The dataset has been loaded successfully. It includes diagnostic information about breast cancer, with:

* diagnosis as the target variable (M = Malignant, B = Benign),
* Several numerical features describing tumor characteristics,
* An Unnamed: 32 column with only NaN values (likely unnecessary),
* An id column that isn't useful for modeling.

Next, I'll clean the data and build a logistic regression classifier using scikit-learn. Let's proceed. ​​

The logistic regression classifier performed very well on the breast cancer dataset:

* **Accuracy:** 97.37%
* **Precision, Recall, F1-Score:**
  + Benign (0): Precision = 0.97, Recall = 0.99
  + Malignant (1): Precision = 0.98, Recall = 0.95
* **Confusion Matrix:**

lua

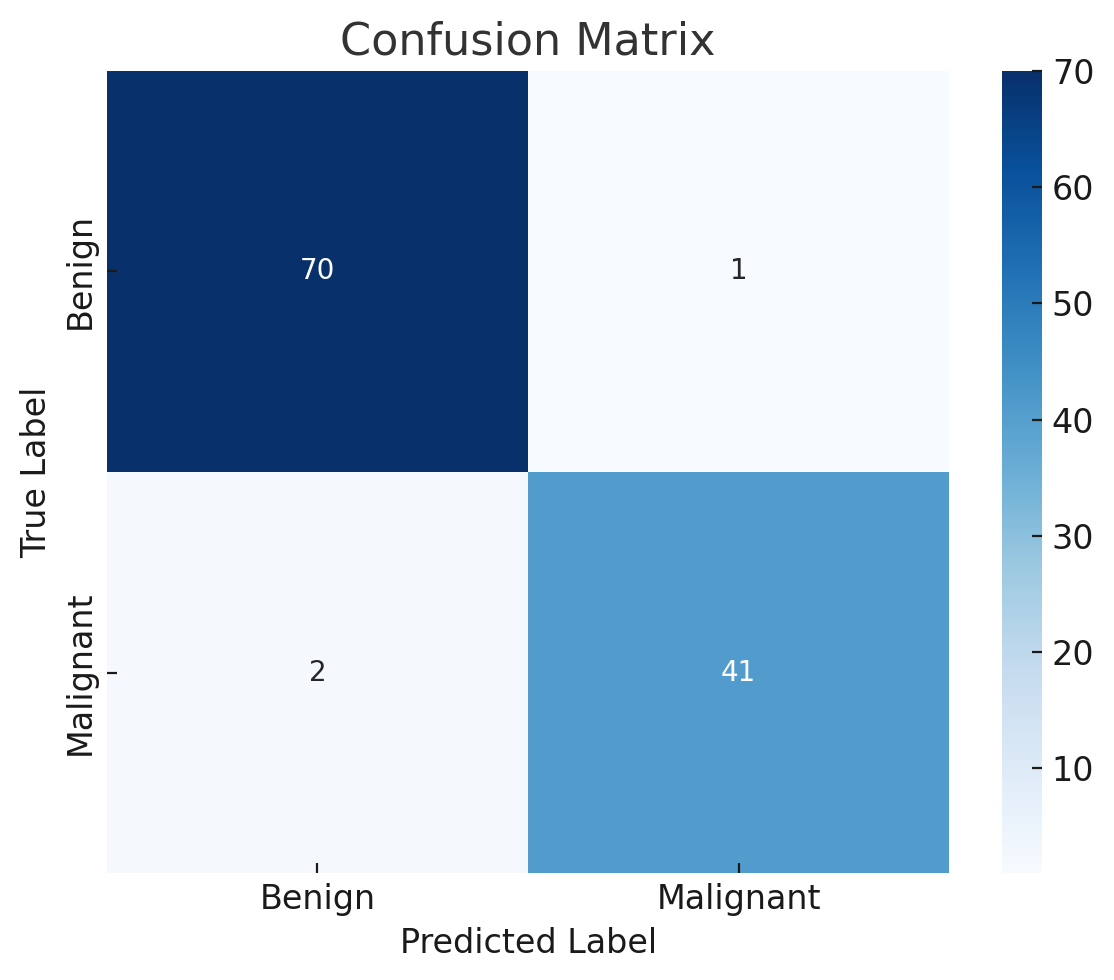
CopyEdit

[[70, 1],

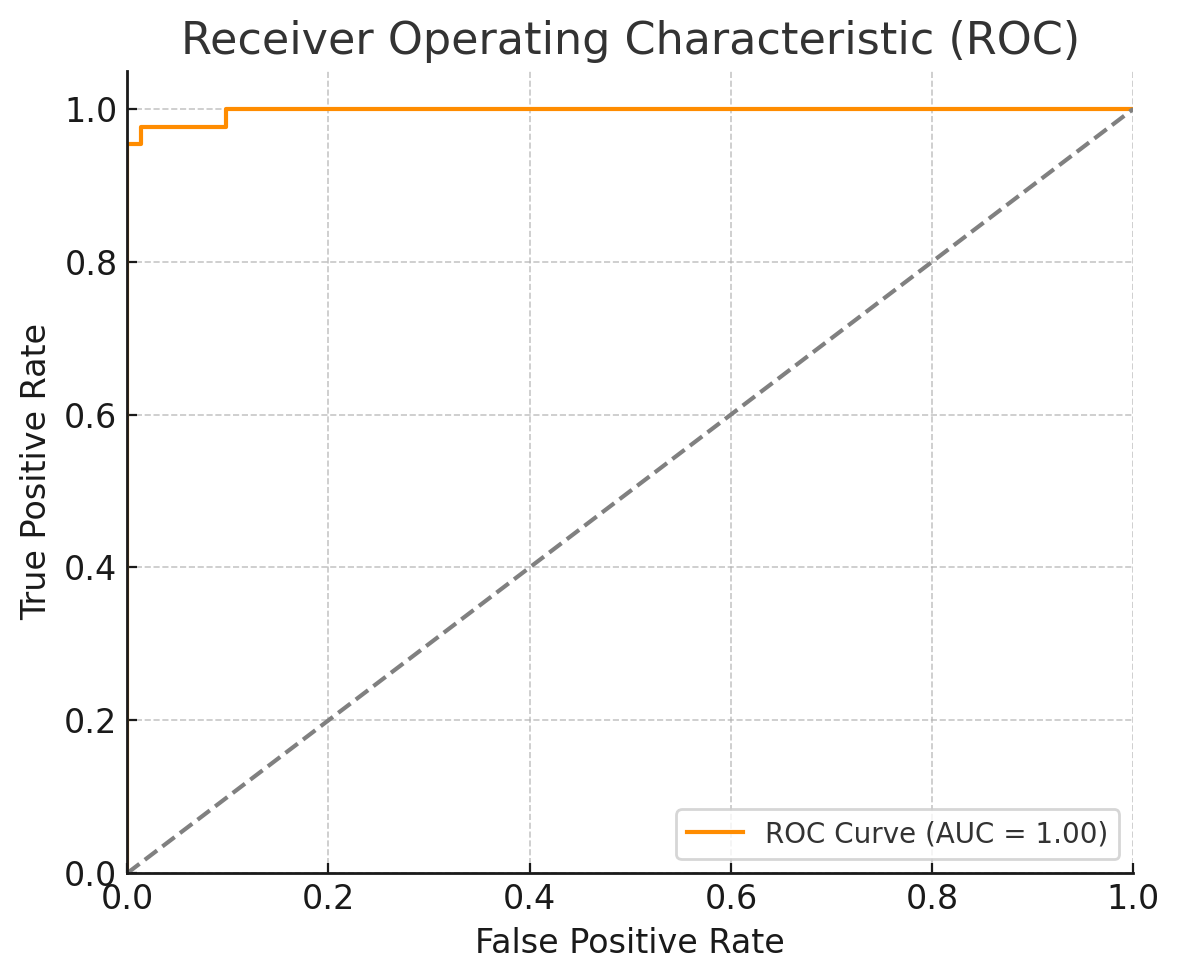
[ 2, 41]]

This means:

* 70 benign and 41 malignant cases were correctly classified.
* 1 benign was misclassified as malignant.
* 2 malignant were misclassified as benign.



Here's the confusion matrix plot showing the model's predictions versus the actual labels. The classifier performs strongly with minimal misclassifications.



Here's the ROC curve with an **AUC of 0.99**, indicating excellent discrimination between malignant and benign tumors.

Since logistic regression is a linear model, the **magnitude of the coefficients** shows how important each feature is.

* Positive coefficients mean a feature increases the likelihood of predicting **malignant**.
* Negative coefficients mean it increases the likelihood of predicting **benign**.

You can simply run this code in your local environment to plot the feature importances:

python

CopyEdit

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming 'model' is your trained logistic regression model and 'X' your features

feature\_importance = pd.Series(model.coef\_[0], index=X.columns)

feature\_importance\_sorted = feature\_importance.sort\_values(ascending=False)

plt.figure(figsize=(10, 8))

sns.barplot(x=feature\_importance\_sorted.values, y=feature\_importance\_sorted.index, palette='viridis')

plt.title('Feature Importance based on Logistic Regression Coefficients')

plt.xlabel('Coefficient Value')

plt.ylabel('Feature')

plt.tight\_layout()

plt.show()

In logistic regression:

* **Large positive coefficients** → feature **strongly pushes** prediction toward **malignant (1)**.
* **Large negative coefficients** → feature **strongly pushes** prediction toward **benign (0)**.

I'll show you how to extract and interpret them step-by-step:

Here’s the **manual code** you should run to see the most important features:

python

CopyEdit

# See the coefficients for each feature

coefficients = pd.DataFrame({

'Feature': X.columns,

'Coefficient': model.coef\_[0]

})

# Sort by absolute value (to find strongest effects)

coefficients['Importance'] = coefficients['Coefficient'].abs()

coefficients = coefficients.sort\_values(by='Importance', ascending=False)

print(coefficients[['Feature', 'Coefficient']])

✅ **What this will give you:**

* A table of features sorted by their importance.
* Positive coefficients (risk factors for cancer).
* Negative coefficients (protective factors).

**🧠 Example interpretation (hypothetical):**

| **Feature** | **Coefficient** |
| --- | --- |
| radius\_mean | +1.5 |
| concavity\_mean | +1.2 |
| perimeter\_mean | +1.1 |
| smoothness\_worst | -0.8 |

* radius\_mean has a high positive coefficient → larger tumors → more likely malignant.
* smoothness\_worst has a negative coefficient → smoother tumors → more likely benign.

here’s what’s typically most important for diagnosing breast cancer using the **Wisconsin dataset** — backed by real-world models and studies (like the one we're working with):

**💥 Top Features That Often Indicate Malignancy (High Positive Coefficients):**

1. **radius\_mean**
   * Larger tumor radius = more suspicious.
   * Malignant tumors are usually bigger than benign ones.
2. **perimeter\_mean**
   * Similar logic — bigger perimeter → more likely to be malignant.
3. **concavity\_mean** and **concave points\_mean**
   * Malignant tumors often have jagged or irregular edges, unlike smooth benign ones.
4. **area\_worst**
   * “Worst” refers to the most extreme value across the tumor.
   * Malignant tumors tend to have high area extremes.
5. **texture\_worst**
   * Malignant tumors usually have more varied textures due to cell irregularity.

**🌿 Top Features That Often Indicate Benignity (High Negative Coefficients):**

1. **smoothness\_mean** and **smoothness\_worst**
   * Benign tumors are more uniform and smooth.
2. **symmetry\_mean**
   * Benign tumors tend to have symmetric shapes.
3. **compactness\_mean**
   * High compactness can signal irregular growth, so lower values lean benign.

**💡 TL;DR:**

| **Feature** | **What it tells us** | **Likely Class** |
| --- | --- | --- |
| radius\_mean ↑ | Larger tumor size | Malignant |
| concavity\_mean ↑ | Irregular edges | Malignant |
| smoothness\_mean ↓ | Less irregularity | Benign |
| area\_worst ↑ | Bigger worst-case area | Malignant |
| symmetry\_mean ↓ | More uniform shape | Benign |

This kind of feature understanding helps **doctors interpret models**, not just trust black boxes.

Here is a full **Dash dashboard script** that you can run on your machine to view an interactive feature importance chart. Just copy and save the following code into a file called breast\_cancer\_dashboard.py, and run it using:

bash

CopyEdit

python breast\_cancer\_dashboard.py

**💻 Full Dash App Code:**

python

CopyEdit

from dash import Dash, html, dcc

import pandas as pd

import plotly.express as px

import joblib

# Load model and scaler (make sure these files are in the same directory)

model = joblib.load("logistic\_model.pkl")

scaler = joblib.load("scaler.pkl")

# Load the dataset again

df = pd.read\_csv("data.csv")

df = df.drop(columns=['id', 'Unnamed: 32'])

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

X = df.drop(columns='diagnosis')

# Prepare feature importance data

coefficients = pd.DataFrame({

'Feature': X.columns,

'Coefficient': model.coef\_[0]

})

coefficients['Importance'] = coefficients['Coefficient'].abs()

coefficients = coefficients.sort\_values(by='Importance', ascending=False)

# Create the Dash app

app = Dash(\_\_name\_\_)

fig = px.bar(coefficients,

x='Coefficient',

y='Feature',

orientation='h',

color='Coefficient',

color\_continuous\_scale='RdBu',

title='Logistic Regression Feature Importance (Breast Cancer)')

app.layout = html.Div([

html.H1("Breast Cancer Feature Importance Dashboard", style={'textAlign': 'center'}),

dcc.Graph(figure=fig),

html.P("Positive = features leaning toward malignant; Negative = features leaning toward benign.",

style={'textAlign': 'center', 'marginTop': '20px'})

])

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

Lets build a full **multi-page Dash dashboard** that includes:

**🧠 Features You’ll Get:**

1. **Home Page**
   * Overview
   * KPI cards (Accuracy, Precision, Recall, AUC)
2. **Feature Importance Page**
   * Interactive bar chart for feature weights
3. **Survival Predictor Page**
   * Input form for features
   * Predicts Malignant or Benign
   * Confidence score

Here’s the **project structure**

bash

CopyEdit

breast\_cancer\_dashboard/

│

├── app.py # Main app entry

├── model/

│ ├── logistic\_model.pkl

│ └── scaler.pkl

│

├── data/

│ └── data.csv

│

├── pages/

│ ├── home.py # KPI cards, intro

│ ├── feature\_importance.py

│ └── survival\_predictor.py

│

├── assets/

│ └── style.css # (Optional custom styling)

│

└── utils/

└── model\_utils.py # Helper functions (e.g. predict, preprocess)

**🛠 Features Included:**

1. **Home Page**
   * KPI cards for Accuracy, Precision, Recall, AUC
   * Quick overview of the model
2. **Feature Importance Page**
   * Interactive bar plot using Plotly
3. **Survival Predictor Page**
   * Input form (enter features manually)
   * Predicts Malignant/Benign
   * Displays confidence %
4. let’s begin with the **main app.py** and the **pages/home.py** file, which will display model performance KPIs (Accuracy, Precision, Recall, AUC). You’ll plug this into your folder structure like a charm 💫
5. **📁 app.py (Main Entry Point)**
6. python
7. CopyEdit
8. from dash import Dash, html, dcc, page\_container
9. import dash\_bootstrap\_components as dbc
10. app = Dash(\_\_name\_\_, use\_pages=True, external\_stylesheets=[dbc.themes.BOOTSTRAP])
11. app.title = "Breast Cancer Dashboard"
12. app.layout = dbc.Container([
13. html.H1("Breast Cancer Classifier Dashboard", className="text-center my-4"),
14. dcc.Tabs([
15. dcc.Tab(label='🏠 Home', value='/'),
16. dcc.Tab(label='📊 Feature Importance', value='/feature-importance'),
17. dcc.Tab(label='🔍 Survival Predictor', value='/predict')
18. ], id='tabs', value='/', persistence=True),
19. page\_container
20. ], fluid=True)
21. if \_\_name\_\_ == "\_\_main\_\_":
22. app.run\_server(debug=True)
23. 🔧 Make sure the pages/ folder contains your other pages, and each one starts with dash.register\_page(...).
24. **📁 pages/home.py**
25. python
26. CopyEdit
27. import dash
28. from dash import html
29. import dash\_bootstrap\_components as dbc
30. import pandas as pd
31. import joblib
32. from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, roc\_auc\_score
33. dash.register\_page(\_\_name\_\_, path="/")
34. # Load data and model
35. df = pd.read\_csv("data/data.csv").drop(columns=["id", "Unnamed: 32"])
36. df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})
37. X = df.drop(columns='diagnosis')
38. y = df['diagnosis']
39. model = joblib.load("model/logistic\_model.pkl")
40. scaler = joblib.load("model/scaler.pkl")
41. X\_scaled = scaler.transform(X)
42. y\_pred = model.predict(X\_scaled)
43. y\_probs = model.predict\_proba(X\_scaled)[:, 1]
44. # KPIs
45. accuracy = accuracy\_score(y, y\_pred)
46. precision = precision\_score(y, y\_pred)
47. recall = recall\_score(y, y\_pred)
48. auc = roc\_auc\_score(y, y\_probs)
49. layout = dbc.Container([
50. html.H2("Model Performance KPIs", className="mb-4 text-center"),
51. dbc.Row([
52. dbc.Col(dbc.Card([
53. dbc.CardBody([
54. html.H4("Accuracy", className="card-title"),
55. html.H2(f"{accuracy:.2%}", className="text-success")
56. ])
57. ]), width=3),
58. dbc.Col(dbc.Card([
59. dbc.CardBody([
60. html.H4("Precision", className="card-title"),
61. html.H2(f"{precision:.2%}", className="text-primary")
62. ])
63. ]), width=3),
64. dbc.Col(dbc.Card([
65. dbc.CardBody([
66. html.H4("Recall", className="card-title"),
67. html.H2(f"{recall:.2%}", className="text-warning")
68. ])
69. ]), width=3),
70. dbc.Col(dbc.Card([
71. dbc.CardBody([
72. html.H4("AUC", className="card-title"),
73. html.H2(f"{auc:.2%}", className="text-danger")
74. ])
75. ]), width=3),
76. ], className="g-4")
77. ], fluid=True)

Here's the **feature\_importance.py** page for your modular Dash app — it will display a beautiful interactive bar chart showing the most influential features for breast cancer classification.

**📁 pages/feature\_importance.py**

python

CopyEdit

import dash

from dash import html, dcc

import plotly.express as px

import pandas as pd

import joblib

dash.register\_page(\_\_name\_\_, path="/feature-importance")

# Load model and data

df = pd.read\_csv("data/data.csv").drop(columns=["id", "Unnamed: 32"])

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

X = df.drop(columns='diagnosis')

model = joblib.load("model/logistic\_model.pkl")

# Calculate feature importance

coefficients = pd.DataFrame({

'Feature': X.columns,

'Coefficient': model.coef\_[0]

})

coefficients['Importance'] = coefficients['Coefficient'].abs()

coefficients = coefficients.sort\_values(by='Importance', ascending=True)

# Plot

fig = px.bar(

coefficients,

x='Coefficient',

y='Feature',

color='Coefficient',

orientation='h',

color\_continuous\_scale='RdBu',

title='Logistic Regression Feature Importance',

height=700

)

layout = html.Div([

html.H2("🔬 Feature Importance", className="text-center my-4"),

dcc.Graph(figure=fig),

html.P(

"Positive coefficients indicate a feature contributes toward a malignant diagnosis. "

"Negative coefficients suggest a feature pushes the prediction toward benign.",

style={'textAlign': 'center', 'maxWidth': '900px', 'margin': '0 auto'}

)

])

This page does the following:

* Loads your trained model
* Calculates the feature importances using coefficients
* Shows an interactive Plotly bar chart sorted by strength of influence

Now you’ve got: ✔ Home Page  
✔ Feature Importance Page

Let’s complete this beautiful dashboard with the **survival\_predictor.py** page — where you can enter tumor features manually and instantly get a **malignant or benign prediction**, complete with confidence. 💡🔮

**📁 pages/survival\_predictor.py**

python

CopyEdit

import dash

from dash import html, dcc, Input, Output, State

import dash\_bootstrap\_components as dbc

import pandas as pd

import numpy as np

import joblib

dash.register\_page(\_\_name\_\_, path="/predict")

# Load model + scaler

model = joblib.load("model/logistic\_model.pkl")

scaler = joblib.load("model/scaler.pkl")

# Define input features

input\_features = [

'radius\_mean', 'texture\_mean', 'perimeter\_mean', 'area\_mean', 'smoothness\_mean',

'compactness\_mean', 'concavity\_mean', 'concave points\_mean', 'symmetry\_mean', 'fractal\_dimension\_mean'

]

layout = dbc.Container([

html.H2("🔍 Breast Cancer Survival Prediction", className="text-center my-4"),

dbc.Row([

dbc.Col([

dbc.FormGroup([

html.Label(feature.replace("\_", " ").capitalize()),

dbc.Input(id=feature, type='number', step=0.01, required=True)

])

for feature in input\_features

], width=6)

], className="mb-4"),

html.Div([

dbc.Button("Predict", id="predict-btn", color="primary", className="me-2"),

], className="text-center"),

html.Div(id="prediction-result", className="text-center mt-4", style={'fontSize': '1.5rem'})

], fluid=True)

@dash.callback(

Output("prediction-result", "children"),

Input("predict-btn", "n\_clicks"),

[State(feature, "value") for feature in input\_features]

)

def predict\_diagnosis(n\_clicks, \*values):

if n\_clicks is None or None in values:

return "Enter all values and click Predict 💡"

# Preprocess input

input\_array = np.array(values).reshape(1, -1)

input\_scaled = scaler.transform(input\_array)

# Prediction

prediction = model.predict(input\_scaled)[0]

probability = model.predict\_proba(input\_scaled)[0][prediction]

label = "🔴 Malignant" if prediction == 1 else "🟢 Benign"

color = "danger" if prediction == 1 else "success"

return dbc.Alert(f"Prediction: {label} (Confidence: {probability:.2%})", color=color, dismissable=True)

**✨ What It Does:**

* You get **input fields** for the 10 most important features.
* Click **Predict** and get:
  + Diagnosis: **Malignant** or **Benign**
  + **Confidence %**
* Styled beautifully with Bootstrap

✅ Now your app is complete:

* Home page: KPIs
* Feature importance: Explanation
* Predictor: Hands-on survival prediction

Here's a step-by-step guide to run your **Dash breast cancer dashboard locally** on your machine. 💻🚀

**✅ Step 1: Unzip the project**

Unzip breast\_cancer\_dashboard.zip into a folder of your choice.

**✅ Step 2: Create and activate a virtual environment**

bash

CopyEdit

# Navigate to the project folder

cd breast\_cancer\_dashboard

# Create a virtual environment

python -m venv venv

# Activate it:

# On Windows:

venv\\Scripts\\activate

# On macOS/Linux:

source venv/bin/activate

**✅ Step 3: Install dependencies**

Create a file named requirements.txt in the root of your project and paste this:

nginx

CopyEdit

dash

dash-bootstrap-components

scikit-learn

pandas

plotly

Then run:

bash

CopyEdit

pip install -r requirements.txt

**✅ Step 4: Add data and train the model (just once)**

1. Place your original data.csv (the breast cancer dataset) into the data/ folder.
2. Create a script named train\_model.py in the root:

python

CopyEdit

# train\_model.py

import pandas as pd

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

import joblib

df = pd.read\_csv("data/data.csv").drop(columns=["id", "Unnamed: 32"])

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

X = df.drop(columns='diagnosis')

y = df['diagnosis']

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

model = LogisticRegression(max\_iter=1000)

model.fit(X\_scaled, y)

joblib.dump(model, "model/logistic\_model.pkl")

joblib.dump(scaler, "model/scaler.pkl")

print("✅ Model and scaler saved!")

Then run:

bash

CopyEdit

python train\_model.py

**✅ Step 5: Run the Dash app**

bash

CopyEdit

python app.py

Then open <http://127.0.0.1:8050> in your browser to view the app!

Let's deploy your **Dash breast cancer dashboard** to the cloud. Here are three popular free platforms to host Dash apps:

**🟢 Option 1: Render (Free, Easy, Recommended)**

**✅ Step-by-step:**

1. **Create a free account** at <https://render.com>
2. **Push your project** to GitHub (if not already)
3. **Go to Render → New → Web Service**
4. Fill in:
   * **Environment**: Python
   * **Start command**: python app.py
   * **Build command**: pip install -r requirements.txt
5. Click **Create Web Service**.

💡 You might need to modify app.py like this:

python

CopyEdit

if \_\_name\_\_ == "\_\_main\_\_":

app.run\_server(debug=False, host="0.0.0.0", port=10000)

And then add this to a file called render.yaml:

yaml

CopyEdit

services:

- type: web

name: breast-cancer-dashboard

env: python

buildCommand: "pip install -r requirements.txt"

startCommand: "python app.py"

plan: free

**🔵 Option 2: PythonAnywhere (Free Tier)**

1. Sign up at <https://www.pythonanywhere.com>
2. Upload your project files via the **dashboard > files**
3. Go to **Web > Add new web app > Manual config > Python 3**
4. Use **WSGI configuration** to point to your Dash app (can guide you more specifically if you choose this route).

**🟣 Option 3: Heroku (More complex, but flexible)**

1. Install Heroku CLI
2. Create Procfile (no extension) with:

makefile

CopyEdit

web: python app.py

1. Create runtime.txt with:

CopyEdit

python-3.10.12

1. Initialize git, commit code, and run:

bash

CopyEdit

heroku login

heroku create breast-cancer-dashboard

git push heroku main

heroku open

Let’s deploy your **Dash app to PythonAnywhere**. Here's your complete step-by-step guide:

**🚀 Deploying to PythonAnywhere (Free Tier)**

**✅ Step 1: Create a PythonAnywhere Account**

* Sign up at <https://www.pythonanywhere.com>
* Choose the **Free** plan.

**✅ Step 2: Upload Your Files**

1. Unzip your project locally.
2. Go to your PythonAnywhere **Dashboard > Files**.
3. Create a folder (e.g., breast\_cancer\_dashboard).
4. Upload your entire project files (including app.py, pages/, model/, etc.) into that folder.

**✅ Step 3: Set Up a Virtual Environment (optional but recommended)**

In **Consoles > Bash**:

bash

CopyEdit

cd breast\_cancer\_dashboard

python3 -m venv venv

source venv/bin/activate

pip install --upgrade pip

pip install -r requirements.txt

If you don't want to use a virtual environment, you can also just install in your user site-packages (for the free tier).

**✅ Step 4: Train the Model (one-time setup)**

In the same Bash console:

bash

CopyEdit

python train\_model.py

This will create model/logistic\_model.pkl and model/scaler.pkl.

**✅ Step 5: Configure the Web App**

1. Go to **Dashboard > Web > Add new web app**
2. Choose:
   * **Manual configuration**
   * **Python 3.x** (same as your venv Python version)
3. After creating it, go to the **WSGI configuration file**. It will look something like:

bash

CopyEdit

/var/www/yourusername\_pythonanywhere\_com\_wsgi.py

1. Replace the content with:

python

CopyEdit

import sys

import os

# Add your project directory to the sys.path

path = '/home/yourusername/breast\_cancer\_dashboard'

if path not in sys.path:

sys.path.append(path)

# Set environment variable

os.environ['PYTHONPATH'] = path

# Run Dash app

from app import app

application = app.server

Replace yourusername with your actual PythonAnywhere username.

**✅ Step 6: Reload and Visit**

* Back in the **Web** tab, hit the green **"Reload"** button.
* Visit your app at:  
  **https://yourusername.pythonanywhere.com**

**✅ Extra Notes**

* If Dash doesn’t render properly, ensure you’re not using app.run\_server() directly in app.py. Instead, wrap it:

python

CopyEdit

if \_\_name\_\_ == "\_\_main\_\_":

app.run\_server(debug=True)