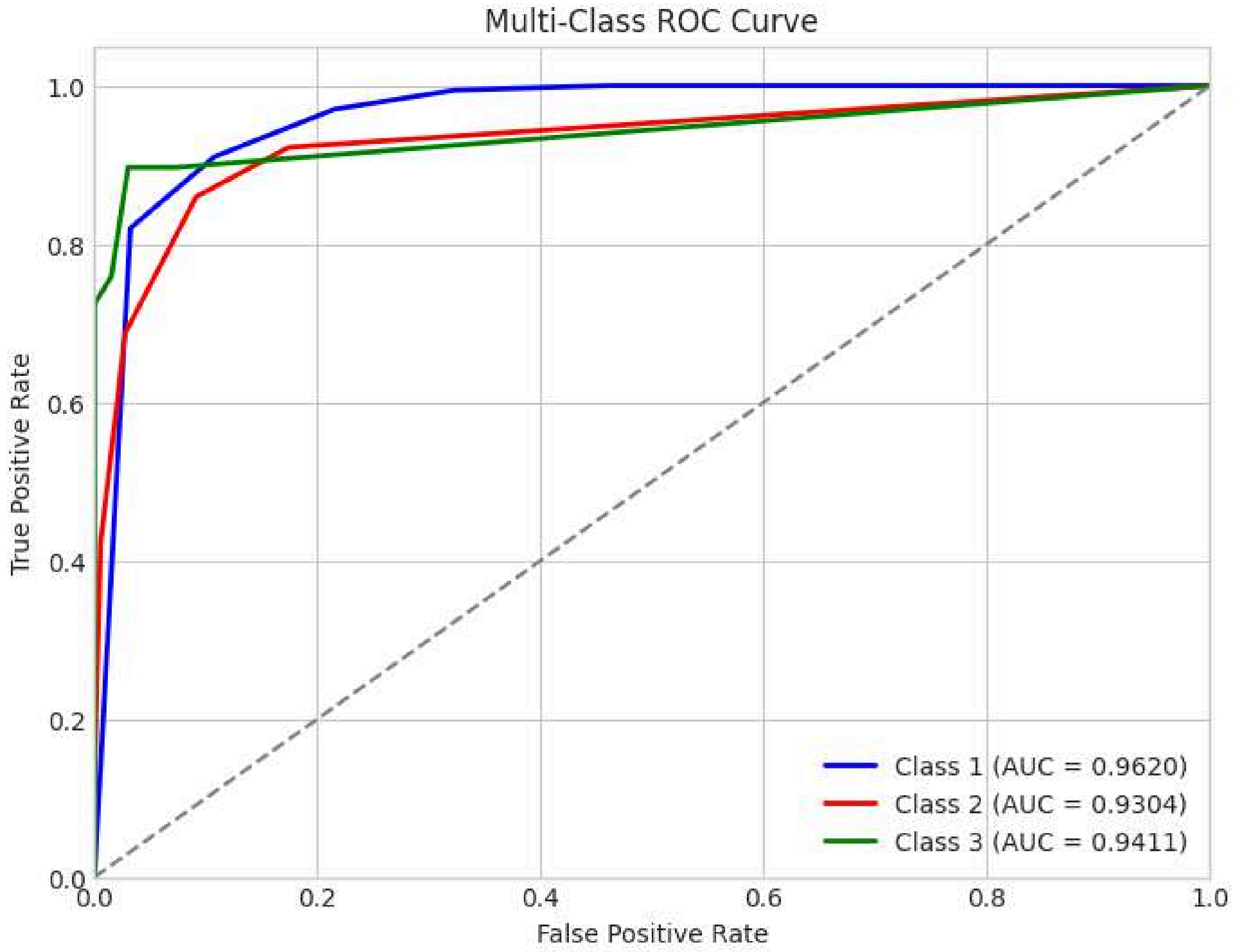
```
# K-Nearest Neighbors (KNN)
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
knn_y_pred = knn_model.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_y_pred)

# K-Nearest Neighbors (KNN) - Classification Report & Confusion Matrix
print("KNN Classification Report")
print(classification_report(y_test, knn_y_pred))
print("KNN Confusion Matrix")
print(confusion_matrix(y_test, knn_y_pred))

# Calculate MCC for the model's predictions
mcc_knn = matthews_corrcoef(y_test,knn_y_pred)

# Print MCC
print(f"Matthews Correlation Coefficient (MCC): {mcc_knn:.4f}")
```

```
KNN Classification Report
              precision    recall  f1-score   support

         0.0       0.94      0.97      0.96       333
         1.0       0.81      0.69      0.75        64
         2.0       0.79      0.76      0.77        29

    accuracy                           0.92       426
   macro avg       0.85      0.81      0.82       426
weighted avg       0.91      0.92      0.91       426

KNN Confusion Matrix
[[324   7   2]
 [ 16  44   4]
 [  4   3  22]]
Matthews Correlation Coefficient (MCC): 0.7575
```

In [23]:
```
plot_confusion_matrix_heatmap(y_test, knn_y_pred, 'KNN')
```



In [24]:
```
plot_multi_class_roc(knn_model ,X_test, y_test, num_classes=3)
```



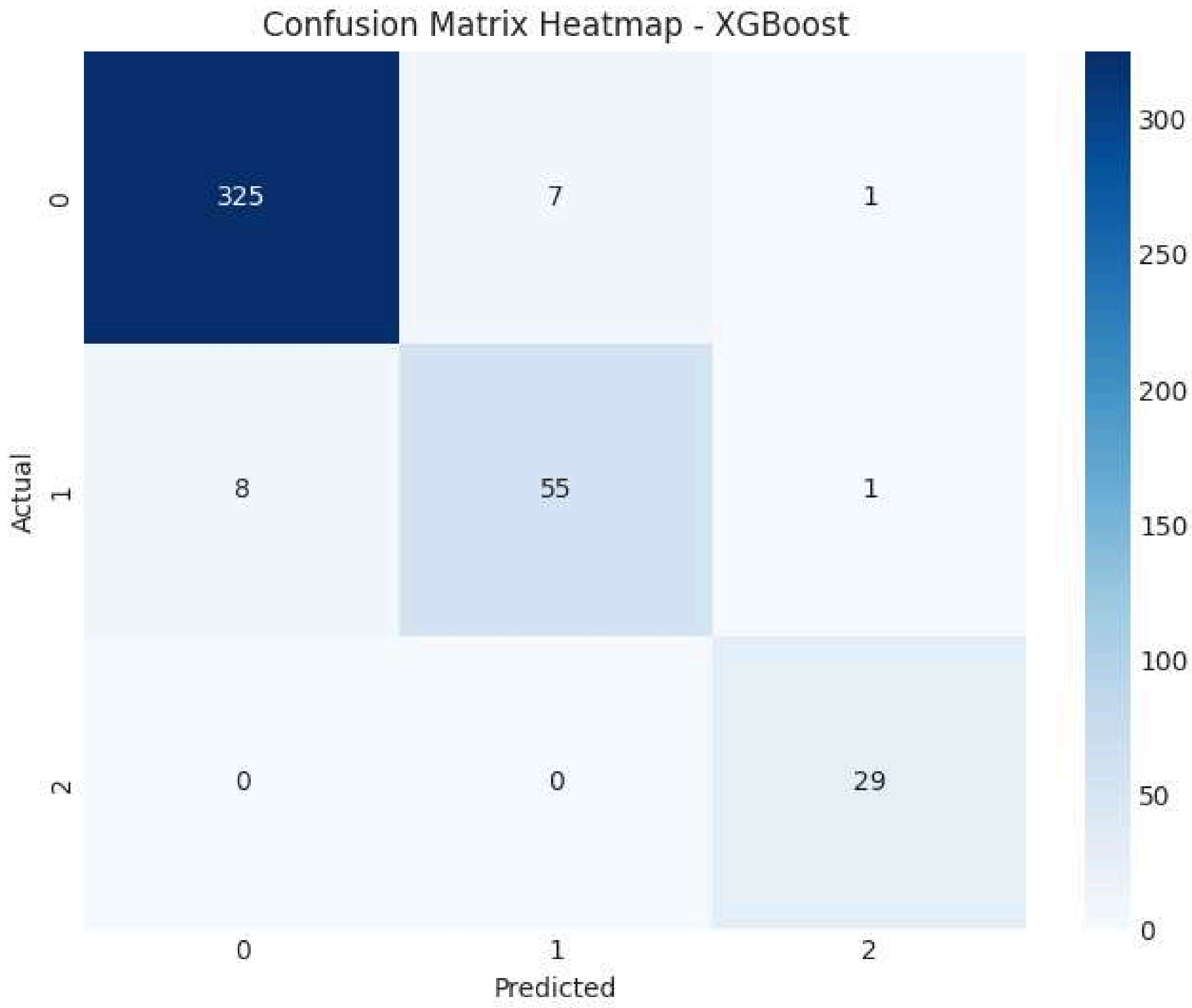
● XGBoost

In [25]:

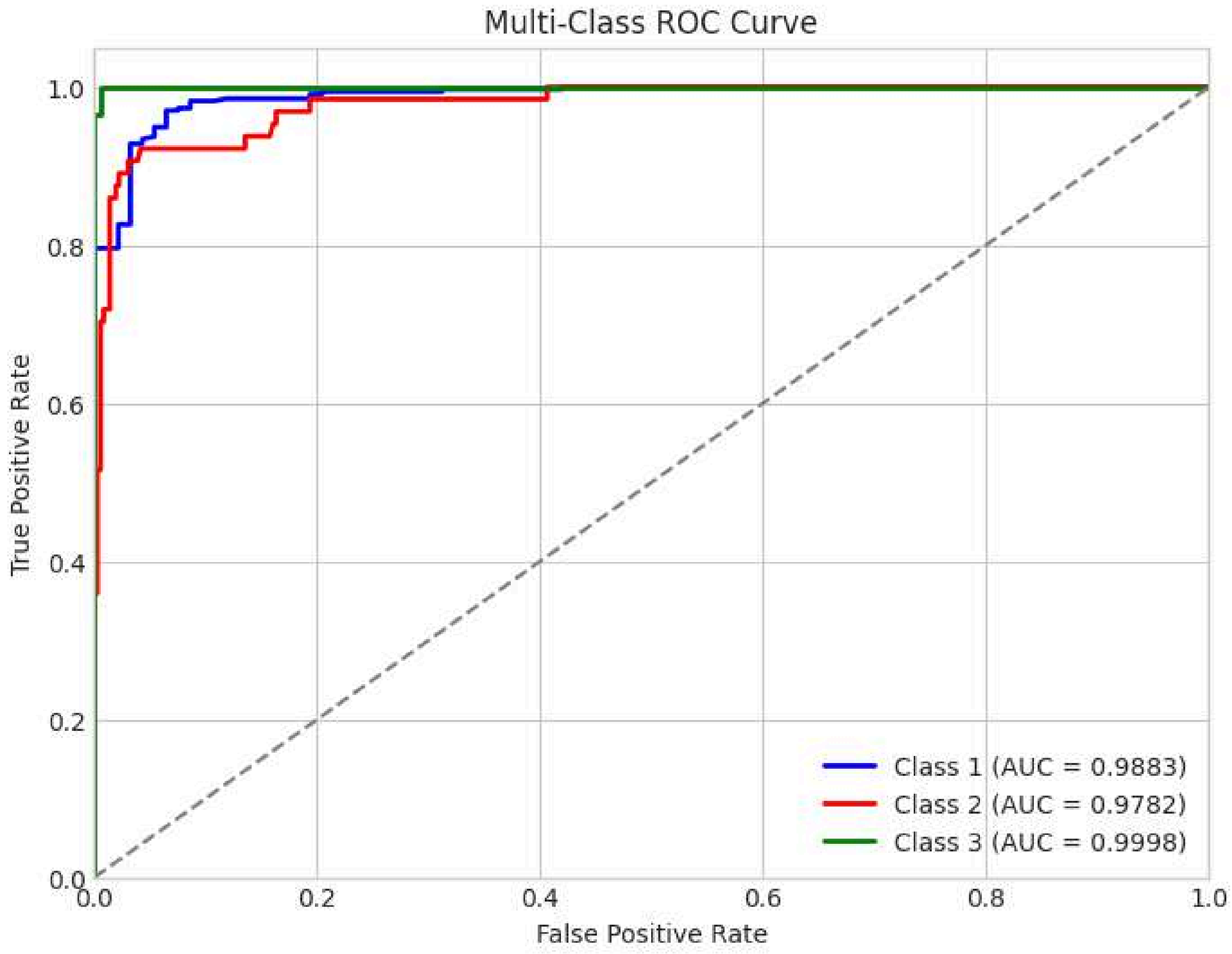
```
# XGBoost
xgb_model = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='mlogloss')
xgb_model.fit(X_train, y_train)
xgb_y_pred = xgb_model.predict(X_test)
xgb_accuracy = accuracy_score(y_test, xgb_y_pred)

# XGBoost - Classification Report & Confusion Matrix

print("XGBoost Classification Report")
print(classification_report(y_test, xgb_y_pred))
print("XGBoost Confusion Matrix")
print(confusion_matrix(y_test, xgb_y_pred))

# Calculate MCC for the model's predictions
mcc_XGb = matthews_corrcoef(y_test,xgb_y_pred)

# Print MCC
print(f"Matthews Correlation Coefficient (MCC): {mcc_XGb:.4f}")
```

```
XGBoost Classification Report
              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98       333
         1.0       0.89      0.86      0.87        64
         2.0       0.94      1.00      0.97        29

    accuracy                           0.96       426
   macro avg       0.93      0.95      0.94       426
weighted avg       0.96      0.96      0.96       426

XGBoost Confusion Matrix
[[325   7   1]
 [  8  55   1]
 [  0   0  29]]
Matthews Correlation Coefficient (MCC): 0.8899
```

```
In [26]: plot_confusion_matrix_heatmap(y_test, xgb_y_pred, 'XGBoost')
```



```
In [27]: plot_multi_class_roc(xgb_model ,X_test, y_test, num_classes=3)
```



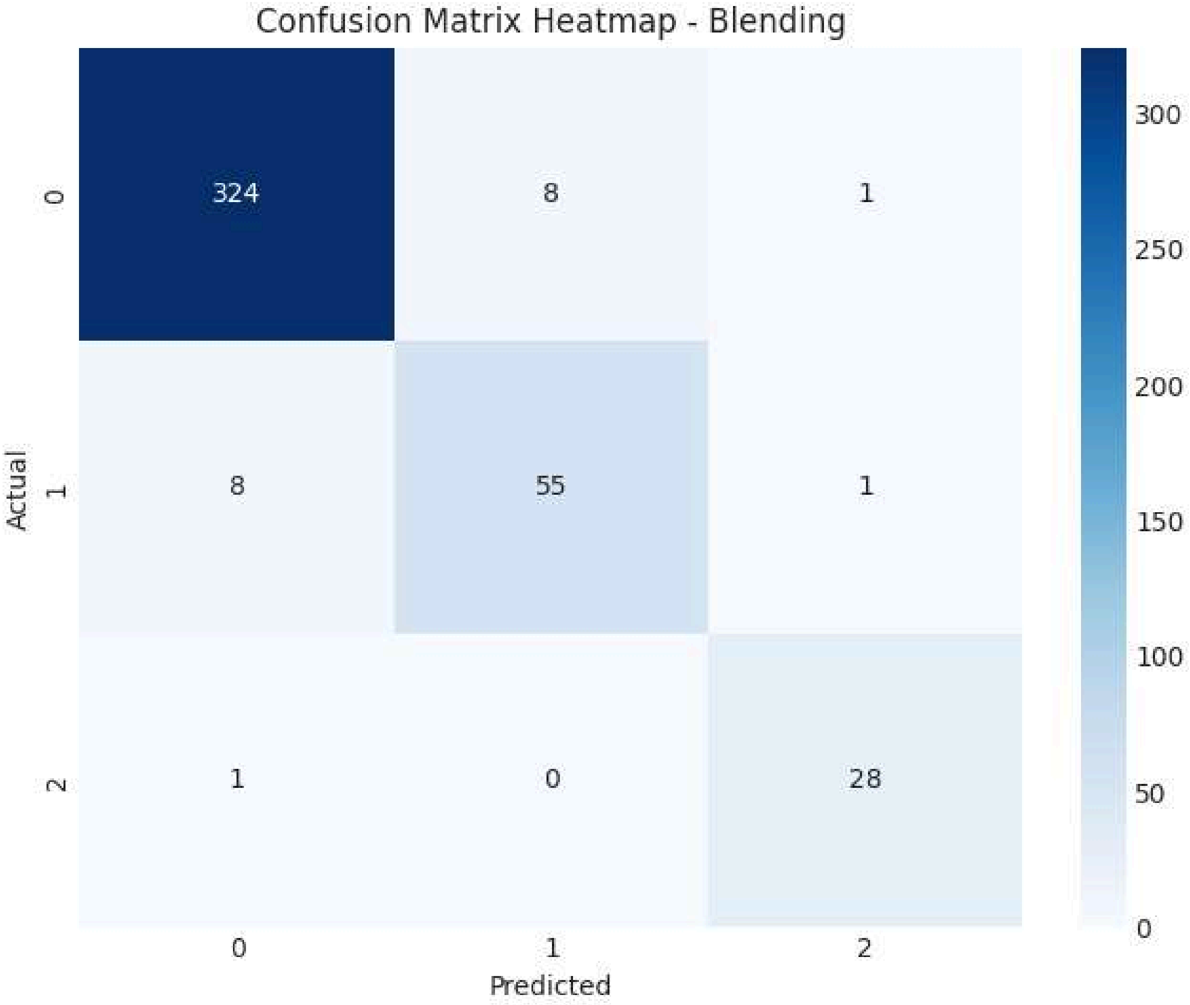
• Blending (Voting Classifier)

In [28]:

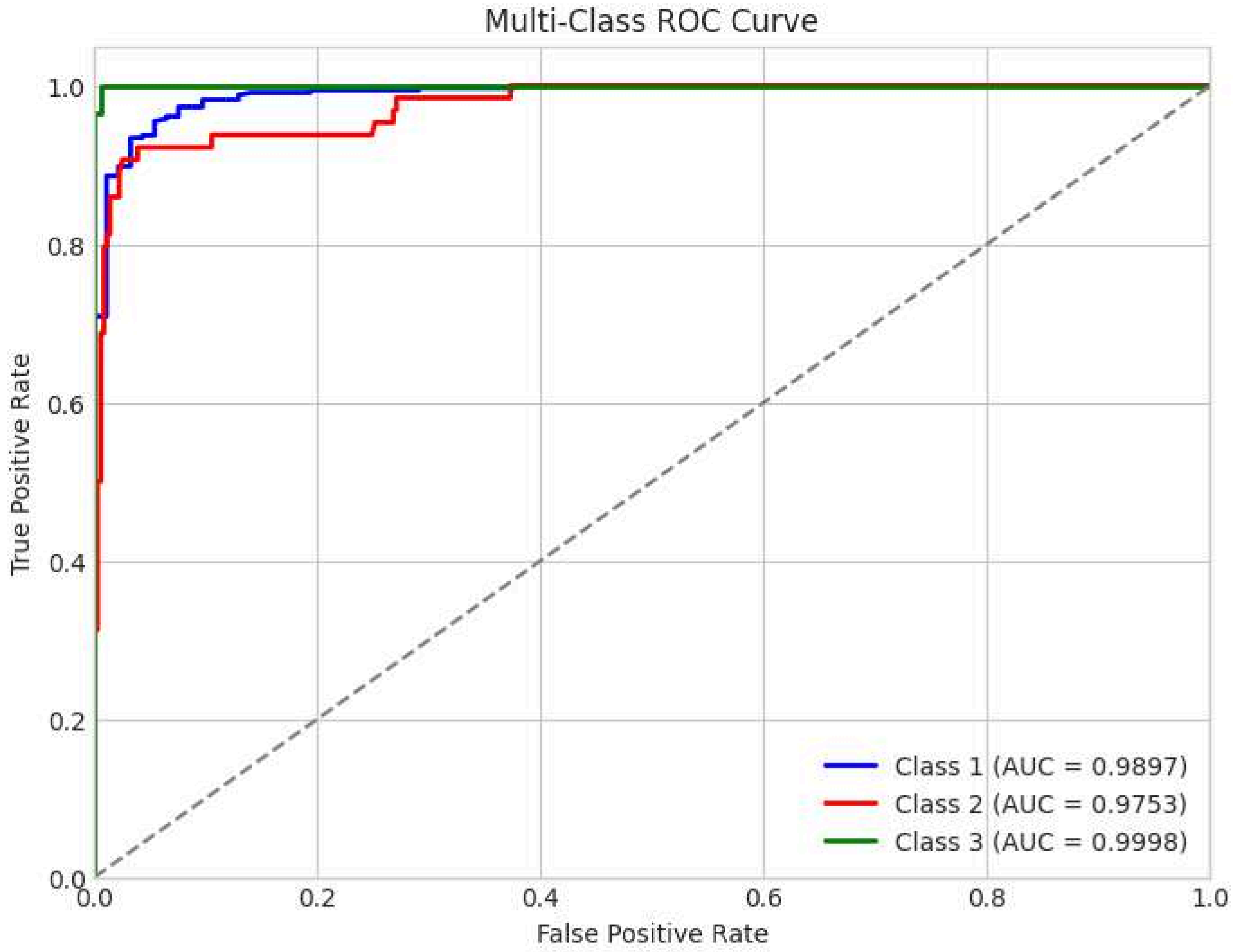
```
# Blending (Voting Classifier)
voting_model = VotingClassifier(
    estimators=[('xgb', xgb_model), ('rf', rf_model)],
    voting='soft'  # soft voting averages the predicted probabilities
)
voting_model.fit(X_train, y_train)

# Predictions
voting_y_pred = voting_model.predict(X_test)

# Accuracy
voting_accuracy = accuracy_score(y_test, voting_y_pred)

# Classification Report & Confusion Matrix for Blending
print("Blending (Voting) Classification Report")
print(classification_report(y_test, voting_y_pred))
print("Blending (Voting) Confusion Matrix")
print(confusion_matrix(y_test, voting_y_pred))

# Accuracy of Blending
print(f"Blending Accuracy: {voting_accuracy:.4f}")

# Calculate MCC for the model's predictions
mcc_voting = matthews_corrcoef(y_test,voting_y_pred)

# Print MCC
print(f"Matthews Correlation Coefficient (MCC): {mcc_voting:.4f}")
```

```
Blending (Voting) Classification Report
              precision    recall  f1-score   support

         0.0       0.97      0.97      0.97       333
         1.0       0.87      0.86      0.87        64
         2.0       0.93      0.97      0.95        29

    accuracy                           0.96       426
   macro avg       0.93      0.93      0.93       426
weighted avg       0.96      0.96      0.96       426

Blending (Voting) Confusion Matrix
[[324   8   1]
 [  8  55   1]
 [  1   0  28]]
Blending Accuracy: 0.9554
Matthews Correlation Coefficient (MCC): 0.8768
```

In [29]:
```
plot_confusion_matrix_heatmap(y_test, voting_y_pred, 'Blending')
```



In [30]:
```
plot_multi_class_roc(voting_model ,X_test, y_test, num_classes=3)
```



• Stacking Classifier with Cross-Validation

In [31]:

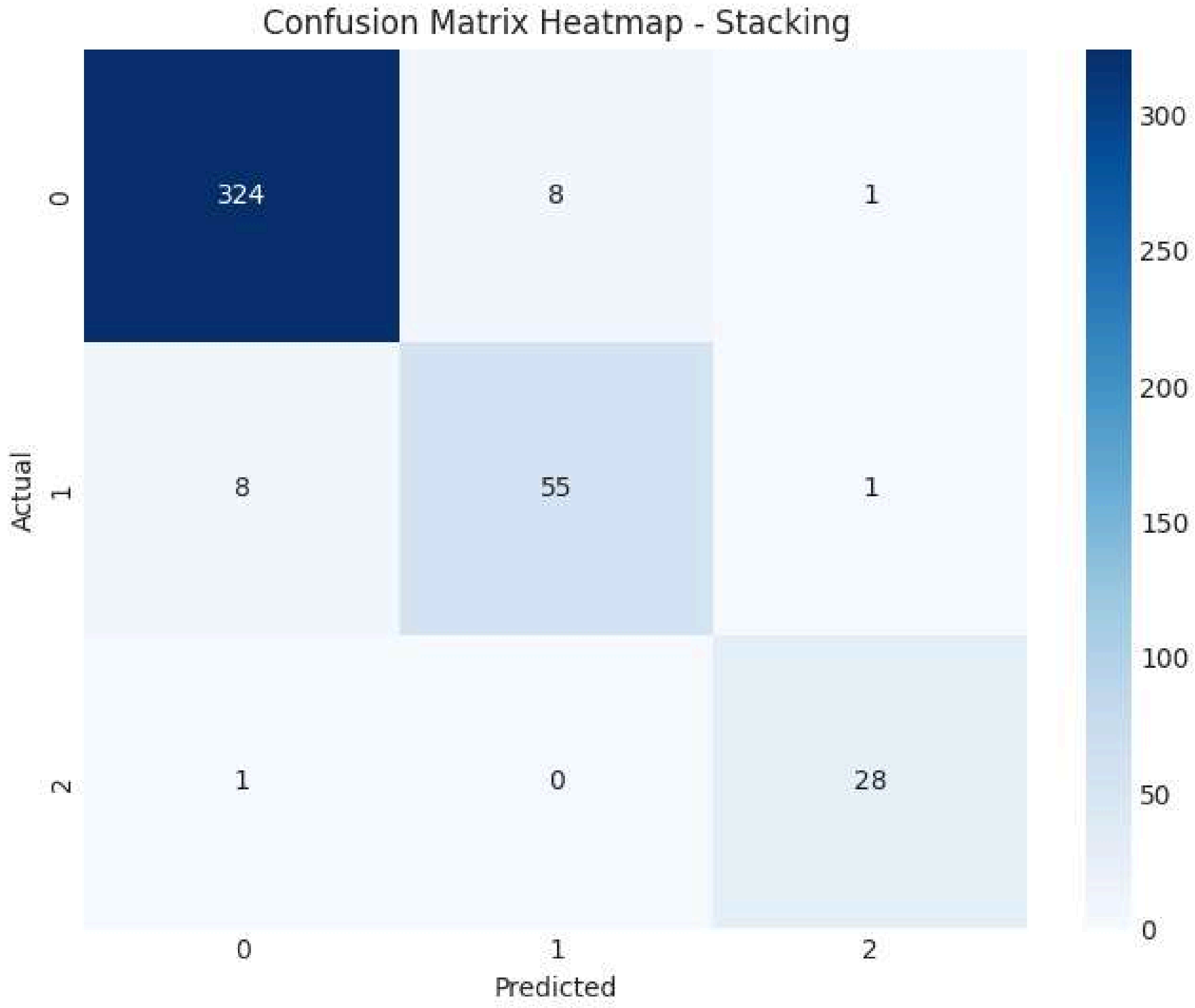
```
# Stacking Classifier with Cross-Validation
stacking_model = StackingClassifier(
    estimators=[('xgb', xgb_model), ('rf', rf_model)],
    final_estimator=LogisticRegression(),  # Meta-model
    cv=5  # Cross-validation folds in Stacking Classifier
)

# Fit Stacking Model
stacking_model.fit(X_train, y_train)

# Cross-Validation Scores
cv_scores = cross_val_score(stacking_model, X_train, y_train, cv=5, scoring='accuracy')

# Predictions on Test Set
stacking_y_pred = stacking_model.predict(X_test)

# Accuracy
stacking_accuracy = accuracy_score(y_test, stacking_y_pred)

# Classification Report & Confusion Matrix for Stacking
print("Stacking Classification Report")
print(classification_report(y_test, stacking_y_pred))
print("Stacking Confusion Matrix")
print(confusion_matrix(y_test, stacking_y_pred))

# Accuracy of Stacking
print(f"Stacking Accuracy: {stacking_accuracy:.4f}")

# Cross-Validation Results
print(f"Cross-Validation Accuracy Scores: {cv_scores}")
print(f"Mean Cross-Validation Accuracy: {cv_scores.mean():.4f}")

# Calculate MCC for the model's predictions
```

```
mcc_stacking = matthews_corrcoef(y_test,stacking_y_pred)

# Print MCC
print(f"Matthews Correlation Coefficient (MCC): {mcc_stacking :.4f}")
```

```
Stacking Classification Report
              precision    recall  f1-score   support

         0.0       0.97      0.97      0.97       333
         1.0       0.87      0.86      0.87        64
         2.0       0.93      0.97      0.95        29

    accuracy                           0.96       426
   macro avg       0.93      0.93      0.93       426
weighted avg       0.96      0.96      0.96       426

Stacking Confusion Matrix
[[324   8   1]
 [  8  55   1]
 [  1   0  28]]
Stacking Accuracy: 0.9554
Cross-Validation Accuracy Scores: [0.94705882 0.93823529 0.93529412 0.95294118 0.95588235]
Mean Cross-Validation Accuracy: 0.9459
Matthews Correlation Coefficient (MCC): 0.8768
```

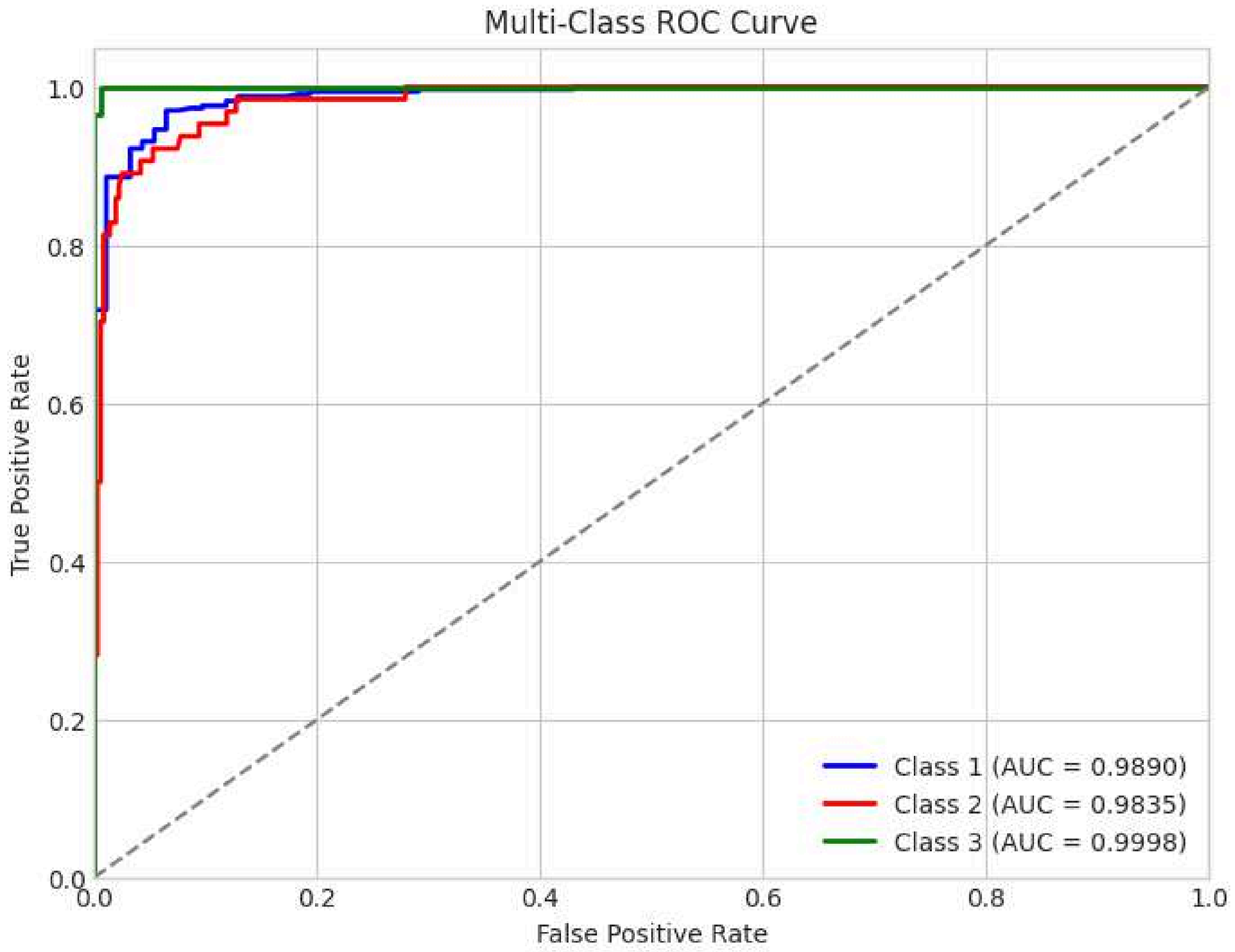
In [32]:

```
plot_confusion_matrix_heatmap(y_test, stacking_y_pred, 'Stacking')
```



In [33]:

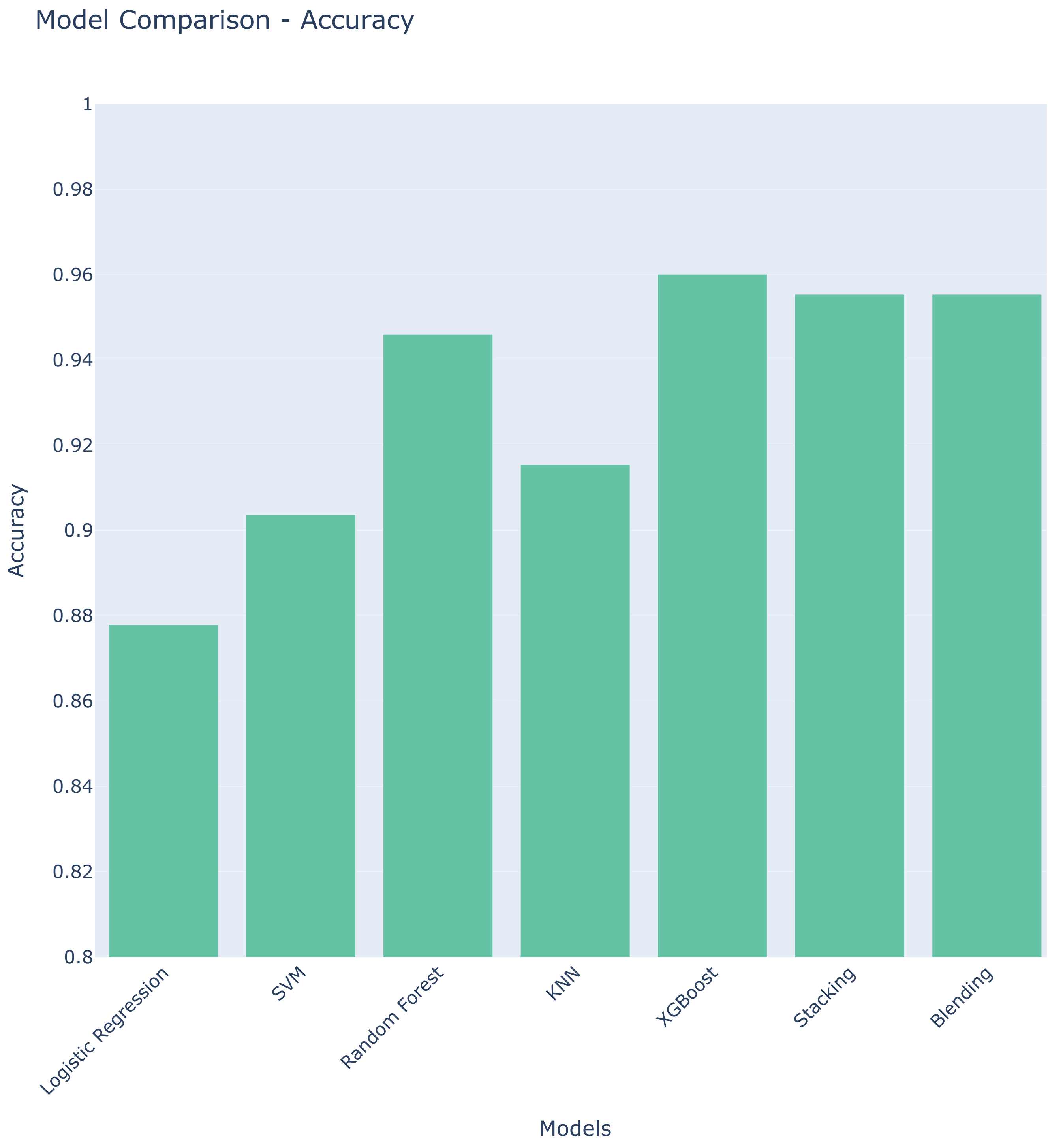
```
plot_multi_class_roc(stacking_model ,X_test, y_test, num_classes=3)
```



• Model Comparison - Accuracy

In [34]:

```
import plotly.graph_objs as go

# Data for the bar plot
models = ['Logistic Regression', 'SVM', 'Random Forest', 'KNN', 'XGBoost', 'Stacking', 'Blending']
accuracies = [log_accuracy, svm_accuracy, rf_accuracy, knn_accuracy, xgb_accuracy, stacking_accurac

# Create a bar plot
bar_fig = go.Figure()

# Add bar trace
bar_fig.add_trace(go.Bar(
    x=models,
    y=accuracies,
    marker_color='#66C2A5'
))

# Update layout with increased size
bar_fig.update_layout(
    title='Model Comparison - Accuracy',
    xaxis_title='Models',
    yaxis_title='Accuracy',
    yaxis=dict(range=[0.8, 1.0]),  # Set y-axis limits
    xaxis_tickangle=-45,  # Rotate x-axis labels for better readability
    width=800,  # Increase width
    height=800   # Increase height
)

# Show the figure
bar_fig.show()
```

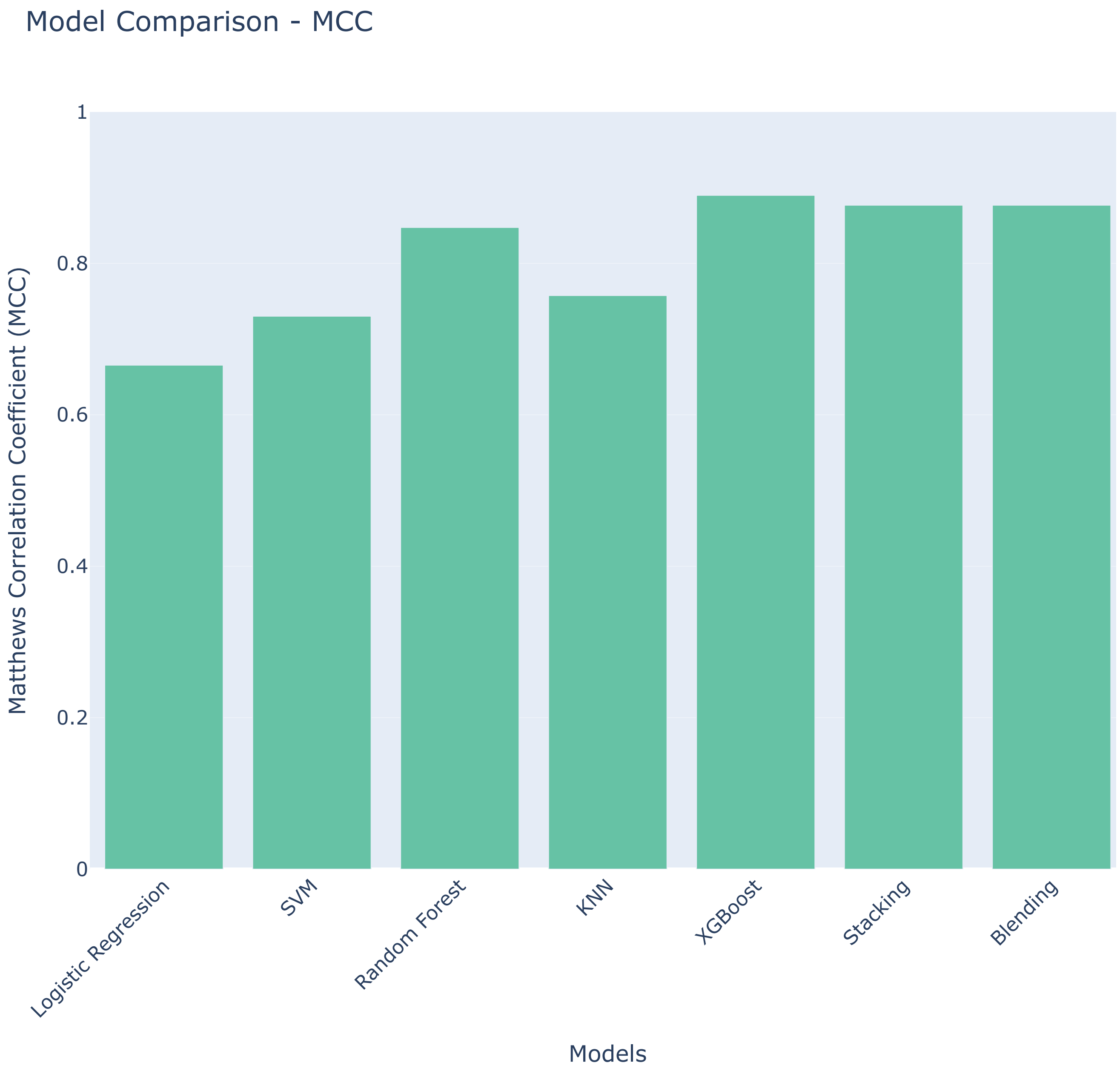


• Model Comparison - Matthews Correlation Coefficient (MCC)

In [35]:

```
# Model names and corresponding MCC values
models = ['Logistic Regression', 'SVM', 'Random Forest', 'KNN', 'XGBoost', 'Stacking', 'Blending']
mcc_values = [mcc_log, mcc_svm, mcc_rf, mcc_knn, mcc_XGb, mcc_stacking, mcc_voting]

# Create a bar plot
mcc_fig = go.Figure()

# Add bar trace
mcc_fig.add_trace(go.Bar(
    x=models,
    y=mcc_values,
    marker_color='#66C2A5'
))

# Update layout with increased size
```

```
mcc_fig.update_layout(
    title='Model Comparison - MCC',
    xaxis_title='Models',
    yaxis_title='Matthews Correlation Coefficient (MCC)',
    yaxis=dict(range=[0.0, 1.0]),  # Set y-axis limits
    xaxis_tickangle=-45,  # Rotate x-axis labels for better readability
    width=800,  # Increase width
    height=700   # Increase height
)

# Show the figure
mcc_fig.show()
```



In [36]:

```
d={'Models':models,'Accuracies':accuracies,'MCC':mcc_values}
Models_comparison=pd.DataFrame(d)


from IPython.display import display, HTML
# Sort the DataFrame by 'Accuracies' and reset the index
Models_comparison_sorted = Models_comparison.sort_values(by='Accuracies', ascending=False).reset_in
# Define the HTML table string with inline CSS styling
html_table = """
<table style="width:60%; margin:auto; border-collapse:collapse; font-family:Georgia, serif; font-si
  <thead>
    <tr style="background-color:#4A90E2; color:white; border:2px solid #66C2A5;">
      <th style="padding:12px;">Models</th>
      <th style="padding:12px;">Accuracies</th>
      <th style="padding:12px;">MCC</th>
    </tr>
  </thead>
  <tbody>
"""
# Add table rows dynamically from the DataFrame
for _, row in Models_comparison_sorted.iterrows():
    html_table += f"""
    <tr style="border-bottom:1px solid #ddd;">
      <td style="padding:12px; background-color:#E6F9E6;">{row['Models']}</td>
      <td style="padding:12px; background-color:#F0F4C3;">{row['Accuracies']:.6f}</td>
      <td style="padding:12px; background-color:#E6F9E6;">{row['MCC']:.6f}</td>
    </tr>
    """
# Close the HTML table
html_table += """
  </tbody>
</table>
"""
# Display the HTML table
display(HTML(html_table))
```

Models	Accuracies	MCC
XGBoost	0.960094	0.889860
Stacking	0.955399	0.876789
Blending	0.955399	0.876789
Random Forest	0.946009	0.847387
KNN	0.915493	0.757509
SVM	0.903756	0.730215
Logistic Regression	0.877934	0.665549

Conclusion

After training and evaluating various machine learning models for fetal health classification, the **XGBoost** model demonstrated superior performance. With an accuracy of **96.01%** and an MCC (Matthews Correlation Coefficient) of **0.8899**, XGBoost stands out as the most reliable option for predicting fetal health status based on CTG data.

Ensemble methods such as **Stacking** and **Blending** also performed well, achieving accuracies of **95.54%** with an MCC of **0.8768**. However, they fell slightly behind XGBoost. The **Random Forest** model achieved an accuracy of **94.60%** and MCC of **0.8474**, positioning it as a strong alternative.

While **K-Nearest Neighbors (KNN)** and **Support Vector Machine (SVM)** provided acceptable results with accuracies of **91.55%** and **90.38%**, their performance was significantly lower compared to the ensemble models. Lastly, **Logistic Regression** yielded the lowest performance with an accuracy of **87.79%** and MCC of **0.6655**.

In conclusion, the **XGBoost** model is the recommended choice for fetal health classification due to its high accuracy and balanced performance. Its use could assist in early diagnosis, helping healthcare professionals intervene before complications arise, ultimately contributing to reducing child and maternal mortality.

Thank You!

We truly appreciate your time and effort in reviewing this analysis. Your valuable feedback and suggestions are always welcomed! Thank you for being a part of this journey toward learning and growth. Together, we can achieve great things in data science!

Feel free to reach out with your comments and insights. Let\'s continue learning and building together!

Best regards,
Faiz Siddiqui