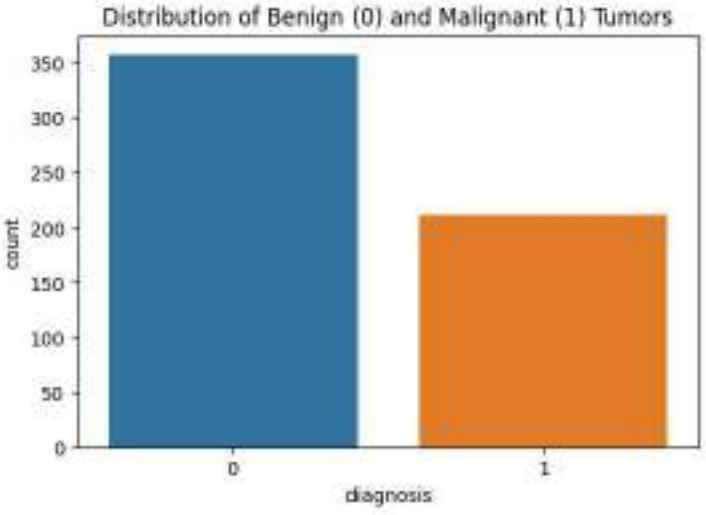
```
!pip install -q gradio scikit-learn pandas matplotlib seaborn
!pip install fpdf
```

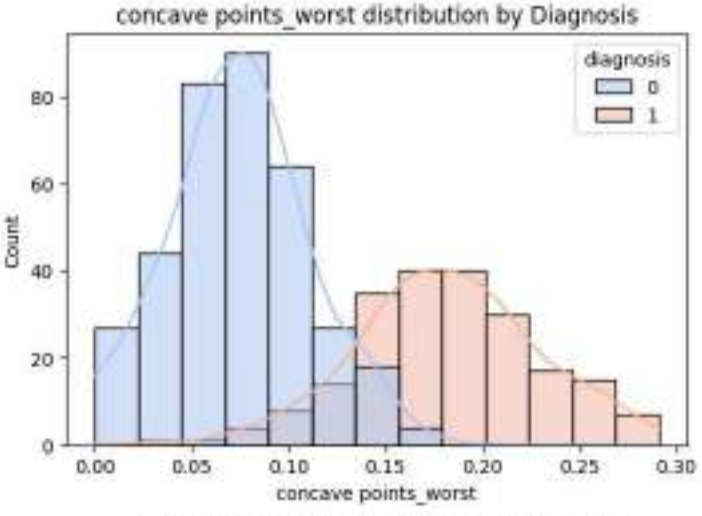
```
Collecting fpdf
  Downloading fpdf-1.7.2.tar.gz (39 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: fpdf
  Building wheel for fpdf (setup.py) ... done
  Created wheel for fpdf: filename=fpdf-1.7.2-py2.py3-none-any.whl size=40704 sha256=f4783add08808ba356ff1b1eaa0
  Stored in directory: /root/.cache/pip/wheels/65/4f/66/bbda9866da446a72e206d6484cd97381cbc7859a7068541c36
Successfully built fpdf
Installing collected packages: fpdf
Successfully installed fpdf-1.7.2
```

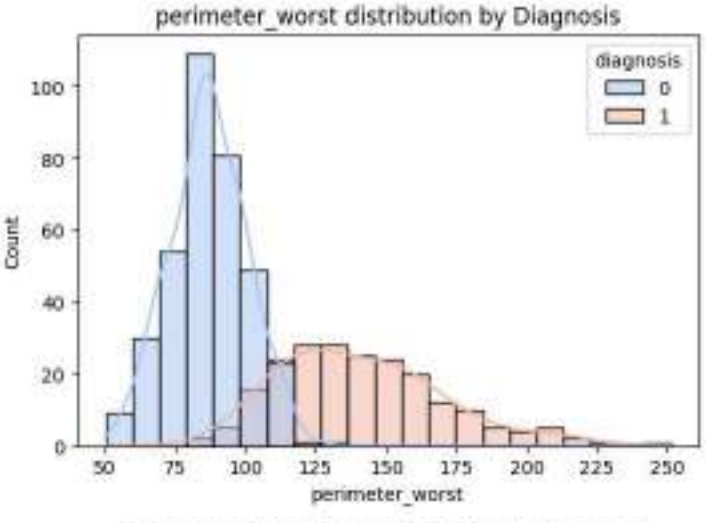
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm

# Load the dataset
url = "/content/data.csv"
df = pd.read_csv(url)

# Clean it
df.drop(columns=['id', 'Unnamed: 32'], inplace=True)
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

# Define X and y
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# Class distribution plot
plt.figure(figsize=(6,4))
sns.countplot(x='diagnosis', data=df, palette=['#1f77b4', '#ff7f0e'])
plt.title("Distribution of Benign (0) and Malignant (1) Tumors")
plt.show()

# Choose top 5 features based on correlation
correlation_matrix = df.corr()
top_5 = correlation_matrix['diagnosis'].abs().sort_values(ascending=False).index[1:6]  # Excluding 'diagnosis'

# Histograms of top 5 features
for feature in top_5:
    plt.figure(figsize=(6,4))
    sns.histplot(data=df, x=feature, hue='diagnosis', kde=True, palette='coolwarm')
    plt.title(f"{feature} distribution by Diagnosis")
    plt.show()

# Correlation Heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(df.corr(), cmap='RdBu_r', center=0, annot=False)
plt.title("Feature Correlation Heatmap")
plt.show()

# Choose a feature (e.g., Radius Mean) and plot the Gaussian distributions
feature = 'radius_mean'
x_vals = np.linspace(df[feature].min(), df[feature].max(), 200)

plt.figure(figsize=(8, 5))
for label, color in zip([0, 1], ['green', 'red']):
```

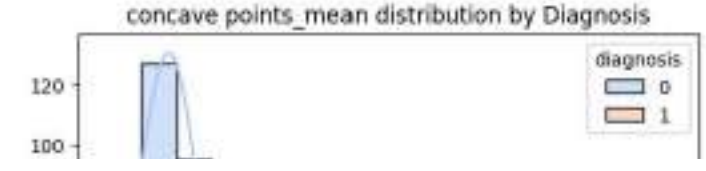
```
        mean = model.theta_[label][X.columns.get_loc(feature)]  # Mean for each class
        std = np.sqrt(model.var_[label][X.columns.get_loc(feature)])  # Standard deviation (square root of variance)
        plt.plot(x_vals, norm.pdf(x_vals, mean, std), label=f"{'Benign' if label==0 else 'Malignant'}", color=color)

    plt.title(f"Gaussian Distribution of {feature}")
    plt.xlabel(feature)
    plt.ylabel("Density")
    plt.legend()
    plt.show()
```

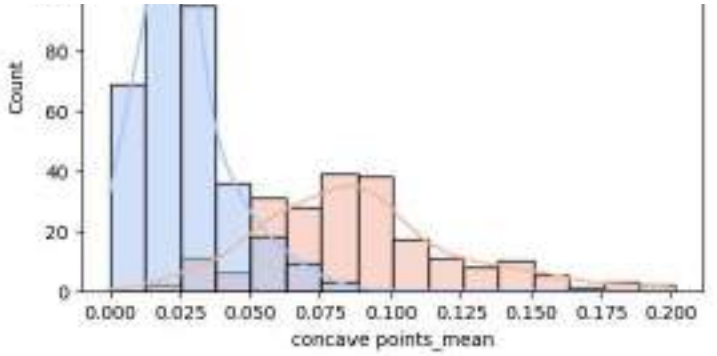
```
<ipython-input-15-efea917eebeb>:30: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variabl

  sns.countplot(x='diagnosis', data=df, palette=['#1f77b4', '#ff7f0e'])
```

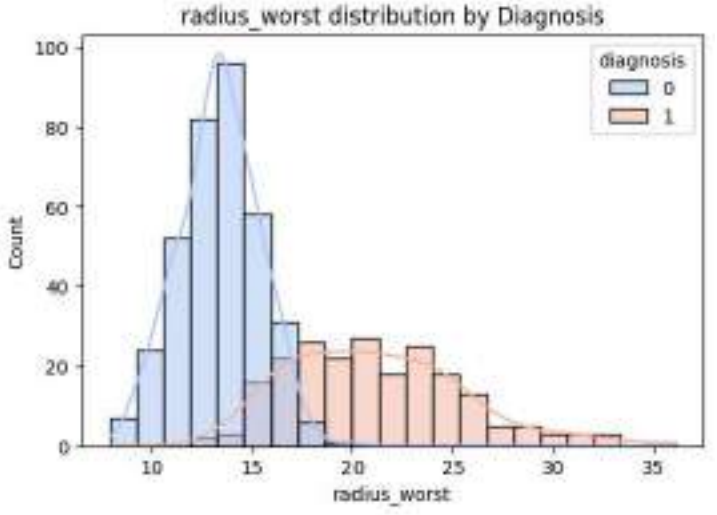


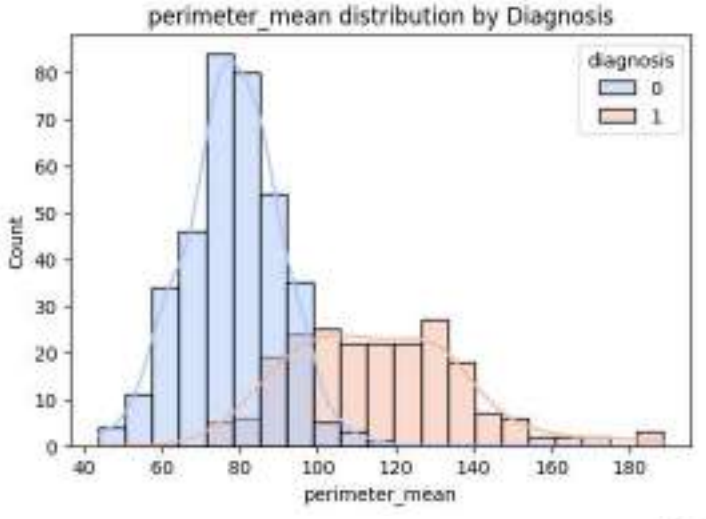


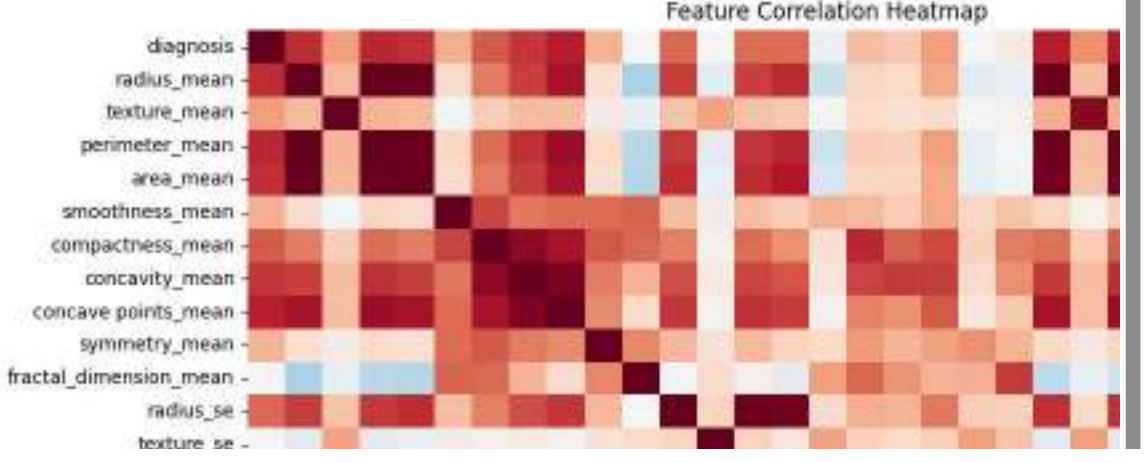


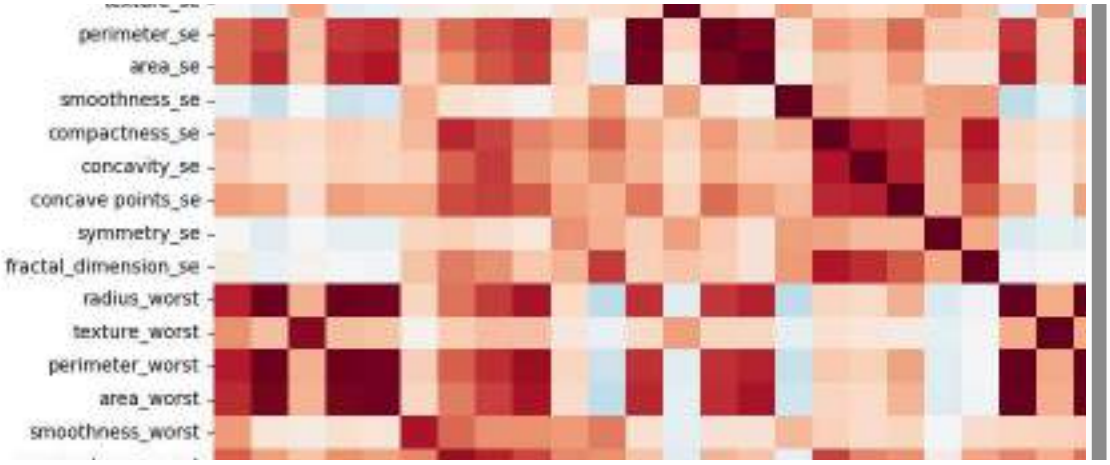



Count
concave points_mean


radius_worst distribution by Diagnosis
diagnosis
0
1
Count
radius_worst


perimeter_mean distribution by Diagnosis
diagnosis
0
1
Count
perimeter_mean


Feature Correlation Heatmap
diagnosis
radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
radius_se
texture_se


perimeter_se
area_se
smoothness_se
compactness_se
concavity_se
concave points_se
symmetry_se
fractal_dimension_se
radius_worst
texture_worst
perimeter_worst
area_worst
smoothness_worst

```
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
import os
import tempfile
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import gradio as gr

# Create temporary directory if it doesn't exist
temp_dir = tempfile.gettempdir()

# Load and clean data - use a more flexible path approach
try:
    # Try the original path first
    df = pd.read_csv("/content/data.csv")
except FileNotFoundError:
    # If that fails, look for the file in the current directory
    try:
        df = pd.read_csv("data.csv")
    except FileNotFoundError:
        # If we still can't find it, use sample data
        # This is a fallback with synthetic data similar to the Wisconsin breast cancer dataset
        np.random.seed(42)
        n_samples = 100

        # Generate synthetic data that resembles breast cancer features
        data = {
            'id': range(n_samples),
            'diagnosis': np.random.choice(['M', 'B'], size=n_samples, p=[0.4, 0.6]),
            'radius_mean': np.random.normal(14.5, 3.5, n_samples),
            'concavity_mean': np.random.normal(0.1, 0.1, n_samples),
            'perimeter_worst': np.random.normal(100, 30, n_samples),
            'area_mean': np.random.normal(600, 300, n_samples),
            'concave points_mean': np.random.normal(0.05, 0.04, n_samples),
        }

        # Create relationships between features and diagnosis to make prediction sensible
        for i in range(n_samples):
            if data['diagnosis'][i] == 'M':  # If malignant, increase the values
                data['radius_mean'][i] += 3
                data['concavity_mean'][i] += 0.1
                data['perimeter_worst'][i] += 20
                data['area_mean'][i] += 200
                data['concave points_mean'][i] += 0.05

        df = pd.DataFrame(data)
        print("Using synthetic dataset as the original file wasn't found.")

# Drop unnecessary columns if they exist
columns_to_drop = []
if 'id' in df.columns:
    columns_to_drop.append('id')
if 'Unnamed: 32' in df.columns:
    columns_to_drop.append('Unnamed: 32')

if columns_to_drop:
    df.drop(columns=columns_to_drop, inplace=True)

# Map diagnosis
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

# Use proper column names from your dataset
selected_features = ['radius_mean', 'concavity_mean', 'perimeter_worst', 'area_mean', 'concave points_mean']

# Verify all features exist in the dataframe
for feature in selected_features:
    if feature not in df.columns:
        raise ValueError(f"Feature '{feature}' not found in dataset. Available columns: {df.columns.tolist()}")

X = df[selected_features]
y = df['diagnosis']

# Train-test split for model evaluation
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Train model
model = GaussianNB()
model.fit(X_train, y_train)

# Evaluate model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, output_dict=True)

# Function to create feature importance visualization
def create_feature_importance_chart():
    # Calculate mean values for each feature by class
    feature_means = {}
    for feature in selected_features:
        feature_means[feature] = [
            df[df['diagnosis'] == 0][feature].mean(),  # Benign
            df[df['diagnosis'] == 1][feature].mean()   # Malignant
        ]

    # Create comparison chart
    plt.figure(figsize=(10, 6))
    x = np.arange(len(selected_features))
    width = 0.35

    plt.bar(x - width/2, [feature_means[f][0] for f in selected_features], width, label='Benign', color='green', alpha=0.7)
    plt.bar(x + width/2, [feature_means[f][1] for f in selected_features], width, label='Malignant', color='red', alpha=0.7)

    plt.xlabel('Features')
    plt.ylabel('Mean Value')
    plt.title('Feature Comparison: Benign vs Malignant')
    plt.xticks(x, [f.replace('_', ' ').title() for f in selected_features], rotation=45, ha='right')
    plt.legend()
    plt.tight_layout()

    # Save plot
    chart_path = os.path.join(temp_dir, "feature_importance.png")
    plt.savefig(chart_path)
```

```
    plt.close()

    return chart_path

# Create confusion matrix visualization
def create_confusion_matrix():
    plt.figure(figsize=(6, 5))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Benign', 'Malignant'],
                yticklabels=['Benign', 'Malignant'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix (Accuracy: {accuracy:.2f})')
    plt.tight_layout()

    # Save plot
    cm_path = os.path.join(temp_dir, "confusion_matrix.png")
    plt.savefig(cm_path)
    plt.close()

    return cm_path

# Generate model summary
model_summary = f"""
## Model Performance

- **Accuracy**: {accuracy:.2f}
- **Precision (Malignant)**: {class_report['1']['precision']:.2f}
- **Recall (Malignant)**: {class_report['1']['recall']:.2f}
- **F1-Score (Malignant)**: {class_report['1']['f1-score']:.2f}

The model was trained on {len(X_train)} samples and tested on {len(X_test)} samples.
"""

# Feature descriptions for educational purposes
feature_descriptions = {
    'radius_mean': 'Average distance from center to points on the perimeter',
    'concavity_mean': 'Severity of concave portions of the contour',
    'perimeter_worst': 'Largest perimeter measurement of the mass',
    'area_mean': 'Average area of the mass',
    'concave points_mean': 'Average number of concave portions of the contour'
}

# Create feature descriptions markdown
feature_desc_md = "## Feature Descriptions\n\n"
for feature, description in feature_descriptions.items():
    feature_desc_md += f"- **{feature.replace('_', ' ').title()}**: {description}\n"

# Predict + plot logic
def predict_with_recommendation(radius, concavity, perimeter, area, concave_pts):
    input_data = np.array([[radius, concavity, perimeter, area, concave_pts]])
    pred = model.predict(input_data)[0]
    probs = model.predict_proba(input_data)[0]
    prob_malignant = probs[1]

    # Recommendation text
    if prob_malignant > 0.85:
        status = "🔴 Malignant"
        advice = "⚠️ Immediate consultation with an oncologist is recommended."
    elif prob_malignant > 0.5:
        status = "🟠 Borderline Malignant"
        advice = "🗓️ Follow-up screening and possible biopsy recommended."
    else:
        status = "🟢 Benign"
        advice = "✅ No malignancy detected. Continue routine checkups."

    report_text = f"""## Diagnosis Results

**Prediction**: {status}
**Probability of Malignancy**: {prob_malignant:.2f} ({int(prob_malignant*100)}%)
**Recommendation**: {advice}

### Value Analysis

"""

    # Add feature analysis
    for i, feature in enumerate(selected_features):
        feature_val = input_data[0][i]
        benign_mean = df[df['diagnosis'] == 0][feature].mean()
        malignant_mean = df[df['diagnosis'] == 1][feature].mean()
        feature_name = feature.replace('_', ' ').title()

        if abs(feature_val - malignant_mean) < abs(feature_val - benign_mean):
            report_text += f"- **{feature_name}**: {feature_val:.2f} (Closer to typical malignant value)\n"
        else:
            report_text += f"- **{feature_name}**: {feature_val:.2f} (Closer to typical benign value)\n"

    # Plot - probability visualization
    plt.figure(figsize=(8, 4))

    # Plot probability bars
    plt.subplot(1, 2, 1)
    plt.barh(['Benign', 'Malignant'], [1 - prob_malignant, prob_malignant], color=['green', 'red'])
    plt.xlim(0, 1)
    plt.title("Malignancy Probability")

    # Plot gauge-style visualization
    plt.subplot(1, 2, 2)
    theta = np.linspace(0, np.pi, 100)
    r = 1

    # Create semicircle
    x = r * np.cos(theta)
    y = r * np.sin(theta)

    # Create the gauge
    plt.plot(x, y, 'k-')
    plt.fill_between(x, 0, y, color='lightgray', alpha=0.3)

    # Add risk zones
    plt.fill_between(x[0:33], 0, y[0:33], color='green', alpha=0.5)
    plt.fill_between(x[33:66], 0, y[33:66], color='orange', alpha=0.5)
    plt.fill_between(x[66:], 0, y[66:], color='red', alpha=0.5)

    # Add needle
    needle_angle = prob_malignant * np.pi
    needle_x = r * np.cos(needle_angle)
```

```
    needle_y = r * np.sin(needle_angle)
    plt.plot([0, needle_x], [0, needle_y], 'k-', linewidth=2)
    plt.plot(0, 0, 'ko', markersize=8)

    # Customize gauge appearance
    plt.axis('equal')
    plt.title("Risk Gauge")
    plt.text(-0.8, -0.2, "Low", fontsize=8)
    plt.text(0, -0.2, "Med", fontsize=8)
    plt.text(0.7, -0.2, "High", fontsize=8)
    plt.xticks([])
    plt.yticks([])

    plt.tight_layout()
    img_path = os.path.join(temp_dir, "prediction_chart.png")
    plt.savefig(img_path)
    plt.close()

    return report_text, img_path

# Information tabs for education
model_info_tab = gr.Markdown("""
# About This Tool

This application uses a Naive Bayes machine learning model to predict the likelihood of a breast tumor being malignant based on measurements from fine needle aspirates (FNA).

## How It Works

The model was trained on the Wisconsin Breast Cancer dataset which contains digitized measurements of breast mass FNAs. The algorithm analyzes patterns in the data to classify tumors as either malignant or benign.

## Disclaimer

This tool is for educational purposes only and should not replace professional medical diagnosis. Always consult with healthcare professionals for medical advice and proper diagnosis.
""")

feature_info_tab = gr.Markdown(feature_desc_md)

# Prepare model evaluation visuals
confusion_matrix_img = create_confusion_matrix()
feature_importance_img = create_feature_importance_chart()

model_eval_tab = gr.Markdown(model_summary)

# Gradio App with tabs
with gr.Blocks(theme=gr.themes.Soft()) as demo:
    gr.Markdown("# 🔬 Breast Cancer Diagnosis Tool")
    gr.Markdown("### Naive Bayes-powered early breast cancer screening assistant")

    with gr.Tabs():
        with gr.TabItem("Diagnosis Tool"):
            with gr.Row():
                with gr.Column(scale=1):
                    gr.Markdown("### Input Tumor Measurements")
                    radius = gr.Slider(5, 30, label="Radius Mean", value=15)
                    concavity = gr.Slider(0.0, 0.5, label="Concavity Mean", value=0.1)
                    perimeter = gr.Slider(50, 250, label="Perimeter Worst", value=100)
                    area = gr.Slider(100, 2500, label="Area Mean", value=600)
                    concave_pts = gr.Slider(0.0, 0.4, label="Concave Points Mean", value=0.05)
                    submit_btn = gr.Button("Analyze", variant="primary")

                with gr.Column(scale=2):
                    gr.Markdown("### Analysis Results")
                    output_text = gr.Markdown()
                    output_chart = gr.Image(label="Probability Analysis")

        with gr.TabItem("Model Performance"):
            with gr.Row():
                with gr.Column():
                    gr.Markdown(model_summary)
                    gr.Image(confusion_matrix_img, label="Confusion Matrix")
                with gr.Column():
                    gr.Markdown("### Feature Importance")
                    gr.Image(feature_importance_img, label="Feature Comparison between Classes")

        with gr.TabItem("Education"):
            with gr.Tabs():
                with gr.TabItem("About Features"):
                    feature_info_tab
                with gr.TabItem("About The Tool"):
                    model_info_tab
                with gr.TabItem("When To Seek Help"):
                    gr.Markdown("""
                    # When to Seek Medical Help

                    ## Warning Signs of Breast Cancer

                    - A new lump or mass in the breast
                    - Thickening or swelling of part of the breast
                    - Irritation or dimpling of breast skin
                    - Redness or flaky skin in the nipple area or the breast
                    - Pulling in of the nipple or pain in the nipple area
                    - Nipple discharge other than breast milk
                    - Any change in the size or the shape of the breast
                    - Pain in any area of the breast

                    ## Remember

                    Early detection is key in breast cancer treatment. Regular self-examinations and screenings as recommended by your healthcare provider are essential.

                    This tool is for educational purposes only and does not replace professional medical advice.
                    """)

    # Connect the button to the prediction function
    submit_btn.click(
        fn=predict_with_recommendation,
        inputs=[radius, concavity, perimeter, area, concave_pts],
        outputs=[output_text, output_chart]
    )

if __name__ == "__main__":
    demo.launch(share=True)
```

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://baecdff42763c2f063.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co/spaces)