🧬 Benign vs Malignant Findings

📋 Summary

This note outlines the clinical differences between **benign** and **malignant** findings, specifically in breast and lung evaluations.

✅ Benign Findings

- **Growth Pattern:** Non-invasive, localized
- **Borders:** Well-defined, smooth margins
- **Growth Rate:** Slow-growing or stable over time
- **Symptoms:** Often asymptomatic
- **Histology:** Normal cellular architecture, no atypia
- **Examples:**
  - **Breast:** Fibroadenoma, cyst, fibrocystic changes
  - **Lung:** Granuloma, hamartoma, post-inflammatory scar

> 💡 **Impression (Example):**
> *No suspicious mass, distortion, or abnormal calcifications. Findings consistent with benign etiology. Routine follow-up recommended.*

⚠️ Malignant Findings

- **Growth Pattern:** Invasive, potential to spread (metastasis)
- **Borders:** Irregular, spiculated or ill-defined
- **Growth Rate:** Rapid progression
- **Symptoms:** May include pain, weight loss, cough, bleeding

- **Histology:** Atypical cells, mitotic activity, abnormal nuclei
- **Examples:**
  - **Breast:** Invasive ductal carcinoma, lobular carcinoma
  - **Lung:** Adenocarcinoma, squamous cell carcinoma, small cell carcinoma

> 🚨 **Impression (Example):**
> *Spiculated mass in the upper outer quadrant with associated lymphadenopathy. Findings suspicious for malignancy. Biopsy recommended.* 🩺 Breast Lump Classification: Benign vs Malignant

📘 Summary

This project explores the classification of breast tumors into benign (non-cancerous) and malignant (cancerous) types using machine learning models. The goal is to identify which algorithm best predicts tumor type based on various diagnostic features.

Importing the libraries

In [27]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, ConfusionMatrixDisplay,recall_score, f
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

READING IN THE DATASET

In [16]:

```
df = pd.read_csv(r'C:\Users\USER\Desktop\breasrcancer datazset.csv')
df
```

Out[16]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	c
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	
...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	

569 rows × 33 columns

In [17]:

```
df.info()
```

```
<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
```

SHAPE OF DATASET

In [18]:
```
df.shape
```

Out[18]:
```
(569, 33)
```

DISTRIBUTION OF TARGET COLUMN

In [19]:
```
df['diagnosis'].value_counts(normalize=True)
```

Out[19]:
```
diagnosis
B    0.627417
M    0.372583
Name: proportion, dtype: float64
```

DROPPING OF NULL COLUMN

In [20]:
```
df.drop(columns='Unnamed: 32', axis=1,inplace=True)
df.isna().sum()
```

Out[20]:
```
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
```

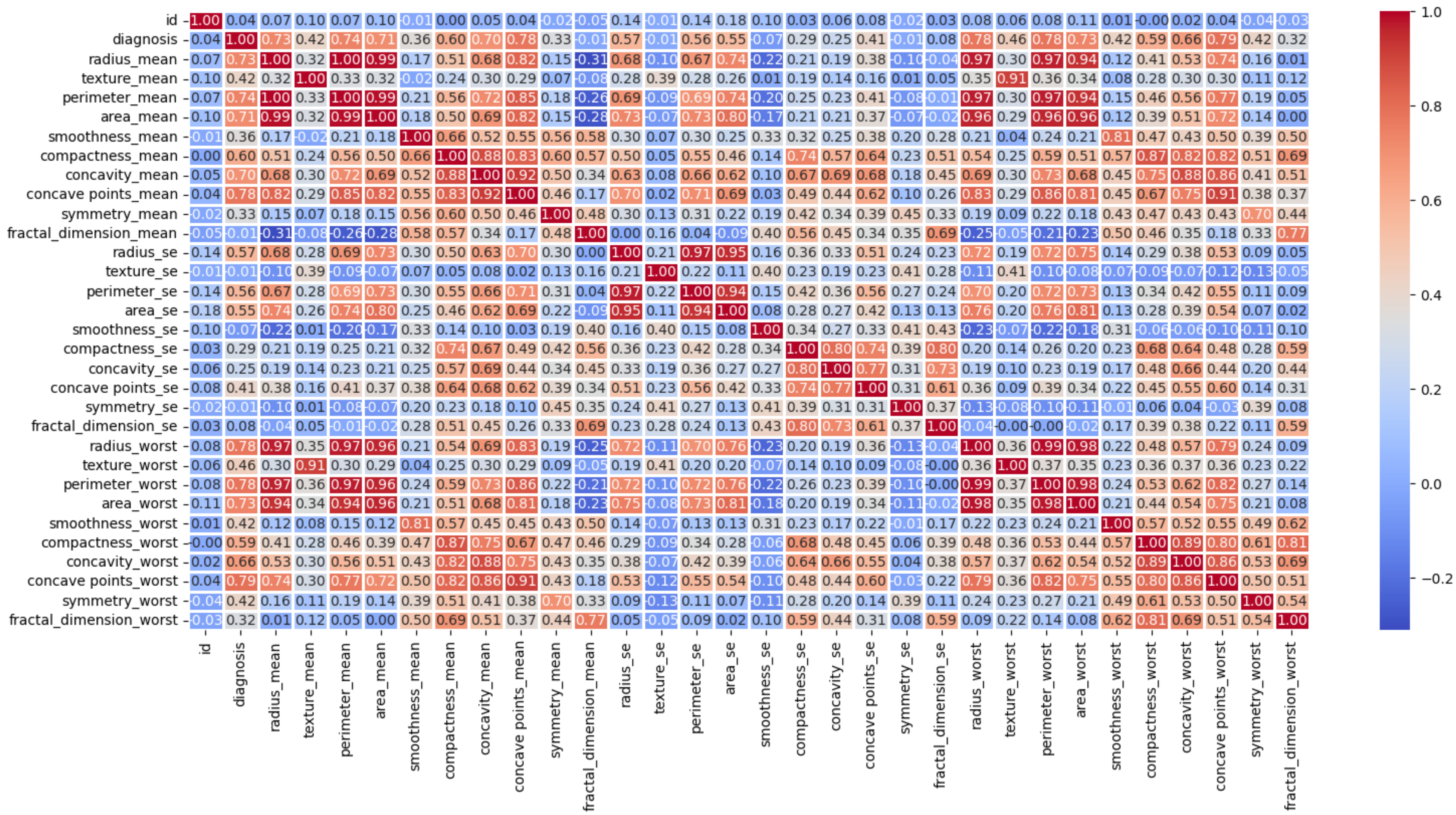
MAPPING TARGET COLUMN TO INT DATA TYPE

In [21]:
```
df['diagnosis'] = df['diagnosis'].map({'B':0, 'M':1})
```

CORRELATION MATRIX

In [22]:
```python
dcorr = df.corr()
plt.figure(figsize=(18, 8))
sns.heatmap(dcorr, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.8)
```

Out[22]: <Axes: >



SPLITTING THE DATASET

In [ ]:
```python
X = df.drop(columns=['id', 'diagnosis',], axis=1)  # predictor variables
y = df['diagnosis']   # target variables

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42) # splitting
```

```
print(X.shape)
print(y.shape)
X_train.head()
```

```
(569, 30)
(569,)
```

Out[ ]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	c point
78	20.18	23.97	143.70	1245.0	0.12860	0.34540	0.37540	
330	16.03	15.51	105.80	793.2	0.09491	0.13710	0.12040	
378	13.66	15.15	88.27	580.6	0.08268	0.07548	0.04249	
213	17.42	25.56	114.50	948.0	0.10060	0.11460	0.16820	
89	14.64	15.24	95.77	651.9	0.11320	0.13390	0.09966	

5 rows × 30 columns

DUE TO SOMEW OF THE MODEL ARE SENSITIVE I DECIDED TO SCALE THE DATASET USING STANDARD SCALER

In [24]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

IMPLEMENTED 4 DIFFERENT MODELS TO CHECK WHICH MODEL IS BEST

In [ ]:
```
lr = LogisticRegression(random_state=42)
knn = KNeighborsClassifier()
rf = RandomForestClassifier(random_state=42)
dt = DecisionTreeClassifier(random_state=42)

# classification models
classifier = [('K nearest neighbor ', knn),
              ('logistic Regression', lr),
              ('Random Forest', rf),
              ('Decision Tree', dt)]
```

```
for class_name, classes in classifier :

    # cross validation score
    scores = cross_val_score(classes, X_scaled, y_train, cv=5, scoring='accuracy')
    print("Cross-validated scores:", scores)
    print("Mean Accuracy: {:.2f}".format(scores.mean()))

    # fitting all the classes
    classes.fit(X_scaled, y_train)

   # predicting the test set
    y_pred_tree = classes.predict(X_test_scaled)

    print('{:s} : {:.2f}'.format(class_name,accuracy_score(y_test, y_pred_tree)))
```

```
Cross-validated scores: [0.9875     0.9875     0.975      0.98734177 0.91139241]
Mean Accuracy: 0.97
K nearest neighbor  : 0.96
Cross-validated scores: [0.9625     1.         0.9875     0.97468354 0.93670886]
Mean Accuracy: 0.97
logistic Regression : 0.97
Cross-validated scores: [0.975      0.9875     0.95       0.96202532 0.88607595]
Mean Accuracy: 0.95
Random Forest : 0.96
Cross-validated scores: [0.9625     0.9625     0.925      0.86075949 0.87341772]
Mean Accuracy: 0.92
Decision Tree : 0.90
```

KNEIGHBORS HAPPENS TO HAVE THE BEST ACCURACY AND CROSS VAL SCORE

In [39]:
```
acc_train = knn.score(X_scaled, y_train)  # training set accuracy score
acc_test = knn.score(X_test_scaled, y_test)  # testing set accuracy score

print(f"Knearest Neighbor - Train Accuracy: {acc_train:.2f}")
print(f"Knearest Neighbor - Test Accuracy: {acc_test:.2f}")

# predicting the test set
y_pred = knn.predict(X_test_scaled)


# cross validation score
scores = cross_val_score(classes, X_scaled, y_train, cv=10, scoring='accuracy')
```

```
print("Cross-validated scores:", scores)
print('-------------------------')

# Evaluation metrics
accuracy_tree = accuracy_score(y_test, y_pred)

roc_auc_tree = roc_auc_score(y_test, y_pred)

precision_tree = precision_score(y_test, y_pred)

recall_tree = recall_score(y_test, y_pred)

f1_tree = f1_score(y_test, y_pred)

print(f"Knearest Neighbor - Accuracy: {accuracy_tree:.2f}")
print('-------------------------')
print(f"Knearest Neighbor - ROC AUC: {roc_auc_tree:.2f}")
print('-------------------------')
print(f"Knearest Neighbor - Precision: {precision_tree:.2f}")
print('-------------------------')
print(f"Knearest Neighbor - Recall: {recall_tree:.2f}")
print('-------------------------')
print(f"Knearest Neighbor - F1 Score: {f1_tree:.2f}")
print('-------------------------')


print("\nClassification Report:")
print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_estimator(knn, X_test_scaled, y_test, cmap='Blues')
```

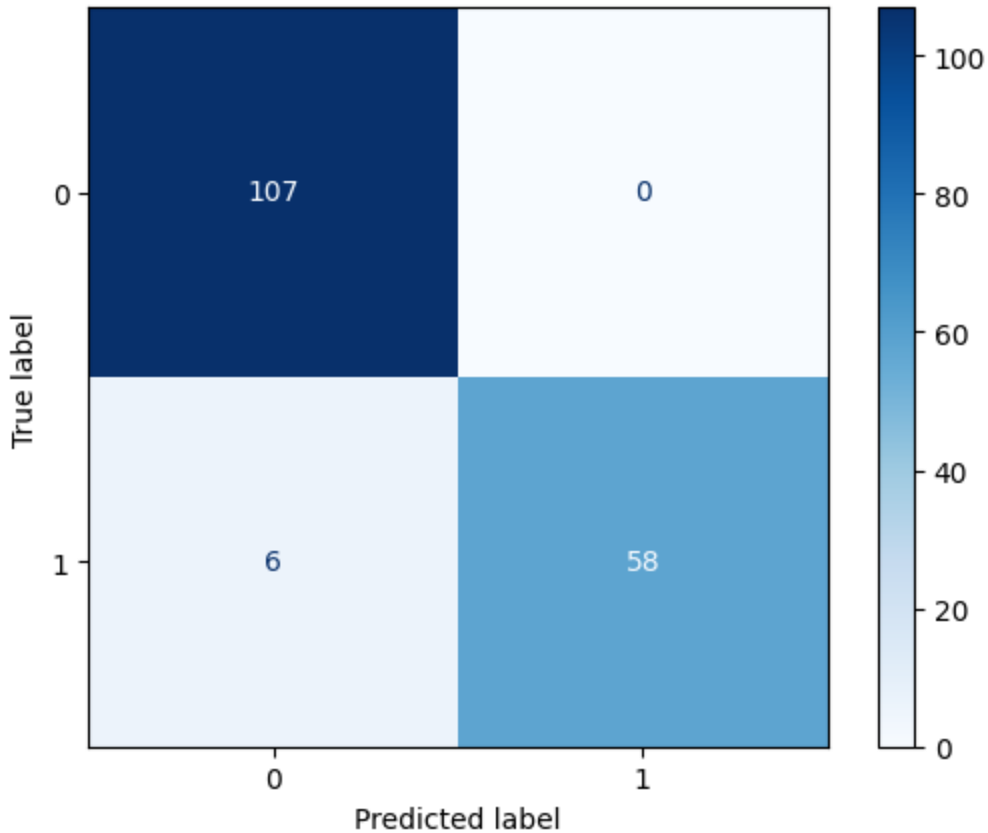
```
Knearest Neighbor - Train Accuracy: 0.97
Knearest Neighbor - Test Accuracy: 0.96
Cross-validated scores: [0.95       0.975      0.95       0.975      0.875      0.925
 0.9        0.925      0.84615385 0.87179487]
-------------------------
Knearest Neighbor - Accuracy: 0.96
-------------------------
Knearest Neighbor - ROC AUC: 0.95
-------------------------
Knearest Neighbor - Precision: 1.00
-------------------------
Knearest Neighbor - Recall: 0.91
-------------------------
Knearest Neighbor - F1 Score: 0.95
-------------------------

Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       107
           1       1.00      0.91      0.95        64

    accuracy                           0.96       171
   macro avg       0.97      0.95      0.96       171
weighted avg       0.97      0.96      0.96       171
```

Out[39]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x27795b91310>



SAVING THE MODEL

In [43]:
```
import pickle
with open('breast_model.pkl', 'wb') as file:
    pickle.dump(knn, file)
```