Table of Contents

1. Introduction

Breast Cancer Prediction involves classifying breast masses as either malignant or benign. The dataset used for this task consists of features derived from digitized images of fine needle aspirates (FNAs) of the breast mass. These features describe various properties of the cell nuclei observed in the images.

Each instance in the dataset is associated with a unique ID number and a diagnosis label, with 'M' representing malignant and 'B' representing benign. Ten real-valued features are computed for each cell nucleus, including radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension. These features provide quantitative measurements that help assess the characteristics of cell nuclei and aid in distinguishing between malignant and benign breast masses.

By training a machine learning model on this dataset, it is possible to develop a predictive model that can assist in early detection and diagnosis of breast cancer. This model leverages the computed features to make accurate predictions, contributing to more effective medical decision-making and potentially improving patient outcomes.

In the following sections, we will delve into the dataset, perform preprocessing tasks, conduct exploratory data analysis, build and evaluate predictive models, and conclude with a summary of our findings and potential future directions.

2. Required Modules

In [29]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
```

3. Data Preprocessing

In this section, we discuss the steps taken to preprocess the dataset. Through preprocessing steps, including data cleaning, handling missing values, and converting categorical variables into numerical formats, we will ensure the dataset is ready for analysis.

3.1 Key Features

The dataset contains the following information for each instance:

1. ID number: A unique identifier for each sample.

2. Diagnosis: The target variable indicating the diagnosis, where 'M' represents malignant and 'B' represents benign.

For each cell nucleus, ten real-valued features are computed, which are:

1. Radius: The mean distance from the center to points on the perimeter of the nucleus.
2. Texture: The standard deviation of gray-scale values in the nucleus.
3. Perimeter: The perimeter of the nucleus.
4. Area: The area of the nucleus.
5. Smoothness: A measure of local variation in radius lengths.
6. Compactness: Computed as the square of the perimeter divided by the area minus 1.0.
7. Concavity: Describes the severity of concave portions of the nucleus contour.
8. Concave points: Represents the number of concave portions of the nucleus contour.
9. Symmetry: Measures the symmetry of the nucleus.
10. Fractal dimension: This feature approximates the "coastline" of the nucleus, using the concept of fractal geometry.

In [3]:
```
# Load the data
data = pd.read_csv('data.csv')
data.head(-1)
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030
...	...	...	...	...	...	...	...
563	926125	M	20.92	25.09	143.00	1347.0	0.10990
564	926424	M	21.56	22.39	142.00	1479.0	0.11100
565	926682	M	20.13	28.25	131.20	1261.0	0.09780
566	926954	M	16.60	28.08	108.30	858.1	0.08455
567	927241	M	20.60	29.33	140.10	1265.0	0.11780

568 rows × 33 columns

In [4]:
```
#dropping unnecessary info
data.drop(['Unnamed: 32','id'], axis=1, inplace=True)
data.head(-1)
```

Out[4]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactr
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
563	M	20.92	25.09	143.00	1347.0	0.10990	
564	M	21.56	22.39	142.00	1479.0	0.11100	
565	M	20.13	28.25	131.20	1261.0	0.09780	
566	M	16.60	28.08	108.30	858.1	0.08455	
567	M	20.60	29.33	140.10	1265.0	0.11780	

568 rows × 31 columns

3.2 Descriptive Statistics

Now we generate the summary statistics:

- Count: The number of non-missing values in each column.
- Mean: The average value of each column.
- Standard Deviation: A measure of the amount of variation or dispersion in each column.
- Minimum: The minimum value in each column.
- 25th Percentile (Q1): The value below which 25% of the data falls.
- Median (50th Percentile or Q2): The middle value in each column. It represents the value below which 50% of the data falls.
- 75th Percentile (Q3): The value below which 75% of the data falls.
- Maximum: The maximum value in each column.

In [5]:
```
data.describe()
```

Out[5]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mea
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.00000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.1043
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.0528
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.0193
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.0649
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.0926
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.1304
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.3454

8 rows × 30 columns

In [6]:

```
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    object 
 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), object(1)
memory usage: 137.9+ KB
```

3.3 Missing Values

In [7]:
```
# Check for missing values in the dataset
data.isnull().sum()
```

Out[7]:
```
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
```

In [8]:
```
#checking the data types of the columns
data.dtypes
```

```
Out[8]: diagnosis                   object
        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
```

3.4 Data Transformation

No need in this project

4. Exploratory Data Analysis

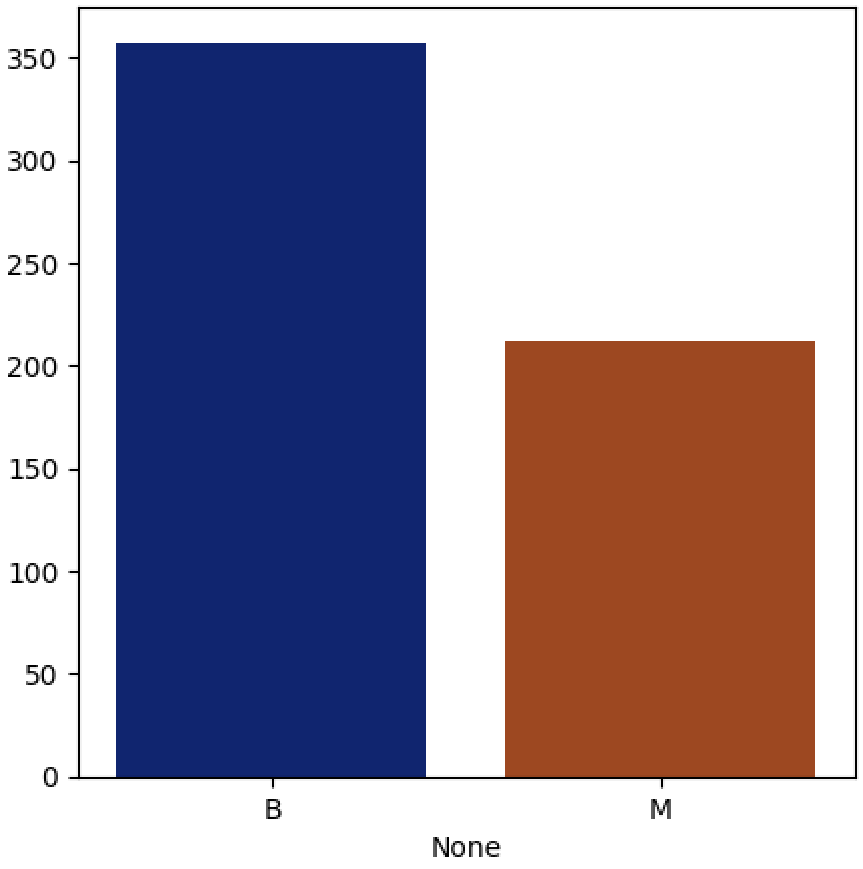
Exploratory data analysis will allow us to gain insights into the distribution of features, detect correlations, and uncover potential patterns and trends related to breast cancer data.

4.1 Visualization

```
In [10]: # bar plot for the number of diagnosis
         plt.figure(figsize=(5,5))
         sns.barplot(x=data['diagnosis'].value_counts().index,y=data['diagnosis'].value_counts(
```

```
<ipython-input-10-557d0eb3533f>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=data['diagnosis'].value_counts().index,y=data['diagnosis'].value_coun
ts().values,palette='dark')
```

```
Out[10]: <Axes: xlabel='None'>
```



4.2 Correlation

```
In [15]: data.corr()
```

```
<ipython-input-15-c44ded798807>:1: FutureWarning: The default value of numeric_only i
n DataFrame.corr is deprecated. In a future version, it will default to False. Select
only valid columns or specify the value of numeric_only to silence this warning.
  data.corr()
```

Out[15]:

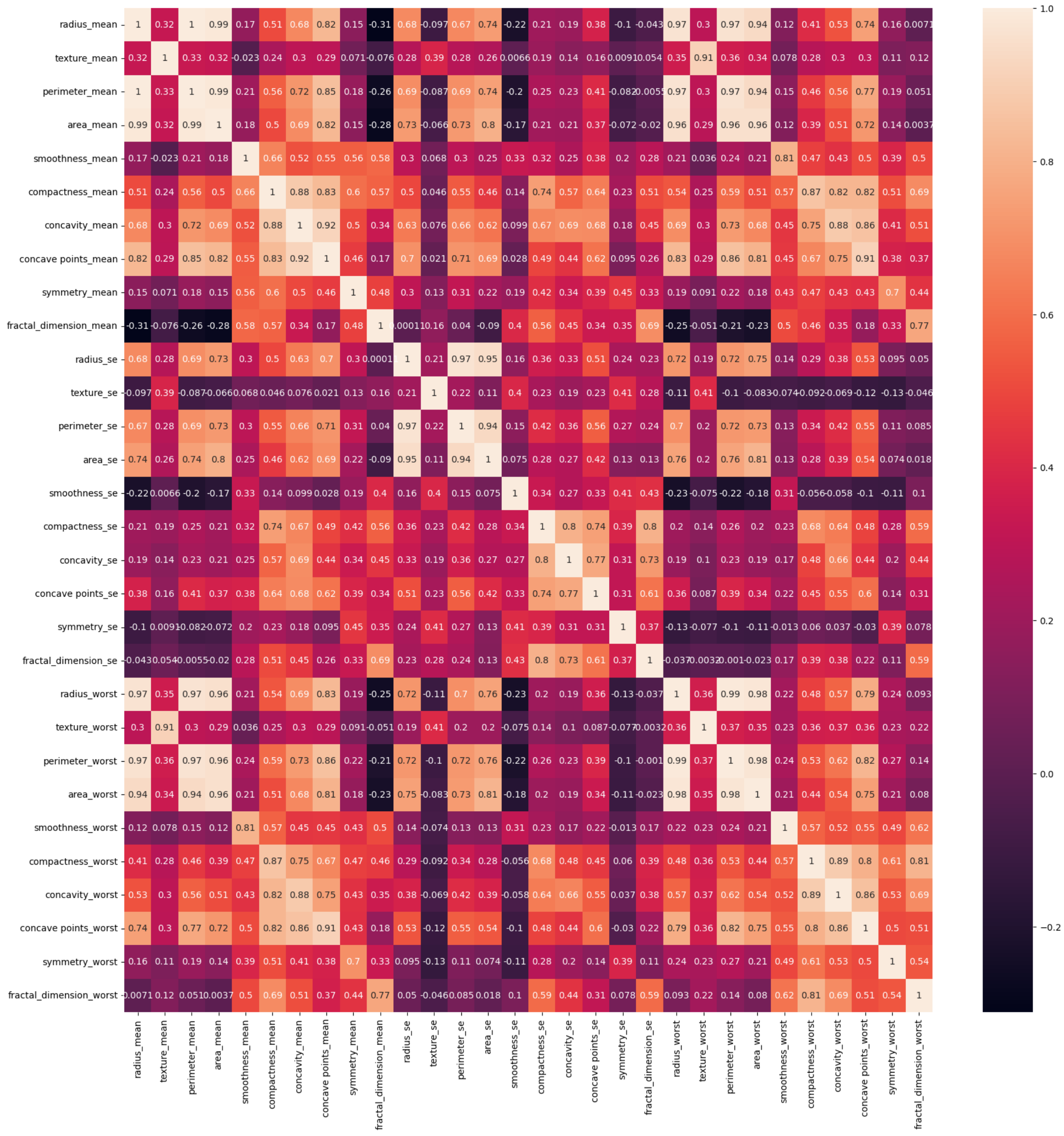
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
radius_mean	1.000000	0.323782	0.997855	0.987357	0.170581
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.023389
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.207278
area_mean	0.987357	0.321086	0.986507	1.000000	0.177028
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.000000
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.659123
concavity_mean	0.676764	0.302418	0.716136	0.685983	0.521984
concave points_mean	0.822529	0.293464	0.850977	0.823269	0.553695
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.557775
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.584792
radius_se	0.679090	0.275869	0.691765	0.732562	0.301467
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0.068406
perimeter_se	0.674172	0.281673	0.693135	0.726628	0.296092
area_se	0.735864	0.259845	0.744983	0.800086	0.246552
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0.332375
compactness_se	0.206000	0.191975	0.250744	0.212583	0.318943
concavity_se	0.194204	0.143293	0.228082	0.207660	0.248396
concave points_se	0.376169	0.163851	0.407217	0.372320	0.380676
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0.200774
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887	0.283607
radius_worst	0.969539	0.352573	0.969476	0.962746	0.213120
texture_worst	0.297008	0.912045	0.303038	0.287489	0.036072
perimeter_worst	0.965137	0.358040	0.970387	0.959120	0.238853
area_worst	0.941082	0.343546	0.941550	0.959213	0.206718
smoothness_worst	0.119616	0.077503	0.150549	0.123523	0.805324
compactness_worst	0.413463	0.277830	0.455774	0.390410	0.472468
concavity_worst	0.526911	0.301025	0.563879	0.512606	0.434926
concave points_worst	0.744214	0.295316	0.771241	0.722017	0.503053
symmetry_worst	0.163953	0.105008	0.189115	0.143570	0.394309
fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738	0.499316

30 rows × 30 columns

In [17]:
```
# create a heatmap to check the correlation
plt.figure(figsize=(20,20))
sns.heatmap(data.corr(),annot=True)
```

```
<ipython-input-17-477ad02c8bb5>:3: FutureWarning: The default value of numeric_only i
n DataFrame.corr is deprecated. In a future version, it will default to False. Select
only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(data.corr(),annot=True)
```

Out[17]:
```
<Axes: >
```



5. Model Building

In thise section we build predictive models using machine learning algorithms. First we have to split the dataset for training and testing.

In [19]:
```
x = data.drop('diagnosis', axis=1)
y = data['diagnosis']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=
print(X_train.shape,y_train.shape)
```

```
(398, 30) (398,)
```

5.1 Decision Tree

In [23]:
```
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)

#predicting the diagnosis
yhat_dtree = dtree.predict(X_test)
```

5.2 Logistic Regression

In [24]:
```
logreg = LogisticRegression()
logreg.fit(X_train,y_train)

#predicting the diagnosis
yhat_logreg = logreg.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

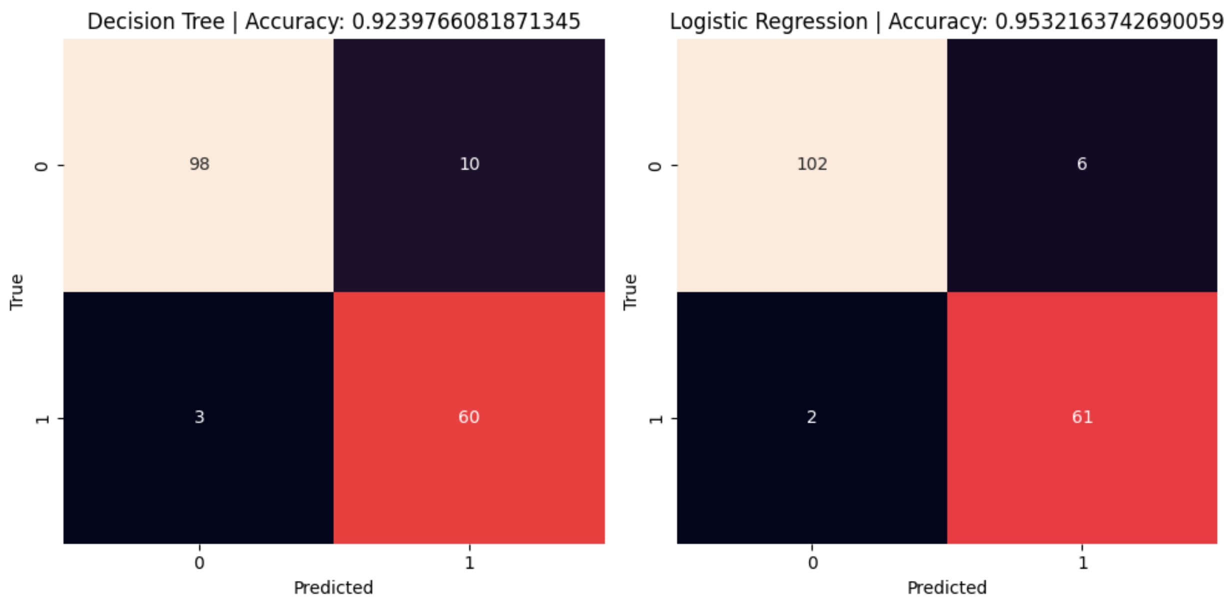
6. Evaluation

In this secton we evaluate the performance of the models and compare them.

6.1 Confusion Matrix

In [27]:
```
# List of model names
model_names = ['Decision Tree', 'Logistic Regression']

# List of predicted labels for each model
predicted_labels = [yhat_dtree, yhat_logreg]

#List of model accuracy
accuracy = [dtree.score(X_test,y_test),logreg.score(X_test,y_test)]

# List of confusion matrices for each model
confusion_matrices = [confusion_matrix(y_test, predicted) for predicted in predicted_l

# Set up the figure and axes
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
```

```
# Iterate over each model and plot the confusion matrix
for i, ax in enumerate(axes.flatten()):
    sns.heatmap(confusion_matrices[i], annot=True, fmt='d', cbar=False, ax=ax)
    ax.set_title("{0} | Accuracy: {1}".format(model_names[i],accuracy[i]))
    ax.set_xlabel('Predicted')
    ax.set_ylabel('True')

# Adjust the layout
plt.tight_layout()

# Show the plot
plt.show()
```



The diagonal boxes in the matrix represent the number of true positive results, indicating the correct predictions made by the model. On the other hand, the off-diagonal boxes represent the number of false positive results, indicating the incorrect predictions made by the model.

6.2 Other Metrics

In [30]:
```
#Decision Tree
print(classification_report(y_test, predicted_labels[0]))
```

```
              precision    recall  f1-score   support

           B       0.97      0.91      0.94       108
           M       0.86      0.95      0.90        63

    accuracy                           0.92       171
   macro avg       0.91      0.93      0.92       171
weighted avg       0.93      0.92      0.92       171
```

The model performs well, achieving a satisfactory accuracy of 92% and an average F1 score of 0.92. It demonstrates its ability to predict breast cancer with a high level of accuracy.

In [31]:
```
#Logistic Regression
print(classification_report(y_test, predicted_labels[1]))
```

```
              precision    recall  f1-score   support

           B       0.98      0.94      0.96       108
           M       0.91      0.97      0.94        63

    accuracy                           0.95       171
   macro avg       0.95      0.96      0.95       171
weighted avg       0.95      0.95      0.95       171
```

The Logistic Regression Classifier model achieves an accuracy of 95% and an average F1 score of 0.95. These metrics indicate that the model is highly effective in predicting breast cancer and outperforms the Decision Tree Classifier in terms of accuracy.

7. Conclusion

Based on the results for breast cancer prediction using a decision tree and logistic regression models, the following conclusions can be drawn:

1. Decision tree results: The decision tree model achieved an accuracy of 92% on the test dataset. It exhibited high precision and recall values for both the benign (B) and malignant (M) classes, indicating that it was able to correctly classify instances from both classes. The model achieved an overall F1-score of 0.92, which indicates a good balance between precision and recall.

2. Logistic regression results: The logistic regression model performed even better, achieving an accuracy of 95% on the test dataset. Similar to the decision tree model, it demonstrated high precision and recall values for both the benign and malignant classes. The model achieved an overall F1-score of 0.95, indicating excellent performance in terms of precision and recall trade-off.

Based on these conclusions, some potential future works for improving breast cancer prediction could include:

1. Feature engineering: Exploring additional feature engineering techniques or extracting more informative features from the breast mass images could potentially enhance the performance of the models. This could involve analyzing different aspects of the cell nuclei or incorporating other relevant image-based features.

2. Ensemble methods: Investigating ensemble methods, such as random forests or gradient boosting, could potentially improve the accuracy and robustness of the breast cancer prediction models. Ensemble methods combine multiple models to make predictions, often resulting in better performance than a single model.

3. Hyperparameter tuning: Fine-tuning the hyperparameters of the models using techniques like grid search or random search could help optimize their performance. By finding the best combination of hyperparameters, the models may achieve even higher accuracy and improve their generalization capabilities.

4. External validation: Validating the trained models on independent datasets from different sources or medical institutions can help assess their generalizability and reliability. This step is crucial to ensure that the models perform consistently across various patient populations and healthcare settings.

5. Interpretability and explainability: Exploring methods to make the models more interpretable and explainable can help build trust and acceptance among medical professionals. Techniques like feature importance analysis or generating decision rules can provide insights into the factors influencing the predictions, making the models more transparent.