MidtermReport

March 10, 2025

1 Analysis of the Diagnostic Wisconsin Breast Cancer Dataset

The dataset I chose is the Wisconsin Breast Cancer Dataset sourced from Hugging Face (https://huggingface.co/datasets/wwydmanski/wisconsin-breast-cancer). This project is a binary classification task aimed at predicting whether a breast tumor is benign (B) or malignant (M). The goal of my analysis is to accurately classify breast tumors based on the features of the cell nuclei present in the breast mass and to identify which features are most indicative of malignancy. Given the binary nature of this problem, I selected logistic regression for modeling due to its high interpretability.

1.1 Dataset Background

The dataset consists of 32 columns, including a patient ID and the response variable, Diagnosis. The remaining 30 predictors are continuos variables, computed from a digitized image of a fine-needle aspirate (FNA) of a breast mass. They describe the characteristics of the cell nuclei. Although only 10 underlying features are measured (such as radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension), each feature is summarized using three statistics:

Mean: The average value across all cell nuclei in the tumor.

Standard Error: The variability or uncertainty in the measurements.

"Worst" Value: The average of the three largest values, highlighting the most extreme measurements.

This approach yields $10 \times 3 = 30$ predictors, providing a comprehensive view of the central tendency, variability, and extremity of the measurements—factors that are critical for distinguishing between benign and malignant tumors. The response variable, Diagnosis, is a categorical variable indicating whether the breast mass is benign (B) or malignant (M).

1.2 Data Cleaning and Set Up

Before starting the preprocessing steps, let's import the necessary libraries for our analysis.

[35]:
```
!pip install --upgrade huggingface_hub
```

```
Requirement already satisfied: huggingface_hub in
/usr/local/lib/python3.11/dist-packages (0.29.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-
packages (from huggingface_hub) (3.17.0)
```

```
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.11/dist-packages (from huggingface_hub) (2024.10.0)
Requirement already satisfied: packaging>=20.9 in
/usr/local/lib/python3.11/dist-packages (from huggingface_hub) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-
packages (from huggingface_hub) (6.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-
packages (from huggingface_hub) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-
packages (from huggingface_hub) (4.67.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.11/dist-packages (from huggingface_hub) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->huggingface_hub) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
packages (from requests->huggingface_hub) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->huggingface_hub) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->huggingface_hub)
(2025.1.31)
```

[36]:
```
import pandas as pd                       # For data manipulation and analysis
import numpy as np                        # For numerical operations and array␣
 ↪handling
import matplotlib.pyplot as plt           # For plotting and visualization
import seaborn as sns                     # For enhanced statistical visualizations
import math                               # For additional mathematical functions
from sklearn.preprocessing import StandardScaler   # For standardizing␣
 ↪continuous features
import statsmodels.api as sm              # For building statistical models (e.g.,␣
 ↪logistic regression)
from statsmodels.stats.outliers_influence import variance_inflation_factor   #␣
 ↪To compute VIF and detect multicollinearity
import sklearn.model_selection as skm   # For splitting data and cross-validation
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,␣
 ↪roc_auc_score  # For model evaluation
from huggingface_hub import notebook_login  # For logging into the Hugging Face␣
 ↪Hub to access datasets/models
```

The dataset was already split into train and test so we import both separately. The train dataset contains 425 observations and the test dataset contains 142 observations.

[38]:
```
df_train = pd.read_csv("hf://datasets/wwydmanski/wisconsin-breast-cancer/train.
 ↪csv")
df_test = pd.read_csv("hf://datasets/wwydmanski/wisconsin-breast-cancer/test.
 ↪csv")
```

[39]:
```python
print("Train Dataset after importing:")
df_train.copy().head()
```

```
Train Dataset after importing:
```

[39]:
```
   Unnamed: 0      0      1       2      3        4        5        6  \
0           0  11.80  17.26   75.26  431.9  0.09087  0.06232  0.02853
1           1  15.75  20.25  102.60  761.3  0.10250  0.12040  0.11470
2           2  14.76  14.74   94.87  668.7  0.08875  0.07780  0.04608
3           3  14.96  19.10   97.03  687.3  0.08992  0.09823  0.05940
4           4  13.43  19.63   85.84  565.4  0.09048  0.06288  0.05858

         7       8  …     21     22      23      24      25      26  \
0  0.01638  0.1847  …  24.49   86.0   562.0  0.1244  0.1726  0.1449
1  0.06462  0.1935  …  30.29  125.9  1088.0  0.1552  0.4480  0.3976
2  0.03528  0.1521  …  17.93  114.2   880.8  0.1220  0.2009  0.2151
3  0.04819  0.1879  …  26.19  109.1   809.8  0.1313  0.3030  0.1804
4  0.03438  0.1598  …  29.87  116.6   993.6  0.1401  0.1546  0.2644

        27      28       29  y
0  0.05356  0.2779  0.08121  B
1  0.14790  0.3993  0.10640  M
2  0.12510  0.3109  0.08187  B
3  0.14890  0.2962  0.08472  B
4  0.11600  0.2884  0.07371  M

[5 rows x 32 columns]
```

[40]:
```python
print("Test Dataset after importing:")
df_test.copy().head()
```

```
Test Dataset after importing:
```

[40]:
```
   Unnamed: 0      0      1       2       3        4        5        6  \
0           0  11.94  18.24   75.71   437.6  0.08261  0.04751  0.01972
1           1  19.40  23.50  129.10  1155.0  0.10270  0.15580  0.20490
2           2  12.18  14.08   77.25   461.4  0.07734  0.03212  0.01123
3           3  11.68  16.17   75.49   420.5  0.11280  0.09263  0.04279
4           4  15.34  14.26  102.50   704.4  0.10730  0.21350  0.20770

          7       8  …     21      22      23      24       25       26  \
0  0.013490  0.1868  …  21.33   83.67   527.2  0.1144  0.08906  0.09203
1  0.088860  0.1978  …  30.53  144.90  1417.0  0.1463  0.29680  0.34580
2  0.005051  0.1673  …  16.47   81.60   513.1  0.1001  0.05332  0.04116
3  0.031320  0.1853  …  21.59   86.57   549.8  0.1526  0.14770  0.14900
4  0.097560  0.2521  …  19.08  125.10   980.9  0.1390  0.59540  0.63050
```

```
        27      28       29  y
0  0.06296  0.2785  0.07408  B
1  0.15640  0.2920  0.07614  M
2  0.01852  0.2293  0.06037  B
3  0.09815  0.2804  0.08024  B
4  0.23930  0.4667  0.09946  M

[5 rows x 32 columns]
```

After we have imported our datasets, the first step is to clean them up before we begin preprocessing. This involves removing unnecessary columns, converting categorical response variables to numerical values and renaming columns to be more descritpive if needed.

For example, we remove the patient ID column from both the train and test datasets since it provides no valuable information for predicting the diagnosis.

[41]:
```
df_train.drop(columns=["Unnamed: 0"],inplace=True)
df_test.drop(columns=["Unnamed: 0"],inplace=True)
```

Since the dataset's columns were initially labeled with numbers, we use the mapping provided on the original website to rename them with their actual descriptive names, making the data easier to understand and work with.

[42]:
```
new_column_names = [
    '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','diagnosis'
]
df_train.columns = new_column_names
df_test.columns = new_column_names
```

We also convert the categorical response variable, Diagnosis to binary values with 0 representing B (benign) and 1 representing M (malignant)

[43]:
```
df_train['diagnosis'] = df_train['diagnosis'].map({'B': 0, 'M': 1})
df_test['diagnosis'] = df_test['diagnosis'].map({'B': 0, 'M': 1})
```

[44]:
```
print("Train Dataset after data cleaning:")
df_train.head()
```

```
Train Dataset after data cleaning:
```

[44]:

```
   radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
0        11.80         17.26           75.26      431.9          0.09087
1        15.75         20.25          102.60      761.3          0.10250
2        14.76         14.74           94.87      668.7          0.08875
3        14.96         19.10           97.03      687.3          0.08992
4        13.43         19.63           85.84      565.4          0.09048

   compactness_mean  concavity_mean  concave_points_mean  symmetry_mean  \
0           0.06232         0.02853              0.01638         0.1847
1           0.12040         0.11470              0.06462         0.1935
2           0.07780         0.04608              0.03528         0.1521
3           0.09823         0.05940              0.04819         0.1879
4           0.06288         0.05858              0.03438         0.1598

   fractal_dimension_mean  …  texture_worst  perimeter_worst  area_worst  \
0                 0.06019  …          24.49             86.0       562.0
1                 0.06303  …          30.29            125.9      1088.0
2                 0.05912  …          17.93            114.2       880.8
3                 0.05852  …          26.19            109.1       809.8
4                 0.05671  …          29.87            116.6       993.6

   smoothness_worst  compactness_worst  concavity_worst  concave_points_worst  \
0            0.1244             0.1726           0.1449               0.05356
1            0.1552             0.4480           0.3976               0.14790
2            0.1220             0.2009           0.2151               0.12510
3            0.1313             0.3030           0.1804               0.14890
4            0.1401             0.1546           0.2644               0.11600

   symmetry_worst  fractal_dimension_worst  diagnosis
0          0.2779                  0.08121          0
1          0.3993                  0.10640          1
2          0.3109                  0.08187          0
3          0.2962                  0.08472          0
4          0.2884                  0.07371          1

[5 rows x 31 columns]
```

[45]:

```
print("Test Dataset after data cleaning:")
df_test.head()
```

```
Test Dataset after data cleaning:
```

[45]:

```
   radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
0        11.94         18.24           75.71      437.6          0.08261
1        19.40         23.50          129.10     1155.0          0.10270
2        12.18         14.08           77.25      461.4          0.07734
3        11.68         16.17           75.49      420.5          0.11280
```

```
4        15.34         14.26          102.50      704.4           0.10730

   compactness_mean  concavity_mean  concave_points_mean  symmetry_mean  \
0           0.04751         0.01972             0.013490         0.1868
1           0.15580         0.20490             0.088860         0.1978
2           0.03212         0.01123             0.005051         0.1673
3           0.09263         0.04279             0.031320         0.1853
4           0.21350         0.20770             0.097560         0.2521

   fractal_dimension_mean  …  texture_worst  perimeter_worst  area_worst  \
0                 0.06110  …          21.33            83.67       527.2
1                 0.06000  …          30.53           144.90      1417.0
2                 0.05649  …          16.47            81.60       513.1
3                 0.06401  …          21.59            86.57       549.8
4                 0.07032  …          19.08           125.10       980.9

   smoothness_worst  compactness_worst  concavity_worst  concave_points_worst  \
0            0.1144            0.08906          0.09203               0.06296
1            0.1463            0.29680          0.34580               0.15640
2            0.1001            0.05332          0.04116               0.01852
3            0.1526            0.14770          0.14900               0.09815
4            0.1390            0.59540          0.63050               0.23930

   symmetry_worst  fractal_dimension_worst  diagnosis
0          0.2785                  0.07408          0
1          0.2920                  0.07614          1
2          0.2293                  0.06037          0
3          0.2804                  0.08024          0
4          0.4667                  0.09946          1

[5 rows x 31 columns]
```

1.3 Exploratory Data Analysis on Raw Data

In this phase, we use tools like correlation heatmaps and summary statistics to uncover patterns and relationships in the raw data. This helps us understand how the predictors interact, highlights key features that differentiate benign from malignant tumors, and guides our subsequent modeling decisions.

1.3.1 Summary Statistics

For the train dataset, we compute the summary statistics for each feature, grouped by diagnosis.

[46]:
```python
df_explore = df_train.copy()
group_stats = df_explore.groupby('diagnosis').describe().T
pd.set_option('display.max_rows', None)
print(group_stats)
```

diagnosis		0	1
radius_mean	count	267.000000	159.000000
	mean	12.133318	17.357673
	std	1.838034	3.247821
	min	6.981000	10.950000
	25%	10.965000	15.070000
	50%	12.250000	17.190000
	75%	13.290000	19.445000
	max	17.850000	28.110000
texture_mean	count	267.000000	159.000000
	mean	17.732921	21.469811
	std	3.849747	3.699337
	min	9.710000	10.380000
	25%	15.075000	19.325000
	50%	17.270000	21.350000
	75%	19.430000	23.655000
	max	30.720000	33.560000
perimeter_mean	count	267.000000	159.000000
	mean	77.980637	114.729182
	std	12.132929	22.221102
	min	43.790000	71.900000
	25%	70.440000	98.710000
	50%	78.310000	112.400000
	75%	85.840000	128.200000
	max	114.600000	188.500000
area_mean	count	267.000000	159.000000
	mean	462.310487	969.093082
	std	138.718787	376.958644
	min	143.500000	361.600000
	25%	369.150000	708.100000
	50%	462.000000	928.300000
	75%	546.350000	1168.000000
	max	992.100000	2501.000000
smoothness_mean	count	267.000000	159.000000
	mean	0.092745	0.102986
	std	0.013295	0.012927
	min	0.064290	0.073710
	25%	0.083085	0.093940
	50%	0.091360	0.102600
	75%	0.100700	0.110950
	max	0.163400	0.144700
compactness_mean	count	267.000000	159.000000
	mean	0.079953	0.145736
	std	0.032982	0.056200
	min	0.023440	0.046050
	25%	0.055930	0.110450
	50%	0.075250	0.131400
	75%	0.097050	0.170950

	max	0.223900	0.345400
concavity_mean	count	267.000000	159.000000
	mean	0.044660	0.160645
	std	0.040951	0.078867
	min	0.000000	0.023980
	25%	0.020540	0.110000
	50%	0.036140	0.147900
	75%	0.059100	0.198050
	max	0.410800	0.426800
concave_points_mean	count	267.000000	159.000000
	mean	0.025270	0.086924
	std	0.015504	0.034916
	min	0.000000	0.020310
	25%	0.014175	0.064465
	50%	0.023440	0.085430
	75%	0.032300	0.102150
	max	0.085340	0.201200
symmetry_mean	count	267.000000	159.000000
	mean	0.173102	0.192105
	std	0.023870	0.027266
	min	0.106000	0.130800
	25%	0.157650	0.173500
	50%	0.170900	0.187600
	75%	0.188200	0.211050
	max	0.254800	0.290600
fractal_dimension_mean	count	267.000000	159.000000
	mean	0.063020	0.062825
	std	0.006809	0.007880
	min	0.051850	0.049960
	25%	0.058500	0.056585
	50%	0.061550	0.061770
	75%	0.066120	0.067415
	max	0.095020	0.097440
radius_se	count	267.000000	159.000000
	mean	0.280735	0.608952
	std	0.104480	0.380426
	min	0.111500	0.193800
	25%	0.207150	0.339250
	50%	0.257700	0.507900
	75%	0.339850	0.762150
	max	0.824500	2.873000
texture_se	count	267.000000	159.000000
	mean	1.201687	1.198211
	std	0.609264	0.480003
	min	0.360200	0.362100
	25%	0.767450	0.874300
	50%	1.081000	1.127000
	75%	1.490500	1.418000

	max	4.885000	3.568000
perimeter_se	count	267.000000	159.000000
	mean	1.986043	4.362019
	std	0.736826	2.838921
	min	0.757000	1.334000
	25%	1.443500	2.555000
	50%	1.893000	3.498000
	75%	2.378500	5.446500
	max	5.004000	21.980000
area_se	count	267.000000	159.000000
	mean	20.813891	73.785723
	std	8.099663	68.715574
	min	6.802000	13.990000
	25%	15.345000	33.450000
	50%	19.630000	53.910000
	75%	24.650000	95.105000
	max	50.950000	542.200000
smoothness_se	count	267.000000	159.000000
	mean	0.007229	0.006930
	std	0.003015	0.003159
	min	0.001713	0.002667
	25%	0.005241	0.005258
	50%	0.006635	0.006369
	75%	0.008532	0.008034
	max	0.020750	0.031130
compactness_se	count	267.000000	159.000000
	mean	0.021106	0.032502
	std	0.015645	0.019792
	min	0.003012	0.008422
	25%	0.011280	0.019335
	50%	0.015870	0.028210
	75%	0.025960	0.038570
	max	0.095860	0.135400
concavity_se	count	267.000000	159.000000
	mean	0.025190	0.042170
	std	0.031288	0.023293
	min	0.000000	0.011010
	25%	0.010890	0.026810
	50%	0.018310	0.035820
	75%	0.030320	0.049775
	max	0.396000	0.143800
concave_points_se	count	267.000000	159.000000
	mean	0.009652	0.014967
	std	0.005539	0.005689
	min	0.000000	0.005174
	25%	0.006467	0.011210
	50%	0.008747	0.014240
	75%	0.011655	0.017545

	max	0.052790	0.040900
symmetry_se	count	267.000000	159.000000
	mean	0.020338	0.020376
	std	0.006798	0.010984
	min	0.009539	0.007882
	25%	0.015555	0.014145
	50%	0.018940	0.017050
	75%	0.023510	0.021665
	max	0.061460	0.078950
fractal_dimension_se	count	267.000000	159.000000
	mean	0.003629	0.004077
	std	0.002916	0.002131
	min	0.000895	0.001087
	25%	0.002099	0.002644
	50%	0.002808	0.003739
	75%	0.004186	0.004985
	max	0.029840	0.012840
radius_worst	count	267.000000	159.000000
	mean	13.381891	21.001635
	std	2.057990	4.414515
	min	7.930000	12.840000
	25%	11.945000	17.675000
	50%	13.450000	20.380000
	75%	14.800000	23.705000
	max	19.820000	36.040000
texture_worst	count	267.000000	159.000000
	mean	23.283184	29.246415
	std	5.381882	5.590107
	min	12.020000	16.670000
	25%	19.515000	25.695000
	50%	22.750000	28.640000
	75%	26.435000	32.210000
	max	41.610000	49.540000
perimeter_worst	count	267.000000	159.000000
	mean	87.084831	140.945975
	std	14.022016	30.536436
	min	50.410000	87.220000
	25%	77.350000	119.000000
	50%	87.240000	134.900000
	75%	96.900000	159.300000
	max	127.100000	251.200000
area_worst	count	267.000000	159.000000
	mean	559.823221	1412.411321
	std	170.102236	625.130303
	min	185.200000	508.100000
	25%	438.300000	935.550000
	50%	551.300000	1272.000000
	75%	673.550000	1679.000000

	max	1210.000000	4254.000000
smoothness_worst	count	267.000000	159.000000
	mean	0.125552	0.146174
	std	0.019287	0.022300
	min	0.071170	0.088220
	25%	0.111800	0.133000
	50%	0.125600	0.144200
	75%	0.137650	0.156800
	max	0.190200	0.222600
compactness_worst	count	267.000000	159.000000
	mean	0.183530	0.378121
	std	0.092653	0.176058
	min	0.027290	0.051310
	25%	0.114400	0.257150
	50%	0.169800	0.356800
	75%	0.232750	0.447900
	max	0.584900	1.058000
concavity_worst	count	267.000000	159.000000
	mean	0.162991	0.452446
	std	0.130330	0.188685
	min	0.000000	0.023980
	25%	0.078115	0.334100
	50%	0.138400	0.402900
	75%	0.210800	0.547400
	max	0.821600	1.170000
concave_points_worst	count	267.000000	159.000000
	mean	0.074317	0.181331
	std	0.035725	0.045913
	min	0.000000	0.028990
	25%	0.051315	0.152350
	50%	0.075300	0.182000
	75%	0.098275	0.209300
	max	0.159900	0.291000
symmetry_worst	count	267.000000	159.000000
	mean	0.270262	0.321391
	std	0.041503	0.075356
	min	0.156600	0.156500
	25%	0.242050	0.274950
	50%	0.267700	0.310900
	75%	0.299450	0.356100
	max	0.412800	0.663800
fractal_dimension_worst	count	267.000000	159.000000
	mean	0.079972	0.092037
	std	0.013842	0.022270
	min	0.055210	0.055040
	25%	0.070260	0.077380
	50%	0.077730	0.089020
	75%	0.085510	0.102800

```
max       0.148600    0.207500
```

Based on the summary statistics, these are the key findings observed:

1. Size and Shape Differences:

- Radius, Perimeter, and Area: Malignant tumors (diagnosis = 1) have significantly higher mean values for radius, perimeter, and area compared to benign tumors. For example, the mean radius is about 12.13 for benign cases and 17.36 for malignant ones, and the mean area nearly doubles.
- Implication: Larger size measurements are strong indicators of malignancy and suggest that these features can play an important role in differentiating tumor types.

2. Structural Characteristics:

- Compactness, Concavity, and Concave Points: The mean values for compactness, concavity, and concave points are significantly higher in malignant tumors. This indicates that malignant tumors tend to have more irregular or indented shapes.
- Implication: These features capture the complexity of tumor borders and irregularities, which are often associated with cancerous growth.

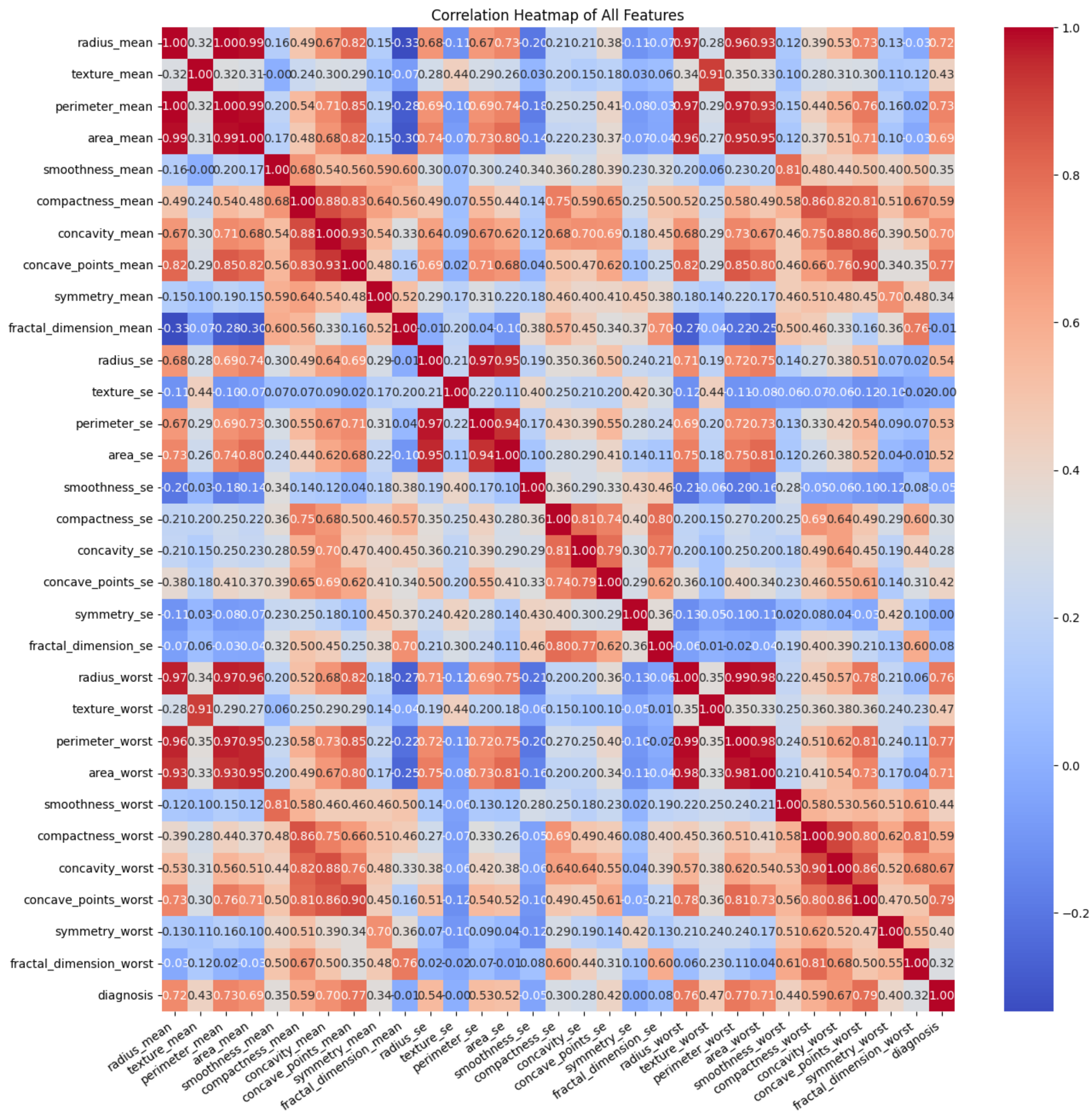
3. Consistency Across Summary Statistics:

- The pattern of malignant tumors having higher measurement values is consistent across the "mean," "standard error," and "worst" measurements for many features.
- Implication: This consistency reinforces that the differences in these features are robust and should be highly informative for differentiating between benign and malignant cases.

1.3.2 Correlation Heatmap

A correlation heatmap visualizes how strongly each pair of predictors in the dataset is related, with the color intensity reflecting the strength of the correlation. We generate a correlation heatmap for the training dataset to illustrate these relationships.

[47]:
```python
corr_matrix = df_explore.corr()
plt.figure(figsize=(15, 15))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.xticks(rotation=35, ha='right')
plt.yticks(rotation=0)
plt.title("Correlation Heatmap of All Features")
plt.show()
```



Legend:

Deep Red (close to +1): Indicates a strong positive correlation.

Pale (close to 0): Indicates little to no correlation.

Deep Blue (close to -1): Indicates a strong negative correlation.

Based on the correlation heatmap, these are the key findings observed:

1. Size vs. Shape Features

- Radius, perimeter, and area (size-based features) show strong correlations with each other overall, but their mean, SE, and worst values are less tightly interlinked. In other words, a tumor's average size can be significantly different from its worst measurements or variability.

- However, compactness, concavity, and concave points (shape-based features) are strongly correlated within each category (mean, SE, worst). If a tumor is highly irregular on average, it typically also has high variability and extreme values in those same shape characteristics.

2. Texture and Fractal Dimension

- Texture_mean and texture_worst have a moderate-to-strong correlation, suggesting these measurements overlap somewhat.
- Fractal dimension features show relatively little-to-no correlations with other variables, implying that they might capture more independent aspects of the tumor.

3. Diagnosis Correlation

- The diagnosis variable (benign vs. malignant) is more strongly correlated with the size and shape based features rather than other features.

In summary, the strong inter-correlations among these features indicate the presence of multicollinearity, which will be dealt with during feature selection.

1.4 Preprocessing

The train and test datasets did not contain any duplicate or missing values, so we simply separate the diagnosis (response) column from the predictor columns.

[48]:
```
y_train = df_train['diagnosis']
y_test = df_test['diagnosis']
X_train = df_train.drop(columns=['diagnosis'])
X_test = df_test.drop(columns=['diagnosis'])
```

We standardize our 30 continuous predictors to ensure they are all on a comparable scale. In our analysis, we fit a StandardScaler on the train dataset to compute the mean and standard deviation for each feature, and then use these parameters to transform both the train and test datasets.

[49]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[50]:
```
df_X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
print("Train Dataset after preprocessing:")
df_X_train_scaled.head()
```

```
Train Dataset after preprocessing:
```

[50]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	\
0	-0.648225	-0.445234	-0.676130	-0.621515	-0.406056	
1	0.473196	0.267550	0.448515	0.310920	0.422769	
2	0.192131	-1.045974	0.130538	0.048796	-0.557140	
3	0.248912	-0.006598	0.219390	0.101448	-0.473759	
4	-0.185461	0.119749	-0.240916	-0.243615	-0.433850	

```
   compactness_mean  concavity_mean  concave_points_mean  symmetry_mean  \
0         -0.788416       -0.736952            -0.826052       0.168377
1          0.297045        0.331766             0.423048       0.497252
2         -0.499109       -0.519289            -0.336666      -1.049956
3         -0.117292       -0.354089            -0.002382       0.287967
4         -0.777950       -0.364259            -0.359970      -0.762190

   fractal_dimension_mean  …  radius_worst  texture_worst  perimeter_worst  \
0               -0.382422  …     -0.573173      -0.165298        -0.625544
1                0.011453  …      0.688440       0.775652         0.552448
2               -0.530819  …      0.215593      -1.229546         0.207021
3               -0.614032  …      0.004980       0.110497         0.056451
4               -0.865058  …      0.362196       0.707514         0.277878

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0   -0.547632         -0.389477          -0.521078        -0.605354
1    0.363809          0.966148           1.196331         0.607497
2    0.004778         -0.495110          -0.344598        -0.268424
3   -0.118249         -0.085781           0.292103        -0.434969
4    0.200235          0.301540          -0.633327        -0.031805

   concave_points_worst  symmetry_worst  fractal_dimension_worst
0             -0.930229       -0.185829                -0.177707
1              0.515565        1.785297                 1.193184
2              0.166147        0.349979                -0.141789
3              0.530890        0.111301                 0.013314
4              0.026686       -0.015345                -0.585873

[5 rows x 30 columns]
```

[51]:
```
y_train.head()
```

[51]:
```
0    0
1    1
2    0
3    0
4    1
Name: diagnosis, dtype: int64
```

[52]:
```
df_X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
print("Test Dataset after preprocessing:")
df_X_test_scaled.head()
```

```
Test Dataset after preprocessing:
```

[52]:
```
   radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
0    -0.608479     -0.211612       -0.657619  -0.605380        -0.994714
```

```
1    1.509447     1.042314        1.538606   1.425369          0.437022
2   -0.540342    -1.203311       -0.594270  -0.538009         -1.370286
3   -0.682294    -0.705078       -0.666669  -0.653785          1.156809
4    0.356796    -1.160401        0.444402   0.149853          0.764846

   compactness_mean  concavity_mean  concave_points_mean  symmetry_mean  \
0         -1.065202       -0.846217            -0.900885       0.246858
1          0.958639        1.450466             1.050705       0.657952
2         -1.352826       -0.951514            -1.119399      -0.481899
3         -0.221950       -0.560093            -0.439204       0.190800
4          2.036999        1.485193             1.275978       2.687260

   fractal_dimension_mean  …  radius_worst  texture_worst  perimeter_worst  \
0               -0.256216  …     -0.645442      -0.677954        -0.694334
1               -0.408773  …      1.119990       0.814588         1.113396
2               -0.895570  …     -0.697063      -1.466405        -0.755448
3                0.147368  …     -0.600016      -0.635774        -0.608716
4                1.022493  …      0.380780      -1.042978         0.528829

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0   -0.607932         -0.829615          -1.042038        -0.859107
1    0.933893          0.574426           0.253440         0.358880
2   -0.632364         -1.459012          -1.264915        -1.103261
3   -0.568772          0.851712          -0.676356        -0.585676
4    0.178229          0.253125           2.115525         1.725317

   concave_points_worst  symmetry_worst  fractal_dimension_worst
0             -0.786171       -0.176087                -0.565736
1              0.645830        0.043107                -0.453627
2             -1.467229       -0.974930                -1.311863
3             -0.246872       -0.145238                -0.230497
4              1.916302        2.879646                 0.815495

[5 rows x 30 columns]
```

[53]:
```
y_test.head()
```

[53]:
```
0    0
1    1
2    0
3    0
4    1
Name: diagnosis, dtype: int64
```

1.5 Feature Selection using VIF values

To identify and remove multicollinearity among our predictors, we use the Variance Inflation Factor (VIF). A higher VIF indicates that the predictor is highly collinear with others.

The code repeatedly removes predictors whose VIF value is higher than a chosen threshold and continues this process until all remaining predictors have a VIF below it. Here, we chose the threshold to be 5 since it is a common rule of thumb to detect moderate multicollinearity.

This iterative process creates a more stable and interpretable model, ensuring that each predictor's effect on the response is measured reliably and accurately. The final set of features is then applied to both the training and test datasets for model building.

[54]:
```
vif_threshold = 5  # Set VIF threshold
iteration = 1

while True:
    # Compute VIF for each feature in the current DataFrame
    vif_data = pd.DataFrame({
        "Feature": df_X_train_scaled.columns,
        "VIF": [variance_inflation_factor(df_X_train_scaled.values, i)
                for i in range(df_X_train_scaled.shape[1])]
    })

    # Check if the highest VIF is above the threshold
    max_vif = vif_data["VIF"].max()
    if max_vif <= vif_threshold:
        break

    # Identify and drop the feature with the highest VIF
    feature_to_drop = vif_data.sort_values("VIF", ascending=False)["Feature"].
 ↪iloc[0]
    #print(f"Iteration {iteration}: Dropping feature '{feature_to_drop}' with␣
 ↪VIF = {max_vif:.2f}")
    df_X_train_scaled = df_X_train_scaled.drop(columns=[feature_to_drop])
    iteration += 1

# Update both training and test sets with the selected features
X_train_scaled_final = df_X_train_scaled
X_test_scaled_final = pd.DataFrame(X_test_scaled, columns=X_test.
 ↪columns)[df_X_train_scaled.columns.tolist()]
print("Final selected features:", X_train_scaled_final.columns.tolist())
```

```
Final selected features: ['texture_mean', 'smoothness_mean', 'symmetry_mean',
'fractal_dimension_mean', 'texture_se', 'perimeter_se', 'smoothness_se',
'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
'area_worst', 'symmetry_worst']
```

[55]:
```
df_X_train_scaled_final = pd.DataFrame(X_train_scaled_final,
    columns=X_train_scaled_final.columns)
print("Train Dataset after feature selection:")
df_X_train_scaled_final.head()
```

Train Dataset after feature selection:

[55]:

	texture_mean	smoothness_mean	symmetry_mean	fractal_dimension_mean	\
0	-0.445234	-0.406056	0.168377	-0.382422	
1	0.267550	0.422769	0.497252	0.011453	
2	-1.045974	-0.557140	-1.049956	-0.530819	
3	-0.006598	-0.473759	0.287967	-0.614032	
4	0.119749	-0.433850	-0.762190	-0.865058	

	texture_se	perimeter_se	smoothness_se	compactness_se	concavity_se	\
0	-0.107217	-0.300473	-0.539654	-0.315563	-0.361989	
1	-0.496215	-0.291660	-0.767025	-0.089356	-0.259170	
2	-1.424413	-0.155767	-0.778116	-0.568253	-0.437503	
3	-0.448101	-0.325518	-0.582388	-0.232252	-0.545041	
4	-0.094789	0.124831	-0.363498	-0.812667	-0.337717	

	concave_points_se	symmetry_se	area_worst	symmetry_worst
0	-1.014988	-0.650644	-0.547632	-0.185829
1	-0.487948	-0.306784	0.363809	1.785297
2	-0.157144	0.148976	0.004778	0.349979
3	0.038116	-0.598191	-0.118249	0.111301
4	-0.356798	-0.600522	0.200235	-0.015345

[56]:
```
print("Shape of final train dataset: " + str(df_X_train_scaled_final.shape[0])
    + " rows x " + str(df_X_train_scaled_final.shape[1]) + " columns")
```

Shape of final train dataset: 426 rows x 13 columns

[57]:
```
df_X_test_scaled_final = pd.DataFrame(X_test_scaled_final,
    columns=X_test_scaled_final.columns)
print("Test Dataset after feature selection:")
df_X_test_scaled_final.head()
```

Test Dataset after feature selection:

[57]:

	texture_mean	smoothness_mean	symmetry_mean	fractal_dimension_mean	\
0	-0.211612	-0.994714	0.246858	-0.256216	
1	1.042314	0.437022	0.657952	-0.408773	
2	-1.203311	-1.370286	-0.481899	-0.895570	
3	-0.705078	1.156809	0.190800	0.147368	
4	-1.160401	0.764846	2.687260	1.022493	

```
   texture_se  perimeter_se  smoothness_se  compactness_se  concavity_se  \
0   -1.007541     -0.627451       0.030242       -0.936805     -0.620892
1    1.068121      0.539931       1.139370        0.395059      0.256948
2   -1.066663     -0.665482      -0.578800       -1.080970     -0.684269
3   -0.082361     -0.147883       0.613513       -0.686322     -0.438851
4   -0.871365      0.237070      -0.107094        1.540439      1.110179

   concave_points_se  symmetry_se  area_worst  symmetry_worst
0          -0.591599    -0.045683   -0.607932       -0.176087
1           0.643423     0.175786    0.933893        0.043107
2          -1.071451    -0.067830   -0.632364       -0.974930
3           0.005572     0.136155   -0.568772       -0.145238
4           1.771051     1.907908    0.178229        2.879646
```

[58]:
```
print("Shape of final test dataset: " + str(df_X_test_scaled_final.shape[0]) +
    " rows x " + str(df_X_test_scaled_final.shape[1]) + " columns")
```

```
Shape of final test dataset: 143 rows x 13 columns
```

1.6 Model Construction

In this stage, we build a logistic regression classifier using our training dataset, which consists of 13 predictors after feature selection. The final predictors are:

- texture_mean
- smoothness_mean
- symmetry_mean
- fractal_dimension_mean
- texture_se
- perimeter_se
- smoothness_se
- compactness_se
- concavity_se
- concave_points_se
- symmetry_se
- area_worst
- symmetry_worst

Firstly, we prepare our data for modeling by adding a constant column to both the train and test datasets. This step ensures that our regression model has an intercept.

[59]:
```
X_train_sm = sm.add_constant(X_train_scaled_final, prepend=False)
X_test_sm = sm.add_constant(X_test_scaled_final, prepend=False)
```

[60]:
```
print("Train Dataset after adding a constant column to the final model:")
X_train_sm.head()
```

```
Train Dataset after adding a constant column to the final model:
```

[60]:

```
   texture_mean  smoothness_mean  symmetry_mean  fractal_dimension_mean  \
0     -0.445234        -0.406056       0.168377               -0.382422
1      0.267550         0.422769       0.497252                0.011453
2     -1.045974        -0.557140      -1.049956               -0.530819
3     -0.006598        -0.473759       0.287967               -0.614032
4      0.119749        -0.433850      -0.762190               -0.865058

   texture_se  perimeter_se  smoothness_se  compactness_se  concavity_se  \
0   -0.107217     -0.300473      -0.539654       -0.315563     -0.361989
1   -0.496215     -0.291660      -0.767025       -0.089356     -0.259170
2   -1.424413     -0.155767      -0.778116       -0.568253     -0.437503
3   -0.448101     -0.325518      -0.582388       -0.232252     -0.545041
4   -0.094789      0.124831      -0.363498       -0.812667     -0.337717

   concave_points_se  symmetry_se  area_worst  symmetry_worst  const
0          -1.014988    -0.650644   -0.547632       -0.185829    1.0
1          -0.487948    -0.306784    0.363809        1.785297    1.0
2          -0.157144     0.148976    0.004778        0.349979    1.0
3           0.038116    -0.598191   -0.118249        0.111301    1.0
4          -0.356798    -0.600522    0.200235       -0.015345    1.0
```

[61]:

```python
print("Test Dataset after adding a constant column to the final model:")
X_test_sm.head()
```

```
Test Dataset after adding a constant column to the final model:
```

[61]:

```
   texture_mean  smoothness_mean  symmetry_mean  fractal_dimension_mean  \
0     -0.211612        -0.994714       0.246858               -0.256216
1      1.042314         0.437022       0.657952               -0.408773
2     -1.203311        -1.370286      -0.481899               -0.895570
3     -0.705078         1.156809       0.190800                0.147368
4     -1.160401         0.764846       2.687260                1.022493

   texture_se  perimeter_se  smoothness_se  compactness_se  concavity_se  \
0   -1.007541     -0.627451       0.030242       -0.936805     -0.620892
1    1.068121      0.539931       1.139370        0.395059      0.256948
2   -1.066663     -0.665482      -0.578800       -1.080970     -0.684269
3   -0.082361     -0.147883       0.613513       -0.686322     -0.438851
4   -0.871365      0.237070      -0.107094        1.540439      1.110179

   concave_points_se  symmetry_se  area_worst  symmetry_worst  const
0          -0.591599    -0.045683   -0.607932       -0.176087    1.0
1           0.643423     0.175786    0.933893        0.043107    1.0
2          -1.071451    -0.067830   -0.632364       -0.974930    1.0
3           0.005572     0.136155   -0.568772       -0.145238    1.0
4           1.771051     1.907908    0.178229        2.879646    1.0
```

1.6.1 K-Fold Cross-Validation on the Final Train Dataset

After adding the constant column to both the train and test datasets to account for the intercept, we evaluate our logistic regression model's performance using 5-fold cross-validation on the final train dataset. In this process, the train dataset is randomly partitioned into five folds. For each fold, the model is trained on four folds and validated on the remaining fold using statsmodels' Generalized Linear Model (GLM) with a binomial family.

We convert the predicted probabilities into class labels (using a threshold of 0.5) and compute the accuracy for each fold. The cross-validation accuracies are then averaged to obtain a mean accuracy, providing an estimate of the model's performance on unseen data before fitting the final model on the entire train dataset.

[62]:
```python
kf = skm.KFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = []

# Loop through the folds
for train_index, val_index in kf.split(X_train_sm):
    # Split the training data into a training fold and a validation fold
    X_train_cv = X_train_sm.iloc[train_index]
    y_train_cv = y_train.iloc[train_index]
    X_val_cv = X_train_sm.iloc[val_index]
    y_val_cv = y_train.iloc[val_index]

    # Fit the model on the current training fold
    model_cv = sm.GLM(y_train_cv, X_train_cv, family=sm.families.Binomial())
    result_cv = model_cv.fit()

    # Predict on the validation fold
    y_val_pred_prob = result_cv.predict(X_val_cv)
    y_val_pred_class = (y_val_pred_prob >= 0.5).astype(int)

    # Compute and store the accuracy for the current fold
    fold_acc = accuracy_score(y_val_cv, y_val_pred_class)
    cv_scores.append(fold_acc)

print("K-Fold Cross-Validation Accuracy Scores:", cv_scores)
print("Mean CV Accuracy:", np.mean(cv_scores))
```

```
K-Fold Cross-Validation Accuracy Scores: [0.9186046511627907,
0.9411764705882353, 0.9176470588235294, 0.9647058823529412, 0.9529411764705882]
Mean CV Accuracy: 0.9390150478796169
```

The cross-validation accuracy scores range from approximately 91.9% to 96.5%, with an average accuracy of about 93.9%. These results indicate that our logistic regression model performs robustly on unseen data and suggests that the selected 13 predictors have strong discriminative power in distinguishing between benign and malignant tumors. This high level of accuracy provides confidence in the model's potential to generalize well to new samples.

1.6.2 Final Model Building: Logistic Regression using Statsmodels

After confirming our approach with cross-validation, we train the final logistic regression model on the entire train dataset using statsmodels' Generalized Linear Model (GLM) with a binomial family. We print a detailed summary of the model, which includes coefficient estimates, standard errors, z-values, and p-values, providing insight into the significance and direction of each predictor's effect.

[63]:
```
# Fit the logistic regression model on the entire training set
glm_binom = sm.GLM(y_train, X_train_sm, family=sm.families.Binomial())
results = glm_binom.fit()

# Print the detailed summary (coefficients, std errors, z-values, p-values, etc.
↪)
print(results.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:              diagnosis   No. Observations:                  426
Model:                            GLM   Df Residuals:                      412
Model Family:                Binomial   Df Model:                           13
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -28.625
Date:                Mon, 10 Mar 2025   Deviance:                       57.249
Time:                        00:31:43   Pearson chi2:                     115.
No. Iterations:                    10   Pseudo R-squ. (CS):             0.6948
Covariance Type:            nonrobust
================================================================================
==========
                             coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----------
texture_mean               2.2856      0.636      3.593      0.000       1.039
3.532
smoothness_mean            2.0688      0.858      2.413      0.016       0.388
3.749
symmetry_mean             -0.6165      0.811     -0.760      0.447      -2.207
0.974
fractal_dimension_mean    -1.1910      1.016     -1.172      0.241      -3.183
0.801
texture_se                 0.0096      0.576      0.017      0.987      -1.119
1.138
perimeter_se               0.9221      0.916      1.006      0.314      -0.874
2.718
smoothness_se              0.9082      0.769      1.181      0.238      -0.600
2.416
compactness_se            -0.9661      0.607     -1.593      0.111      -2.155
0.223
```

```
concavity_se                 0.7583      0.651      1.165      0.244      -0.518
2.034
concave_points_se            1.6286      1.174      1.388      0.165      -0.672
3.929
symmetry_se                 -2.6360      0.856     -3.079      0.002      -4.314
-0.958
area_worst                  10.2753      2.443      4.206      0.000       5.487
15.064
symmetry_worst               4.1134      1.169      3.517      0.000       1.821
6.406
const                       -0.2506      0.791     -0.317      0.752      -1.802
1.301
================================================================================
==========
```

Next, we used the fitted model to predict probabilities on the test dataset, converting these probabilities to binary predictions (benign or malignant) with a threshold of 0.5. Finally, we evaluate the model's performance on the test dataset by computing the overall accuracy, which gives a clear measure of how well the model generalizes to new data.

[64]:
```
# Predict probabilities on the test set using the final model
y_pred_prob = results.predict(X_test_sm)  # Predicted probability for class "1"␣
↪(Malignant)
y_pred_class = (y_pred_prob >= 0.5).astype(int)

# Evaluate the predictions on the test set
acc = accuracy_score(y_test, y_pred_class)
print("Test Accuracy:", acc)
```

```
Test Accuracy: 0.972027972027972
```

The table above displays each predictor's coefficient, standard error, z-value, and p-value. A lower p-value (< 0.05) indicates that the predictor contributes significantly to explaining whether a tumor is malignant. The significant predictors in this model are:

- texture_mean
- smoothness_mean
- symmetry_se
- area_worst
- symmetry_worst

area_worst has a notably large positive coefficient (10.2753), suggesting that higher values of this feature strongly increase the log-odds of malignancy. Meanwhile, symmetry_se is negative (-2.6360), implying that greater variability in symmetry lowers the log-odds of malignancy, whereas symmetry_worst (4.1134) indicates that extreme symmetry values raise the likelihood of malignancy.

Some predictors, such as texture_se, perimeter_se, and symmetry_mean, have higher p-values, meaning we do not have strong statistical evidence that they significantly affect the odds of malignancy in the presence of other variables. Overall, the Pseudo R-squared of 0.6948 indicates that the model explains a substantial portion of the variation in tumor diagnosis.

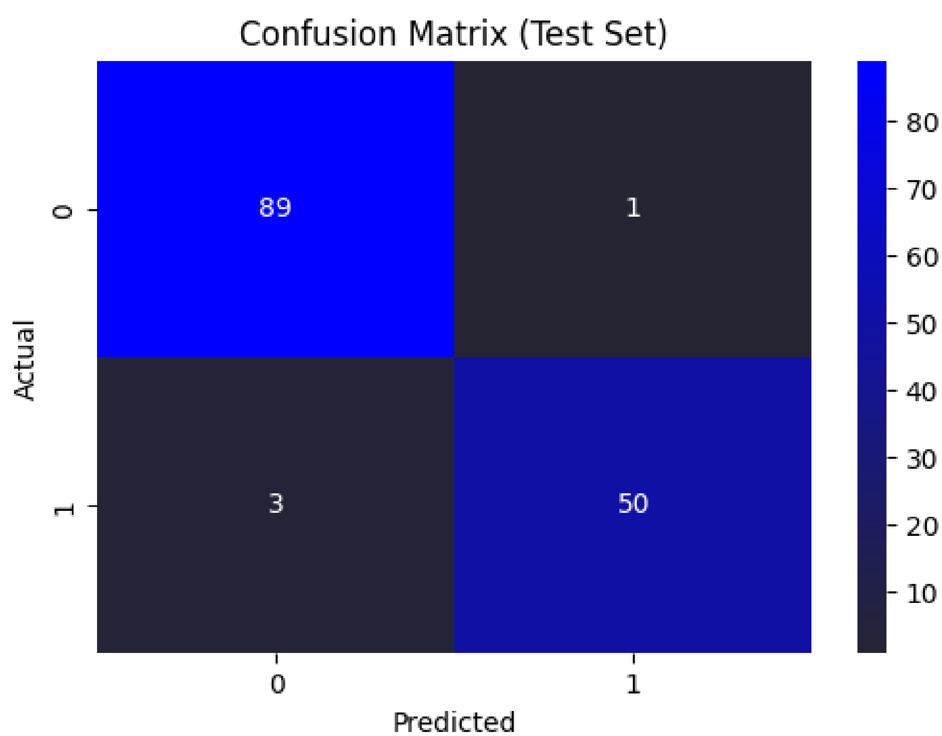
The final logistic regression model achieves a 97.20% accuracy on the test dataset, meaning that it correctly classifies 97.20% of the cases as either benign or malignant.

1.7 Data Visualization

Now that we have fit our model, we evaluate and visualize its performance using the following:

1.7.1 Confusion Matrix

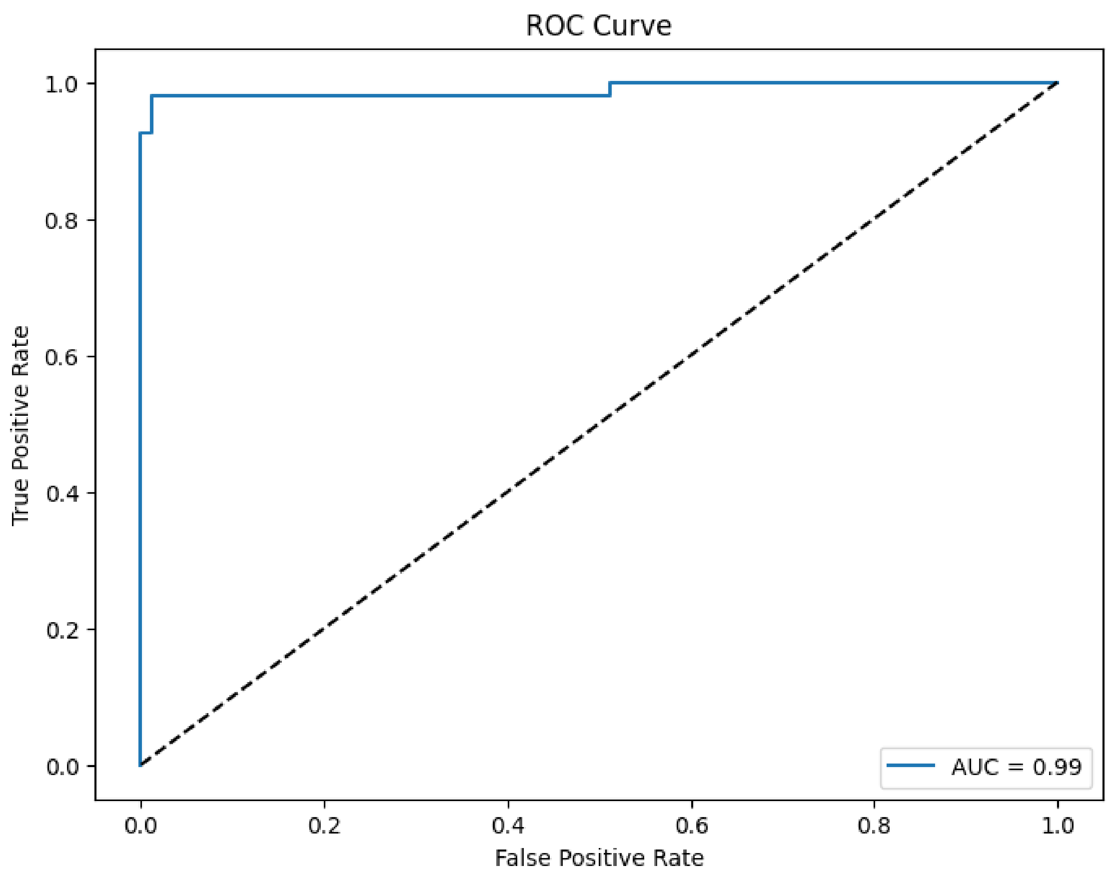
[65]:
```
cm = confusion_matrix(y_test, y_pred_class)
dark_cmap = sns.dark_palette("blue", as_cmap=True)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap=dark_cmap)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix (Test Set)")
plt.show()
```



The confusion matrix shows that out of all benign tumors, 89 are correctly identified (true negatives) and only 1 is misclassified (false positive). Among malignant tumors, 50 are correctly identified (true positives), while 3 are missed (false negatives). These results suggest that the model does a great job in distinguishing benign from malignant tumors in this dataset, although the few false negatives highlight the importance of further refinement in real-world medical applications.

1.7.2 ROC Curve and AUC

[66]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
auc = roc_auc_score(y_test, y_pred_prob)
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
plt.plot([0,1], [0,1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



This curve hugs the top-left corner of the graph, indicating high sensitivity for a relatively low false positive rate.This means that the model is very good at correctly identifying malignant tumors while rarely misclassifying benign ones. The AUC value of 0.99 also shows the model's ability to discriminate between benign and malignant cases.

1.7.3 Odd Ratios Bar Plot

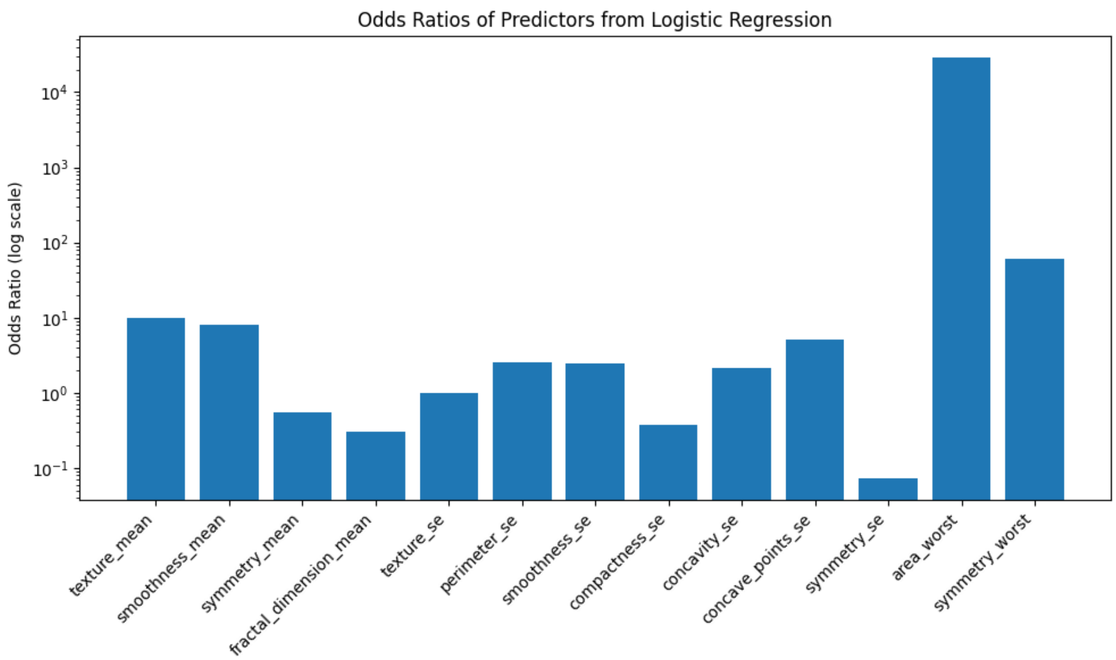
To better understand each predictor's impact, we exponentiated the coefficients to obtain odds ratios, shown in the bar chart on a logarithmic scale. Since the features were standardized, an odds ratio reflects how the odds of malignancy change when the predictor increases by one standard deviation.

- An odds ratio above 1 means that an increase in that feature raises the odds of malignancy.
- An odds ratio below 1 means that an increase in that feature lowers the odds of malignancy.

[67]:
```python
param_names = results.params.index

# Drop 'const' from both names and coefficients
param_names = param_names.drop('const')
coef = results.params.drop('const')
odds_ratios = np.exp(coef)

# Now param_names and coef (or odds_ratios) have matching order
print("Parameter Names:", param_names)
print("Odds Ratios:")
print(odds_ratios)

# Plot in the statsmodels order
x_positions = np.arange(len(param_names))

plt.figure(figsize=(10, 6))
plt.bar(x_positions, odds_ratios, align='center')
plt.yscale('log')  # Use log scale
plt.xticks(x_positions, param_names, rotation=45, ha='right')
plt.ylabel("Odds Ratio (log scale)")
plt.title("Odds Ratios of Predictors from Logistic Regression")
plt.tight_layout()
plt.show()
```

```
Parameter Names: Index(['texture_mean', 'smoothness_mean', 'symmetry_mean',
       'fractal_dimension_mean', 'texture_se', 'perimeter_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
       'area_worst', 'symmetry_worst'],
      dtype='object')
Odds Ratios:
texture_mean              9.832010
smoothness_mean           7.915127
symmetry_mean             0.539837
fractal_dimension_mean    0.303908
texture_se                1.009653
perimeter_se              2.514690
smoothness_se             2.479856
compactness_se            0.380557
concavity_se              2.134739
```

```
concave_points_se            5.096790
symmetry_se                  0.071650
area_worst               29006.124849
symmetry_worst              61.152919
dtype: float64
```



Below are the key insights from the odds ratios:

1. Strong Positive Indicators

- area_worst (29,000) is exceptionally large, indicating that higher worst-area measurements dramatically increase the odds of malignancy.
- symmetry_worst (61), texture_mean (9.83), smoothness_mean (7.92), and concave_points_se (5.10) also have odds ratios well above 1. This means that tumors with higher values in these features are far more likely to be malignant. However, concave_points_se is the only predictor that is not statistically significant among these.

2. Predictors That Decrease the Odds

- symmetry_se (0.07) is well below 1, suggesting that greater variability in symmetry reduces the odds of malignancy.
- fractal_dimension_mean (0.30), compactness_se (0.38), and symmetry_mean (0.54) are also below 1, indicating negative associations. However, only symmetry_se is statistically significant among these.

3. Less Influential or Non-Significant Predictors

- Some predictors with odds ratios near 1 (e.g., texture_se 1.01) or moderate odds ratios but high p-values (e.g., perimeter_se 2.51, concavity_se 2.13) are not statistically significant, so we cannot confidently conclude they have a meaningful effect.

In summary, area_worst, symmetry_worst, texture_mean, and smoothness_mean are strong positive indicators of malignancy (with odds ratios much greater than 1), while symmetry_se stands out as a strong negative indicator (with an odds ratio much less than 1). This pattern confirms that certain size and shape related features play a critical role in distinguishing malignant tumors from benign ones.

1.8 Conclusion

Our analysis aims to classify breast tumors based on features of cell nuclei and identify which predictors are most indicative of malignancy. We first assess the statistical significance of each predictor using p-values from our logistic regression model, which highlights that features such as area_worst, symmetry_worst, texture_mean, and smoothness_mean, symmetry_se are significant.

By exponentiating the coefficients to obtain odds ratios, we determine the direction and magnitude of these associations. Increases in area_worst, symmetry_worst, texture_mean, and smoothness_mean substantially raise the odds of a tumor being malignant, while an increase in symmetry_se is associated with decreased odds. Although concave_points_se exhibits a high odds ratio (5.10), its p-value of 0.165 indicates that its effect is not statistically significant and should be interpreted with caution.

The final model achieves a test accuracy of approximately 97.2%, and the mean cross-validation accuracy is around 93.9%. The close agreement between these scores indicates that our model generalizes well to unseen data.

Overall, these results confirm that certain size and shape related features, particularly area_worst and symmetry_worst, play a critical role in distinguishing malignant tumors from benign ones. This model could serve as a reliable tool for early breast cancer detection, potentially aiding medical practitioners in diagnosis, although further refinement is necessary before clinical implementation.