Jupyter **Breast_Cancer_Classification** Last Checkpoint: a minute ago (unsaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

📌 Step 1: Import Required Libraries

In [1]:
```
# ✅ Data handling libraries
import numpy as np
import pandas as pd

# ✅ Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

# ✅ Machine Learning libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.tree import DecisionTree
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# ✅ Model Evaluation libraries
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score

# ✅ Utility libraries
import os, sys, pickle, joblib
import os.path, shutil, tempfile # For temp files, directory management
import time, datetime # For timing operations
import io # Handle in-memory file operations
from IPython.display import display, clear_output # Display dataframes, widgets, and text inputs in Jupyter/Colab
from IPython.display import display # Alternate displaying widgets, images, HTML, and other rich content in Jupyter/Colab
from google.colab import files # Enable file upload and download functionalities in Google Colab

import warnings
warnings.filterwarnings("ignore") # Ignore unnecessary warning messages

sns.set_style("whitegrid") # Enhances readability of plots with a clean grid background
```

In [2]:
```
!pip install gradio
import gradio as gr # Create web-based interfaces for ML models
```

```
Collecting gradio
  Downloading gradio-5.29.0-py3-none-any.whl.metadata (16 KB)
Collecting aiofiles<24.0,>=22.0 (from gradio)
  Downloading aiofiles-23.2.1-py3-none-any.whl (15 KB)
Requirement already satisfied: anyio<4.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.7.1)
Collecting fastapi<0.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 KB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.6.0-py3-none-any.whl.metadata (3.8 KB)
Collecting gradio-client==1.9.0 (from gradio)
  Downloading gradio_client-1.9.0-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: httpcore~=1.0.5 in /usr/local/lib/python3.11/dist-packages (from gradio) (1.0.5)
Requirement already satisfied: huggingface-hub~=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: jinja2<3.1.5,>=2.11.4 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.4)
Collecting markupsafe~=2.0 (from gradio)
  Downloading MarkupSafe-2.1.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.0 kB)
Requirement already satisfied: numpy~=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (1.26.4)
```

📌 Step 2: Load the Dataset

Mount Google Drive to access the dataset

In [3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Define dataset path

In [4]:
```
dataset_path='/content/drive/MyDrive/Breast Cancer Classification/Wisconsin Train Dataset/Train Dataset.csv'
```

Load the Wisconsin Breast Cancer Dataset

In [5]:
```
data=pd.read_csv(dataset_path) # Read the original dataframe which will remain the same without removing the id col.
```

📌 Step 3: Exploratory Data Analysis (EDA)

Get Insights About the Dataset

Shows column names, data types, and missing values

In [6]:
```
# ✅ Pro tip: Create a summary of dataset
def check_dataset(df):
    print("📌 Dataset Overview:\n")
    print(f"{'Column Name':<20} {'Data Type':<15} {'Non-Null Count':<15} {'Unique Values'}")
    print("-"*70)
    for col in df.columns:
        dtype = str(df[col].dtype)
        non_null = df[col].notnull().sum()
        unique = df[col].nunique()
        print(f"{col:<20} {dtype:<15} {non_null:<15} {unique}")

# Display the data
print("\n🔍 First 5 rows:\n")
print(data[dataset_path].head(), sep="\n")

## OR Use print(data.info())

# OR Use pandas_profiling
print("\n📊 Generating detailed report...")
profile = ydata_profiling.ProfileReport(df, minimal=True)
profile.to_file("dataset_report.html")
print("✅ Report saved as dataset_report.html")
```

```
📌 Dataset Overview:

Column Name          Data Type       Non-Null Count  Unique Values
----------------------------------------------------------------------
id                   int64           569             569
diagnosis            object          569             2
radius_mean          float64         569             456
texture_mean         float64         569             479
perimeter_mean       float64         569             522
area_mean            float64         569             539
smoothness_mean      float64         569             474
compactness_mean     float64         569             537
concavity_mean       float64         569             537
concave points_mean  float64         569             542
symmetry_mean        float64         569             432
fractal_dimension_mean float64       569             499
radius_se            float64         569             540
texture_se           float64         569             519
perimeter_se         float64         569             533
area_se              float64         569             528
smoothness_se        float64         569             547
compactness_se       float64         569             541
concavity_se         float64         569             533
concave points_se    float64         569             507
symmetry_se          float64         569             498
fractal_dimension_se float64         569             459
radius_worst         float64         569             457
texture_worst        float64         569             511
perimeter_worst      float64         569             514
area_worst           float64         569             544
smoothness_worst     float64         569             411
compactness_worst    float64         569             529
concavity_worst      float64         569             539
concave points_worst float64         569             492
symmetry_worst       float64         569             500
fractal_dimension_worst float64      569             535

🔍 First 5 rows of the dataset:

\n\n\n
```

🔍 **Check the unique values in the 'diagnosis' column**

In [16]:
```
# This helps to identify the different categories in the target variable
print("Unique values in 'diagnosis' column:", cancerdata.diagnosis.unique())
```

```
Unique values in 'diagnosis' column: ['M' 'B']
```

📊 **Display feature names**

In [34]:
```
print("📊 Feature Names in Dataset:")
print(list(cancerdata.columns)) #Printed vertically, making it easier to read

# OR print(cancerdata.columns) OUTPUT Index(['name', 'age', 'factory', 'Department',
```

```
📊 Feature Names in Dataset:
['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', 'compactness_worst', 'concavity_worst', 'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst']
```

In [17]:
```
# benign_count = cancerdata['diagnosis'].value_counts()['B']
# malignant_count = cancerdata['diagnosis'].value_counts()['M']

print("Number of Benign (B) cases: {benign_count}")
print("Number of Malignant (M) cases: {malignant_count}")
```

```
Number of Benign (B) cases: 357
Number of Malignant (M) cases: 212
```

🥧 **Pie chart for Benign vs. Malignant**

In [18]:
```
# Define labels and colors
labels = ['Benign', 'Malignant']
colors = ['#66b3ff', '#ff9999'] # Blue for Benign, Red for Malignant
sizes = [benign_count, malignant_count] # Values for pie chart

# Create pie chart
plt.pie(sizes, labels=labels, colors=colors,
        autopct='%1.1f%%',
        startangle=140, explode=(0, 0.1)) # Explode Malignant slice
# autopct='%1.1f%%' shows percentage values inside slices, 1f: 1 decimal place (e.g., 25.3), 1f0 = Displays the % symbol.
# startangle=140 rotates the chart for a specific angle, helps adjust slice orientation for better readability.
plt.title('Diagnosis Distribution')
plt.show()
```

📊 **Countplot (Target Class Distribution)**

NOTE : Seaborn's countplot automatically handles categorical variables, even if they are in string format ('M' and 'B').

In [19]:
```
# sns.countplot(x='diagnosis', data=cancerdata, palette=['#FF9999', '#66B3FF'])
plt.title('Diagnosis Count')
plt.xlabel('Diagnosis (M: Malignant, B: Benign)')
plt.ylabel('Count')
plt.show()
```

📏 **Count number of features & traget**

In [20]:
```
num_features = data.shape[1] - 1 # Excluding target column, .shape[1] count the number of columns
target_column = 'diagnosis'
total_patients = data.shape[0] # .shape[0] count the number of rows
```

In [21]:
```
# Display dataset details
print("📊 Dataset Information:")
print("🔹 Number of Features: {num_features}")
print("🎯 Target Column: {target_column}")
print("🧮 Total Number of Patients: {total_patients}")
print("🟢 Total Benign (B) Patients: {benign_count}")
print("🔴 Total Malignant (M) Patients: {malignant_count}")
```

```
📊 Dataset Information:
🔹 Number of Features: 31
🎯 Target Column: diagnosis
🧮 Total Number of Patients: 468
🟢 Total Benign (B) Patients: 289
🔴 Total Malignant (M) Patients: 179
```

🔤 **Label Encoding**

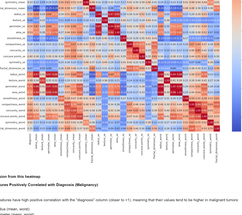
In [22]:
```
# cancerdata['diagnosis'] = cancerdata['diagnosis'].map({'M': 1, 'B': 0})
# print("✅ Diagnosis converted to numerical values.")
print("🔍 Unique values in 'diagnosis' after conversion:", cancerdata["diagnosis"].unique())

# print("🔍 Unique values in 'diagnosis' after conversion: [1 0]")
```

🔥 **Correlation Heatmap**

- Red (1.0) = Highly correlated
- Blue (-1.0) = Strong negative correlation
- White (0) = No correlation

In [23]:
```
# To analyze correlations between features and the target
plt.figure(figsize=(14, 10))
sns.heatmap(cancerdata.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap of Features')
plt.show()
```



Conclusion from this heatmap

- **Features Positively Correlated with Diagnosis (Malignancy):**

These features have high positive correlation with the "diagnosis" column (closer to +1), meaning that their values tend to be higher in malignant tumors:

- Radius (mean, worst)
- Perimeter (mean, worst)
- Area (mean, worst)
- Concavity (mean, worst)
- Concave Points (mean, worst)
- Compactness (mean, worst)

✅ **Conclusion:**

Higher values of these features are associated with **malignant tumors**. Larger tumor size and irregular shapes indicate malignancy.

- **Features Negatively Correlated with Diagnosis (Malignancy):**

These features have high negative correlation with the "diagnosis" column (closer to -1), meaning that their values tend to be lower in benign tumors:

- Smoothness (mean, worst)
- Symmetry (mean, worst)
- Fractal Dimension (mean, worst)

✅ **Conclusion:** Higher values of these features are associated with benign tumors. Benign tumors tend to have smoother, more symmetrical structures with less complexity.

📌 Step 4: Data Preprocessing & Data Splitting

1) Define features (X) and target variable (y)

In [24]:
```
X = cancerdata.drop(columns=['diagnosis'])
y = cancerdata['diagnosis']
```

2) ✂️ Split into training and test sets (80% training, 20% testing)

In [25]:
```
# Splitting data
X_train, X_test, y_train, y_test = train_test_split(
    cancerdata.drop(columns=['diagnosis']),
    cancerdata['diagnosis'],
    test_size=0.2,
    random_state=42)

print("Shape of training set:", X_train.shape)
print("Shape of test set:", X_test.shape)
```

```
Shape of training set: (374, 30)
Shape of test set: (94, 30)
```

3) ⚖️ Apply Z-score Standardization

💡 If you don't save the scaler, your trained model is unusable because it can't scale new data properly.

⚠️ Always save the scaler along with your trained model to ensure the same preprocessing is applied consistently.

In [26]:
```
# Define the scaler path
scaler_path = "/content/drive/MyDrive/Breast Cancer Classification/scaler.pkl"

# Initialize and fit the scaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train) # Fit and transform training data

# Save the scaler to the specified path
joblib.dump(sc, scaler_path)

# Check if the file was saved successfully
if os.path.exists(scaler_path):
    print("✅ Scaler saved successfully at: {scaler_path}")
else:
    print("❌ Error: Scaler could not be saved at {scaler_path}.")

# Transform the test data using the same scaler (to prevent data leakage)
X_test = sc.transform(X_test)

✅ Scaler saved successfully at: /content/drive/MyDrive/Breast Cancer Classification/scaler.pkl
```

📌 Step 5: Train Machine Learning Models

In [27]:
```
# Store results in an empty dictionary and update it dynamically after model training for saving the model accuracies for comparis
results = {}
```

1️⃣ Logistic Regression

1) Train the Logistic Regression model

In [28]:
```
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
```

Out[28]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

2) Predictions & Evaluation

In [29]:
```
y_pred_lr = lr_model.predict(X_test)
y_prob_lr = lr_model.predict_proba(X_test)[:, 1]
```

3) Metrics

NOTE : AUC-ROC Score

- AUC-ROC (Area Under the Receiver Operating Characteristic Curve) is a metric used to evaluate classification models.
- It measures how well the model distinguishes between classes (e.g., benign vs. malignant).
- AUC-ROC ranges from 0 to 1, where:
  - 1.0 = Perfect classifier
  - 0.5 = Random guess
  - 0.0 = Worst classifier

In [30]:
```
print("📊 Logistic Regression Performance:")

# Training Accuracy
y_train_pred_lr = lr_model.predict(X_train)
train_acc_lr = accuracy_score(y_train, y_train_pred_lr)
print("🔹 Training Accuracy: {train_acc_lr:.4f}")

# Test Accuracy
y_test_pred = lr_model.predict(X_test)
test_acc_lr = accuracy_score(y_test, y_pred_lr)
print("🎯 Test Accuracy: {test_acc_lr:.4f}")

# ROC-AUC Score
roc_auc = roc_auc_score(y_test, y_prob_lr)
print("📈 ROC-AUC Score: {roc_auc:.4f}")

# Classification Report
print(classification_report(y_test, y_pred_lr))

# Save Results
models_results["Logistic Regression"] = {train_acc_lr, test_acc_lr, roc_auc}
```

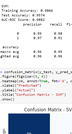
```
📊 Logistic Regression Performance:
🔹 Training Accuracy: 0.9893
🎯 Test Accuracy: 0.9574
📈 ROC-AUC Score: 0.9962
              precision    recall  f1-score   support

           0       0.95      0.98      0.97        68
           1       0.97      0.93      0.95        26

    accuracy                           0.96        94
   macro avg       0.96      0.95      0.96        94
weighted avg       0.96      0.96      0.96        94
```

4) Confusion Matrix

In [31]:
```
cm = confusion_matrix(y_test, y_pred_lr)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])
plt.title('Confusion Matrix - Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



2️⃣ SUPPORT VECTOR MACHINE (SVM)

In [32]:
```
model_svm = SVC(probability=True)
model_svm.fit(X_train, y_train)
```

Out[32]:
```
SVC(probability=True)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [33]:
```
# Predictions
y_train_pred_svm = model_svm.predict(X_train)
y_pred_svm = model_svm.predict(X_test)
y_prob_svm = model_svm.predict_proba(X_test)[:, 1]
```

In [34]:
```
# Training & Test Accuracy
train_acc_svm = accuracy_score(y_train, y_train_pred_svm)
test_acc_svm = accuracy_score(y_test, y_pred_svm)
```

In [35]:
```
# Evaluation Metrics
print("📊 SVM Results:")
roc_auc = roc_auc_score(y_test, y_prob_svm)
print("🔹 Training Accuracy: {train_acc_svm:.4f}")
print("🎯 Test Accuracy: {test_acc_svm:.4f}")
print("📈 ROC-AUC Score: {roc_auc:.4f}")
print(classification_report(y_test, y_pred_svm))

# Save Results
models_results["SVM"] = {train_acc_svm, test_acc_svm, roc_auc}
```

```
📊 SVM Results:
🔹 Training Accuracy: 0.9866
🎯 Test Accuracy: 0.9574
📈 ROC-AUC Score: 0.9982
              precision    recall  f1-score   support

           0       0.95      0.98      0.97        68
           1       0.97      0.92      0.94        26

    accuracy                           0.96        94
   macro avg       0.96      0.95      0.96        94
weighted avg       0.96      0.96      0.96        94
```

In [36]:
```
cm = confusion_matrix(y_test, y_pred_svm)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])
plt.title('Confusion Matrix - SVM')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

5️⃣ K-NEAREST NEIGHBORS (KNN)

```
In [37]: # model_knn = ClassifierClassifier(n_neighbors=5)
         model_knn.fit(X_train, y_train)
```

```
Out[37]: ClassifierClassifier()
```

In a jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [38]: # Predictions
         y_train_pred_knn = model_knn.predict(X_train)
         y_pred_knn = model_knn.predict(X_test)
```

```
In [39]: # Training & Test Accuracy
         train_acc_knn = accuracy_score(y_train, y_train_pred_knn)
         test_acc_knn = accuracy_score(y_test, y_pred_knn)
```

```
In [40]: print("\n📊 KNN Performance:")
         print(f"Training Accuracy: {train_acc_knn:.4f}")
         print(f"Test Accuracy: {test_acc_knn:.4f}")
         print("\n📋 Classification Report:")
         print(f"\n{classification_report(y_test, y_pred_knn)}")

         # 🔥 Bonus: ROC-AUC Score
         y_test_proba = model_knn.predict_proba(X_test)[:, 1]
         roc_auc = roc_auc_score(y_test, y_test_proba)
         print(f"\n🔥 ROC-AUC Score: {roc_auc:.4f}")
         print(f"\n🎯 Sample probabilities: {y_test_proba[:5]}")
```

```
📊 KNN Performance:
Training Accuracy: 0.9723
Test Accuracy: 0.9574

📋 Classification Report:
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        60
           1       1.00      0.88      0.94        34

    accuracy                           0.96        94
   macro avg       0.97      0.94      0.95        94
weighted avg       0.96      0.96      0.96        94

🔥 ROC-AUC Score: 0.9941

🎯 Sample probabilities: [0.2 0.2 0.2 0.2 0.2]
```

Class 1 Recall (0.88) is lower than Precision (1.00) — This suggests that while all predicted positives are correct, some actual positives are missed (model is slightly conservative in predicting Class 1).

```
In [41]: # cm = confusion_matrix(y_test, y_pred_knn)
         cm = np.array([[60, 0], [4, 30]])
         sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
         plt.title('Confusion Matrix - KNN')
         plt.show()
```

📌 Step 6: Compare Model Accuracies

```
In [42]: # Best_model_name = max(model_results, key=lambda k: model_results[k][2])  # Select model with highest test accuracy
         best_model = model[f"{best_model_name.lower().replace(' ', '_')}"]  # Retrieve model object dynamically

         # Save the best model
         joblib.dump(best_model, "best_model.pkl")
         print(f"💾 Best Model Saved: {best_model_name} with Test Accuracy: {model_results[best_model_name][2]:.4f}")

         # ═══════════════ COMPARISON TABLE ═══════════════
         comparison_df = pd.DataFrame(model_results, index=['Training Accuracy', 'Test Accuracy', 'ROC-AUC'].T
         print("\n📊 Model Comparison Table:")
         print(comparison_df)
```

```
💾 Best Model Saved: Soft with Test Accuracy: 0.9574

📊 Model Comparison Table:
                   Training Accuracy  Test Accuracy  ROC-AUC
Logistic Regression           0.9648         0.9681   0.9954
SVM                           0.9841         0.9787   0.9922
KNN                           0.9723         0.9574   0.9941%
```

📌 Step 7: Saving the best model

```
In [43]: # Save the trained model
         if os.path.exists(colab_path):
             joblib.dump(best_model, model_path)

         # Verify if the model is saved
         if os.path.exists(model_path):
             print("✅ Model saved successfully at: {model_path}")
         else:
             print("❌ Model saving failed!")
```

```
✅ Model saved successfully at: /content/drive/MyDrive/Breast Cancer Classification/svm_model.pkl
```

```
In [44]: # Load the Scaler
         if os.path.exists(scaler_path):
             scaler = joblib.load(scaler_path)
             print("📏 Scaler loaded successfully from: {scaler_path}")
         else:
             print("❌ Error: Scaler file not found at {scaler_path}. Exiting...")
             exit()

         # Upload CSV Button
         upload_button = widgets.FileUpload(accept='.csv', multiple=False)
         display(upload_button)

         # Function to process uploaded CSV & predict
         def process_and_predict(change):
             global uploaded_file
             uploaded_file = list(upload_button.value.values())[0]  # Get uploaded file
             file_content = io.BytesIO(uploaded_file['content'])  # Read file content

             try:
                 # Load CSV into DataFrame
                 new_data = pd.read_csv(file_content)
             except Exception as e:
                 print("❌ Error loading CSV: {e}")
                 return

             # Remove 'ID' column if present
             if 'id' in new_data.columns:
                 new_data = new_data.drop(columns=['id'])

             # Extract Expected Column Directly from the Uploaded CSV
             feature_columns = new_data.columns.tolist()

             # Convert data to numeric (handle string errors)
             for col in new_data.columns:
                 new_data[col] = pd.to_numeric(new_data[col], errors='coerce')  # Convert to numeric
             new_data = new_data.fillna(0)  # Replace missing values with 0

             # Apply StandardScaler
             new_data_scaled = scaler.transform(new_data)

             # Make predictions using SVM model
             predictions = model_svm.predict(new_data_scaled)

             # Save predictions
             new_data['Predicted_Class'] = predictions
             output_file = "/content/svm_predictions.csv"
             new_data.to_csv(output_file, index=False)

             print("💾 Predictions saved to", output_file)
             download_button.layout.visibility = 'visible'

         # Bind function to upload button
         upload_button.observe(process_and_predict, names='value')

         # Download CSV Button (initially hidden)
         download_button = widgets.Button(
             description="Download Predictions",
             disabled=False,
             button_style='success'
         )
         download_button.layout.visibility = 'hidden'
         display(download_button)

         📏 Scaler loaded successfully from: /content/drive/MyDrive/Breast Cancer Classification/scaler.pkl

         FileUpload(value={}, accept='.csv', description='Upload')

         Button(description='Download Predictions', layout=Layout(visibility='hidden'), style=ButtonStyle())
```

📌 Step 8: Make ⚠️ Breast Cancer Classification Web App using Gradio

```
In [45]: # ✅ Load Model & Scaler
         if os.path.exists(model_path):
             scaler_path = "/content/drive/MyDrive/Breast Cancer Classification/scaler.pkl"

         # Load the trained SVM model
         if os.path.exists(model_path):
             model_svm = joblib.load(model_path)
         else:
             raise FileNotFoundError("❌ Model file not found at {model_path}")

         # Load the scaler
         if os.path.exists(scaler_path):
             scaler = joblib.load(scaler_path)
         else:
             raise FileNotFoundError("❌ Scaler file not found at {scaler_path}")

         # Expected features
         expected_columns = [
             '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', 'compactness_worst', 'concavity_worst',
             'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst'
         ]


         def predict_from_csv(file):
             df = pd.read_csv(file)

             # Preserve 'id' column if present
             if 'id' in df.columns:
                 ids = df['id']  # Keep 'id' as Series (not DataFrame)
                 df = df.drop(columns=['id'])
             else:
                 ids = None

             # Remove ID column from processing
             if 'id' in expected_columns:
                 df = df.drop(columns=['id'])

             # Check for missing columns
             missing_columns = [col for col in expected_columns if col not in df.columns]
             if missing_columns:
                 return f"❌ Error: Missing columns: {missing_columns}"

             # Convert data to numeric & handle missing values
             df = df[expected_columns].apply(pd.to_numeric, errors='coerce').fillna(0)

             # Scale data
             df_scaled = scaler.transform(df)

             # Predict
             predictions = model_svm.predict(df_scaled)

             # Add Predictions to DataFrame
             df["Predicted_Class"] = ["🟢 Benign" if p == 0 else "🔴 Malignant" for p in predictions]

             # Restore 'id' column if available
             if ids is not None:
                 df.insert(0, 'id', ids)  # Insert 'id' as the first column

             # Save Predictions
             output_file = "/content/Patient_Classification.csv"
             df.to_csv(output_file, index=False)

             return output_file  # Return the file path for download


         def manual_input(*inputs):
             inputs = [pd.to_numeric(val, errors='coerce') if isinstance(val, (int, float, str)) else 0 for val in inputs]

             return pd.DataFrame(
                 [dict(zip(expected_columns, inputs))],
                 index=[0],
                 columns=expected_columns
             ).fillna(0).to_csv(
                 "/content/Patient_Classification.csv",
                 index=False
             )


         # ✅ Gradio Interface
         with gr.Blocks():
             gr.Markdown(
                 """
                 <h1 style="text-align: center;">🔬 Breast Cancer Classification 🔬</h1>

                 <p style="text-align: center;">Uploading detailed medical records helps early detection saves lives!</br></br></br>

                 <p style="text-align: center;">Women aged 40+ should have regular check-ups to detect cancer in its early stages.

                 <p style="text-align: center;">Uploading</br>a</br>or</br>manually</br>enter</br>an option from the above 🔼</br></br></br>

                 <p style="text-align: center;">
                 """
             )


         # ✅ CSV Upload Page
         def upload_csv_page():
             return gr.Interface(
                 fn=predict_from_csv,
                 inputs=gr.FileUpload("📂 Upload CSV File", type="filepath"),  #
                 outputs=gr.File(label="📥 Download Predictions"),
                 title="📂 Upload CSV for Bulk Predictions",
                 description="Upload a CSV file with patient data. The system will process and return the predicted results.",
                 live=False
             )


         def predict_from_manual_input(*inputs):
             """
             ✅ Ensure inputs are converted to numeric values (handle empty or incorrect inputs)
             processed_inputs = []
             for i in inputs:
                 try:
                     processed_inputs.append(float(i))
                 except ValueError:
                     return "❌ Error: Please enter valid numerical values!"

             # Convert list to NumPy array and reshape
             data = np.array(processed_inputs).reshape(1, -1)

             # Convert to DataFrame with correct feature names
             df = pd.DataFrame(data, columns=expected_columns)

             # Voice the input using the loaded scaler
             df_scaled = scaler.transform(df)

             # Make prediction using the SVM model
             prediction = model_svm.predict(df_scaled)[0]

             # Convert output to readable label
             return "🟢 Benign" if prediction == 0 else "🔴 Malignant"

             except Exception as e:
             return f"❌ Error: {str(e)}"

         # ✅ Launch App
         if __name__ == "__main__":
             [uploaded_page], [manual_input_page], manual_input_page[1], _ # Call functions to return gr.Blocks
             ]*sys.argv[1:] if len(sys.argv) > 1 else ["📂 Upload CSV", "✍️ Manual Entry"]
             )

         demo.launch(share=True)
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://ccf7b6b1c0f05c2.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the Terminal or from
to setting directory to display in mapping face boxes (https://huggingface.co/spaces)
```

Out[45]: