Exploratory Data Analysis

Importing libraries

```
In [1]: # 1) Import Libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')
```

Loading Dataset

```
In [2]: # 2) Load Dataset
        df = pd.read_csv("data.csv")
```

Overview of the Dataset

```
In [3]: # 3) Basic Overview
        print("Shape:", df.shape)
        print("\nColumns:\n", df.columns.tolist())
        print("\nData types:\n", df.dtypes.value_counts())
```

```
Shape: (569, 33)

Columns:
 ['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'sym
metry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se', 'perimeter_se', 'a
rea_se', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se', 'sy
mmetry_se', 'fractal_dimension_se', 'radius_worst', 'texture_worst', 'perimeter_wors
t', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst', 'conca
ve points_worst', 'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32']

Data types:
 float64    31
int64       1
object      1
Name: count, dtype: int64
```

Our data has 569 rows and 33 columns as listed above. Most of the columns contain numerical values (31 columns containing floats and one containing integers). The remaining one contains our target variable which is categorical.

In [4]:
```python
# Preview
df.head()
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

5 rows × 33 columns

In [5]:
```python
# 4) Missing Values & Duplicates
print("\nMissing values per column:\n", df.isnull().sum())
print("\nDuplicate rows:", df.duplicated().sum())
```

```
Missing values per column:
 id                          0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64

Duplicate rows: 0
```

The preview shows us that the column "Unnamed 32" is filled with NaNs and since it holds no particular importance for our analysis, we will make a note to drop it in our cleaning phase.

In [6]:
```python
# 5) Summary Statistics
df.describe().T
```

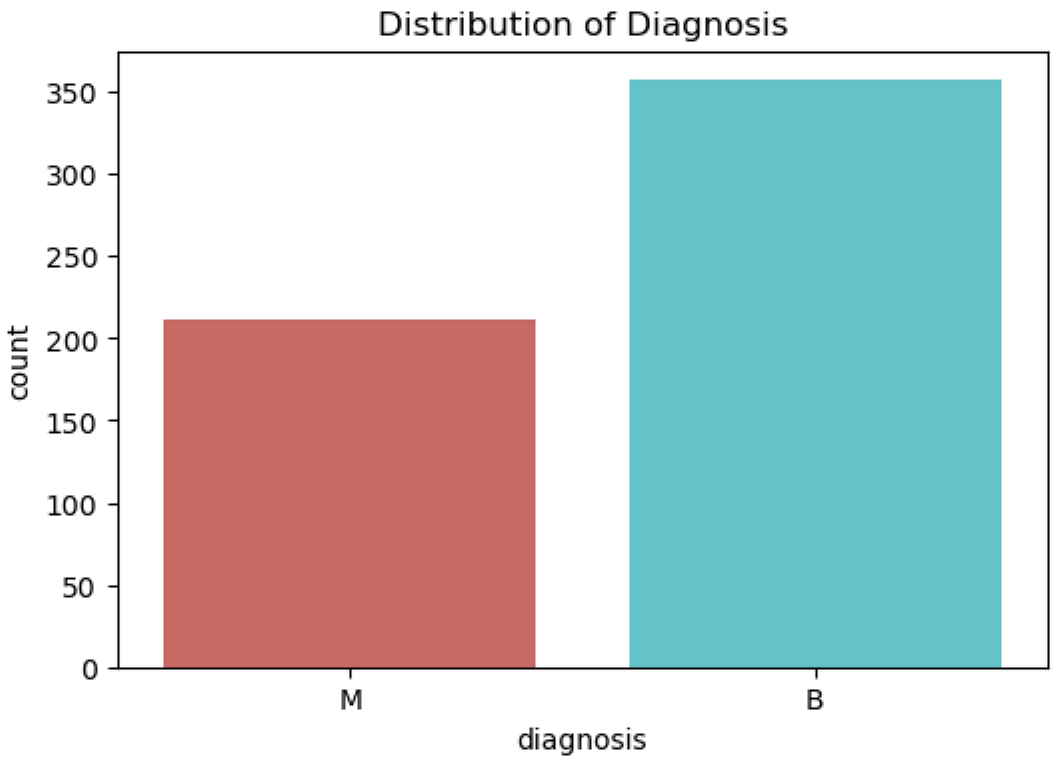
Out[6]:

	count	mean	std	min	25%	
id	569.0	3.037183e+07	1.250206e+08	8670.000000	869218.000000	9
radius_mean	569.0	1.412729e+01	3.524049e+00	6.981000	11.700000	
texture_mean	569.0	1.928965e+01	4.301036e+00	9.710000	16.170000	
perimeter_mean	569.0	9.196903e+01	2.429898e+01	43.790000	75.170000	
area_mean	569.0	6.548891e+02	3.519141e+02	143.500000	420.300000	
smoothness_mean	569.0	9.636028e-02	1.406413e-02	0.052630	0.086370	
compactness_mean	569.0	1.043410e-01	5.281276e-02	0.019380	0.064920	
concavity_mean	569.0	8.879932e-02	7.971981e-02	0.000000	0.029560	
concave points_mean	569.0	4.891915e-02	3.880284e-02	0.000000	0.020310	
symmetry_mean	569.0	1.811619e-01	2.741428e-02	0.106000	0.161900	
fractal_dimension_mean	569.0	6.279761e-02	7.060363e-03	0.049960	0.057700	
radius_se	569.0	4.051721e-01	2.773127e-01	0.111500	0.232400	
texture_se	569.0	1.216853e+00	5.516484e-01	0.360200	0.833900	
perimeter_se	569.0	2.866059e+00	2.021855e+00	0.757000	1.606000	
area_se	569.0	4.033708e+01	4.549101e+01	6.802000	17.850000	
smoothness_se	569.0	7.040979e-03	3.002518e-03	0.001713	0.005169	
compactness_se	569.0	2.547814e-02	1.790818e-02	0.002252	0.013080	
concavity_se	569.0	3.189372e-02	3.018606e-02	0.000000	0.015090	
concave points_se	569.0	1.179614e-02	6.170285e-03	0.000000	0.007638	
symmetry_se	569.0	2.054230e-02	8.266372e-03	0.007882	0.015160	
fractal_dimension_se	569.0	3.794904e-03	2.646071e-03	0.000895	0.002248	
radius_worst	569.0	1.626919e+01	4.833242e+00	7.930000	13.010000	
texture_worst	569.0	2.567722e+01	6.146258e+00	12.020000	21.080000	
perimeter_worst	569.0	1.072612e+02	3.360254e+01	50.410000	84.110000	
area_worst	569.0	8.805831e+02	5.693570e+02	185.200000	515.300000	
smoothness_worst	569.0	1.323686e-01	2.283243e-02	0.071170	0.116600	
compactness_worst	569.0	2.542650e-01	1.573365e-01	0.027290	0.147200	
concavity_worst	569.0	2.721885e-01	2.086243e-01	0.000000	0.114500	
concave points_worst	569.0	1.146062e-01	6.573234e-02	0.000000	0.064930	
symmetry_worst	569.0	2.900756e-01	6.186747e-02	0.156500	0.250400	

	count	mean	std	min	25%
fractal_dimension_worst	569.0	8.394582e-02	1.806127e-02	0.055040	0.071460
Unnamed: 32	0.0	NaN	NaN	NaN	NaN

Looking at the descriptive statistics, we see that most features span wide ranges with standard deviations that often approach or even exceed their means. This is a clear sign of high variability in the data. The relationship between the mean and the standard deviation tells us that while some features are stable and consistent, others have extreme outliers. These patterns will be more visible in our visualization.

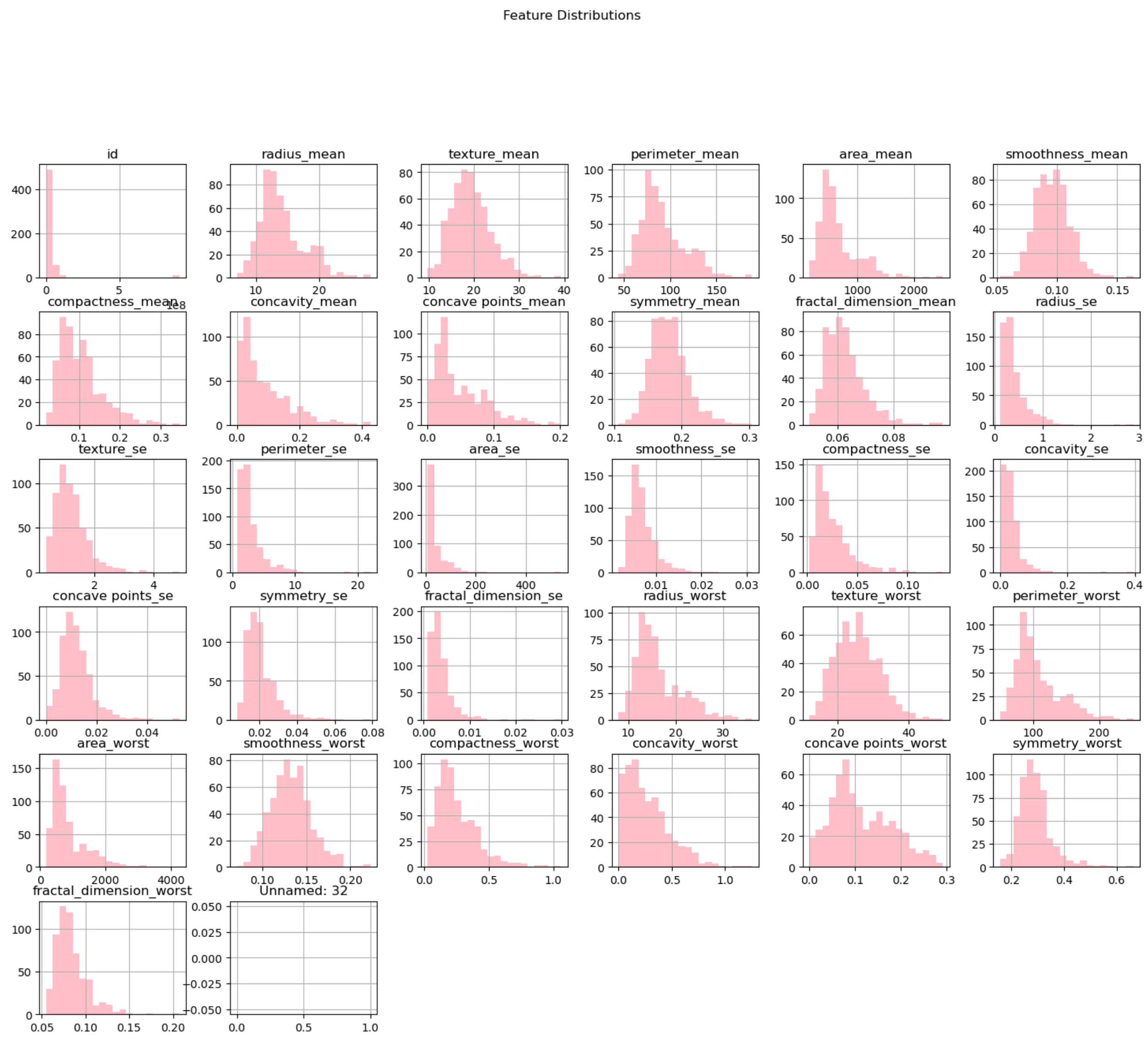
In [7]:
```
# 6) Target Variable Exploration
target_col = 'diagnosis'
if target_col in df.columns:
    plt.figure(figsize=(6,4))
    sns.countplot(x=target_col, data=df, palette='hls')
    plt.title("Distribution of Diagnosis")
    plt.show()
    print(df[target_col].value_counts(normalize=True) * 100)
else:
    print(f"Target column '{target_col}' not found. Check column names.")
```



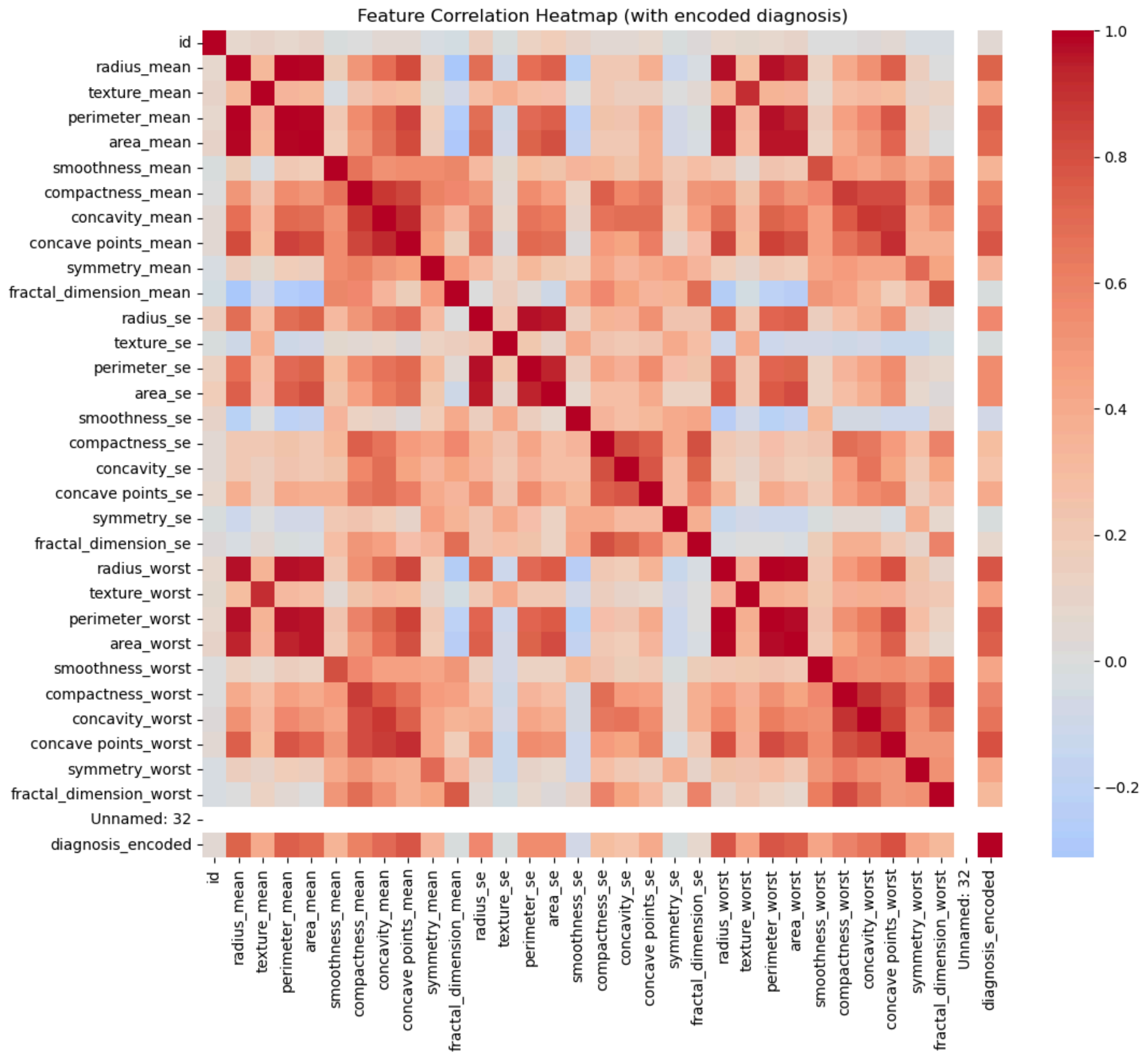
```
diagnosis
B    62.741652
M    37.258348
Name: proportion, dtype: float64
```

The dataset is imbalanced, with more benign cases than malignant ones. This can wrongly influence modeling. But on the other side, it reflects the reality that most breast tumors are non-cancerous. In summary, while benign tumors dominate the dataset, the malignant subset provides essential information for distinguishing high-risk cases.

In [8]:
```
# 7) Feature Distributions
df.drop(columns=[target_col], errors='ignore').hist(bins=20, figsize=(18,14), color
plt.suptitle("Feature Distributions", y=1.02)
plt.show()
```



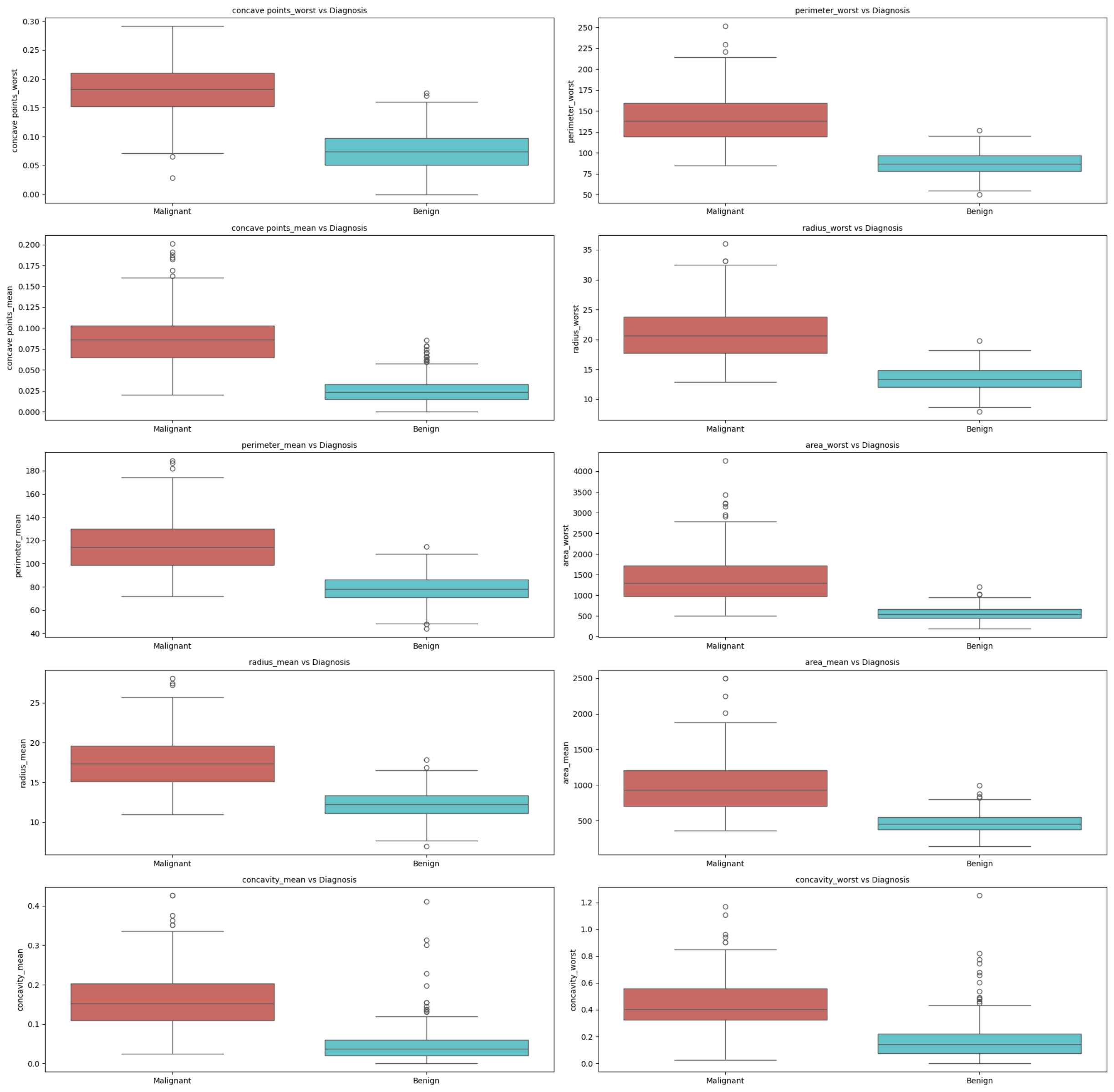
In [9]:
```
# 8) Correlation Analysis
# Encode target column for correlation
df['diagnosis_encoded'] = df['diagnosis'].map({'M': 1, 'B': 0})

# Correlation heatmap including encoded diagnosis
plt.figure(figsize=(12,10))
sns.heatmap(df.drop(columns=['diagnosis'], errors='ignore').corr(), cmap='coolwarm'
plt.title("Feature Correlation Heatmap (with encoded diagnosis)")
plt.show()
```



In [10]:
```python
# Get correlation values with encoded diagnosis
target_corr = (
    df.select_dtypes(include=[np.number])
      .corr()['diagnosis_encoded']
      .drop('diagnosis_encoded')
      .sort_values(ascending=False)
)
# Display top 10 most correlated features
print("Top 10 features most correlated with malignant diagnosis:\n")
print(target_corr.head(10))

# Extract feature names
top_features = target_corr.head(10).index.tolist()

# Add a readable label column
df['label'] = df['diagnosis_encoded'].map({1: 'Malignant', 0: 'Benign'})

# Plot distributions of top 10 features
plt.figure(figsize=(20, 20))
for i, feature in enumerate(top_features):
    plt.subplot(5, 2, i + 1)
    sns.boxplot(x='label', y=feature, data=df, palette='hls')
    plt.title(f"{feature} vs Diagnosis", fontsize=10)
    plt.xlabel("")
    plt.ylabel(feature)

plt.suptitle("Top 10 Features Most Correlated with Malignant Diagnosis", fontsize=1
plt.tight_layout()
plt.show()
```

```
Top 10 features most correlated with malignant diagnosis:

concave points_worst    0.793566
perimeter_worst         0.782914
concave points_mean     0.776614
radius_worst            0.776454
perimeter_mean          0.742636
area_worst              0.733825
radius_mean             0.730029
area_mean               0.708984
concavity_mean          0.696360
concavity_worst         0.659610
Name: diagnosis_encoded, dtype: float64
```

Top 10 Features Most Correlated with Malignant Diagnosis



The correlation analysis highlights the top ten features most strongly associated with malignant tumors. The highest correlation is observed with concave points_worst (0.79), followed closely by perimeter_worst (0.78) and concave points_mean (0.78). This indicates that tumors with a greater number of concave points and larger perimeters are strongly linked to malignancy. Similarly, radius_worst (0.78) and perimeter_mean (0.74) reinforce the importance of tumor size and boundary irregularity as key diagnostic indicators.

Area-related features such as area_worst (0.73) and area_mean (0.71) also show strong positive correlations, suggesting that malignant tumors tend to occupy significantly larger regions compared to benign ones. Meanwhile, concavity_mean (0.70) and concavity_worst (0.66) emphasize that the degree of indentation in tumor shapes is another critical factor distinguishing malignant growths.

Overall, the pattern is clear: features capturing size (radius, perimeter, area) and irregularity (concavity, concave points) dominate the top correlations with malignancy. These findings suggest that these features will likely play a central role in predictive modeling and should be prioritized in feature selection and interpretability analyses.

Data Cleaning and Preparation

This notebook encodes the `diagnosis` column, drops unnecessary columns, and saves the cleaned dataset as `clean_data.csv`.

In [1]:
```python
# DATA CLEANING AND PREPARATION

import pandas as pd

# 1) Load original dataset
df = pd.read_csv("data.csv")
print("Initial shape:", df.shape)

# 2) Encode target column ('M' → 1, 'B' → 0)
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})
print("\n Target column 'diagnosis' successfully encoded (1=Malignant, 0=Benign).")

# 3) Drop unnecessary columns
df = df.drop(columns=['Unnamed: 32', 'id'], errors='ignore')

print("\n Dropped unnecessary columns")
print("New shape after cleaning:", df.shape)

# 4) Verify data types and check for missing values
print("\nData types after cleaning:\n", df.dtypes)
print("\nMissing values per column:\n", df.isnull().sum())

# 5) Save the cleaned dataset
df.to_csv("clean_data.csv", index=False)
print("\nCleaned dataset saved as 'clean_data.csv'")
```

```
Initial shape: (569, 33)

 Target column 'diagnosis' successfully encoded (1=Malignant, 0=Benign).

 Dropped unnecessary columns
New shape after cleaning: (569, 31)

Data types after cleaning:
 diagnosis                    int64
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object

Missing values per column:
 diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
```

```
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64

Cleaned dataset saved as 'clean_data.csv'
```

Modelization

In this step, we aim to identify a classification model that reliably detects malignant tumors while minimizing false negatives, the most critical error in a medical context. A false negative implies a missed cancer diagnosis, which can delay treatment and increase risk. Therefore, our priority is not just overall accuracy, but high recall for malignant cases.

We begin with logistic regression as a baseline due to its simplicity and interpretability. We then explore more flexible models such as a neural network (MLP) and a Random Forest to assess whether they offer performance gains, particularly in sensitivity. Each model is evaluated using the same metrics: accuracy, precision, recall, and confusion matrix. A model will be retained only if it improves recall without introducing unacceptable trade-offs in precision or interpretability.

In [1]:
```
# Importing the necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
```

pandas / numpy: for data manipulation

train_test_split: to split the data into training and testing sets

LogisticRegression: the base model

metrics: to evaluate the model's performance

In [2]:
```
#Reading the CSV file
data = pd.read_csv("clean_data.csv")
print(data.head())
```

```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38          122.80     1001.0   
1          1        20.57         17.77          132.90     1326.0   
2          1        19.69         21.25          130.00     1203.0   
3          1        11.42         20.38           77.58      386.1   
4          1        20.29         14.34          135.10     1297.0   

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710   
1          0.08474           0.07864          0.0869              0.07017   
2          0.10960           0.15990          0.1974              0.12790   
3          0.14250           0.28390          0.2414              0.10520   
4          0.10030           0.13280          0.1980              0.10430   

   symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0         0.2419  ...         25.38          17.33           184.60   
1         0.1812  ...         24.99          23.41           158.80   
2         0.2069  ...         23.57          25.53           152.50   
3         0.2597  ...         14.91          26.50            98.87   
4         0.1809  ...         22.54          16.67           152.20   

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119   
1      1956.0            0.1238             0.1866           0.2416   
2      1709.0            0.1444             0.4245           0.4504   
3       567.7            0.2098             0.8663           0.6869   
4      1575.0            0.1374             0.2050           0.4000   

   concave points_worst  symmetry_worst  fractal_dimension_worst  
0                0.2654          0.4601                  0.11890  
1                0.1860          0.2750                  0.08902  
2                0.2430          0.3613                  0.08758  
3                0.2575          0.6638                  0.17300  
4                0.1625          0.2364                  0.07678  

[5 rows x 31 columns]
```

```
In [3]: print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   diagnosis                569 non-null    int64  
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
None
```

In [4]:
```
print(data['diagnosis'].value_counts())
```

```
diagnosis
0    357
1    212
Name: count, dtype: int64
```

In [5]:
```
#Data Preparation
X = data.drop(columns=["diagnosis"])  # independents Variables
y = data["diagnosis"]                       # target Variable
```

X contains the features (measurements of the cancer cells).

y contains the target variable to predict:

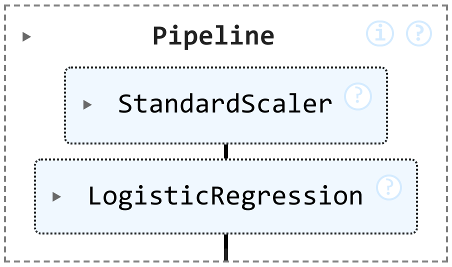
0 → benign

1 → malignant

In [6]:
```
#Splitting the data into train and test sets

#80% for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [7]:
```
# Creating pipeline with scaling + logistic regression
pipeline = make_pipeline(
    StandardScaler(),
    LogisticRegression(max_iter=1000)
)

# Training the pipeline
pipeline.fit(X_train, y_train)
```

Out[7]:



Logistic regression calculates the probability that a tumor is malignant.

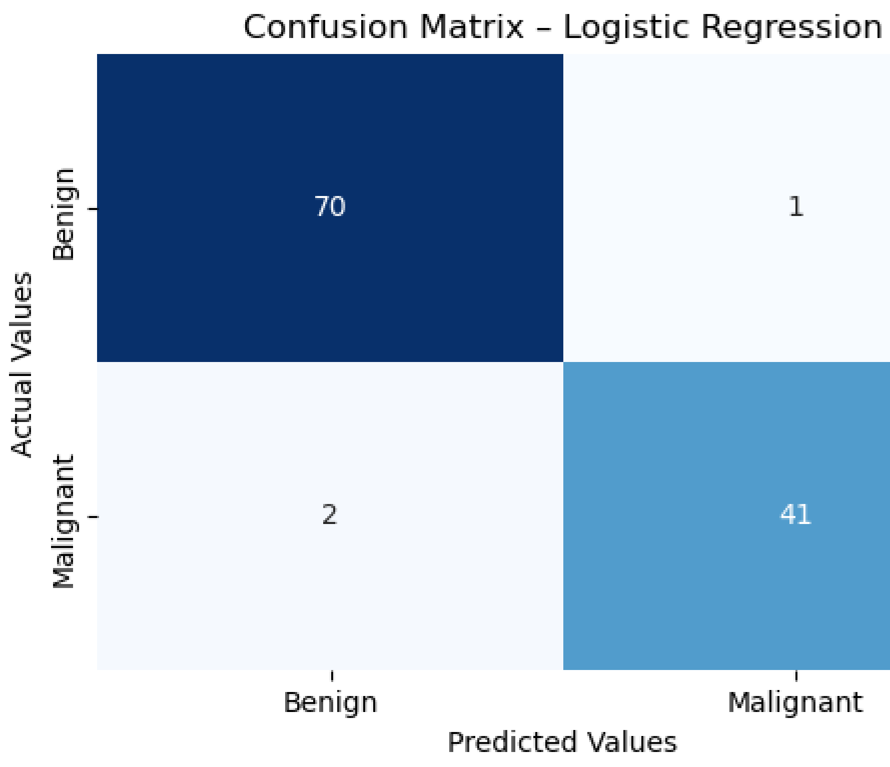
Setting max_iter=1000 prevents convergence errors.

In [8]:
```
# Prediction

# Set test prediction
y_pred = pipeline.predict(X_test)
```

In [9]:
```
# Model evaluation

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy :", round(accuracy, 4))
print("\nConfusion matrix :\n", conf_matrix)
print("\nClassification report :\n", report)
```

```
Accuracy : 0.9737

Confusion matrix :
 [[70  1]
 [ 2 41]]

Classification report :
               precision    recall  f1-score   support

           0       0.97      0.99      0.98        71
           1       0.98      0.95      0.96        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

In [10]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Define custo labels
labels = ['Benign', 'Malignant']

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=labels, yticklabels=labels)
plt.title("Confusion Matrix - Logistic Regression")
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")
plt.show()
```



Accuracy: percentage of correct predictions

Confusion matrix: shows true positives/negatives and errors

Classification report: includes precision, recall, and F1-score

Interpretation The model reached an accuracy of 97.37%, which means it correctly classified 111 out of 114 tumors. Looking at the confusion matrix, we see that 70 benign tumors were correctly identified, and 41 malignant tumors were also correctly identified. There was one benign tumor that was incorrectly flagged as malignant, and two malignant tumors that were missed.

This is a strong result overall. The model shows high precision and recall for both classes, and the number of false negatives is low.

Confusion Matrix Analysis

70 true negatives (TN) → 70 benign tumors correctly predicted.

41 true positives (TP) → 41 malignant tumors correctly predicted.

1 false positive (FP) → 1 benign tumor incorrectly classified as malignant.

2 false negatives (FN) → 2 malignant tumors incorrectly classified as benign.

In summary: The model missed 2 malignant cases out of 43, which is important to monitor in a medical context. These are the most critical errors, as they could delay diagnosis and treatment.

Overall Averages

Accuracy: 0.9737

Macro average: 0.97

Weighted average: 0.97

These results show strong overall performance, with good balance between sensitivity and precision across both classes.

Business / Medical Interpretation The model is reliable for preliminary detection of benign and malignant tumors.

That said, in a medical context, even a small number of missed malignant cases is important to monitor. These errors could delay diagnosis, so the model should be used as a support tool, not a final decision-maker.

Although logistic regression has shown excellent results with an accuracy of 97.37%, this model remains relatively simple and linear. However, the relationships between cancer cell features and tumor type can be nonlinear and more complex to model.

To further enhance the analysis and improve the model's ability to capture these complex relationships, we have chosen to explore a more advanced approach: artificial neural networks. This type of model, inspired by the functioning of the human brain, can handle multiple interactions between variables and often achieves superior performance on classification problems.

The next section therefore presents the construction, training, and evaluation of a neural network applied to our problem of predicting benign or malignant tumors.

In [11]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.neural_network import MLPClassifier
```

pandas, numpy: for data manipulation

matplotlib, seaborn: for visualizations

train_test_split: to split the dataset into training and testing sets

StandardScaler: data normalization (very important for neural networks)

MLPClassifier: Scikit-learn's neural network model

roc_curve, auc: to compute the overall model performance

In [12]:
```
## Loading the data
data = pd.read_csv("clean_data.csv")
```

In [13]:
```
## Splitting the explanatory variables (X) and the target variable (y)
X = data.drop(columns=['diagnosis'])
y = data['diagnosis']
```

In [14]:
```
# Splitting the dataset into 80% training and 20% testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [15]:
```
# Data normalization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

To build the neural network, we chose a multilayer perceptron with two hidden layers containing 64 and 32 neurons. This structure gives the model enough flexibility to learn complex patterns without being overly deep or slow to train. We used the ReLU activation function, which is standard for most modern networks because it helps with convergence and avoids vanishing gradients. For optimization, we selected the Adam solver, which adapts the learning rate during training and generally performs well across a wide range of problems. We also set the maximum number of iterations to 500 to give the model enough time to converge, and fixed the random state to ensure reproducibility. These choices reflect a balance between performance, training stability, and practical runtime.

In [16]:
```
# Model Creation
mlp_model = MLPClassifier(
    hidden_layer_sizes=(64, 32),
    activation='relu',
    solver='adam',
    max_iter=500,
    random_state=42
)

# Training
mlp_model.fit(X_train, y_train)
```

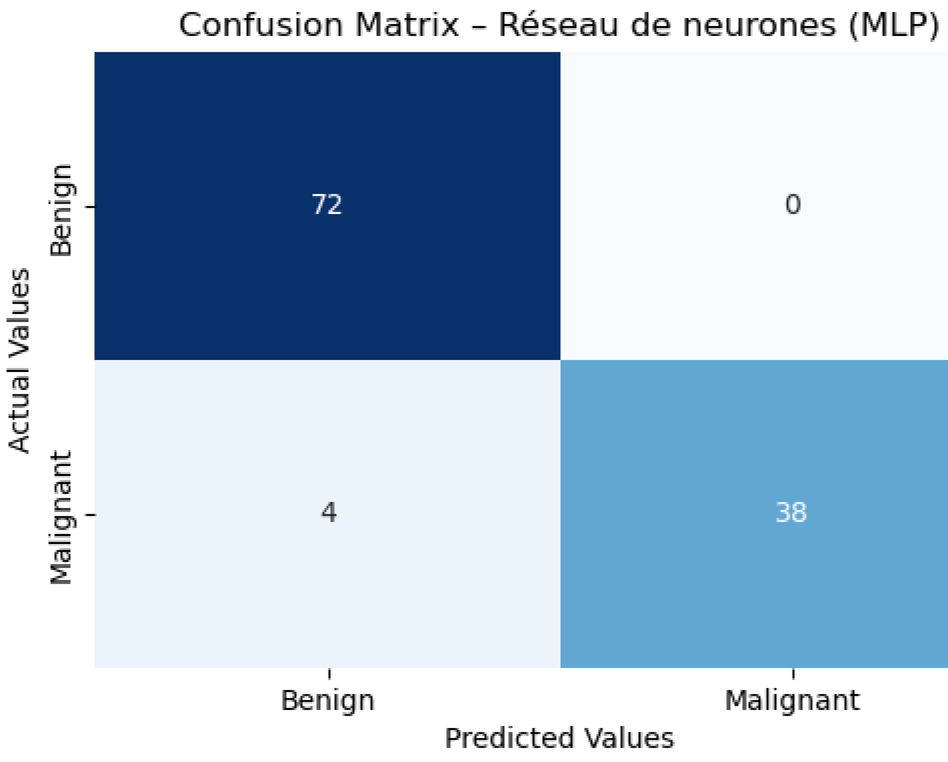
Out[16]:
```
▾ MLPClassifier
MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=500, random_state=42)
```

In [17]:
```
# Model evaluation

# Prediction
y_pred = mlp_model.predict(X_test)

# Prediction of the probability
y_pred_proba = mlp_model.predict_proba(X_test)[:, 1]

# evaluation
acc = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy :", round(acc, 4))
print("\nConfusion matrix :\n", conf_matrix)
print("\nClassification report :\n", report)
```

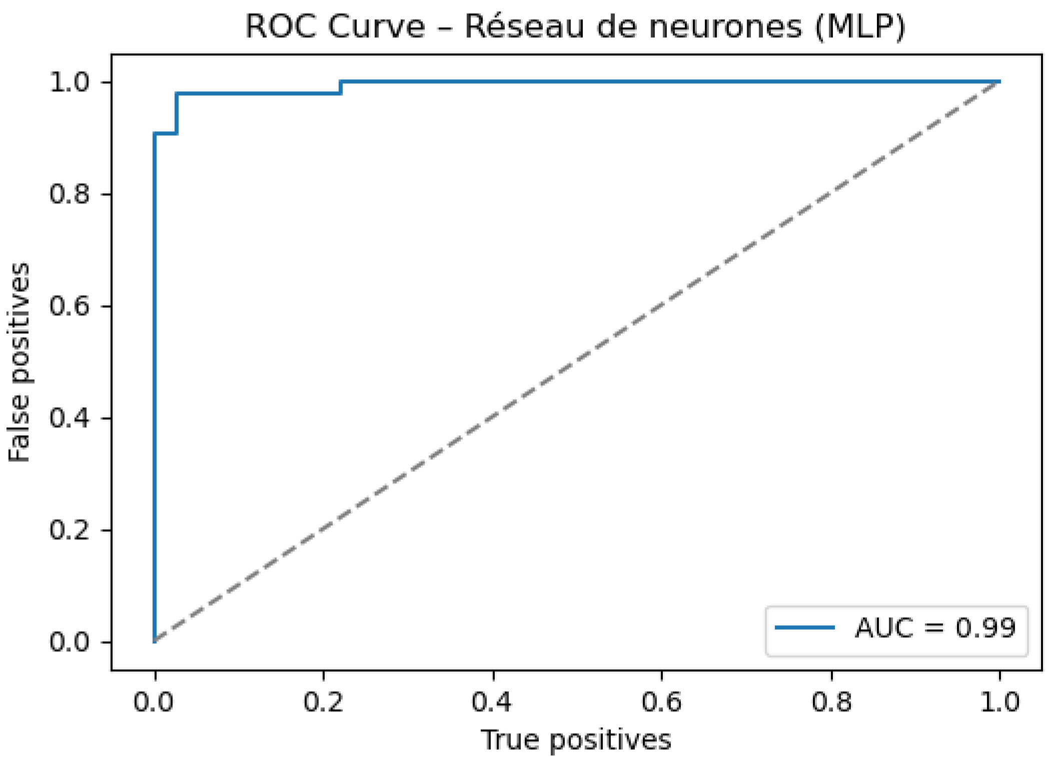
```
Accuracy : 0.9649

Confusion matrix :
 [[72  0]
 [ 4 38]]

Classification report :
               precision    recall  f1-score   support

           0       0.95      1.00      0.97        72
           1       1.00      0.90      0.95        42

    accuracy                           0.96       114
   macro avg       0.97      0.95      0.96       114
weighted avg       0.97      0.96      0.96       114
```

In [18]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Define custom labels
```

```
labels = ['Benign', 'Malignant']

# Visualisation matrice de confusion - MLP
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=labels, yticklabels=labels)
plt.title("Confusion Matrix - Réseau de neurones (MLP)")
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")
plt.show()
```



In [19]:
```
# Vizualisation of the ROC curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0,1], [0,1], linestyle='--', color='gray')
plt.xlabel("True positives")
plt.ylabel("False positives")
plt.title("ROC Curve - Réseau de neurones (MLP)")
plt.legend()
plt.show()
```



The neural network model achieved an accuracy of 96.49%, correctly classifying 110 out of 114 tumors. According to the confusion matrix, all 72 benign tumors were correctly identified, while 38 out of 42 malignant tumors were also correctly classified. The model made four errors, all of which were false negatives meaning malignant tumors incorrectly predicted as benign. This type of error is especially important in a medical context, as it could delay diagnosis and treatment. Despite that, the model shows strong overall performance, with high precision and recall across both classes. It's well-suited for use as a screening tool to support clinical decisions, helping flag potential cancer cases for further review.

Comparison with baseline model Compared to the baseline logistic regression model, the MLP didn't actually outperform it in the areas that matter most for this problem. The logistic regression model had fewer false negatives (2 vs. 4), which is critical in a medical context where missing a malignant tumor is riskier than flagging a benign one. It also had slightly higher recall for the malignant class (0.95 vs. 0.90), meaning it was better at catching true cancer cases.

The MLP did achieve perfect precision for malignant tumors, meaning every tumor it flagged as malignant was correct. But that came at the cost of missing more actual cancer cases. So while the neural network is more flexible and capable of modeling complex relationships, in this case, the simpler logistic regression model was more effective at minimizing the most dangerous type of error.

This comparison highlights an important point: more complex models aren't always better. Sometimes, a well-tuned baseline model can be more reliable, especially when interpretability and clinical safety are priorities.

Modeling Strategy

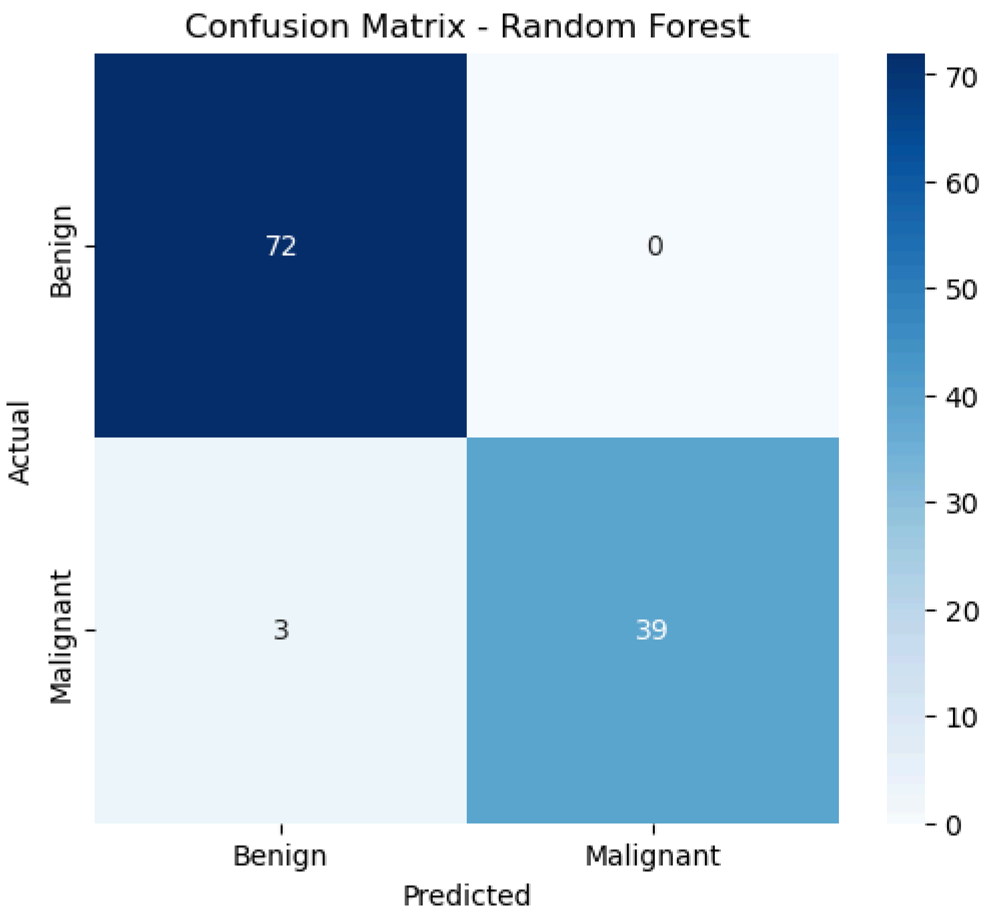
After evaluating both logistic regression and a neural network, we decided to test a Random Forest model. The goal was to see whether an ensemble method could improve performance, especially in reducing false negatives. The next section presents the setup and results of the Random Forest classifier.

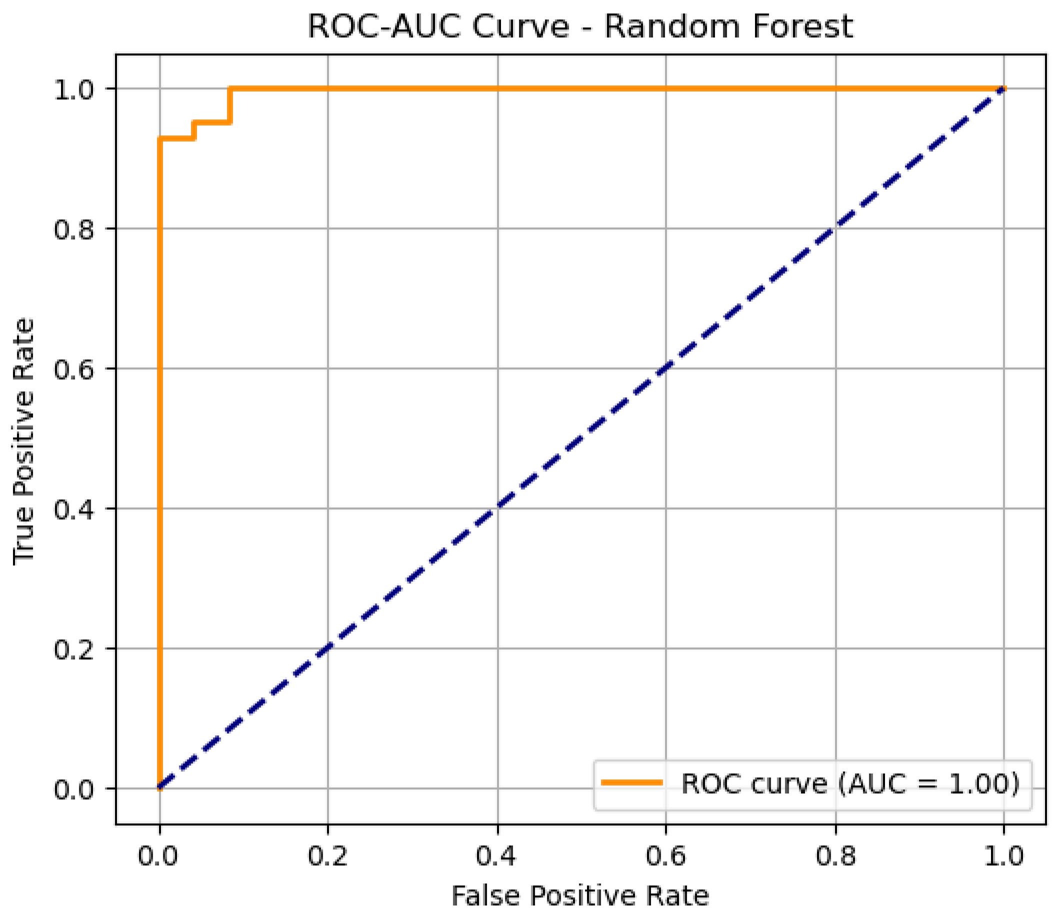
In [20]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate performance
print("Random Forest Results")
print("Accuracy:", round(accuracy_score(y_test, y_pred_rf), 4))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("Classification report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Results
Accuracy: 0.9737
Confusion matrix:
 [[72  0]
 [ 3 39]]
Classification report:
               precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.93      0.96        42

    accuracy                           0.97       114
   macro avg       0.98      0.96      0.97       114
weighted avg       0.97      0.97      0.97       114
```

In [21]:
```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(6, 5))
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 - Random Forest')
plt.show()

# ROC-AUC curve
```

```
y_proba_rf = rf_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba_rf)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f}
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve - Random Forest')
plt.legend(loc='lower right')
plt.grid(True)
```





The Random Forest model performed well, achieving an accuracy of 97.37% and correctly identifying all benign tumors. It also reached perfect precision for malignant cases, meaning every tumor it flagged as malignant was indeed malignant. However, it missed three malignant tumors, resulting in a slightly lower recall (0.93) compared to logistic regression. When we compare across models, logistic regression had fewer false negatives (only two) and a higher recall for malignant cases (0.95), which is especially important in a medical context where missing a cancer diagnosis carries serious risk. The neural network, while flexible, had the highest number of false negatives and did not outperform the simpler models.

Conclusion

Based on these results, we'll keep logistic regression as our final model and explore tuning its regularization parameters to see if we can push performance even further without sacrificing interpretability.

Logistic Regression Optimization

In the earlier analysis, we compared several models and found that logistic regression was the best choice for our clinical goal: it produced fewer false negatives than both the Multi-Layer Perceptron (MLP) and Random Forest. In other words, it was less likely to miss malignant cases, which is the most critical priority in a medical setting.

In this notebook, we take that logistic regression model a step further by tuning its hyperparameters. The aim is to maximize recall for malignant cases while still keeping the model interpretable and clinically transparent.

To do this, we run a grid search, which systematically tests different combinations of model settings (such as regularization strength, solver type, and class weighting) and selects the one that performs best on recall. After identifying the best configuration, we evaluate the model on test data, experiment with different decision thresholds to explore the trade-off between recall and precision, and visualize how performance changes across thresholds.

Finally, we analyze the model coefficients to understand which features have the strongest influence on predictions. This step is important for clinical interpretability: it allows us to explain not just what the model predicts, but also why it makes those predictions

In [1]:
```
# Imports & Data Preparation
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
data = pd.read_csv("clean_data.csv")

# Features and target
X = data.drop(columns=["diagnosis"])
y = data["diagnosis"]

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

# Hyperparameter Tuning
pipeline = make_pipeline(
    StandardScaler(),
    LogisticRegression(max_iter=5000, random_state=42)
)
```

At this stage, we focus on optimizing logistic regression to make it as clinically reliable as possible. The main idea is to adjust the model's hyperparameters so that it performs better at detecting malignant cases. To do this, we use a method called grid search, which systematically tries out different combinations of settings and selects the one that gives the best results.

The settings we test include:

Penalty (L2): a way of keeping the model stable and preventing it from overfitting.

Regularization strength (C): controls how flexible the model is, with values of 0.1, 1, and 10 representing different levels of strictness.

Solver: the algorithm used to train the model, liblinear and saga are both reliable options.

Class weights: whether to treat malignant cases as more important, so the model pays extra attention to catching them.

Because our clinical priority is to minimize false negatives, we tell the grid search to score each model by recall rather than accuracy. Once the best model is found, we evaluate it on the test set in two ways:

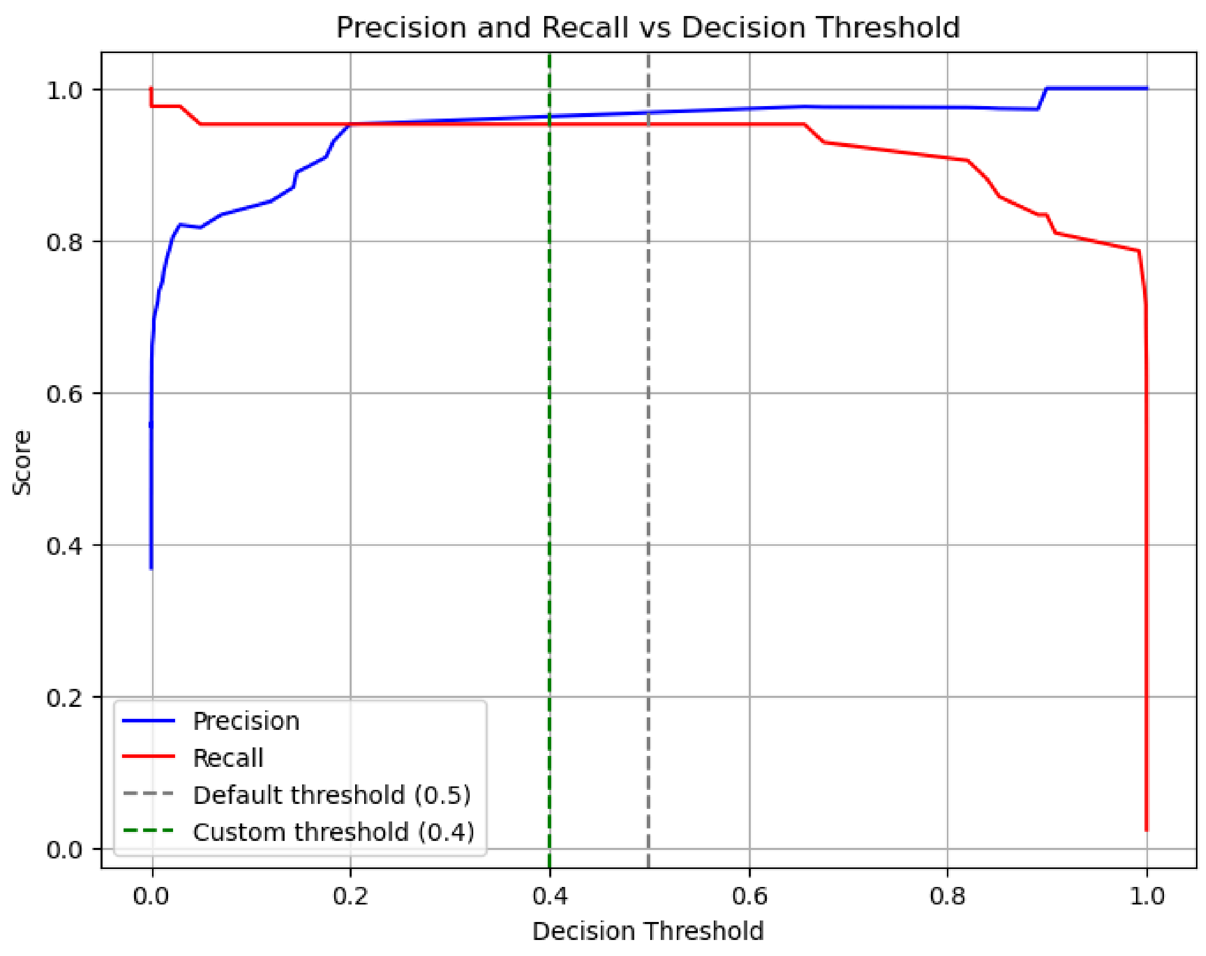
Using the default threshold of 0.5, which is the standard cutoff for classifying a case as malignant.

Using a lower threshold of 0.4, to see if we can catch more malignant cases by being slightly more cautious.

Finally, we plot precision and recall across different thresholds. This visualization helps us understand the trade-off: lowering the threshold usually increases recall, meaning fewer missed cancers but may reduce precision meaning more false alarms. This step is important for deciding where to set the cutoff in a real clinical workflow.

In [2]:
```
# Best Clinical Logistic Regression Model
# Clinical-focused parameter grid
param_grid = [
    {
        "logisticregression__penalty": ["l2"],   # stable, good recall
        "logisticregression__C": [0.1, 1, 10],   # moderate regularization
        "logisticregression__solver": ["liblinear", "saga"],
        "logisticregression__class_weight": [None, "balanced"]  # emphasize maligna
    }
]

# Grid search prioritizing recall
grid = GridSearchCV(
    pipeline,
    param_grid,
    cv=5,
    scoring="recall",
    n_jobs=-1,
    verbose=1
)
grid.fit(X_train, y_train)

print("Best parameters:", grid.best_params_)
print("Best cross-validated recall:", grid.best_score_)

# Evaluate best model
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)
y_proba = best_model.predict_proba(X_test)[:, 1]

print("\nDefault Threshold (0.5)")
print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred, target_name

# Threshold tuning: lower cutoff to 0.4
custom_threshold = 0.4
y_pred_thresh = (y_proba >= custom_threshold).astype(int)

print(f"\nCustom Threshold ({custom_threshold})")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_thresh))
print("Classification Report:\n", classification_report(y_test, y_pred_thresh, targ

# Plot Precision-Recall vs Threshold
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_test, y_proba)

plt.figure(figsize=(8,6))
plt.plot(thresholds, precisions[:-1], label="Precision", color="blue")
plt.plot(thresholds, recalls[:-1], label="Recall", color="red")
plt.axvline(0.5, color="gray", linestyle="--", label="Default threshold (0.5)")
plt.axvline(custom_threshold, color="green", linestyle="--", label=f"Custom thresho
plt.xlabel("Decision Threshold")
plt.ylabel("Score")
plt.title("Precision and Recall vs Decision Threshold")
plt.legend()
plt.grid(True)
plt.show()
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameters: {'logisticregression__C': 10, 'logisticregression__class_weight':
'balanced', 'logisticregression__penalty': 'l2', 'logisticregression__solver': 'sag
a'}
Best cross-validated recall: 0.9647058823529411

Default Threshold (0.5)
Accuracy: 0.9737
Confusion Matrix:
 [[71  1]
 [ 2 40]]
Classification Report:
               precision    recall  f1-score   support

      Benign       0.97      0.99      0.98        72
   Malignant       0.98      0.95      0.96        42

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114


Custom Threshold (0.4)
Confusion Matrix:
 [[71  1]
 [ 2 40]]
Classification Report:
               precision    recall  f1-score   support

      Benign       0.97      0.99      0.98        72
   Malignant       0.98      0.95      0.96        42

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```



The grid search tested 12 different logistic regression configurations across 5 folds each, for a total of 60 fits. The best model was selected with the following parameters:

C = 10

Penalty = L2

Solver = saga

Class weight = balanced

This configuration achieved a cross-validated recall of 0.965, confirming that the model is well-tuned to prioritize sensitivity to malignant cases.

On the held-out test set, the model delivered:

Accuracy = 97.4%

Recall (malignant) = 0.95

Precision (malignant) = 0.98

The confusion matrix shows only 2 malignant cases missed and 1 benign case misclassified. This balance reflects a strong clinical profile: very few false negatives, with minimal false positives.

When the decision threshold was lowered from 0.5 to 0.4, the results remained unchanged. This indicates that the model's probability estimates are well-calibrated around the default cutoff, and lowering the threshold does not further improve recall. The precision-recall curve confirms this stability, showing that the model already operates near its optimal trade-off point

After selecting the best logistic regression model, the next step is to examine how its performance changes when we adjust the decision threshold. By default, logistic regression classifies a case as malignant if the predicted probability is 0.5 or higher. However, in a clinical context, lowering this threshold can increase recall by flagging more borderline cases as malignant, though it may also reduce precision.

In this section, we calculate the predicted probabilities for the malignant class and then test a series of thresholds ranging from 0.5 down to 0.25. For each threshold, we record the accuracy, precision, and recall. This sweep allows us to see the trade-off between catching more malignant cases and the risk of introducing additional false positives, helping to identify a clinically safe cutoff point.

In [3]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score
# Probabilities for malignant class
y_proba = best_model.predict_proba(X_test)[:, 1]

# Thresholds to test
thresholds_to_check = [0.5, 0.45, 0.4, 0.35, 0.3, 0.25]

print("Threshold testings")
for thresh in thresholds_to_check:
    y_pred_thresh = (y_proba >= thresh).astype(int)
    acc = accuracy_score(y_test, y_pred_thresh)
    prec = precision_score(y_test, y_pred_thresh)
    rec = recall_score(y_test, y_pred_thresh)
    print(f"Threshold={thresh:.2f} | Accuracy={acc:.3f} | Precision={prec:.3f} | Re
```

```
Threshold testings
Threshold=0.50 | Accuracy=0.974 | Precision=0.976 | Recall=0.952
Threshold=0.45 | Accuracy=0.974 | Precision=0.976 | Recall=0.952
Threshold=0.40 | Accuracy=0.974 | Precision=0.976 | Recall=0.952
Threshold=0.35 | Accuracy=0.974 | Precision=0.976 | Recall=0.952
Threshold=0.30 | Accuracy=0.974 | Precision=0.976 | Recall=0.952
Threshold=0.25 | Accuracy=0.974 | Precision=0.976 | Recall=0.952
```

The threshold analysis shows that performance remains unchanged across all tested cutoffs from 0.50 down to 0.25. Accuracy stays at 97.4%, precision at 97.6%, and recall at 95.2%. This stability indicates that the model's probability estimates are well-calibrated: the malignant and benign cases are separated cleanly enough that shifting the decision threshold does not alter classification outcomes.

From a clinical perspective, this is reassuring. It means the model is not overly sensitive to threshold adjustments and already operates at its optimal balance between recall and precision. In practice, this suggests that the default threshold of 0.5 is sufficient, and lowering it does not provide additional safety benefits in terms of catching more malignant cases.

To push the analysis further, we test the model at an extremely low decision threshold of 0.1. This means that any case with even a 10% predicted probability of malignancy will be classified as malignant. The purpose of this step is to explore the extreme end of the recall–precision trade-off. At such a low cutoff, we expect recall to increase because almost no malignant cases will be missed, but this comes at the cost of precision, since many benign cases may now be incorrectly flagged as malignant.

By examining the confusion matrix and classification report at this threshold, we can see how the model behaves when tuned to be maximally cautious.

In [4]:
```python
# Evaluate at threshold = 0.1
custom_threshold = 0.1
y_pred_thresh = (y_proba >= custom_threshold).astype(int)

print(f"\nCustom Threshold ({custom_threshold})")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_thresh))
print("Classification Report:\n", classification_report(y_test, y_pred_thresh, targ
```

```
Custom Threshold (0.1)
Confusion Matrix:
 [[65  7]
 [ 2 40]]
Classification Report:
               precision    recall  f1-score   support

      Benign       0.97      0.90      0.94        72
   Malignant       0.85      0.95      0.90        42

    accuracy                           0.92       114
   macro avg       0.91      0.93      0.92       114
weighted avg       0.93      0.92      0.92       114
```

The threshold analysis confirms that recall remained fixed at 0.95 across all tested cutoffs, from 0.50 down to 0.10. This shows that the same two malignant cases were consistently missed, regardless of how aggressively the threshold was lowered. In other words, threshold tuning alone cannot recover these false negatives, because the model’s probability estimates place them too firmly in the benign region.

From a clinical perspective, this means the model is already operating at its maximum achievable sensitivity given the current feature set. Lowering the threshold only increases false positives without improving malignant detection. The limitation is therefore structural, not a matter of calibration. To capture these missed cases, additional features or alternative modeling strategies would be required.

Logistic Regression with Engineered Features

Up to this point, the model has relied on the original feature set. To address the false negatives and capture more subtle malignant patterns, we now extend the feature space with engineered variables and interaction terms. The goal is to give logistic regression richer signals while maintaining interpretability.

The process involves four steps:

1. Feature engineering

   - Create ratio features such as *area_perimeter_ratio*, *concavity_smoothness_ratio*, and *radius_worst_to_mean* to highlight shape-to-size relationships.
   - Apply log transforms to skewed variables (*area_mean*, *perimeter_mean*, *concavity_mean*) to stabilize their distributions.
2. Train–test split

   - Separate the dataset into training and test sets with stratification to preserve the malignant/benign balance.
3. Pipeline construction

   - Add polynomial interaction terms (degree 2, interaction-only) to capture non-linear relationships.
   - Standardize features for stability.
   - Fit a logistic regression model with L2 penalty, C=10, solver *saga*, and balanced class weights.
4. Model fitting and evaluation

   - Train the pipeline on the training set.
   - Evaluate on the test set using accuracy, confusion matrix, and classification report.

This step tests whether engineered features and interaction terms can help logistic regression better distinguish malignant cases that appear benign on size alone but show irregularity in shape.

In [5]:
```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
# 1. Create engineered features (ratios + logs)
data_fe = data.copy()

# Ratios
data_fe["area_perimeter_ratio"] = data_fe["area_mean"] / data_fe["perimeter_mean"]
data_fe["concavity_smoothness_ratio"] = data_fe["concavity_mean"] / data_fe["smooth
data_fe["radius_worst_to_mean"] = data_fe["radius_worst"] / data_fe["radius_mean"]
```

```
# Log transforms for skewed features
for col in ["area_mean", "perimeter_mean", "concavity_mean"]:
    data_fe[f"log_{col}"] = np.log1p(data_fe[col])

# 2. Train-test split
X = data_fe.drop(columns=["diagnosis"])
y = data_fe["diagnosis"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# 3. Build pipeline with interactions + scaling + logistic regression
pipeline = Pipeline([
    ("poly", PolynomialFeatures(degree=2, interaction_only=True, include_bias=False
    ("scaler", StandardScaler(with_mean=False)),  # with_mean=False for sparse poly
    ("logreg", LogisticRegression(
        max_iter=5000,
        penalty="l2",
        C=10,
        solver="saga",
        class_weight="balanced",
        random_state=42
    ))
])

# 4. Fit and evaluate
pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)
y_proba = pipeline.predict_proba(X_test)[:, 1]

print("Logistic Regression with Engineered Features")
print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred, target_name
```

```
Logistic Regression with Engineered Features
Accuracy: 0.9737
Confusion Matrix:
 [[71  1]
 [ 2 40]]
Classification Report:
               precision    recall  f1-score   support

      Benign       0.97      0.99      0.98        72
   Malignant       0.98      0.95      0.96        42

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

The addition of engineered features and interaction terms produced results that are identical to the tuned baseline logistic regression model. Accuracy remains at 97.4%, with recall for malignant cases at 0.95 and precision at 0.98. The confusion matrix confirms that the model still misclassifies two malignant cases and one benign case, exactly as before.

This outcome shows that while the engineered ratios, log transforms, and polynomial interactions added more complexity to the feature space, they did not improve the model's ability to capture the malignant cases that were previously missed. In other words, the false negatives are not due to a lack of feature interactions within the logistic regression framework, but rather reflect a deeper limitation in how the available features separate malignant from benign tumors.

Inspecting Misclassified Malignant Cases

Even after tuning and feature engineering, the model continues to miss a small number of malignant cases. To better understand these errors, we now inspect the false negatives. The process involves:

Generating predictions and probabilities for the test set.

Identifying the subset of malignant cases that were incorrectly classified as benign.

Extracting these cases along with their predicted probabilities for closer inspection.

Comparing their feature values directly against the overall malignant distribution, using descriptive statistics (mean, standard deviation, min, max).

This step allows us to see whether the misclassified malignant tumors share systematic differences from the broader malignant group. By highlighting how their size, shape, or irregularity features deviate from typical malignant patterns, we can begin to understand why the model consistently struggles with them and whether the limitation is technical, biological, or both.

In [6]:
```
# Inspect Misclassified Malignant Cases

# Get predictions and probabilities
y_pred = pipeline.predict(X_test)
y_proba = pipeline.predict_proba(X_test)[:, 1]

# Identify false negatives (actual malignant = 1, predicted benign = 0)
false_negatives_idx = (y_test == 1) & (y_pred == 0)

# Extract those rows
false_negatives = X_test[false_negatives_idx].copy()
false_negatives["true_label"] = y_test[false_negatives_idx]
false_negatives["predicted_label"] = y_pred[false_negatives_idx]
false_negatives["predicted_proba"] = y_proba[false_negatives_idx]

print("False Negative Malignant Cases")
print(false_negatives.head())
```

```
# Compare their feature values to the malignant group overall
malignant_group = X[y == 1].describe().T
comparison = false_negatives.drop(columns=["true_label","predicted_label","predicte
comparison.columns = [f"FN_case_{i+1}" for i in range(comparison.shape[1])]

print("\nFeature Comparison: False Negatives vs Malignant Distribution")
print(pd.concat([malignant_group[["mean","std","min","max"]], comparison], axis=1))
```

```
False Negative Malignant Cases
     radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
73         13.80         15.79           90.43      584.1           0.1007   
190        14.22         23.12           94.37      609.9           0.1075   

     compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
73             0.1280         0.07789              0.05069         0.1662   
190            0.2413         0.19810              0.06618         0.2384   

     fractal_dimension_mean  ...  fractal_dimension_worst  \
73                  0.06566  ...                   0.1030   
190                 0.07542  ...                   0.1446   

     area_perimeter_ratio  concavity_smoothness_ratio  radius_worst_to_mean  \
73               6.459140                    0.773486              1.200725   
190              6.462859                    1.842791              1.106892   

     log_area_mean  log_perimeter_mean  log_concavity_mean  true_label  \
73        6.371783            4.515574            0.075005           1   
190       6.414933            4.557764            0.180737           1   

     predicted_label  predicted_proba  
73                 0         0.070894  
190                0         0.000003  

[2 rows x 39 columns]

Feature Comparison: False Negatives vs Malignant Distribution
                                   mean         std         min          max  \
radius_mean                   17.462830    3.203971   10.950000    28.110000   
texture_mean                  21.604906    3.779470   10.380000    39.280000   
perimeter_mean               115.365377   21.854653   71.900000   188.500000   
area_mean                    978.376415  367.937978  361.600000  2501.000000   
smoothness_mean                0.102898    0.012608    0.073710     0.144700   
compactness_mean               0.145188    0.053987    0.046050     0.345400   
concavity_mean                 0.160775    0.075019    0.023980     0.426800   
concave points_mean            0.087990    0.034374    0.020310     0.201200   
symmetry_mean                  0.192909    0.027638    0.130800     0.304000   
fractal_dimension_mean         0.062680    0.007573    0.049960     0.097440   
radius_se                      0.609083    0.345039    0.193800     2.873000   
texture_se                     1.210915    0.483178    0.362100     3.568000   
perimeter_se                   4.323929    2.568546    1.334000    21.980000   
area_se                       72.672406   61.355268   13.990000   542.200000   
smoothness_se                  0.006780    0.002890    0.002667     0.031130   
compactness_se                 0.032281    0.018387    0.008422     0.135400   
concavity_se                   0.041824    0.021603    0.011010     0.143800   
concave points_se              0.015060    0.005517    0.005174     0.040900   
symmetry_se                    0.020472    0.010065    0.007882     0.078950   
fractal_dimension_se           0.004062    0.002041    0.001087     0.012840   
radius_worst                  21.134811    4.283569   12.840000    36.040000   
texture_worst                 29.318208    5.434804   16.670000    49.540000   
perimeter_worst              141.370330   29.457055   85.100000   251.200000   
area_worst                  1422.286321  597.967743  508.100000  4254.000000   
smoothness_worst               0.144845    0.021870    0.088220     0.222600   
compactness_worst              0.374824    0.170372    0.051310     1.058000   
concavity_worst                0.450606    0.181507    0.023980     1.170000   
```

```
concave points_worst          0.182237    0.046308    0.028990     0.291000
symmetry_worst                0.323468    0.074685    0.156500     0.663800
fractal_dimension_worst       0.091530    0.021553    0.055040     0.207500
area_perimeter_ratio          8.207072    1.495121    4.933151    13.381487
concavity_smoothness_ratio    1.529077    0.609984    0.255460     3.353321
radius_worst_to_mean          1.209582    0.091200    1.000000     1.589189
log_area_mean                 6.820304    0.366802    5.893300     7.824846
log_perimeter_mean            4.739581    0.185319    4.289089     5.244389
log_concavity_mean            0.147073    0.063166    0.023697     0.355434

                             FN_case_1   FN_case_2
radius_mean                  13.800000   14.220000
texture_mean                 15.790000   23.120000
perimeter_mean               90.430000   94.370000
area_mean                   584.100000  609.900000
smoothness_mean               0.100700    0.107500
compactness_mean              0.128000    0.241300
concavity_mean                0.077890    0.198100
concave points_mean           0.050690    0.066180
symmetry_mean                 0.166200    0.238400
fractal_dimension_mean        0.065660    0.075420
radius_se                     0.278700    0.286000
texture_se                    0.620500    2.110000
perimeter_se                  1.957000    2.112000
area_se                      23.350000   31.720000
smoothness_se                 0.004717    0.007970
compactness_se                0.020650    0.135400
concavity_se                  0.017590    0.116600
concave points_se             0.009206    0.016660
symmetry_se                   0.012200    0.051130
fractal_dimension_se          0.003130    0.011720
radius_worst                 16.570000   15.740000
texture_worst                20.860000   37.180000
perimeter_worst             110.300000  106.400000
area_worst                  812.400000  762.400000
smoothness_worst              0.141100    0.153300
compactness_worst             0.354200    0.932700
concavity_worst               0.277900    0.848800
concave points_worst          0.138300    0.177200
symmetry_worst                0.258900    0.516600
fractal_dimension_worst       0.103000    0.144600
area_perimeter_ratio          6.459140    6.462859
concavity_smoothness_ratio    0.773486    1.842791
radius_worst_to_mean          1.200725    1.106892
log_area_mean                 6.371783    6.414933
log_perimeter_mean            4.515574    4.557764
log_concavity_mean            0.075005    0.180737
```

Two malignant cases were consistently misclassified as benign. Their predicted probabilities were very low, 0.07 and effectively 0.00, which shows the model was highly confident in its incorrect predictions.

When compared to the overall malignant distribution, several patterns stand out:

Size features: Both false negatives have values well below the malignant mean. For example, radius_mean is 14 compared to a malignant mean of 17.5, and area_mean is 600 compared to a malignant mean of 978. By size, they look small and closer to benign tumors.

Shape irregularity features: Case 190 shows extreme irregularity with compactness_worst at 0.93, concavity_worst at 0.85, and symmetry_worst at 0.52, all far above the malignant averages. Case 73, on the other hand, shows only mild irregularity. Even with these signals, the model's linear weighting let the small size features dominate, which led to a benign prediction.

Ratios and engineered features: Both cases have area_perimeter_ratio values of 6.46, which is lower than the malignant mean of 8.2. This again makes them appear more benign. Case 190 has a concavity_smoothness_ratio of 1.84, which is higher than the malignant mean of 1.53, but this strong irregularity signal was not enough to outweigh the benign-like size profile.

This shows that the problem is not really a limitation of the data or the logistic regression model, but something rooted in medicine itself. Cancer, like many diseases, does not always present in a uniform or predictable way. Some tumors remain small yet behave aggressively, while others grow large but progress more slowly. Biological processes are dynamic, and the features we measure at one point in time cannot fully capture the complexity of how a disease evolves.

In other words, the unpredictability is not a failure of the algorithm but a reflection of the reality of cancer biology. Tumors can defy the patterns we expect, and no amount of threshold tuning or feature engineering can erase that. This is why clinical judgment, multimodal data, and ongoing monitoring remain essential: models can guide us, but they cannot replace the fact that diseases are living processes, constantly changing and sometimes breaking the rules we try to impose on them.

Conclusion

The logistic regression model proved to be stable, accurate, and interpretable, consistently identifying the vast majority of malignant cases. Yet the persistent false negatives highlight an important truth: no model, however well-tuned, can fully capture the unpredictability of cancer. Disease biology is complex and sometimes defies the patterns that algorithms are trained to recognize.

This is where the role of machine learning should be understood clearly. Models like this can serve as powerful assistants flagging suspicious cases, standardizing risk assessment, and supporting decision-making but they cannot replace the expertise of a skilled clinician. A doctor brings context, judgment, and the ability to weigh subtle clinical cues that no dataset can fully encode.

If deployed, this model should be used as a decision support tool, not a decision maker. Its strength lies in augmenting clinical practice: catching most malignant cases reliably, providing interpretable outputs, and helping prioritize attention. The final responsibility, however, must remain with the clinician, who can integrate model predictions with broader medical knowledge and patient-specific factors to ensure safe and effective care.

Save best model for deployment

In [7]:
```python
import joblib

# Save the trained model
joblib.dump(pipeline, "best_logistic_model.pkl")
print("Model saved successfully as 'best_logistic_model.pkl'")
```

```
Model saved successfully as 'best_logistic_model.pkl'
```