Title

Breast Cancer Diagnosis Prediction

Overview of Problem Statement

This project analyzes a breast cancer dataset to distinguish between benign (B) and malignant (M tumors. The aim is to preprocess the dataset, explore data distributions, visualize patterns, and b machine learning models to accurately classify diagnoses, which is critical for early cancer detec treatment planning.

Objective

Load and preprocess the dataset.

Perform EDA to understand distributions, detect outliers, and explore correlations.

Handle missing and irrelevant columns.

Build classification models to predict cancer diagnosis.

Evaluate models with performance metrics such as accuracy, confusion matrix, and classificatior

Importing necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix, accu
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

Data collection

In [3]:
```
# Load the dataset
df = pd.read_csv("data (1).csv")
```

In [4]:
```
# Display the first few rows of the dataset to understand its structure
print("First few rows of the dataset:")
df.head()
```

```
First few rows of the dataset:
```

Out[4]:

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

5 rows × 33 columns

In [5]:
```
# Display the last few rows of the dataset
print("Last few rows of the dataset:")
df.tail()
```

```
Last few rows of the dataset:
```

Out[5]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smooth
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

5 rows × 33 columns

Data Description

In [6]:
```
# Display the size of the dataset (number of rows and columns)
print("Dataset size:")
print(df.shape)
```

```
Dataset size:
(569, 33)
```

EDA (Exploratory Data Analysis)

In [7]:
```
# Display the columns of the dataset
columns = df.columns
print("Columns in the dataset:",columns)
```

```
Columns in the dataset: Index(['id', 'diagnosis', 'radius_mean', 'texture_mea
'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', 'Unnamed: 32'],
      dtype='object')
```

In [8]:
```
# Numerical columns
numerical_features = df.select_dtypes(include=['int64', 'float64']).columı
print(numerical_features)
```

```
Index(['id', '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', 'Unnamed: 32'],
      dtype='object')
```

In [9]:
```
# Categorical columns
categorical_features= df.select_dtypes(include=['object','datetime']).colı
print(categorical_features)
```

```
Index(['diagnosis'], dtype='object')
```

In [10]:
```
# Get a summary of the dataset
print("Summary of the dataset:")
df.info()
```

```
Summary of the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 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
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [11]:
```
# Display statistical summary for all columns within the dataFram
df.describe(include='all')
```

Out[11]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mea
count	5.690000e+02	569	569.000000	569.000000	569.000000	569.0000(
unique	NaN	2	NaN	NaN	NaN	Na
top	NaN	B	NaN	NaN	NaN	Na
freq	NaN	357	NaN	NaN	NaN	Na
mean	3.037183e+07	NaN	14.127292	19.289649	91.969033	654.8891(
std	1.250206e+08	NaN	3.524049	4.301036	24.298981	351.9141:
min	8.670000e+03	NaN	6.981000	9.710000	43.790000	143.5000(
25%	8.692180e+05	NaN	11.700000	16.170000	75.170000	420.3000(
50%	9.060240e+05	NaN	13.370000	18.840000	86.240000	551.1000(
75%	8.813129e+06	NaN	15.780000	21.800000	104.100000	782.7000(
max	9.113205e+08	NaN	28.110000	39.280000	188.500000	2501.0000(

11 rows × 33 columns

In [12]:
```
# Describe the numerical features
print("Statistical description of numerical features:")
df.describe()
```

```
Statistical description of numerical features:
```

Out[12]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	5
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	

8 rows × 32 columns

In [13]:
```
# Describe the numerical features
print("Statistical description of numerical features:")
df.describe()
```

```
Statistical description of numerical features:
```

Out[13]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	5
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	

8 rows × 32 columns

In [14]:
```
# Check for Null Values
print("Checking for missing values:")
print(df.isnull().sum())
```

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

Column 'Unnamed: 32' contains 569 missing values.This column has only missing values and pr
no useful information.B

In [15]:
```
# Drop unnecessary columns
df = df.drop(columns=['id', 'Unnamed: 32'], errors='ignore')
```

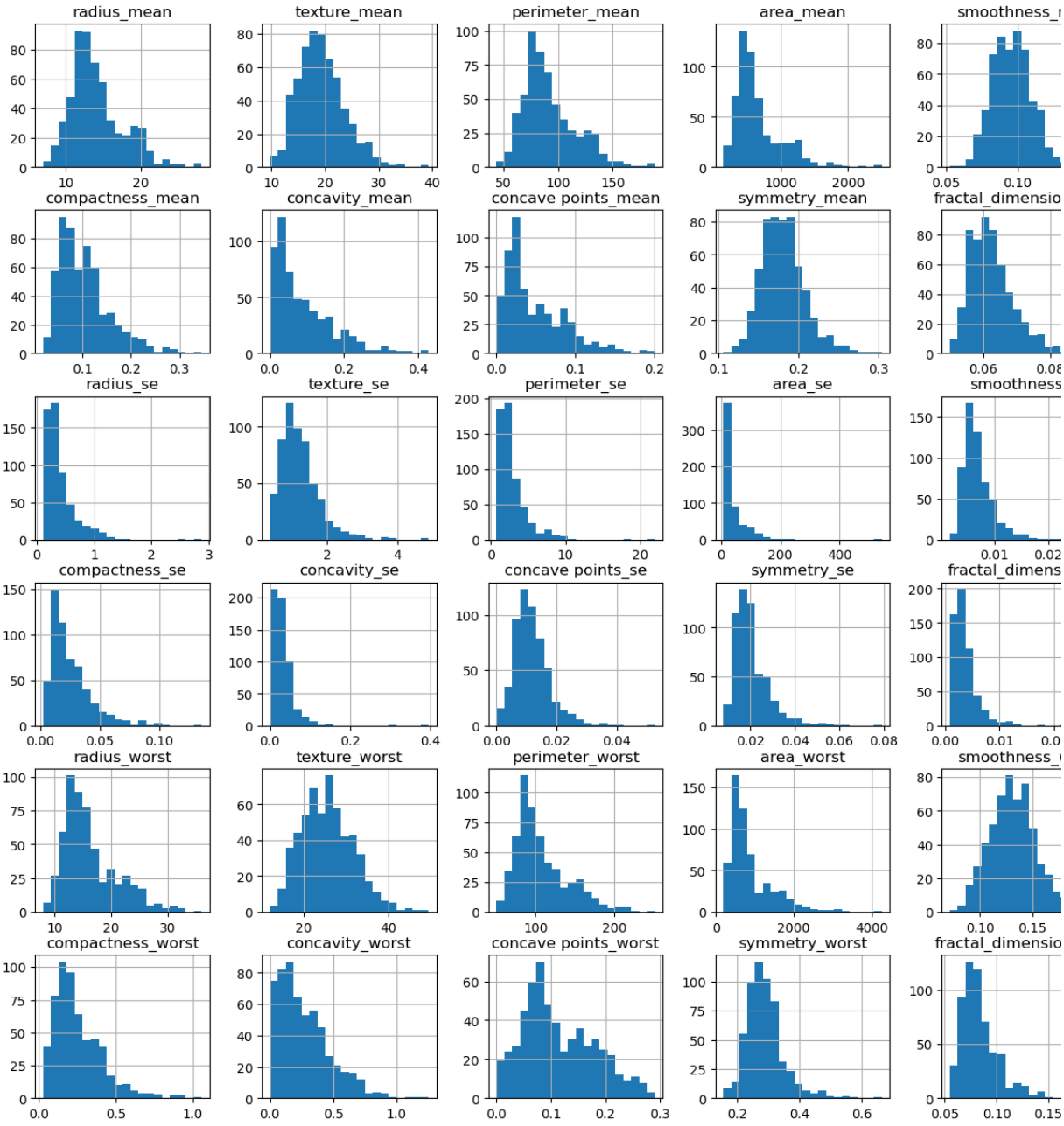
In [16]:
```
# Check for Null Duplicates
print("Checking for duplicate records:")
print(df.duplicated().sum())
```

```
Checking for duplicate records:
0
```

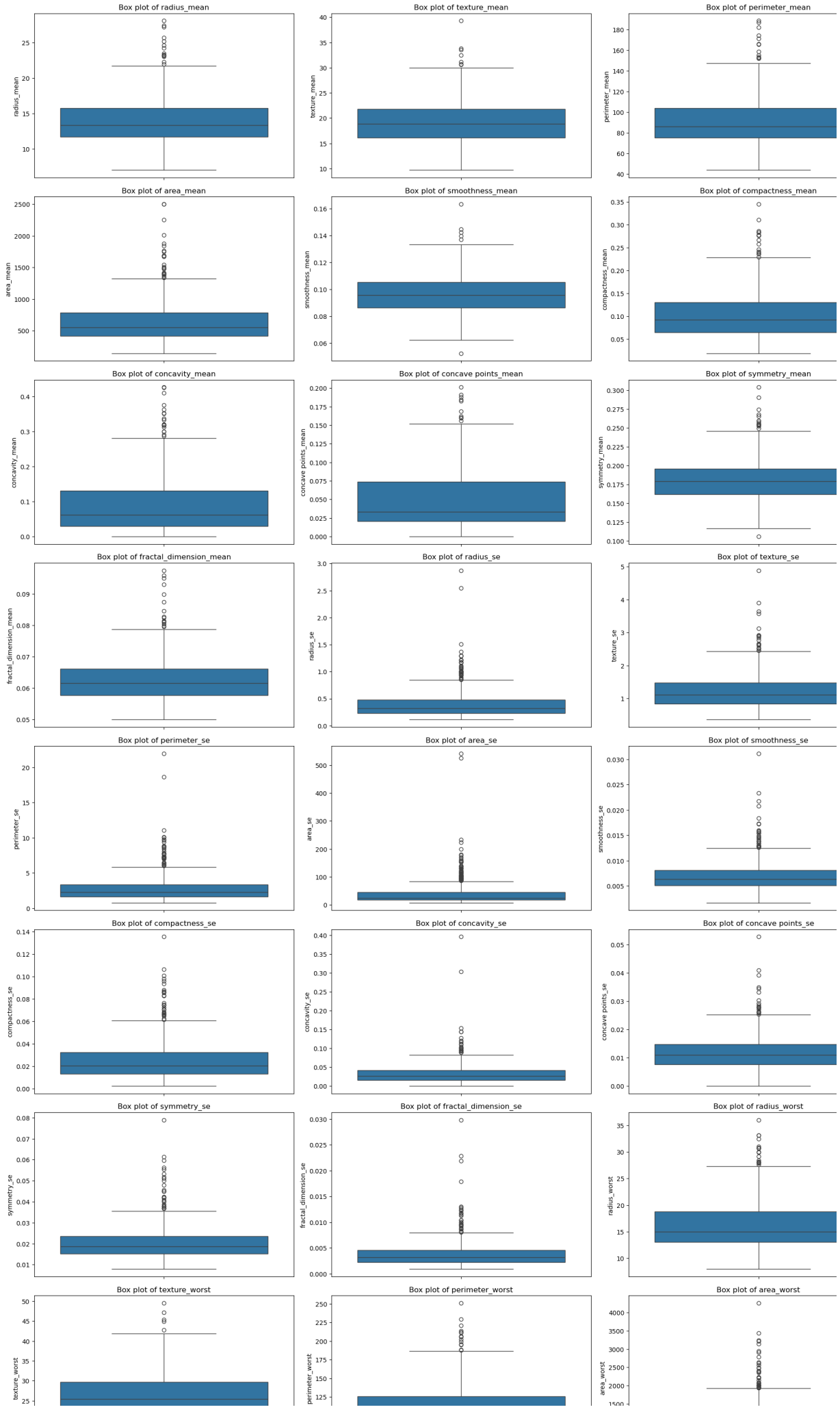
Data Visualization

In [17]:
```
# Get all numerical columns
numerical_features = df.select_dtypes(include=['number']).columns

# Histograms for numerical features
plt.figure(figsize=(20, 20))
df[numerical_features].hist(bins=20, figsize=(15, 15))
plt.suptitle('Histograms of Numerical Features')
plt.show()
```

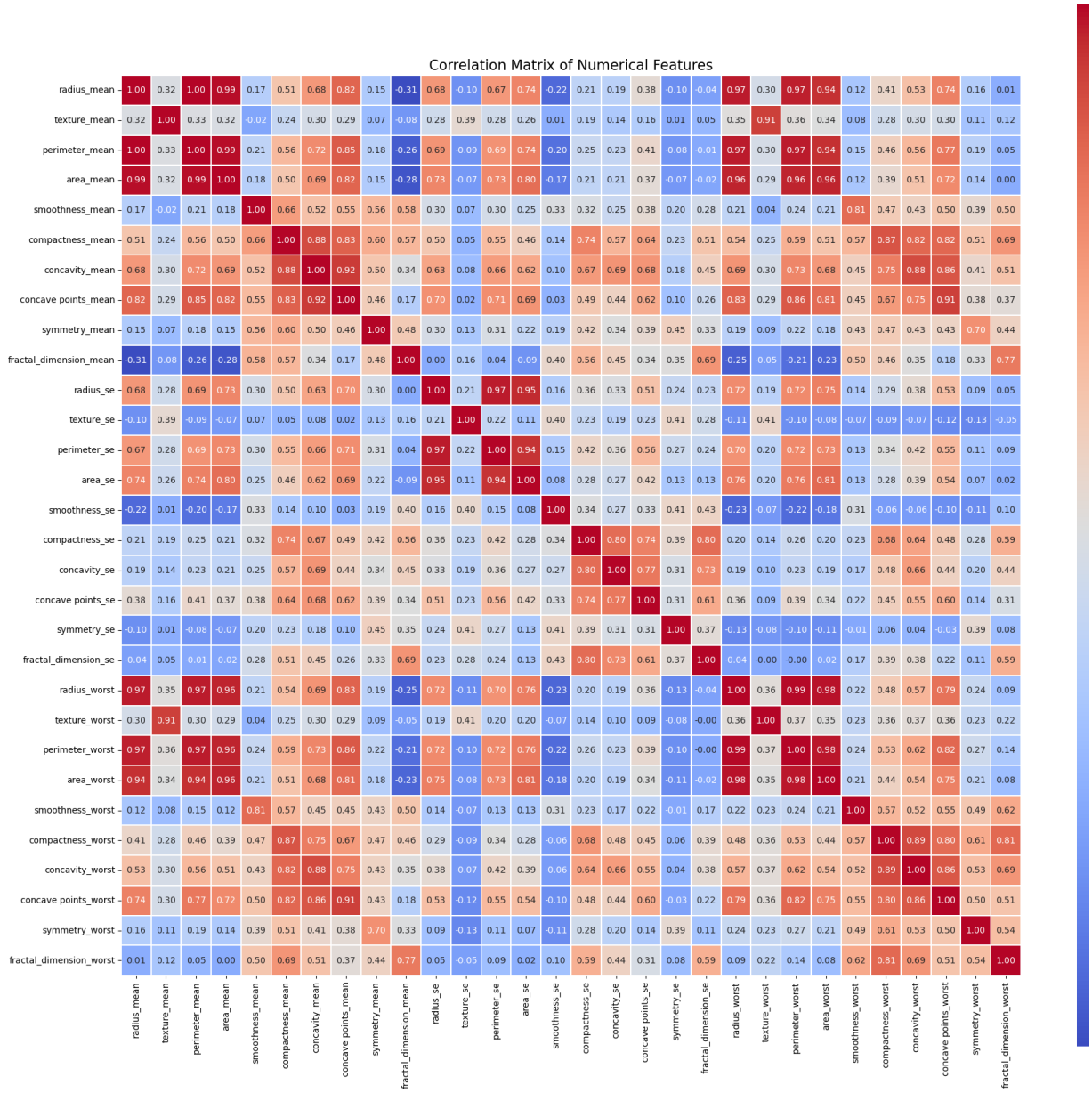
```
<Figure size 2000x2000 with 0 Axes>
```



In [18]:
```
# Get all numerical columns
numerical_features = df.select_dtypes(include=['number']).columns

# Create box plots
plt.figure(figsize=(20, 45))  # Adjust figure size for readability
for i, col in enumerate(numerical_features, 1):
    plt.subplot(11, 3, i)  # 11 rows × 3 columns
    sns.boxplot(y=df[col])  # Changed from x= to y= to avoid the error
    plt.title(f'Box plot of {col}')
    plt.xlabel('')  # Optional: to keep it clean
    plt.tight_layout()

plt.show()
```


Box plot of radius_mean
radius_mean
Box plot of texture_mean
texture_mean
Box plot of perimeter_mean
perimeter_mean
Box plot of area_mean
area_mean
Box plot of smoothness_mean
smoothness_mean
Box plot of compactness_mean
compactness_mean
Box plot of concavity_mean
concavity_mean
Box plot of concave points_mean
concave points_mean
Box plot of symmetry_mean
symmetry_mean
Box plot of fractal_dimension_mean
fractal_dimension_mean
Box plot of radius_se
radius_se
Box plot of texture_se
texture_se
Box plot of perimeter_se
perimeter_se
Box plot of area_se
area_se
Box plot of smoothness_se
smoothness_se
Box plot of compactness_se
compactness_se
Box plot of concavity_se
concavity_se
Box plot of concave points_se
concave points_se
Box plot of symmetry_se
symmetry_se
Box plot of fractal_dimension_se
fractal_dimension_se
Box plot of radius_worst
radius_worst
Box plot of texture_worst
texture_worst
Box plot of perimeter_worst
perimeter_worst
Box plot of area_worst
area_worst

In [19]:
```
# Compute correlation matrix
corr_matrix = df[numerical_features].corr()

# Plot the heatmap with an adjusted figure size
plt.figure(figsize=(20, 18))  # Increased size for 31 features
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=T
plt.title('Correlation Matrix of Numerical Features', fontsize=16)
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



Target Variable Distribution

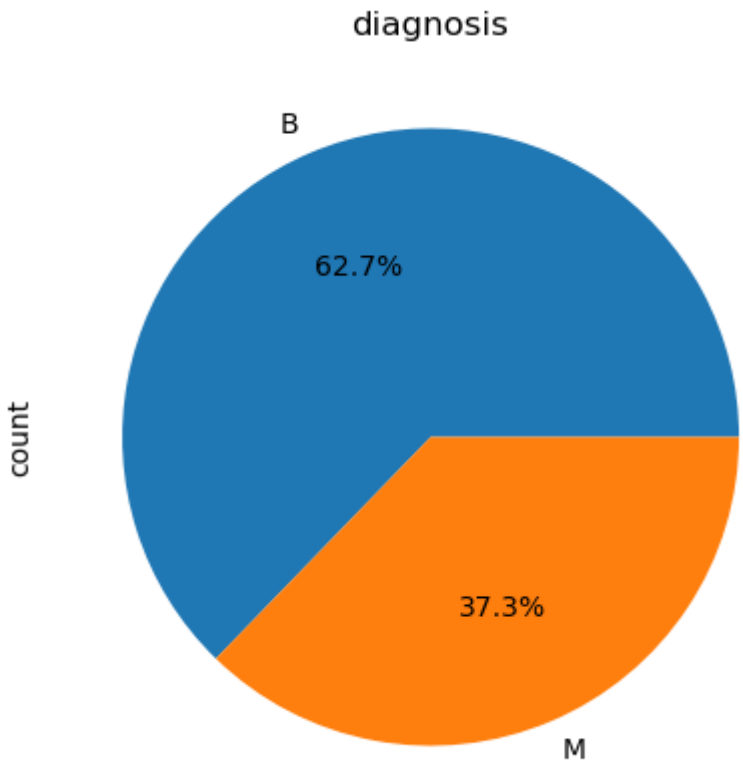
In [20]:
```
#Types of Cancer: Malignant and Benign
print(df["diagnosis"].unique())
```

```
['M' 'B']
```

1. Pie Chart for Target Variable

In [21]:
```
#pie chart for the target variable (Osteoporosis)
plt.figure(figsize=(5,5))
df['diagnosis'].value_counts().plot.pie(autopct='%1.1f%%').set_title('dia
```

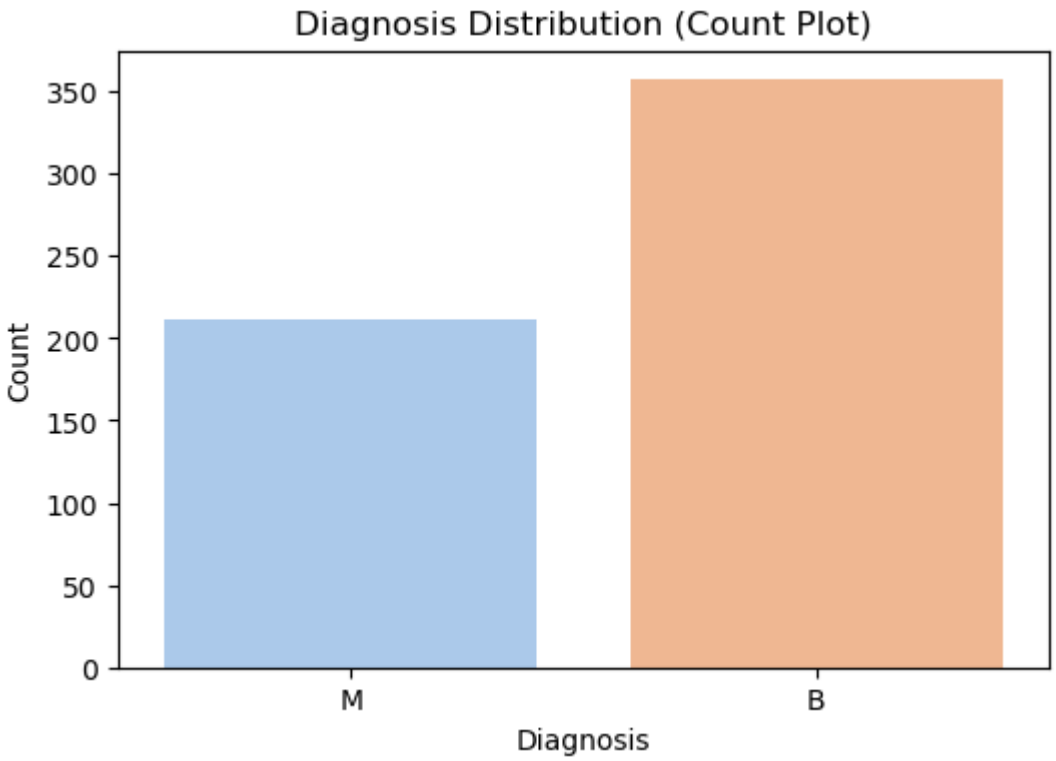
Out[21]:
```
Text(0.5, 1.0, 'diagnosis')
```



The pie chart shows the distribution of cancer diagnoses in the dataset. About 62.7% of the case Benign (B), while 37.3% are Malignant (M). This indicates a moderate class imbalance, which is important to consider during model training.

2. Count Plot for Target Variable

In [22]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='diagnosis', data=df, palette='pastel')
plt.title('Diagnosis Distribution (Count Plot)')
plt.xlabel('Diagnosis')
plt.ylabel('Count')
plt.show()
```



Data Preprocessing

In [23]:
```
# Encode target variable
df['diagnosis'] = df['diagnosis'].map({'B': 0, 'M': 1})
```

In [24]:
```
# Function to remove outliers using IQR method
def remove_outliers_iqr(df, columns):
    mask = pd.Series(True, index=df.index)
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower = Q1 - 1.5 * IQR
        upper = Q3 + 1.5 * IQR
        mask &= df[col].between(lower, upper)
    return df[mask]

# Example usage on all numerical columns (excluding 'id' or 'diagnosis' i
numerical_cols = df.select_dtypes(include='number').columns
df = remove_outliers_iqr(df, numerical_cols)
```

Feature Scaling

In [25]:
```
from sklearn.preprocessing import StandardScaler

# Select numerical features (excluding the target column if needed)
numerical_cols = df.select_dtypes(include='number').columns
numerical_cols = numerical_cols.drop('diagnosis')  # Exclude target colum

# Initialize and fit the scaler
scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

In [26]:
```
df.head()
```

Out[26]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
6	1	1.964461	0.377377	1.973596	2.108920	0.11275
7	1	0.134540	0.594978	0.225722	0.033027	2.12668
10	1	1.065623	1.211941	0.968866	1.020884	-0.93031
11	1	0.968887	-0.157666	1.022372	0.945414	0.31771
13	1	0.997102	1.393701	1.028317	0.953051	-0.76850

5 rows × 31 columns

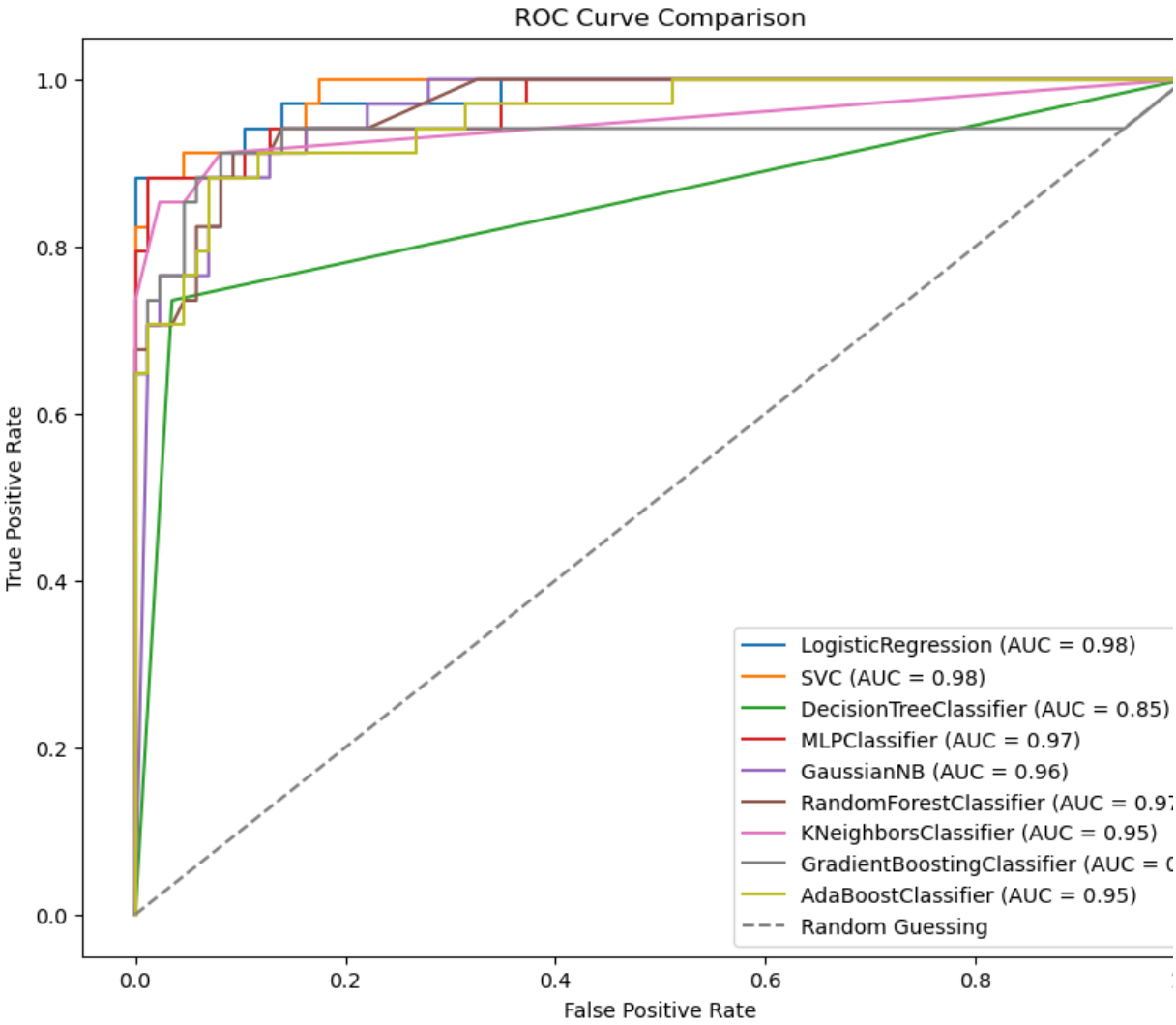
Data Splitting

In [27]:
```
# Split dataset into features and target variable
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']
```

In [28]:
```
# Splitting the dataset into 70-30 that is , 70% of the data is for train

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
```

Model Training & Evaluation (Without feature Selection & Hyperparameter Tuning)

In [29]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score, precision_score, recall_score



from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClas
from sklearn.neighbors import KNeighborsClassifier


## Initialize models
# -----------------------------
models = [
    LogisticRegression(),
    SVC(probability=True),
    DecisionTreeClassifier(),
    MLPClassifier(max_iter=500),
    GaussianNB(),
    RandomForestClassifier(),
    KNeighborsClassifier(),
    GradientBoostingClassifier(),
    AdaBoostClassifier()
]

# -----------------------------
# Evaluate models
# -----------------------------
model_names = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_aucs = []

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

for model in models:
    model.fit(X_train, y_train)
    pred = model.predict(X_test)

    model_names.append(model.__class__.__name__)
    accuracy_scores.append(accuracy_score(y_test, pred))
    precision_scores.append(precision_score(y_test, pred))
    recall_scores.append(recall_score(y_test, pred))
    f1_scores.append(f1_score(y_test, pred))

    if hasattr(model, "predict_proba"):
        proba = model.predict_proba(X_test)[:, 1]
    else:
        proba = model.decision_function(X_test)
        proba = (proba - proba.min()) / (proba.max() - proba.min())

    roc_auc = roc_auc_score(y_test, proba)
    roc_aucs.append(roc_auc)
```



Model-Name	Accuracy	Precision	Recall	F1-Score	ROC AUC
LogisticRegression	0.966667	1.000000	0.882353	0.937500	0.980164
SVC	0.950000	0.937500	0.882353	0.909091	0.983242
MLPClassifier	0.950000	0.937500	0.882353	0.909091	0.970930
KNeighborsClassifier	0.941667	0.935484	0.852941	0.892308	0.947161
RandomForestClassifier	0.908333	0.960000	0.705882	0.813559	0.965800
GradientBoostingClassifier	0.908333	0.896552	0.764706	0.825397	0.928865
DecisionTreeClassifier	0.900000	0.892857	0.735294	0.806452	0.850205
GaussianNB	0.891667	0.800000	0.823529	0.811594	0.962038
AdaBoostClassifier	0.883333	0.857143	0.705882	0.774194	0.953146

Results

The project tested several classifiers including Logistic Regression, SVC, Decision Tree, Randon
Gradient Boosting, AdaBoost, and others. Among them, ensemble models (Random Forest, Grad
Boosting, and AdaBoost) achieved the best performance with high accuracy, precision, recall, F1
and ROC-AUC values. Simpler models like Logistic Regression and Naive Bayes performed reas
but were less effective compared to ensembles.

Conclusion

The analysis shows that machine learning models can reliably classify breast cancer cases into b
and malignant. Ensemble methods provided the most accurate and balanced results, making the
suitable for medical decision support. Overall, the project highlights the potential of ML in early d
and diagnosis of breast cancer.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: