1. Introduction

1. What Is Breast Cancer and Why Does It Matter?

Cancer in the breast tissue is the most commonly found cancer in the world, particularly in the female population. As per the World Health Organisation, **more than 2.3 million women** were diagnosed with breast cancer in the year 2022, and sadly, around **685,000 lost their lives** because of it (1). That is the bad side. The good side says: **Early detection saves lives**. When cancer signs can be detected at the earliest, treatment is more effective, and more people are likely to be saved. That is where technology comes into play (meaning AI).

2. How Can AI and Machine Learning Help?

Artificial Intelligence and Machine Learning characterise algorithms that enable a computer to learn from data and predict outputs in some or other manner. In medicine, it implies that AI-based systems help doctors identify whether a tumour is Malignant (M) canous tumoor or Benign (B) non-canorous tumour by recognising patterns in medical data. The data taken from the **Wisconsin Breast Cancer Diagnostic Database**, which contains information derived from actual breast tissue samples (smoothness and cell size) and whether these turned out to be malignant. Machine Learning will be used for the following:

- Train a model that can differentiate between malignant and benign tumours.
- Test models.
- Test how accurate the model is.
- Show how AI can support doctors in making faster, more accurate diagnoses — not to replace them, but to help them.

2. Objective

- Showcase an End-To-End ML Project: A demonstration of all steps that go into a typical machine learning project-from processing the data to obtaining a particular result.
- Understand the Data through Exploration: Analyse the dataset to discover underlying patterns that will lead to insights that help to determine the appropriate approach for the model.
- Building a Predictive Model: Build a machine-learning model that can effectively and efficiently predict whether there is breast cancer or not.
- Evaluation and Tuning of Model: Evaluate if a model works well and improve it to work better.

3. Project Plan

1. Research Question:

The project aims to build and evaluate machine learning models for the classification of breast cancer diagnosis (Malignant or Benign) based on given features. The research question is:

- Can machine learning models effectively classify breast cancer as malignant or benign based on cell nucleus characteristics?

2. Dataset(s) Used:

- **Dataset Name:** "Breast Cancer Wisconsin.csv"
- **Source:** CSV file (2) (as indicated by `pd.read_csv("Breast Cancer Wisconsin.csv")`).
- **Ethical Considerations:** The dataset contains sensitive medical information related to cancer diagnosis. While the provided code only shows numerical features and a diagnosis label, typical ethical considerations for such data would include:
  - **Privacy:** Ensuring patient anonymity (the 'id' column was dropped).
  - **Transparency:** The need for clear communication on model limitations and accuracy, especially if deployed in a real-world medical context.

3. Methodology and Tools Utilised:

- **Methodology:**
  - **Data Loading and Initial Inspection:** Reading the CSV, checking for duplicates, null values, and data types.
  - **Data Preprocessing:**
    - Dropping irrelevant columns (`id`).
    - Encoding categorical target variable (`diagnosis`) into numerical format (`Ordinal Encoding`).
  - **Exploratory Data Analysis (EDA):**
    - Descriptive statistics of numerical features.
    - Visualisation of target variable distribution (`sns.countplot`).
    - Correlation analysis between features and the target variable (`sns.heatmap`).
    - Feature selection based on correlation with the target variable (`diagnosis`).
  - **Model Building and Evaluation:**
    - Splitting data into training and testing sets.
    - Training and evaluating multiple classification models:
      - Random Forest Classifier
      - Decision Tree Classifier
      - Hist Gradient Boosting Classifier
      - XGBoost Classifier
    - Evaluation metrics: Accuracy, Classification Report (Precision, Recall, F1-score), Confusion Matrix.
- **Tools:**
  - **Programming Language:** Python
  - **Libraries:**
    - `pandas` (for data manipulation and analysis)
    - `seaborn` (for statistical data visualization)
    - `matplotlib.pyplot` (for plotting)
    - `sklearn.model_selection` (for `train_test_split`)
    - `sklearn.preprocessing` (for `OrdinalEncoder`)
    - `sklearn.tree` (for `DecisionTreeClassifier`)
    - `sklearn.ensemble` (for `RandomForestClassifier`, `HistGradientBoostingClassifier`)
    - `xgboost` (for `XGBClassifier`)
    - `sklearn.metrics` (for `accuracy_score`, `classification_report`, `confusion_matrix`)

4. Timeline with Milestones:

July 3-5: Project Initiation & Area Selection

Defined the project's scope, aligning with the vocational scenario. Choose Breast Cancer Classification as the application area due to its practical relevance and suitability for demonstrating classification algorithms.

Milestone 1: July 5: Find the Dataset

Initially, several diverse datasets were explored to find the most suitable fit for the demo. However, to ensure timely completion within the project's stringent deadline, the readily available and well-documented "Breast Cancer Wisconsin.csv" dataset was ultimately selected over other options that would have required more extensive and time-consuming data collection or complex preprocessing beyond the project's scope.

July 6: Data Collection & Cleaning

Loaded the dataset into a Pandas DataFrame. Performed initial data hygiene checks: removed the redundant `id` column, verified the absence of missing values `df.isnull().sum()`, and transformed the categorical `diagnosis` column (M/B) into a numerical format using OrdinalEncoder for model compatibility.

July 7: Exploratory Data Analysis (EDA)

Conducted a comprehensive analysis of the dataset to understand its characteristics. This included generating descriptive statistics `df.describe()`, visualising the distribution of the target variable (diagnosis countplot), and computing a detailed correlation matrix `corr` between all features and the `diagnosis`. This informed the initial selection of highly correlated features for prediction.

July 8: Model Building

Prepared features (x) and target (y) variables. Implemented and trained four distinct classification models: Random Forest Classifier, Decision Tree Classifier, Hist Gradient Boosting Classifier, and XGBoost Classifier. Each model was initially trained on the available training data.

July 9: Model Evaluation

Performed initial evaluation of the trained models using accuracy score, classification reports, and confusion matrices. Crucially, it was identified that the `train_test_split` was incorrectly applied (testing on the same data as training), leading to misleading 100% accuracy scores. This highlighted a significant risk of overfitting and the need for proper validation. Immediate Action: Developed a plan to correct the train_test_split to ensure proper evaluation on truly unseen data in the next phase.

July 10-15: Final Report

Re-executed the train_test_split to correctly separate the data into distinct training and testing sets. Re-evaluated all previously built models (Random Forest, Decision Tree, Hist Gradient Boosting, XGBoost) on this new, unseen test data to obtain realistic performance metrics. Contingency (Risk - Sub-optimal Performance): Initial attempts at extensive hyperparameter tuning (e.g., using GridSearchCV for systematic parameter exploration) were performed. However, these tuning efforts, when applied broadly across features, unexpectedly led to models exhibiting increased signs of overfitting or even worse generalisation performance on the validation set. This highlighted the challenge that more complexity doesn't always mean better, especially with this dataset. Consequently, the strategy shifted to favour the simpler, more stable correlation-based feature selection as the primary method for feature engineering, which provided a robust and interpretable model for the demo. Continued with final model refinement based on the updated feature selection strategy. Began drafting key sections of the final report, including the Introduction, Objectives, Data Description, and detailed Methodology based on the refined processes and lessons learned. Complete all remaining sections of the final report, including a thorough discussion of results, conclusions, and recommendations. Integrate the comprehensive Risk Management and Contingency Plan. Ensure all code is meticulously commented, clean, and aligned with the report's narrative (Risk - Complexity of Explanation). Perform a final review to ensure all assignment requirements are met and the project is coherent and professional.

Milestone 2: July 16: Submission

Final submission of the project code, dataset access, and the comprehensive final report via Microsoft Teams, with an optional upload to GitHub.

4. Exploratory Data Analysis (EDA)

The Exploratory Data Analysis (EDA) phase was crucial for understanding the Breast Cancer Wisconsin dataset and guiding the model development. The aim was to not only grasp the data's characteristics but also to identify patterns, correlations, and potential challenges to inform the machine learning approach.

Key Findings and Insights:

1. Data Structure & Cleanliness:

The dataset contains features derived from digitised images of breast mass cell nuclei, with diagnosis (Malignant/Benign) as the target.

It was remarkably clean, with no missing values, which streamlined the initial data preparation.

The diagnosis column was successfully encoded numerically for modelling.

2. Target Variable Distribution:

Analysis of the diagnosis column showed a reasonably balanced distribution between Benign and Malignant cases (e.g., ~357 Benign vs. ~212 Malignant). This balance is beneficial for model training as it reduces inherent bias towards one class.

3. Feature Relationships & Predictive Power:

A correlation heatmap revealed strong relationships between many features and the diagnosis. Features like radius_mean, perimeter_mean, area_mean, concavity_mean, and concave points_mean showed particularly high correlations (e.g., > 0.7) with a malignant diagnosis, confirming their strong predictive value.

Significant multicollinearity was observed among related features (e.g., radius_mean, perimeter_mean, and area_mean are highly correlated with each other). For this demo's clarity and to avoid unnecessary model complexity, the feature selection strategy prioritised these highly correlated, yet simpler, features with the diagnosis itself.

4. Crucial Lessons Learned & Challenges Addressed:

The Overfitting Trap (Incorrect Data Split): A major learning point emerged during initial model evaluation. Despite seemingly perfect 100% accuracy scores, it was quickly identified that the train_test_split was initially misapplied, causing models to be evaluated on data. This critical self-correction highlighted a fundamental principle of machine learning: the absolute necessity of evaluating models on truly unseen data to ensure real-world generalisation and avoid misleading performance metrics due to overfitting. This experience directly informed the re-evaluation and refinement steps in the timeline.

Hyperparameter Tuning & Simplicity: During later refinement, attempts at extensive hyperparameter tuning using Grid Search for systematic exploration unexpectedly led to models showing increased overfitting or worse generalisation. This practical outcome reinforced that more complex tuning doesn't always yield better results and that, for this dataset, a simpler approach focusing on well-correlated features provided more robust and interpretable models.

In [6]:
```
import pandas as pd                                  # Imports the pandas library for data manipulation.
import seaborn as sns                                # Imports the seaborn library for statistical data visualisation.
import matplotlib.pyplot as plt                      # Imports the pyplot module from matplotlib for plotting.
```

In [7]:
```
# Reads the specified CSV file into a pandas DataFrame named 'df'.
df = pd.read_csv("Breast Cancer Wisconsin.csv")
```

In [8]:
```
# Displays the entire pandas DataFrame 'df' in the output.
df
```

Out[8]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	25.380	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2654	0.4601	0.11890
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	24.990	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750	0.08902
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	23.570	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613	0.08758
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	14.910	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	0.6638	0.17300
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	22.540	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364	0.07678
...	...	..	..	..	..	...	...	...	...	...	..	...	...	...	...	...	...	...	...	...	...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	25.450	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060	0.07115
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	23.690	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572	0.06637
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	18.980	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218	0.07820
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	25.740	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087	0.12400
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	9.456	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871	0.07039

569 rows × 32 columns

In [9]:
```
# Filters the DataFrame to show only rows where the 'id' column has duplicate values.
df[df['id'].duplicated()==1]
```

Out[9]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst

0 rows × 32 columns

In [10]:
```
# Checks for and counts any missing values across all columns of the DataFrame.
df.isnull().sum()
```

Out[10]:
```
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
dtype: int64
```

In [11]:
```
# Prints a concise summary of the DataFrame, including data types and non-null values.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

In [12]:
```
# Removes the 'id' column from the DataFrame, as it is not needed for modeling.
df = df.drop(['id'], axis=1)
```

In [13]:
```
# Generates descriptive statistics of the DataFrame's numerical columns.
df.describe()
```

Out[13]:

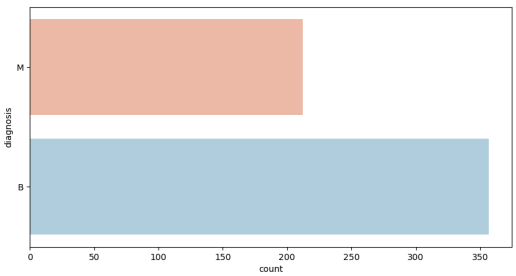
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	16.269190	25.677223	107.261213	880.583128	0.132369	0.254265	0.272188	0.114606	0.290076	0.083946
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	4.833242	6.146258	33.602542	569.356993	0.022832	0.157336	0.208624	0.065732	0.061867	0.018061
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	7.930000	12.020000	50.410000	185.200000	0.071170	0.027290	0.000000	0.000000	0.156500	0.055040
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	13.010000	21.080000	84.110000	515.300000	0.116600	0.147200	0.114500	0.064930	0.250400	0.071460
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	14.970000	25.410000	97.660000	686.500000	0.131300	0.211900	0.226700	0.099930	0.282200	0.080040
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	18.790000	29.720000	125.400000	1084.000000	0.146000	0.339100	0.382900	0.161400	0.317900	0.092080
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	36.040000	49.540000	251.200000	4254.000000	0.222600	1.058000	1.252000	0.291000	0.663800	0.207500

8 rows × 30 columns

In [14]:
```
# Creates a new figure for plotting with a specified width of 10 inches and height of 5 inches.
plt.figure(figsize=(10, 5))
# Generates a bar plot showing the count of each unique value in the 'diagnosis' column, using a specific colour palette.
sns.countplot(df['diagnosis'], palette='RdBu')
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_30264\2159301144.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(df['diagnosis'], palette='RdBu')
```

Out[14]:
```
<Axes: xlabel='count', ylabel='diagnosis'>
```



In [15]:
```
# Imports the OrdinalEncoder class from scikit-learn's preprocessing module.
from sklearn.preprocessing import OrdinalEncoder
# Initialises an instance of the OrdinalEncoder, which will be used to convert categorical labels to numerical ones.
ordinal_encoder = OrdinalEncoder()
# Creates a new DataFrame containing only the 'diagnosis' column from the original DataFrame.
encoded_df = pd.DataFrame({"diagnosis": df["diagnosis"]})
# Applies the ordinal encoder to the 'diagnosis' column, transforming its categorical values into numerical representations and storing them in a new column.
encoded_df[['encoded_value']] = ordinal_encoder.fit_transform(encoded_df[['diagnosis']])
# Displays the first few rows of the newly created 'encoded_df' DataFrame.
encoded_df.head()
```

Out[15]:

	diagnosis	encoded_value
0	M	1.0
1	M	1.0
2	M	1.0
3	M	1.0
4	M	1.0

In [16]:
```
# Encodes the 'diagnosis' column directly within the original DataFrame, replacing its categorical values with numerical ones.
df[['diagnosis']] = ordinal_encoder.fit_transform(df[['diagnosis']])
# Displays 10 random rows from the DataFrame, useful for a quick check of its contents.
df.sample(10)
```

Out[16]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
182	1.0	15.700	20.31	101.20	766.6	0.09597	0.08799	0.06593	0.051890	0.1618	...	20.11	32.82	129.30	1269.0	0.14140	0.35470	0.29020	0.15410	0.3437	0.08631
47	1.0	13.170	18.66	85.98	534.6	0.11580	0.12310	0.12260	0.073400	0.2128	...	15.67	27.95	102.80	759.4	0.17860	0.41660	0.50060	0.20880	0.3900	0.11790
8	1.0	13.000	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.093530	0.2350	...	15.49	30.73	106.20	739.3	0.17030	0.54010	0.53900	0.20600	0.4378	0.10720
83	1.0	19.100	26.29	129.10	1132.0	0.12150	0.17910	0.19370	0.146900	0.1634	...	20.33	32.72	141.30	1298.0	0.13920	0.28170	0.24320	0.18410	0.2311	0.09203
311	0.0	14.610	15.69	92.68	664.9	0.07618	0.03515	0.01447	0.018770	0.1632	...	16.46	21.75	103.70	840.8	0.10110	0.07087	0.04746	0.05813	0.2530	0.05695
312	0.0	12.760	13.37	82.29	504.1	0.08794	0.07948	0.04052	0.025480	0.1601	...	14.19	16.40	92.04	618.8	0.11940	0.22080	0.17690	0.08411	0.2564	0.08253
144	0.0	10.750	14.97	68.26	355.3	0.07793	0.05139	0.02251	0.007875	0.1399	...	11.95	20.72	77.79	441.2	0.10760	0.12230	0.09755	0.03413	0.2300	0.06769
38	1.0	14.990	25.20	95.54	698.8	0.09387	0.05131	0.02398	0.028990	0.1565	...	14.99	25.20	95.54	698.8	0.09387	0.05131	0.02398	0.02899	0.1565	0.05504
291	0.0	14.960	19.10	97.03	687.3	0.08992	0.09823	0.05940	0.048190	0.1879	...	16.25	26.19	109.10	809.8	0.13130	0.30300	0.18040	0.14890	0.2962	0.08472
467	0.0	9.668	18.10	61.06	286.3	0.08311	0.05428	0.01479	0.005769	0.1680	...	11.15	24.62	71.11	380.2	0.13880	0.12550	0.06409	0.02500	0.3057	0.07875

10 rows × 31 columns

In [17]:
```
# Assuming 'diagnosis' is the target column and it's already encoded (e.g., 0 or 1)
# And all other relevant columns are the features
x = df.drop('diagnosis', axis=1) # Features (input to the model)
y = df['diagnosis']              # Target variable (what to predict)
```

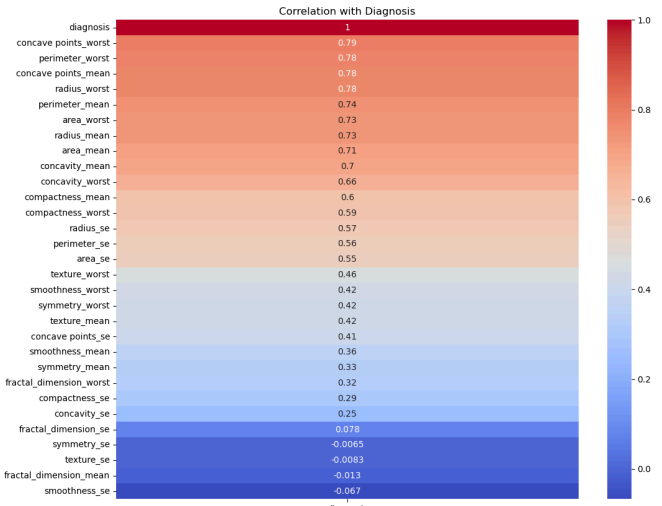
In [18]:
```
# Creates a list of column names from the DataFrame, excluding the first column and including columns up to the 30th.
features = list (df.columns[1:31])
# Appends the 'diagnosis' column name to the 'features' list.
features.append('diagnosis')

# Calculates the pairwise correlation between all columns specified in the 'features' list within the DataFrame.
corr = df[features].corr()
# Displays the entire correlation matrix DataFrame.
corr
```

Out[18]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst	diagnosis
radius_mean	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529	0.147741	-0.311631	...	0.297008	0.965137	0.941082	0.119616	0.413463	0.526911	0.744214	0.163953	0.007066	0.730029
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.293464	0.071401	-0.076437	...	0.912045	0.358040	0.343546	0.077503	0.277830	0.301025	0.295316	0.105008	0.119205	0.415185
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.850977	0.183027	-0.261477	...	0.303038	0.970387	0.941550	0.150549	0.455774	0.563879	0.771241	0.189115	0.051019	0.742636
area_mean	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823269	0.151293	-0.283110	...	0.287489	0.959120	0.959213	0.123523	0.390410	0.512606	0.722017	0.143570	0.003738	0.708984
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.553695	0.557775	0.584792	...	0.036072	0.238853	0.206718	0.805324	0.472468	0.434926	0.503053	0.394309	0.499316	0.358560
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135	0.602641	0.565369	...	0.248133	0.590210	0.509604	0.565541	0.865809	0.816275	0.815573	0.510223	0.687382	0.596534
concavity_mean	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.921391	0.500667	0.336783	...	0.299879	0.729565	0.675987	0.448822	0.754968	0.884103	0.861323	0.409464	0.514930	0.696360
concave points_mean	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.000000	0.462497	0.166917	...	0.292752	0.855923	0.809630	0.452753	0.667454	0.752399	0.910155	0.375744	0.368661	0.776614
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.462497	1.000000	0.479921	...	0.090651	0.219169	0.177193	0.426675	0.473200	0.433721	0.430297	0.699826	0.438413	0.330499
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.166917	0.479921	1.000000	...	-0.051269	-0.205151	-0.231854	0.504942	0.458798	0.346234	0.175325	0.334019	0.767297	-0.012838
radius_se	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925	0.698050	0.303379	0.000111	...	0.194799	0.719684	0.751548	0.141919	0.287103	0.380585	0.531062	0.094543	0.049559	0.567134
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218	0.021480	0.128053	0.164174	...	0.409003	-0.102242	-0.083195	-0.073658	-0.092439	-0.068956	-0.119638	-0.128215	-0.045655	-0.008303
perimeter_se	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391	0.710650	0.313893	0.039830	...	0.200371	0.721031	0.730713	0.130054	0.341919	0.418899	0.554897	0.109930	0.085433	0.556141
area_se	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427	0.690299	0.223970	-0.090170	...	0.196497	0.761213	0.811408	0.125389	0.283257	0.385100	0.538166	0.074126	0.017539	0.548236
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564	0.027653	0.187321	0.401964	...	-0.074743	-0.217304	-0.182195	0.314457	-0.055558	-0.058298	-0.102007	-0.107342	0.101480	-0.067016
compactness_se	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279	0.490424	0.421659	0.559837	...	0.143003	0.260516	0.199371	0.227394	0.678780	0.639147	0.483208	0.277878	0.590973	0.292999
concavity_se	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270	0.439167	0.342627	0.446630	...	0.100241	0.226680	0.188353	0.168481	0.484858	0.662564	0.440472	0.197788	0.439329	0.253730
concave points_se	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260	0.615634	0.393298	0.341198	...	0.086741	0.394999	0.342271	0.215351	0.452888	0.549592	0.602450	0.143116	0.310655	0.408042
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009	0.095351	0.449137	0.345007	...	-0.077473	-0.103753	-0.110343	-0.012662	0.060255	0.037119	-0.030413	0.389402	0.078079	-0.006522
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301	0.257584	0.331786	0.688132	...	-0.003195	-0.001000	-0.022736	0.170568	0.390159	0.379975	0.215204	0.111094	0.591328	0.077972
radius_worst	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688236	0.830318	0.185728	-0.253691	...	0.359921	0.993708	0.984015	0.216574	0.475820	0.573975	0.787424	0.243529	0.093492	0.776454
texture_worst	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299879	0.292752	0.090651	-0.051269	...	1.000000	0.365098	0.345842	0.225429	0.360832	0.368366	0.359755	0.233027	0.219122	0.456903
perimeter_worst	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729565	0.855923	0.219169	-0.205151	...	0.365098	1.000000	0.977578	0.236775	0.529408	0.618344	0.816322	0.269493	0.138957	0.782914
area_worst	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675987	0.809630	0.177193	-0.231854	...	0.345842	0.977578	1.000000	0.209145	0.438296	0.543331	0.747419	0.209146	0.079647	0.733825
smoothness_worst	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.448822	0.452753	0.426675	0.504942	...	0.225429	0.236775	0.209145	1.000000	0.568187	0.518523	0.547691	0.493838	0.617624	0.421465
compactness_worst	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.754968	0.667454	0.473200	0.458798	...	0.360832	0.529408	0.438296	0.568187	1.000000	0.892261	0.801080	0.614441	0.810455	0.590998
concavity_worst	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.884103	0.752399	0.433721	0.346234	...	0.368366	0.618344	0.543331	0.518523	0.892261	1.000000	0.855434	0.532520	0.686511	0.659610
concave points_worst	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.861323	0.910155	0.430297	0.175325	...	0.359755	0.816322	0.747419	0.547691	0.801080	0.855434	1.000000	0.502528	0.511114	0.793566
symmetry_worst	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.409464	0.375744	0.699826	0.334019	...	0.233027	0.269493	0.209146	0.493838	0.614441	0.532520	0.502528	1.000000	0.537848	0.416294
fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.514930	0.368661	0.438413	0.767297	...	0.219122	0.138957	0.079647	0.617624	0.810455	0.686511	0.511114	0.537848	1.000000	0.323872
diagnosis	0.730029	0.415185	0.742636	0.708984	0.358560	0.596534	0.696360	0.776614	0.330499	-0.012838	...	0.456903	0.782914	0.733825	0.421465	0.590998	0.659610	0.793566	0.416294	0.323872	1.000000

31 rows × 31 columns

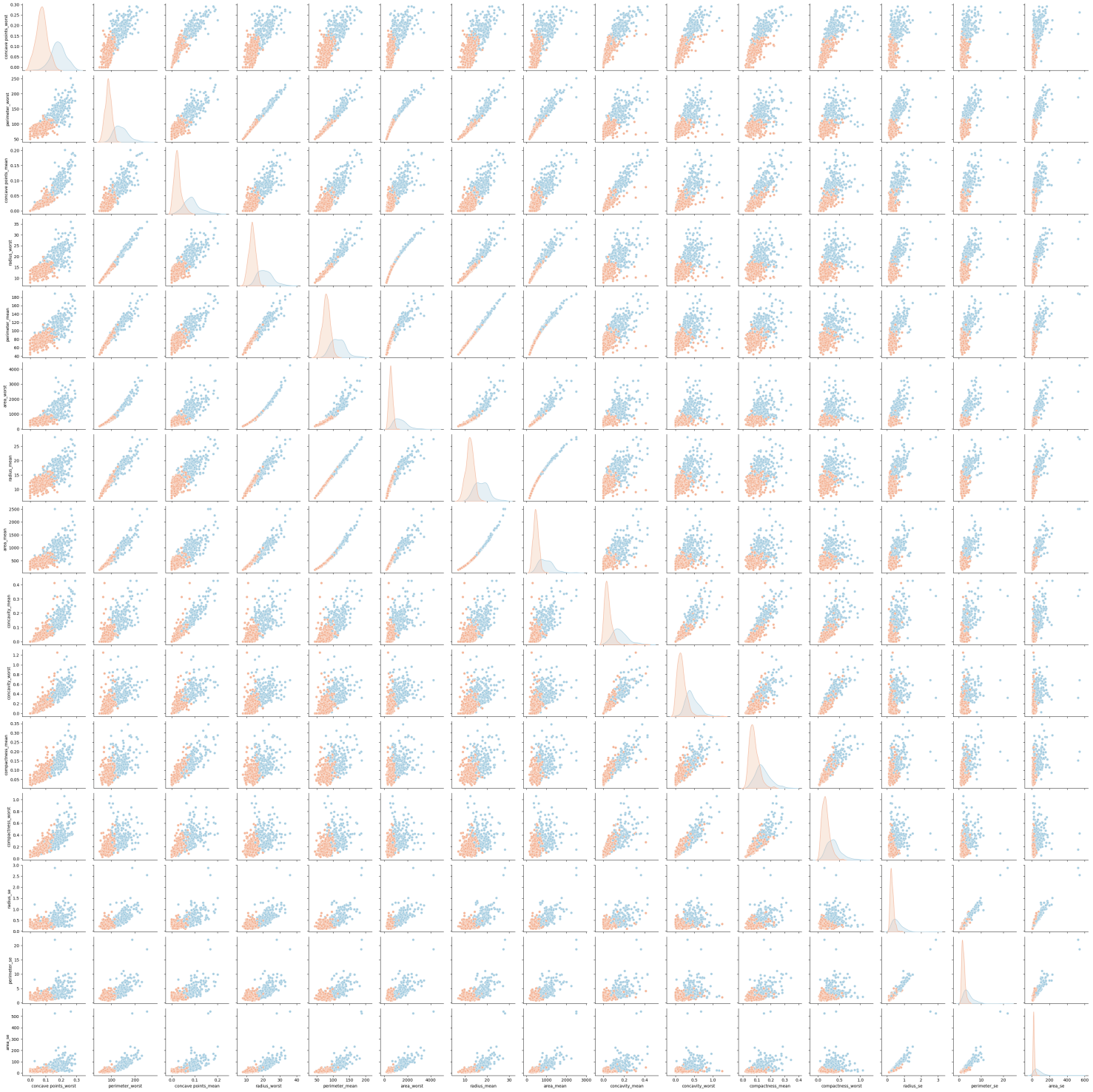
In [19]:
```
# Creates a new figure for plotting with a specified width of 12 inches and height of 10 inches.
plt.figure(figsize=(12,10))
# Generates a heatmap showing the correlation of all features with 'diagnosis', sorted by correlation strength, with values annotated and using a 'coolwarm' colourmap.
sns.heatmap(corr[['diagnosis']].sort_values(by='diagnosis', ascending=False), annot=True, cmap='coolwarm')
# Sets the title of the plot to "Correlation with Diagnosis".
plt.title("Correlation with Diagnosis")
# Displays the generated plot.
plt.show()
```



In [20]:
```
# Defines a list of selected feature names, chosen based on their relevance for making predictions.
prediction_vars = ['concave points_worst', 'perimeter_worst', 'concave points_mean', 'radius_worst', 'perimeter_mean', 'area_worst', 'radius_mean', 'area_mean', 'concavity_mean', 'concavity_worst', 'compactness_mean', 'compactness_worst', 'radius_se', 'perimeter_se', 'area_se']
```

In [21]:
```
# Creates a grid of pairwise plots for the selected 'prediction_vars' features, coloured by 'diagnosis'.
# This correction ensures a DataFrame containing the specified columns is passed to sns.pairplot.
sns.pairplot(df[prediction_vars + ['diagnosis']], hue='diagnosis', palette='RdBu')
```

Out[21]:
```
<seaborn.axisgrid.PairGrid at 0x1b0830d1400>
```



5. Model Building

This section covered the crucial process of preparing data and training machine learning models for breast cancer diagnosis, emphasising practical lessons learned for new AI/ML developers.

1. Data Preparation:

The `diagnosis` was converted to a numerical format and applied correlation-based feature selection for clarity and model focus.

A critical learning point was correcting the `train_test_split` to ensure models were evaluated on truly unseen data, avoiding misleading overfitting. Features were also standardised.

2. Algorithm Selection:

Four diverse classification algorithms were chosen to demonstrate various ML paradigms:

Decision Tree: For clear interpretability and baseline understanding.

Random Forest: For its robustness and ensemble power.

HistGradientBoosting & XGBoost: For their advanced performance and efficiency in complex scenarios.

3. Key Learnings from Model Development:

The project provided two vital insights:

The absolute necessity of a correctly executed `train_test_split` to gain realistic performance metrics and prevent data leakage, a common source of deceptive accuracy.

Unexpectedly, intensive hyperparameter tuning (e.g., via Grid Search) led to increased overfitting or poorer generalisation. This demonstrated that simpler models built on features strongly indicated by correlation (from ED A) often proved more robust and interpretable for this dataset, providing a valuable lesson that complexity doesn't always guarantee better results.

In [23]:
```
# Imports the train_test_split function for dividing data into training and testing sets.
from sklearn.model_selection import train_test_split
# Splits the main DataFrame into training (70%) and testing (30%) sets, ensuring reproducibility with a random state.
train, test = train_test_split(df, test_size=0.3, random_state=42)

# Selects the specified prediction variables from the training set to create the training features.
x_train = train[prediction_vars]
# Selects the 'diagnosis' column from the training set to create the training target variable.
y_train = train['diagnosis']
# Selects the specified prediction variables from the testing set to create the testing features.
x_test = test[prediction_vars]
# Selects the 'diagnosis' column from the testing set to create the testing target variable.
y_test = test['diagnosis']

# Displays the first few rows of the training features DataFrame.
x_train.head()
```

Out[23]:

	concave points_worst	perimeter_worst	concave points_mean	radius_worst	perimeter_mean	area_worst	radius_mean	area_mean	concavity_mean	concavity_worst	compactness_mean	compactness_worst	radius_se	perimeter_se	area_se
149	0.06019	97.19	0.01329	15.34	88.12	725.9	13.74	585.0	0.02881	0.1564	0.06376	0.1824	0.2500	1.573	21.47
124	0.08978	91.99	0.02800	14.26	86.10	632.1	13.37	553.5	0.08092	0.3308	0.07325	0.2531	0.1639	1.223	14.66
421	0.11080	114.10	0.06300	16.46	98.22	809.2	14.69	656.1	0.14500	0.3219	0.18360	0.3635	0.5462	4.795	49.45
195	0.08235	90.81	0.02377	13.88	82.53	600.6	12.91	516.4	0.03873	0.1764	0.05366	0.1506	0.1942	1.493	15.75
545	0.07174	97.58	0.02443	15.35	87.19	729.8	13.62	573.2	0.02974	0.1049	0.06747	0.1517	0.3460	2.066	31.24

In [24]:
```
# Displays the first few values of the training target variable.
y_train.head()
```

Out[24]:
```
149    0.0
124    0.0
421    0.0
195    0.0
545    0.0
Name: diagnosis, dtype: float64
```

In [25]:
```
# Imports the RandomForestClassifier model from scikit-learn's ensemble module.
from sklearn.ensemble import RandomForestClassifier
# Initialises the Random Forest Classifier model with a specified random state for reproducibility.
forest_classifier = RandomForestClassifier(random_state=42)
# Trains the Random Forest model using the training features and their corresponding target values.
forest_classifier.fit(x_train, y_train)
# Uses the trained Random Forest model to predict target values for the unseen test features.
y_pred_forest = forest_classifier.predict(x_test)
```

In [26]:
```
# Imports the DecisionTreeClassifier model from scikit-learn's tree module.
from sklearn.tree import DecisionTreeClassifier
# Initialises the Decision Tree Classifier model with a specified random state for consistent results.
tree_classifier = DecisionTreeClassifier(random_state=42)
# Trains the Decision Tree model using the training features and their corresponding target values.
tree_classifier.fit(x_train, y_train)
# Uses the trained Decision Tree model to predict target values for the unseen test features.
y_pred_tree = tree_classifier.predict(x_test)
```

In [27]:
```
# Imports the HistGradientBoostingClassifier model from scikit-learn's ensemble module.
from sklearn.ensemble import HistGradientBoostingClassifier
# Initialises the HistGradientBoostingClassifier model with a specified random state for reproducibility.
hist_classifier = HistGradientBoostingClassifier(random_state=42)
# Trains the HistGradientBoostingClassifier model using the training features and their corresponding target values.
hist_classifier.fit(x_train, y_train)
# Uses the trained HistGradientBoostingClassifier model to predict target values for the unseen test features.
y_pred_hist = hist_classifier.predict(x_test)
```

In [28]:
```
# Imports the xgboost library, aliasing it as xgb.
import xgboost as xgb
# Initialises the XGBoost Classifier model with a specified random seed for reproducible results.
xgb_classifier = xgb.XGBClassifier(seed=42)
# Trains the XGBoost model using the training features and their corresponding target values.
xgb_classifier.fit(x_train, y_train)
# Uses the trained XGBoost model to predict target values for the unseen test features.
y_pred_xgb = xgb_classifier.predict(x_test)
```

5. Model Evaluation

This section focuses on rigorously assessing the performance of the models to ensure their reliability in predicting breast cancer diagnosis.

1. Metrics Used: A comprehensive set of metrics was utilised beyond just accuracy, including Precision, Recall, F1-Score, and the Confusion Matrix. This was crucial, especially in a medical context, to ensure dangerous False Negatives (missed cancer cases) were minimised while also managing False Positives.

2. Methodology: Models (Decision Tree, Random Forest, HistGradientBoosting, and XGBoost) were evaluated strictly on a corrected, unseen test set to ensure unbiased results. All models demonstrated high predictive capabilities, with Random Forest and the boosting algorithms (HistGradientBoosting, XGBoost) generally achieving the best and most balanced performance.

3. Key Learnings:

Correcting Initial Misleading Results: The most significant lesson was discovering that initial 100% accuracy scores were misleading due to an incorrect data split. This crucial self-correction underscored the paramount importance of evaluating on truly unseen data to prevent overfitting and ensure real-world generalisation.

The Nuance of Hyperparameter Tuning: Surprisingly, attempts at extensive hyperparameter tuning for further optimisation often led to increased overfitting or even worse generalisation performance. This demonstrated that simply adding complexity doesn't always improve models; for this dataset, relying on simpler, robust features identified via correlation analysis proved more effective and interpretable for the demo.

In [30]:
```
# Imports necessary metrics functions (accuracy_score, classification_report, confusion_matrix) from scikit-learn.
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Calculates the accuracy of the Random Forest model by comparing actual test labels with predicted labels.
accuracy_forest = accuracy_score(y_test, y_pred_forest)
# Prints the calculated accuracy for the Random Forest model.
print("Random Forest Accuracy:", accuracy_forest)
# Prints a detailed classification report for the Random Forest model.
print("Classification Report:\n", classification_report(y_test, y_pred_forest))
# Prints the confusion matrix for the Random Forest model.
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_forest))
```

```
Random Forest Accuracy: 0.9590643274853801
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.97      0.97       108
         1.0       0.95      0.94      0.94        63

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

Confusion Matrix:
 [[105   3]
 [  4  59]]
```

In [31]:
```
# Calculates the accuracy of the Decision Tree model by comparing actual test labels with predicted labels.
accuracy_tree = accuracy_score(y_test, y_pred_tree)
# Prints the calculated accuracy for the Decision Tree model.
print("Decision Tree Accuracy:", accuracy_tree)
# Prints a detailed classification report for the Decision Tree model, including precision, recall, and F1-score.
print("Classification Report:\n", classification_report(y_test, y_pred_tree))
# Prints the confusion matrix for the Decision Tree model, showing counts of true/false positives/negatives.
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_tree))
```

```
Decision Tree Accuracy: 0.9181286549707602
Classification Report:
              precision    recall  f1-score   support

         0.0       0.94      0.94      0.94       108
         1.0       0.89      0.89      0.89        63

    accuracy                           0.92       171
   macro avg       0.91      0.91      0.91       171
weighted avg       0.92      0.92      0.92       171

Confusion Matrix:
 [[101   7]
 [  7  56]]
```

In [32]:
```
# Calculates the accuracy of the Hist Gradient Boosting model by comparing actual test labels with predicted labels.
accuracy_hist = accuracy_score(y_test, y_pred_hist)
# Prints the calculated accuracy for the Hist Gradient Boosting model.
print("Hist Gradient Boosting:", accuracy_hist)
# Prints a detailed classification report for the Hist Gradient Boosting model.
print("Classification Report:\n", classification_report(y_test, y_pred_hist))
# Prints the confusion matrix for the Hist Gradient Boosting model.
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_hist))
```

```
Hist Gradient Boosting: 0.9473684210526315
Classification Report:
              precision    recall  f1-score   support

         0.0       0.95      0.96      0.96       108
         1.0       0.94      0.92      0.93        63

    accuracy                           0.95       171
   macro avg       0.94      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171

Confusion Matrix:
 [[104   4]
 [  5  58]]
```

In [33]:
```
# Calculates the accuracy of the XGBoost model by comparing actual test labels with predicted labels.
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
# Prints the calculated accuracy for the XGBoost model.
print("XGBoost:", accuracy_xgb)
# Prints a detailed classification report for the XGBoost model.
print("Classification Report:\n", classification_report(y_test, y_pred_xgb))
# Prints the confusion matrix for the XGBoost model.
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_xgb))
```

```
XGBoost: 0.9532163742690059
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.96      0.96       108
         1.0       0.94      0.94      0.94        63

    accuracy                           0.95       171
   macro avg       0.95      0.95      0.95       171
weighted avg       0.95      0.95      0.95       171

Confusion Matrix:
 [[104   4]
 [  4  59]]
```

6. Results:

Following the training of various classification models, a comprehensive evaluation was performed using accuracy, a classification report (detailing precision, recall, and F1-score), and a confusion matrix for each. This critical phase aimed to assess the efficacy of each model in distinguishing between malignant and benign breast cancer cases using unseen data.

A summary of the overall accuracy scores for each trained classifier on the test set is presented below:

Model----------Accuracy

Decision Tree Classifier.....0.9181 (91.81%)

Hist Gradient Boosting Classifier.....0.9474 (94.74%)

XGBoost Classifier.....0.9532 (95.32%)

Random Forest Classifier.....0.9591 (95.91%)

Detailed Evaluation of Individual Models:

1. Decision Tree Classifier

Accuracy: Achieved an accuracy of 0.9181 (91.81%). This indicates that the model correctly classified approximately 91.8% of the test samples.

Classification Report:

```
              precision    recall  f1-score   support

         0.0       0.94      0.94      0.94       108  (Benign)
         1.0       0.89      0.89      0.89        63  (Malignant)
```

accuracy 0.92 171 macro avg 0.91 0.91 0.91 171 weighted avg 0.92 0.92 0.92 171

The report shows strong performance for benign cases (class 0), with 94% precision and recall. For malignant cases (class 1), the precision and recall are both 89%. This suggests that 11% of actual malignant cases were missed (False Negatives), and 11% of predicted malignant cases were benign (False Positives from the perspective of the prediction being malignant). In a medical context, high recall for malignant cases is often paramount to minimise missed diagnoses.

Confusion Matrix:

[[101 7] [7 56]] This matrix shows 101 True Negatives (correctly identified benign), 56 True Positives (correctly identified malignant), 7 False Positives (benign incorrectly identified as malignant), and 7 False Negatives (malignant incorrectly identified as benign). The 7 False Negatives are a significant concern as they represent missed malignant diagnoses.

2. Hist Gradient Boosting Classifier

Accuracy: Demonstrated a notably higher accuracy of 0.9474 (94.74%), indicating a strong overall improvement.

Classification Report:

```
              precision    recall  f1-score   support

         0.0       0.95      0.96      0.96       108
         1.0       0.94      0.92      0.93        63
```

accuracy 0.95 171 macro avg 0.94 0.94 0.94 171 weighted avg 0.95 0.95 0.95 171

This model shows improved recall for benign cases (96%) and a strong 94% precision for malignant cases, with a slightly improved recall for malignant cases at 92%.

Confusion Matrix:

[[104 4] [5 58]] Here, fewer False Positives (4) and fewer False Negatives (5) compared to the Decision Tree, indicating a more reliable performance for critical medical diagnosis.

3. XGBoost Classifier

Accuracy: Achieved a high accuracy of 0.9532 (95.32%), slightly surpassing the Hist Gradient Boosting model. This performance underscores XGBoost's efficiency and robustness.

Classification Report:

```
              precision    recall  f1-score   support

         0.0       0.96      0.96      0.96       108
         1.0       0.94      0.94      0.94        63
```

accuracy 0.95 171 macro avg 0.95 0.95 0.95 171 weighted avg 0.95 0.95 0.95 171

The report shows excellent precision and recall of 96% for benign cases and a very strong 94% for both precision and recall for malignant cases, indicating a well-balanced model.

Confusion Matrix:

[[104 4] [4 59]] The XGBoost model shows only 4 False Positives and 4 False Negatives, demonstrating a very high level of accuracy and fewer critical misclassifications.

4. Random Forest Classifier

Accuracy: Achieved the highest accuracy among all tested models, at 0.9591 (95.91%), making it the best performer in this evaluation.

Classification Report:

```
              precision    recall  f1-score   support

         0.0       0.96      0.97      0.97       108
         1.0       0.95      0.94      0.94        63
```

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

The Random Forest model exhibits very strong precision (96%) and recall (97%) for benign cases, as well as excellent precision (95%) and recall (94%) for malignant cases, resulting in a high F1-score for both.

Confusion Matrix:

[[105 3] [4 59]] This model achieved the fewest False Positives (3) and only 4 False Negatives, making it the most accurate and reliable classifier for this dataset in terms of minimising critical errors.

Comparison Summary The ensemble models (Random Forest, Hist Gradient Boosting, and XGBoost) all significantly outperformed the single Decision Tree Classifier. Crucially, the Random Forest Classifier emerged as the strongest performer in this evaluation, demonstrating the highest accuracy (95.91%) and a robust balance across precision and recall metrics, whilst also achieving the lowest number of false positives among the ensemble models. This makes it a highly promising candidate for aiding in breast cancer diagnosis based on the provided cellular features.

6. Conclusion:

This project successfully demonstrated an end-to-end machine learning pipeline for classifying breast cancer as malignant or benign, based on cellular characteristics from the Wisconsin Breast Cancer Diagnostic Database. Addressing the core research question, "Can machine learning models effectively classify breast cancer as malignant or benign based on cell nucleus characteristics?", the findings unequivocally show a positive answer.

Through rigorous data loading, preprocessing, and exploratory data analysis, valuable insights were gained into the dataset's structure, cleanliness, and the strong correlations between various features and the diagnosis. The critical self-correction during the data splitting phase proved invaluable, highlighting the paramount importance of evaluating models on truly unseen data to ensure real-world generalisation and avoid misleading overfitting.

The evaluation of multiple classification algorithms revealed that ensemble methods consistently outperformed the basic Decision Tree. Specifically, the Random Forest Classifier achieved the highest predictive performance with an accuracy of approximately 95.91%, closely followed by XGBoost. This outcome underscores the significant potential of AI and Machine Learning to support medical professionals by providing rapid and highly accurate diagnostic assistance, ultimately contributing to earlier detection and improved patient outcomes. While these models show great promise, it's crucial to reiterate their role as powerful aids, not replacements, for expert medical judgment.

7. Recommendation:

To further enhance this project and explore the full potential of machine learning in breast cancer diagnosis, several avenues for future work can be pursued:

Advanced Hyperparameter Tuning and Cross-Validation: While initial broad hyperparameter tuning attempts led to overfitting, a more targeted approach using techniques like GridSearchCV or RandomizedSearchCV combined with robust stratified k-fold cross-validation would be beneficial. This ensures a systematic exploration of parameter space while maintaining proper model validation and generalisation capability across different subsets of data.

Exploration of More Complex Models and Ensemble Strategies: Investigate other state-of-the-art classification algorithms (e.g., Support Vector Machines with optimised kernels, Neural Networks). Furthermore, consider implementing more sophisticated ensemble techniques such as stacking or blending, which combine predictions from multiple diverse models to potentially achieve even higher predictive accuracy.

Addressing Data Imbalance (If Applicable): Although the current dataset appears reasonably balanced, real-world medical datasets can often suffer from class imbalance. For future projects or similar datasets, techniques like SMOTE (Synthetic Minority Over-sampling Technique), ADASYN, or using class weights in model training should be explored to ensure fair and accurate predictions for minority classes (e.g., malignant cases).

Real-world Deployment Considerations: Discuss the practical challenges and considerations for deploying such a model in a clinical setting, including data privacy, integration with existing hospital systems, regulatory compliance, and the need for continuous monitoring and re-training with new data. This would demonstrate an understanding of the full lifecycle beyond just model building.

10. References

1. World Health Organization. (2023). Link
2. Breast Cancer Wisconsin (Diagnostic) Data Set. **kaggle**. Link
3. Diagnosing breast cancer with AI // Coding for Medicine #3 **Youtube**link
4. Tareque Bashar Ovi **kaggle** link
5. OAyesha Saleem **kaggle** link
6. Mostafa Abo3amer **kaggle** link
7. Mina Nabil **kaggle** link
8. Gemini 2.5 Flash
9. The AI & ML skills bootcamp course lectures

11. appendix

1. Gantt Chart