Breast Cancer Prediction

Breast Cancer Prediction is a classification task aimed at predicting the diagnosis of a breast mass as either malignant or benign. The dataset used for this prediction consists of features computed from a digitized image of a fine needle aspirate (FNA) of the breast mass. These features describe various characteristics of the cell nuclei present in the image.

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.

These features provide quantitative measurements that can be used to 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 the early detection and diagnosis of breast cancer.

In [1]:
```python
# Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [2]: # loading Dataset
        df = pd.read_csv("/content/Breast_Cancer_Dataset.csv")
```

EDA

```
In [3]: df.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mea
0	842302	M	17.99	10.38	122.80	1001.
1	842517	M	20.57	17.77	132.90	1326.
2	84300903	M	19.69	21.25	130.00	1203.
3	84348301	M	11.42	20.38	77.58	386.
4	84358402	M	20.29	14.34	135.10	1297.

5 rows × 33 columns

```
In [4]: df.shape
```

```
Out[4]: (569, 33)
```

```
In [5]: 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
```

In [6]:
```
# Duplicate values check
df.duplicated().sum()
```

Out[6]:
```
np.int64(0)
```

The Breast Cancer dataset consists of 569 observations and 33 columns with no duplicated rows, combining an identifier, a binary target variable, and multiple numerical features derived from cell nucleus measurements. As there are no missing values are present in the core features. But one column (`Unnamed: 32`) contains only null values and provides no information. `Unnamed: 32` should be dropped during preprocessing

Target Variable Analysis

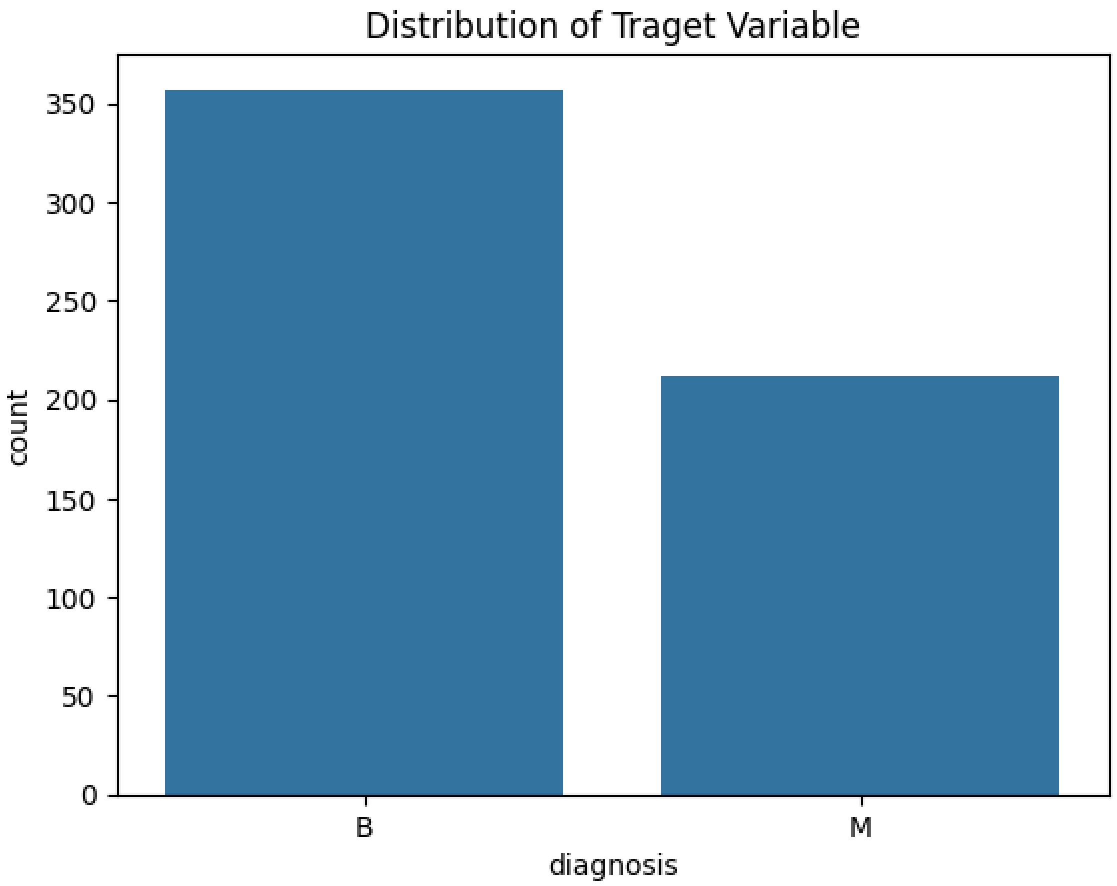
```
In [7]: # Class distribution (M vs B)
        df['diagnosis'].value_counts()
```

Out[7]:

diagnosis	count
B	357
M	212

dtype: int64

```
In [8]: # Visualizing Class Distribution
        sns.countplot(data=df, x='diagnosis', order=df['diagnosis'].value_counts().inc
        plt.title('Distribution of Traget Variable')
        plt.show()
```



The dataset contains a moderate class imbalance, with Benign (B) cases occuring more frequently than Malignant (M) cases.

Univariate Analysis

In [9]:
```
df.describe()
```

Out[9]:

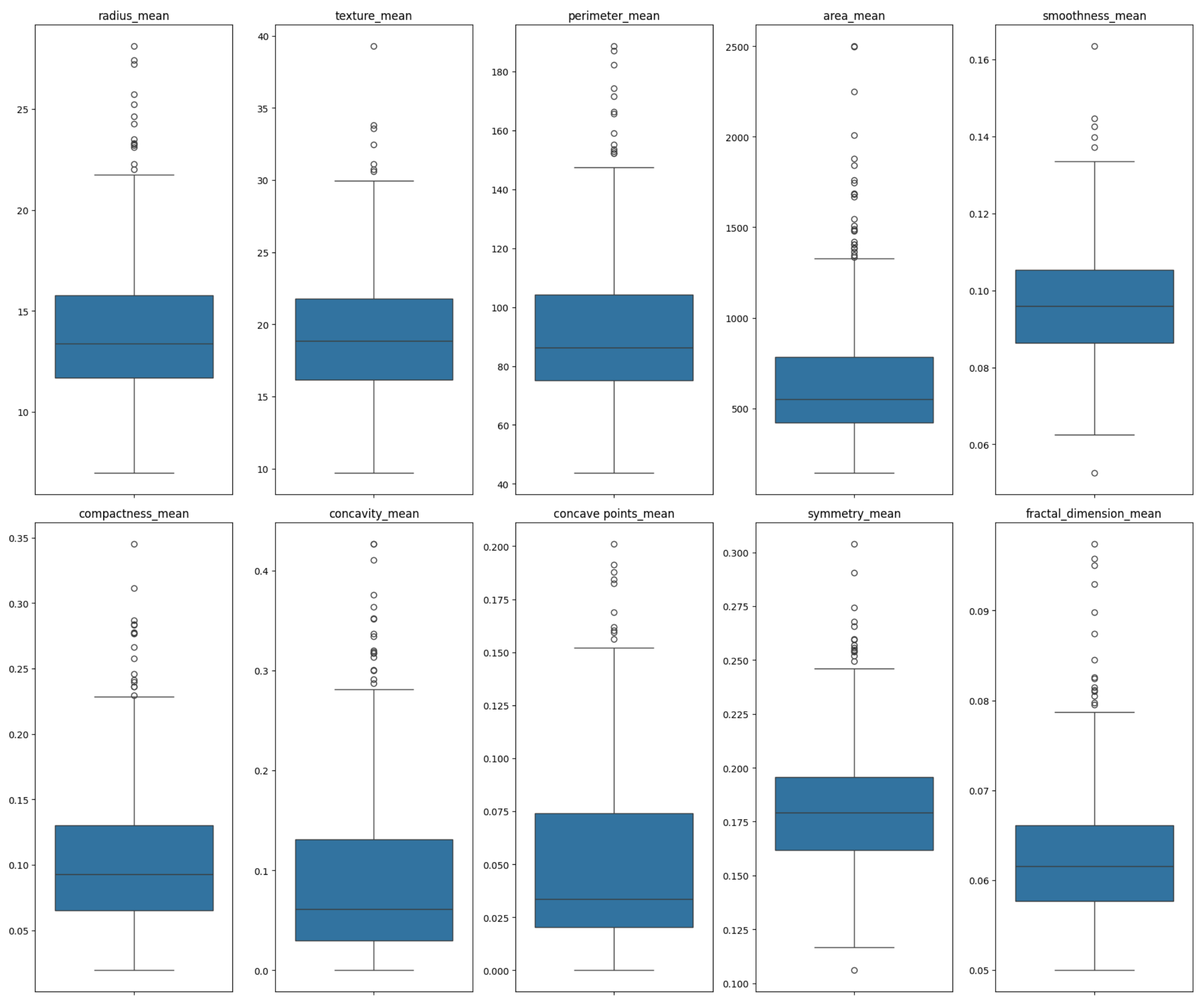
	id	radius_mean	texture_mean	perimeter_mean	area_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000
25%	8.692180e+05	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

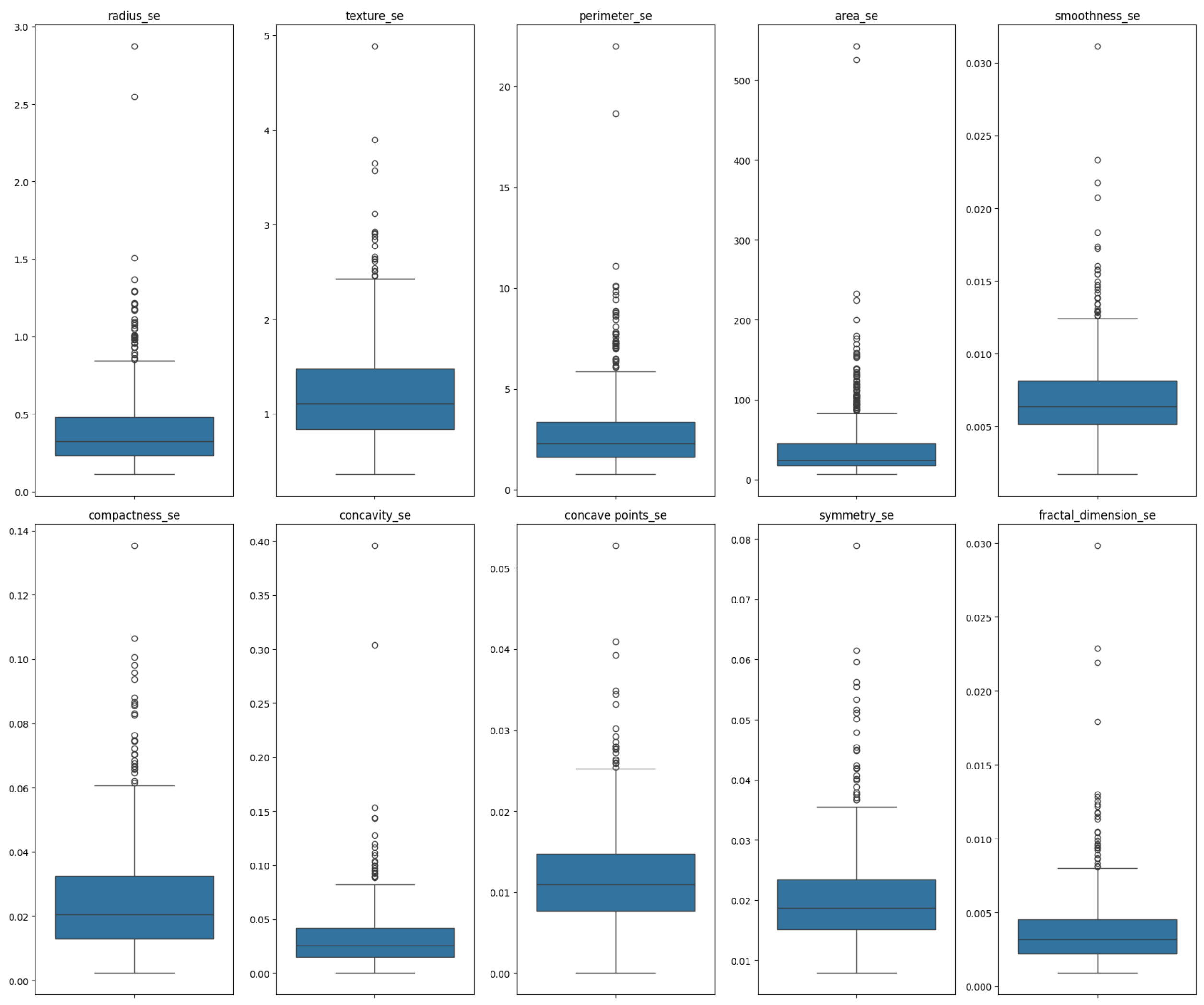
In [10]:
```
df.columns
```

Out[10]:
```
Index(['id', 'diagnosis', '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')
```

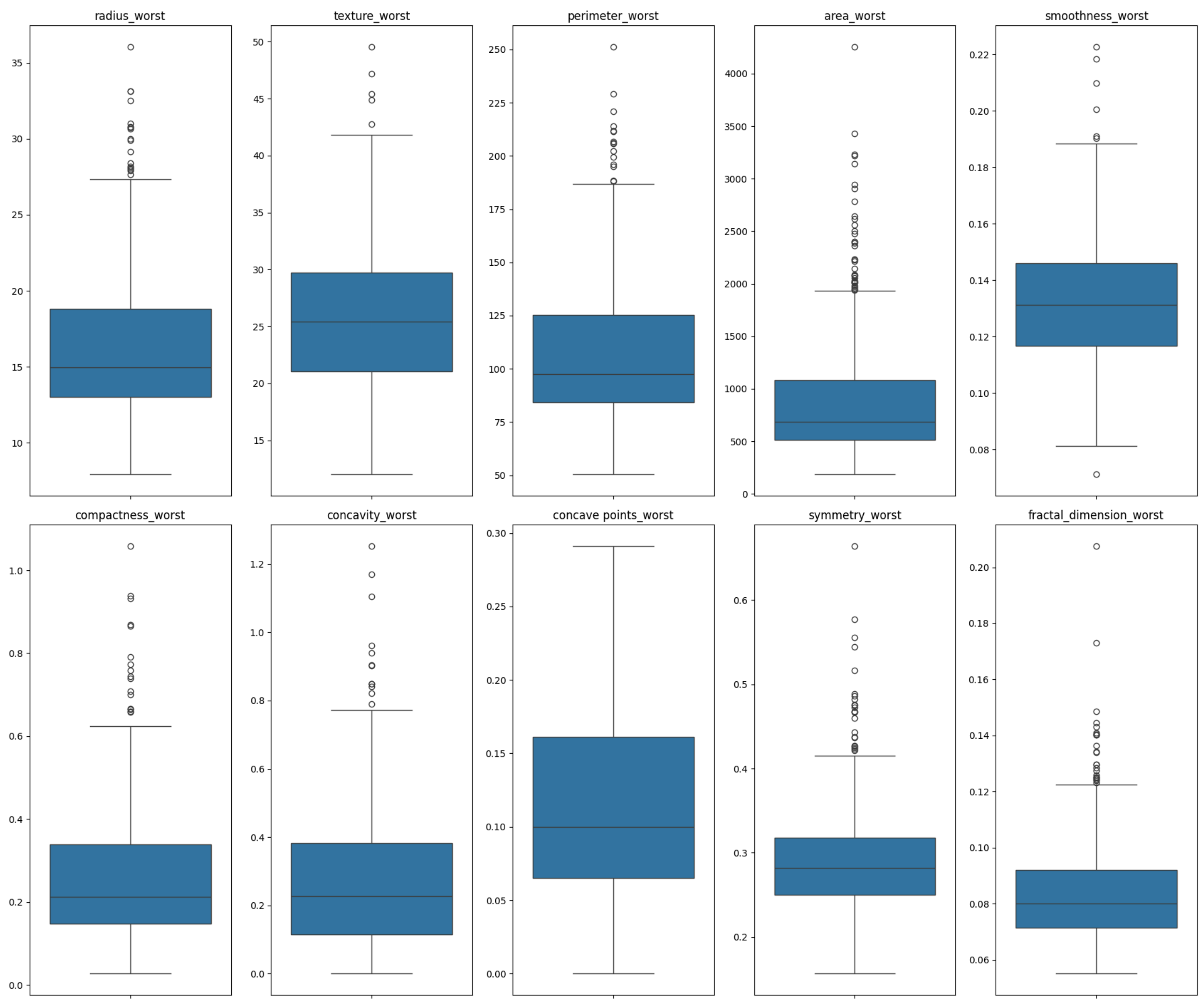
In [11]:
```
# Cheking Outliers for _mean columns
mean_cols = [c for c in df.columns if c.endswith('_mean')]
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(18,15))
axes = axes.flatten()

for ax, col in zip(axes, mean_cols):
  sns.boxplot(y=df[col], ax=ax)
  ax.set_title(col)
  ax.set_ylabel('')

plt.tight_layout()
plt.show()
```



In [12]:
```
# Cheking Outliers for _se columns
se_cols = [c for c in df.columns if c.endswith('_se')]
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(18,15))
axes = axes.flatten()

for ax, col in zip(axes, se_cols):
  sns.boxplot(y=df[col], ax=ax)
  ax.set_title(col)
  ax.set_ylabel('')

plt.tight_layout()
plt.show()
```



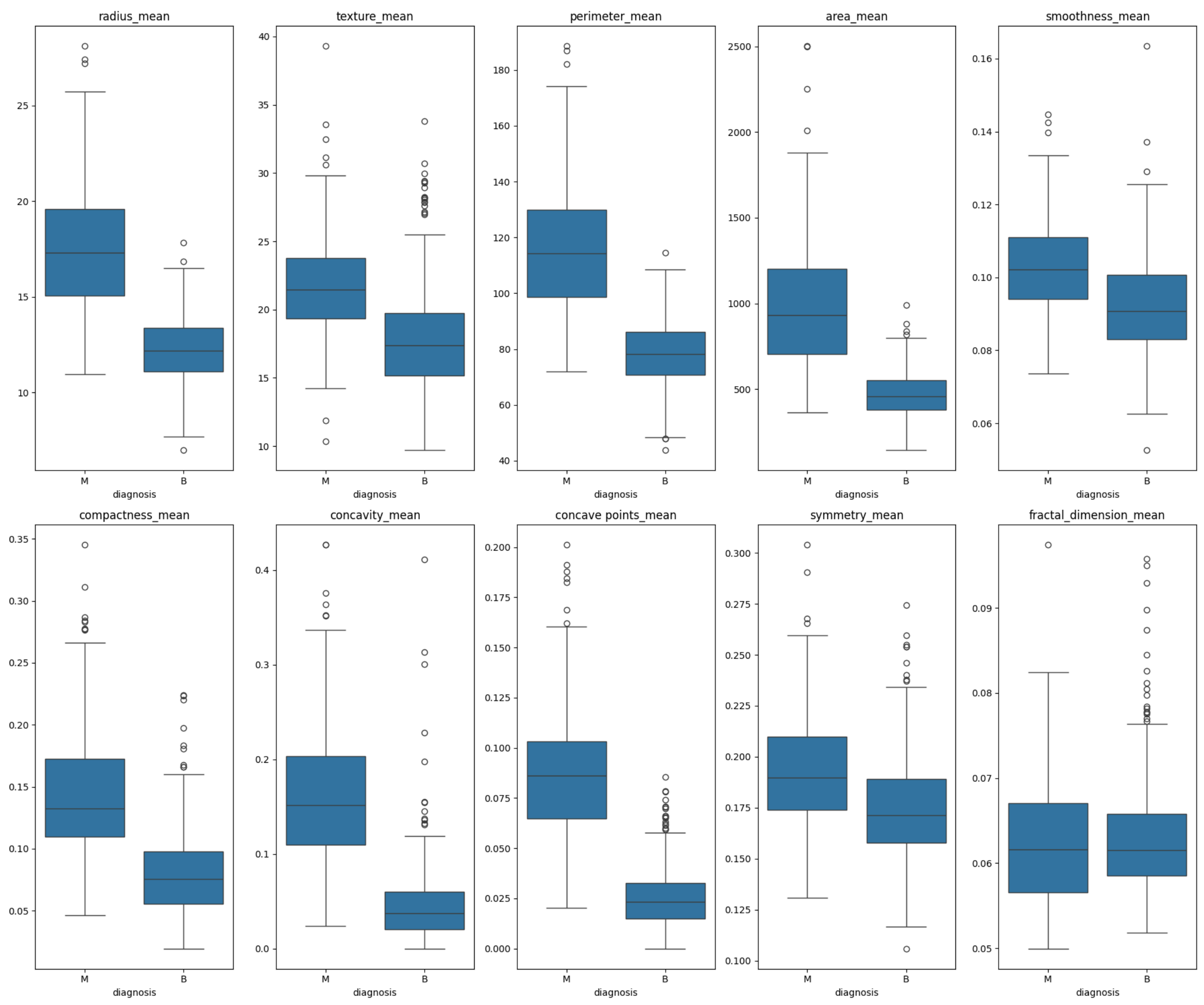
```
In [13]: # Cheking Outliers for _worst columns
worst_cols = [c for c in df.columns if c.endswith('_worst')]
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(18,15))
axes = axes.flatten()

for ax, col in zip(axes, worst_cols):
  sns.boxplot(y=df[col], ax=ax)
  ax.set_title(col)
  ax.set_ylabel('')

plt.tight_layout()
plt.show()
```



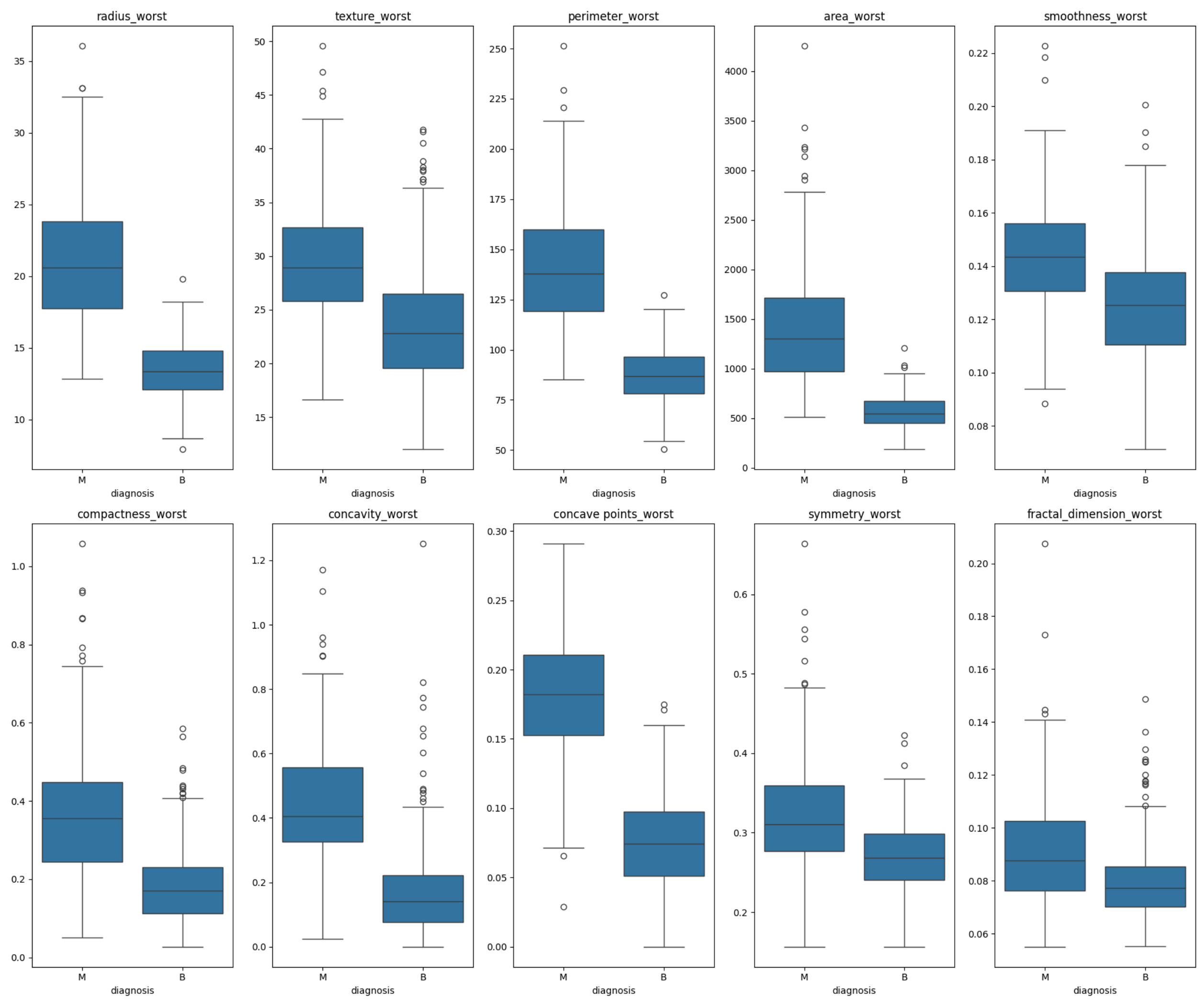
Boxplots reveal the presence of extreme values across several size and shape related features. These outliers reflect genuine biological variation rather than data quality issues. Consequently, no outlier removal was performed. Although feature scaling is required to ensure fair model traning.

Feature vs Target Analysis

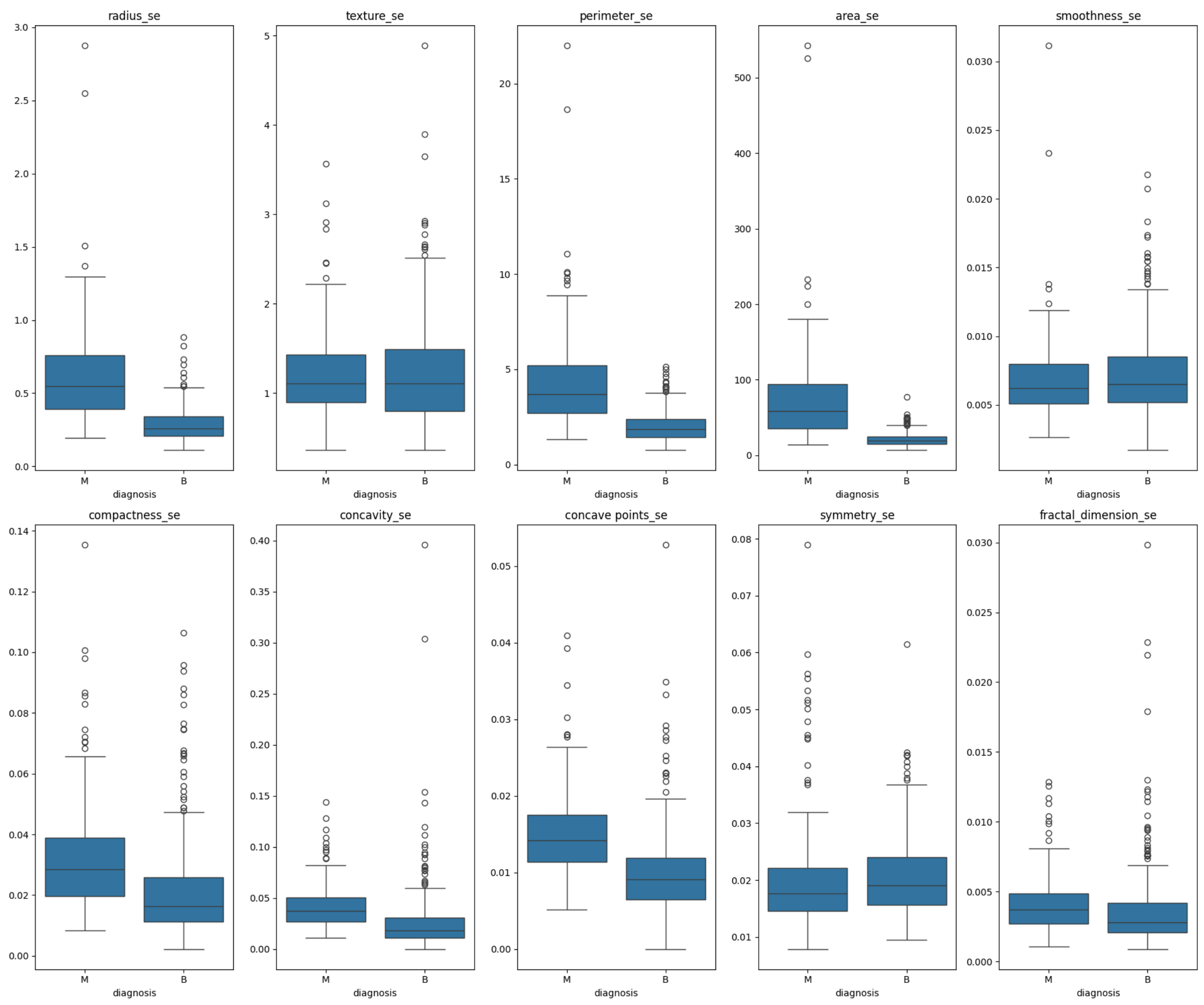
In [14]:
```
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(18,15))
axes = axes.flatten()

for ax, col in zip(axes, mean_cols):
  sns.boxplot(data=df, y=col, x='diagnosis', ax=ax)
  ax.set_title(col)
  ax.set_ylabel('')
  ax.set_ylabel('')

plt.tight_layout()
plt.show()
```



```
In [15]: fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(18,15))
         axes = axes.flatten()

         for ax, col in zip(axes, worst_cols):
           sns.boxplot(data=df, y=col, x='diagnosis', ax=ax)
           ax.set_title(col)
           ax.set_ylabel('')
           ax.set_ylabel('')

         plt.tight_layout()
         plt.show()
```



In [16]:
```
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(18,15))
axes = axes.flatten()

for ax, col in zip(axes, se_cols):
  sns.boxplot(data=df, y=col, x='diagnosis', ax=ax)
  ax.set_title(col)
  ax.set_ylabel('')
  ax.set_ylabel('')

plt.tight_layout()
plt.show()
```



- Mean features (`*_mean`) show clear separation between classes, with malignant tumors exhibiting consistently higher values for size- and shape-related attributes (e.g., radius, perimeter, area, concavity).
- Worst-case features (`*_worst`) provide the strongest discrimination, capturing extreme tumor characteristics that are more prevalent in malignant cases.
- Standard error features (`*_se`) display substantial overlap between classes, indicating weaker standalone discriminative power and a more supportive role.

Correlation Analysis

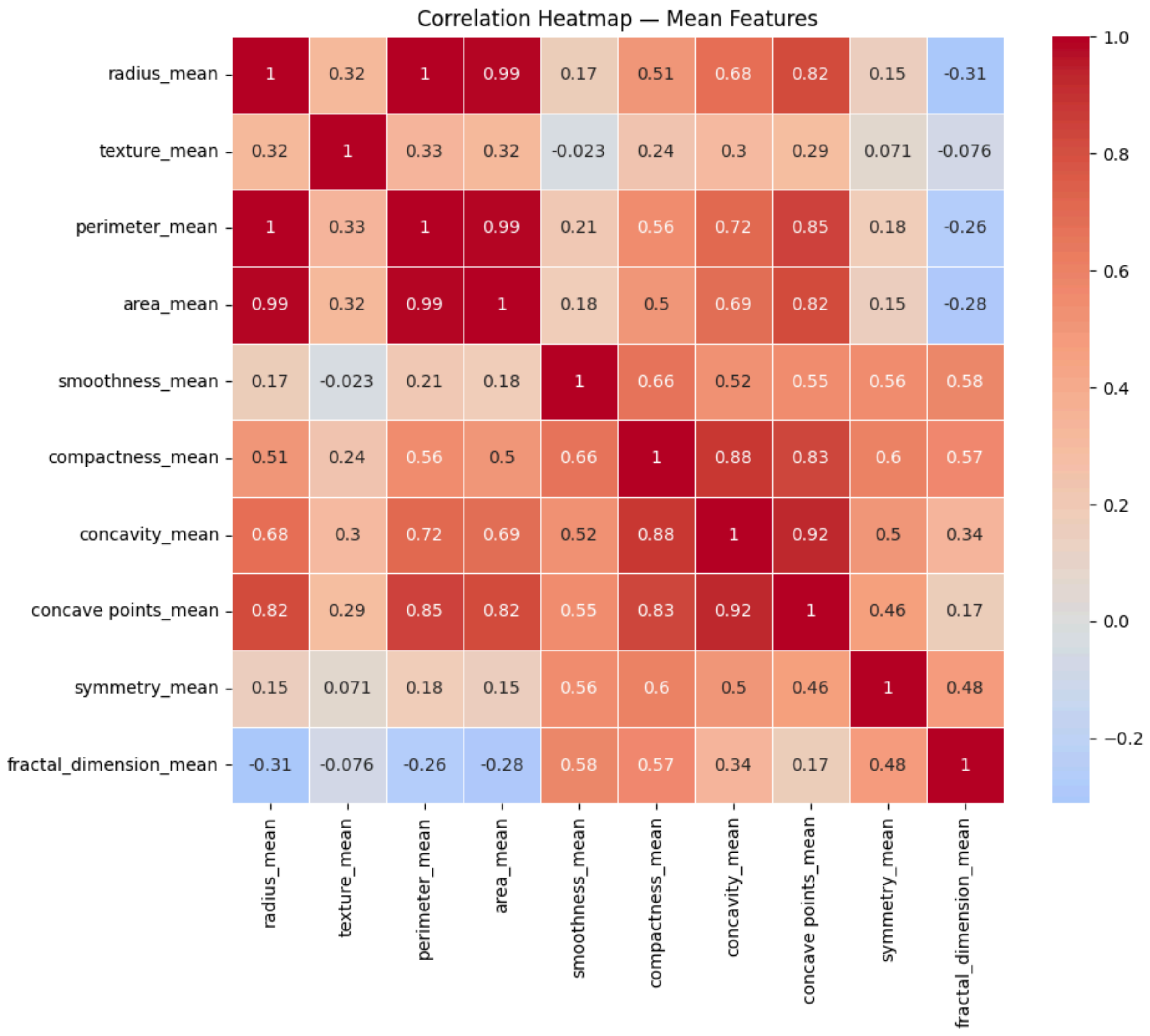
In [16]:

```

```

In [17]:

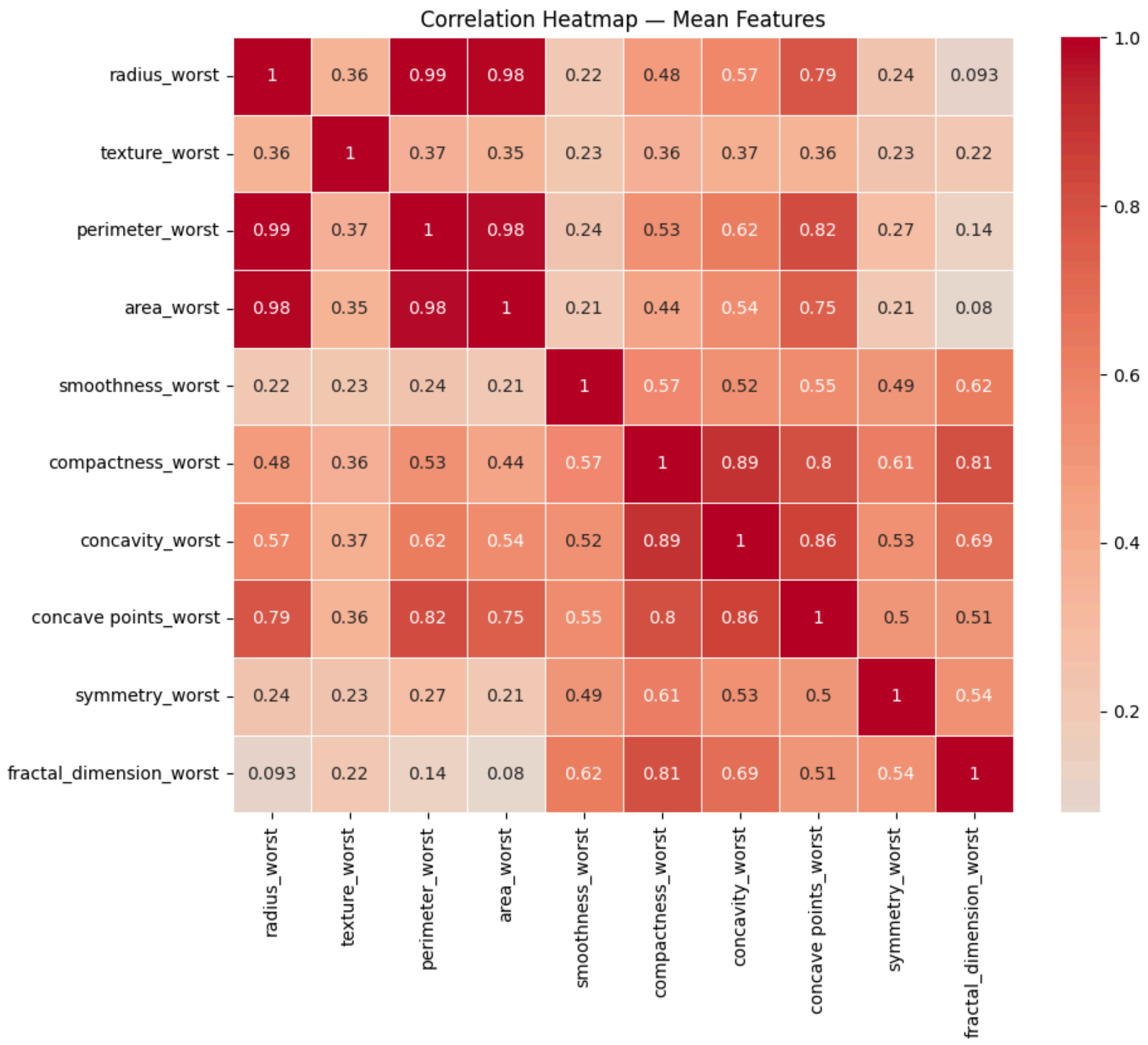
```
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(
    df[mean_cols].corr(),
    cmap='coolwarm',
    annot=True,
    center=0,
    linewidths=0.5
)
plt.title("Correlation Heatmap — Mean Features")
plt.show()
```



	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean
radius_mean	1	0.32	1	0.99	0.17	0.51	0.68	0.82	0.15	-0.31
texture_mean	0.32	1	0.33	0.32	-0.023	0.24	0.3	0.29	0.071	-0.076
perimeter_mean	1	0.33	1	0.99	0.21	0.56	0.72	0.85	0.18	-0.26
area_mean	0.99	0.32	0.99	1	0.18	0.5	0.69	0.82	0.15	-0.28
smoothness_mean	0.17	-0.023	0.21	0.18	1	0.66	0.52	0.55	0.56	0.58
compactness_mean	0.51	0.24	0.56	0.5	0.66	1	0.88	0.83	0.6	0.57
concavity_mean	0.68	0.3	0.72	0.69	0.52	0.88	1	0.92	0.5	0.34
concave points_mean	0.82	0.29	0.85	0.82	0.55	0.83	0.92	1	0.46	0.17
symmetry_mean	0.15	0.071	0.18	0.15	0.56	0.6	0.5	0.46	1	0.48
fractal_dimension_mean	-0.31	-0.076	-0.26	-0.28	0.58	0.57	0.34	0.17	0.48	1

In [18]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(
    df[worst_cols].corr(),
    cmap='coolwarm',
    annot=True,
    center=0,
    linewidths=0.5
)
plt.title("Correlation Heatmap — Mean Features")
plt.show()
```



Correlation analysis revealed strong multicollinearity among size-related features such as radius, perimeter, and area, as well as among shape irregularity measures including concavity and concave points. These correlations are expected due to the physical relationships between features and justify the use of feature scaling and regularization during modeling rather than aggressive feature removal.

Pre-processing and Modeling

I will be using following models for this current project

- Logistic Regression
- Support Vector Machine (SVM)
- Random Forest
- K-Nearest Neighbours (KNN)

In [19]:
```
# Importing model libraries
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, roc_auc_score, accuracy_sco
```

In [20]:
```
df.columns
```

Out[20]:
```
Index(['id', 'diagnosis', '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 [21]:
```
# Features and target
X = df.drop(columns=['id', 'diagnosis', 'Unnamed: 32'])
y = df['diagnosis'].map({'B':0, 'M':1})

# Train/Test split (80%/20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, strat

# Common preprocessing
preprocessor = Pipeline(steps=[
    ('scaler', StandardScaler())
])
```

Baseline Models

In [22]:
```
# Logistic Regression
baseline_log_reg = Pipeline(steps=[
    ('preprocessing', preprocessor),
    ('model', LogisticRegression(max_iter=1000, class_weight='balanced', rando
])

# Support Vector Machine (RBF)
baseline_svm = Pipeline(steps=[
    ('preprocessing', preprocessor),
    ('model', SVC(kernel='rbf', probability=True, class_weight='balanced', rar
])

# Random Forest Classifier - Random Forest does not require scaling, so prepro
baseline_rf = Pipeline(steps=[
    ('model', RandomForestClassifier(n_estimators=200, random_state=42, class_
])

# KNN
baseline_knn = Pipeline(steps=[
    ('preprocessing', preprocessor),
    ('model', KNeighborsClassifier(n_neighbors=5))
])

models = {
    'Logistic Regression': baseline_log_reg,
    'SVM (RBF)': baseline_svm,
    'Random Forest Classification': baseline_svm,
    'KNN': baseline_knn
}
```

In [23]:
```
# Model Evaluation
for name, pipeline in models.items():
  pipeline.fit(X_train, y_train)
  y_pred = pipeline.predict(X_test)
  y_proba = pipeline.predict_proba(X_test)[:, 1]

  print(f'\n{name}')
  print('ROC-AUC:', roc_auc_score(y_test, y_proba))
  print(classification_report(y_test, y_pred))
```

```
Logistic Regression
ROC-AUC: 0.9953703703703703
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        72
           1       0.98      0.95      0.96        42

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114


SVM (RBF)
ROC-AUC: 0.9953703703703705
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        72
           1       0.98      0.98      0.98        42

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114


Random Forest Classification
ROC-AUC: 0.9953703703703705
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        72
           1       0.98      0.98      0.98        42

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114


KNN
ROC-AUC: 0.982308201058201
              precision    recall  f1-score   support

           0       0.95      0.99      0.97        72
           1       0.97      0.90      0.94        42

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

Final Project Conclusion

Summary of Findings

- The dataset is **clean**, **well-structured**, and **information-rich**, with strong signal present in size- and shape-related features.
- Exploratory Data Analysis revealed:
    - Clear but overlapping separation between classes
    - Strong multicollinearity among related features
- No data quality issues requiring outlier removal
- Multiple baseline models were evaluated using a consistent preprocessing and evaluation framework.

Model Performance Overview

- **Logistic Regression** achieved strong performance with **high ROC–AUC** and **good malignant recall**, serving as a reliable and interpretable baseline.
- **Support Vector Machine (RBF)** and **Random Forest** slightly outperformed Logistic Regression, **achieving near-perfect ROC–AUC** and **higher malignant recall**.
- **KNN underperformed relative to other models**, particularly in malignant recall, and was therefore not considered a final candidate.

Across all strong models, **ROC–AUC scores** were already near saturation **(~0.995)**, and differences in accuracy and recall were marginal.

Final Model Consideration

Logistic Regression remains valuable for **interpretability** and **transparency**.

SVM (RBF) and **Random Forest** provide the best balance of **performance** and **robustness**.

Given the negligible performance differences, **model choice can be guided by deployment constraints (interpretability vs flexibility)** rather than raw metrics alone.

```
In [23]:
```