

```
# Aishah Abdullha Saeed Al Dhanhani
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
df = pd.read_csv('/content/drive/MyDrive/First project /data.csv')
```

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364

5 rows × 33 columns

```
df.isnull().sum()
```

	0
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

```
missing_percentage = df.isnull().mean() * 100
columns_with_missing_30 = missing_percentage[missing_percentage > 30]
print("Columns with more than 30% missing values:")
print(columns_with_missing_30)
```

Columns with more than 30% missing values:
Unnamed: 32 100.0
dtype: float64

```
missing_percentage = df.isnull().mean() * 100

threshold = 30

# Filter columns with more than the threshold percentage of missing values
columns_to_drop = missing_percentage[missing_percentage > threshold].index

# Drop the columns
df = df.drop(columns=columns_to_drop)

# Display the resulting dataframe after removing columns with too many missing values
print(f"Columns dropped: {list(columns_to_drop)}")
```

```
print(f"Remaining columns: {list(df.columns)}")
```

Columns dropped: ['Unnamed: 32']
Remaining columns: ['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']

Start coding or [generate](#) with AI.

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
for column in df.select_dtypes(include=['object']).columns:
```

```
    df[column] = label_encoder.fit_transform(df[column])
df.head()
```

	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
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625

5 rows × 32 columns

```
# Fill missing values in numeric columns with mean
for column in df.select_dtypes(include=['float64']).columns:
    mean_value = df[column].mean()
    df[column] = df[column].fillna(mean_value)
```

```
# Fill missing values in categorical columns with mode
for column in df.select_dtypes(include=['object']).columns:
    mode_value = df[column].mode()[0] # Get the mode (most frequent value)
    df[column] = df[column].fillna(mode_value)
```

```
df.isnull().sum()
```

	0
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

```
import numpy as np # Ensure numpy is imported for np.number
data=df.copy()
def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Return a boolean series indicating if the value is an outlier
    return (column < lower_bound) | (column > upper_bound)

# Identify columns to exclude from outlier detection (e.g., ID, target variable)
columns_to_exclude = ['id', 'diagnosis']

# Iterate only over numeric columns that are not in the exclude list
for col in data.select_dtypes(include=np.number).columns:
    if col not in columns_to_exclude:
        data[f'{col}_Outlier'] = detect_outliers_iqr(data[col])
outliers_only = data[data.filter(like='_Outlier').any(axis=1)]
outliers_only
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst_Outlier	texture_worst_Outlier	perimeter_worst_Outlier	area_worst_Outlier	smoothness_worst_Outlier	compactness_worst_Outlier
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	False	False	False	True	False	
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	False	False	False	True	False	
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	False	False	False	False	False	
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	False	False	False	False	True	
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
563	926125	1	20.92	25.09	143.00	1347.0	0.10990	0.22360	0.3174	0.14740	...	False	False	False	False	False	
564	926424	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.2439	0.13890	...	False	False	False	True	False	
565	926682	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.1440	0.09791	...	False	False	False	False	False	
567	927241	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.3514	0.15200	...	False	False	False	False	False	
568	92751	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.0000	0.00000	...	False	False	False	False	False	

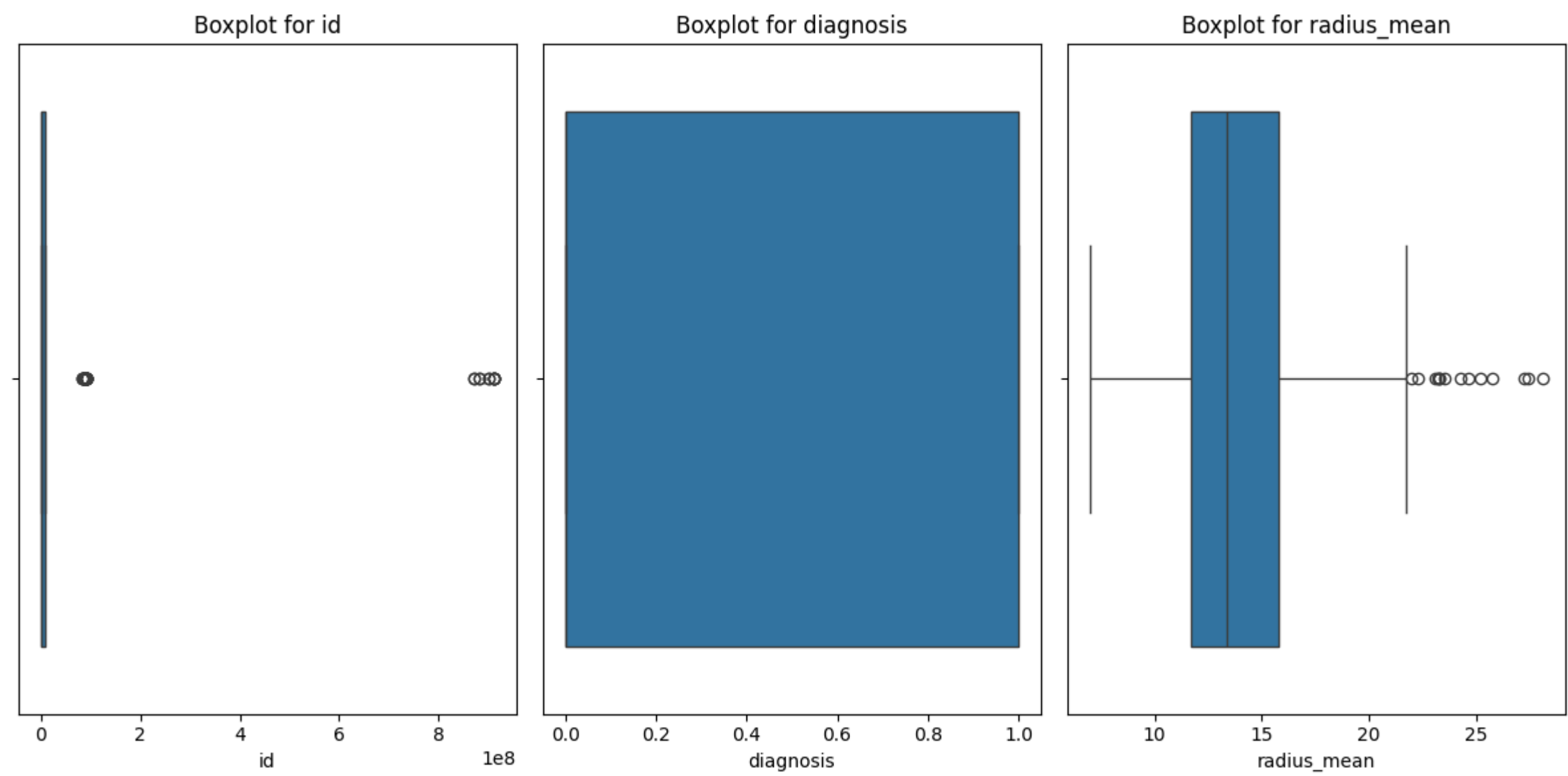
171 rows × 62 columns

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 6))

for i, col in enumerate(df.columns[:3]): # Only plotting the first 3
    plt.subplot(1, 3, i+1) # Create subplots for each feature
    sns.boxplot(x=df[col])
    plt.title(f"Boxplot for {col}")

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



```
import numpy as np # Ensure numpy is imported for np.number
data=df.copy()
def detect_outliers_zscore(column, threshold=3):
    mean = np.mean(column)
    std_dev = np.std(column)
    z_scores = (column - mean) / std_dev
    return np.abs(z_scores) > threshold

# Identify columns to exclude from outlier detection (e.g., ID, target variable)
```

```
columns_to_exclude = ['id', 'diagnosis']

# Iterate only over numeric columns that are not in the exclude list
for col in data.select_dtypes(include=np.number).columns:
    if col not in columns_to_exclude:
        data[f'{col}_Outlier'] = detect_outliers_zscore(data[col])
outliers_only = data[data.filter(like='_Outlier').any(axis=1)]
outliers_only
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst_Outlier	texture_worst_Outlier	perimeter_worst_Outlier	area_worst_Outlier	smoothness_worst_Outlier	compactness_worst_Outlier
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	False	False	False	False	False	False
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	False	False	False	False	False	True
9	84501001	1	12.46	24.04	83.97	475.9	0.11860	0.23960	0.2273	0.08543	...	False	False	False	False	False	False
12	846226	1	19.17	24.80	132.40	1123.0	0.09740	0.24580	0.2065	0.11180	...	False	False	False	False	False	False
14	84667401	1	13.73	22.61	93.60	578.3	0.11310	0.22930	0.2128	0.08025	...	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
559	925291	0	11.51	23.93	74.52	403.5	0.09261	0.10210	0.1112	0.04105	...	False	False	False	False	False	False
561	925311	0	11.20	29.37	70.67	386.0	0.07449	0.03558	0.0000	0.00000	...	False	False	False	False	False	False
562	925622	1	15.22	30.62	103.40	716.9	0.10480	0.20870	0.2550	0.09429	...	False	False	False	False	False	False
567	927241	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.3514	0.15200	...	False	False	False	False	False	False
568	92751	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.0000	0.00000	...	False	False	False	False	False	False

74 rows × 62 columns

```
import numpy as np
import pandas as pd # Ensure pandas is imported for select_dtypes

data=df.copy()

def replace_outliers_with_nan(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Replace outliers with NaN
    return column.where((column >= lower_bound) & (column <= upper_bound), np.nan)

# Identify columns to exclude from outlier detection/replacement
# 'id' and 'diagnosis' are typically not for outlier treatment
columns_to_exclude = ['id', 'diagnosis']

# Replace outliers with NaN for each applicable numeric feature
for col in data.select_dtypes(include=np.number).columns:
    if col not in columns_to_exclude:
        data[col] = replace_outliers_with_nan(data[col])

null_counts = data.isnull().sum()
print("Sum of null values in each column:")
print(null_counts)
```

```
Sum of null values in each column:
id                0
diagnosis         0
radius_mean       14
texture_mean       7
perimeter_mean    13
area_mean         25
smoothness_mean   6
compactness_mean  16
concavity_mean    18
concave points_mean 10
symmetry_mean     15
fractal_dimension_mean 15
radius_se         38
texture_se        20
perimeter_se      38
area_se          65
smoothness_se     30
compactness_se    28
concavity_se      22
concave points_se  19
```

```
symmetry_se      27
fractal_dimension_se  28
radius_worst     17
texture_worst     5
perimeter_worst  15
area_worst       35
smoothness_worst  7
compactness_worst 16
concavity_worst  12
concave_points_worst 0
symmetry_worst   23
fractal_dimension_worst 24
dtype: int64
```

```
from sklearn.impute import KNNImputer

# Separate numeric and non-numeric columns
numeric_cols = data.select_dtypes(include=np.number).columns.tolist()
categorical_cols = data.select_dtypes(exclude=np.number).columns.tolist()

# Exclude 'id' and 'diagnosis' from imputation if they are in numeric_cols (shouldn't be after previous steps, but good practice)
# and ensure they are handled as non-imputed columns
columns_to_impute = [col for col in numeric_cols if col not in ['id', 'diagnosis']]

# Prepare the data for imputation (only numeric columns that might have NaNs)
data_to_impute = data[columns_to_impute]

imputer = KNNImputer(n_neighbors=2)
data_imputed_values = imputer.fit_transform(data_to_impute)

# Create a DataFrame from the imputed values
data_imputed_df = pd.DataFrame(data_imputed_values, columns=columns_to_impute, index=data.index)

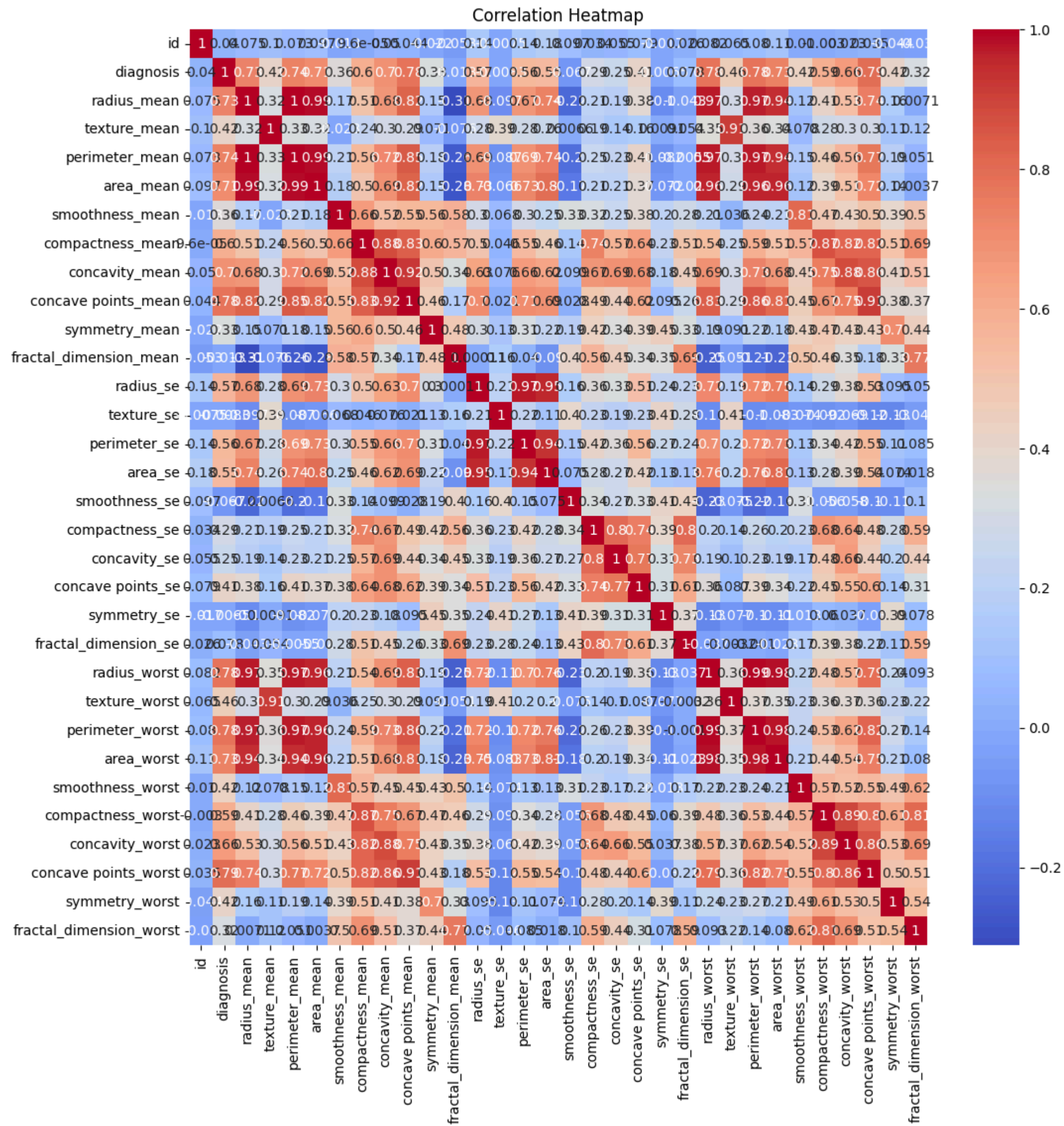
# Re-combine with the non-imputed columns
data_final = data[categorical_cols + ['id', 'diagnosis']].copy() # Include original non-imputed columns
for col in columns_to_impute:
    data_final[col] = data_imputed_df[col]

# Verify no more null values in the imputed columns
print(data_final.isnull().sum())

data = data_final # Update data to the fully processed DataFrame
```

```
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
```

```
plt.figure(figsize=(12, 12))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



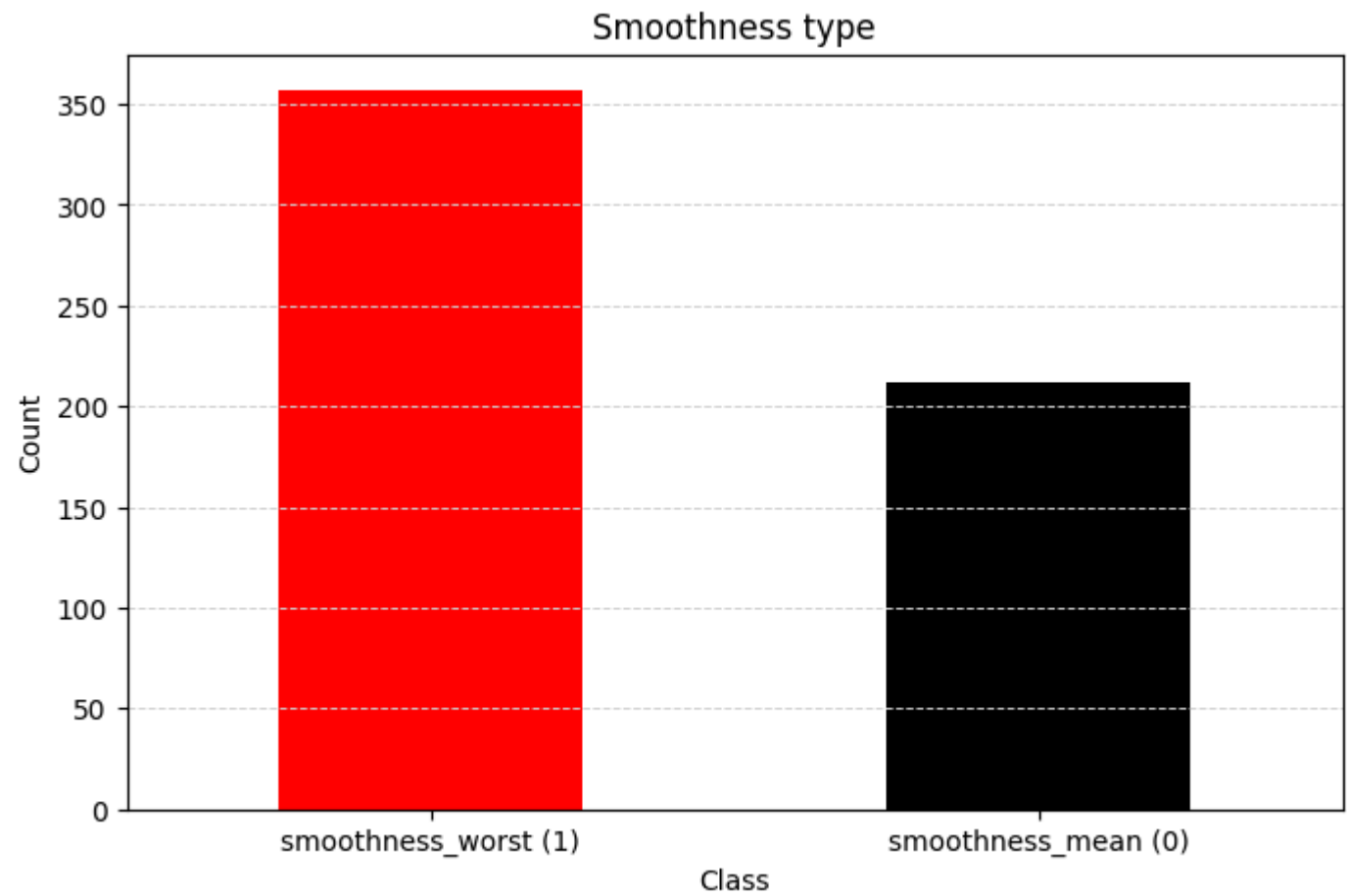
```
df['diagnosis'].value_counts()
```

count	
diagnosis	
0	357
1	212

dtype: int64

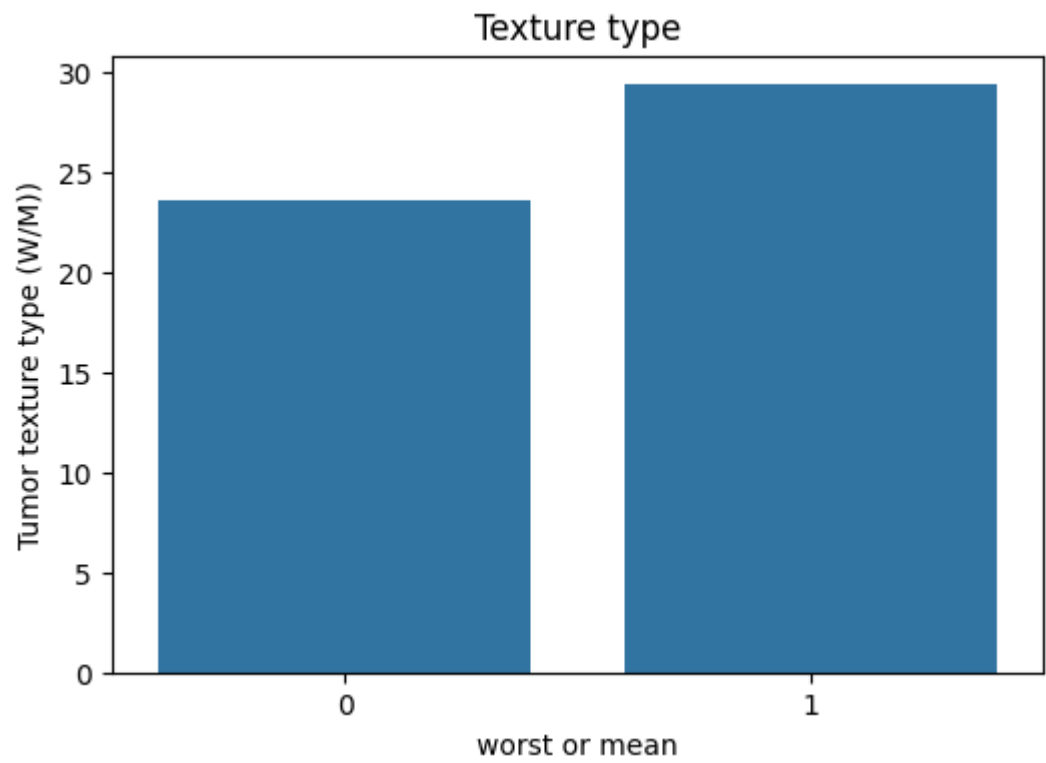
```
plt.figure(figsize=(8, 5))
df['diagnosis'].value_counts().plot(kind='bar', color=['red', 'black'])
plt.xticks([0, 1], ['smoothness_worst (1)', 'smoothness_mean (0)'], rotation=0)
plt.title('Smoothness type')
plt.xlabel('Class')
plt.ylabel('Count')
plt.grid(axis='y', color='lightgray', linestyle='--', linewidth=0.7)
```

plt.show()



```
import seaborn as sns

plt.figure(figsize=(6, 4))
sns.barplot(x='diagnosis', y='texture_worst', data=df, estimator='mean', ci=None)
plt.title("Texture type")
plt.xlabel("worst or mean")
plt.ylabel("Tumor texture type (W/M)")
plt.show()
```



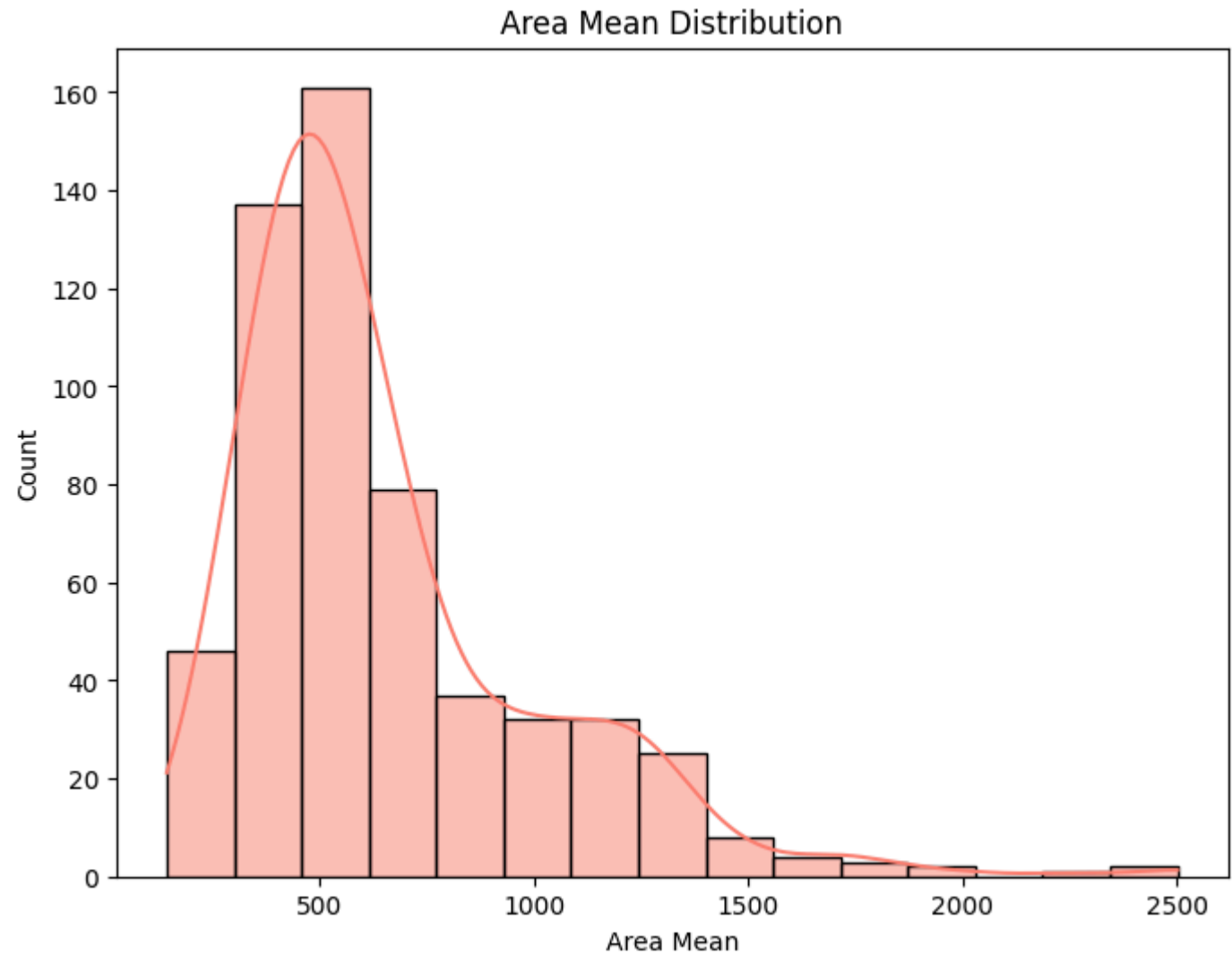
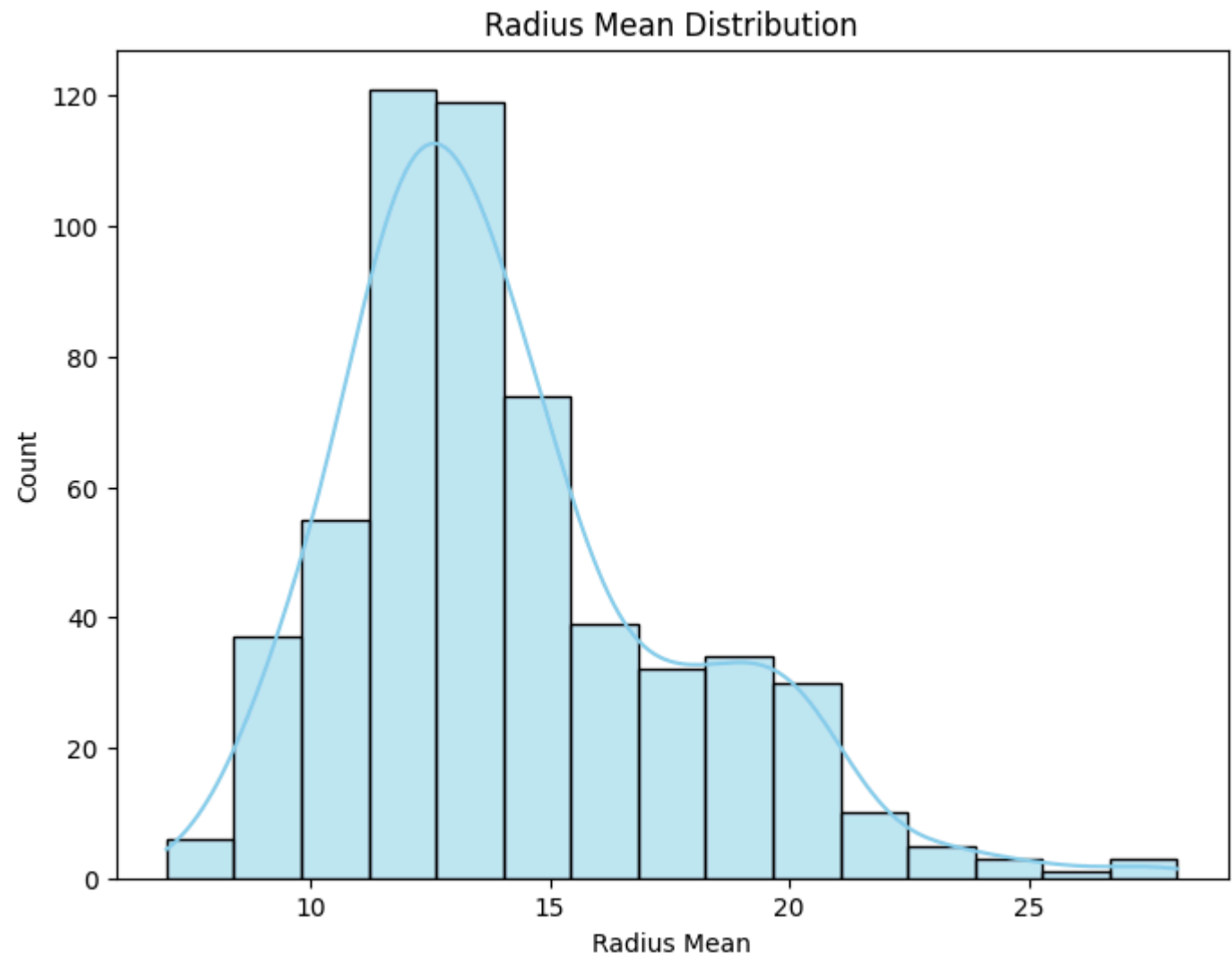
```
cat_cols = [col for col in df.columns if df[col].dtype == 'object']
num_cols = [col for col in df.columns if df[col].dtype != 'object']
```

```
for col in cat_cols:
    print(f"{col} has {df[col].unique()}")
```

```
plt.figure(figsize=(8, 6))
sns.histplot(df['radius_mean'], bins=15, kde=True, color='skyblue')
plt.title('Radius Mean Distribution')
plt.xlabel('Radius Mean')
plt.ylabel('Count')
plt.show()
```

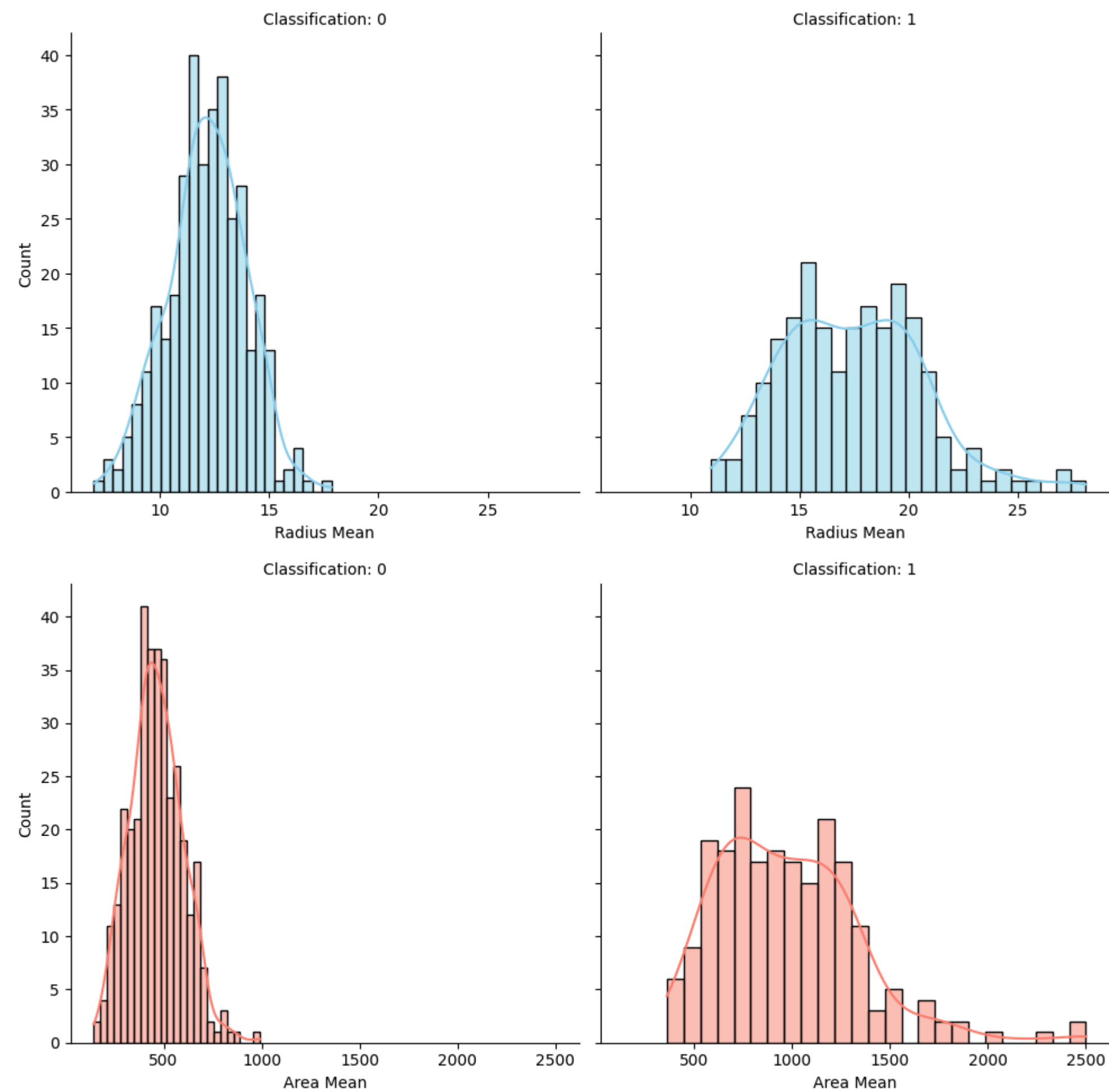
```
plt.figure(figsize=(8, 6))
sns.histplot(df['area_mean'], bins=15, kde=True, color='salmon')
plt.title('Area Mean Distribution')
plt.xlabel('Area Mean')
```

```
plt.ylabel('Count')
plt.show()
```



```
# لكل فئة من radius_mean رسم توزيع classification
g = sns.FacetGrid(df, col='diagnosis', height=5, aspect=1)
g.map(sns.histplot, 'radius_mean', bins=25, kde=True, color='skyblue')
g.set_axis_labels("Radius Mean", "Count")
g.set_titles("Classification: {col_name}")
plt.show()

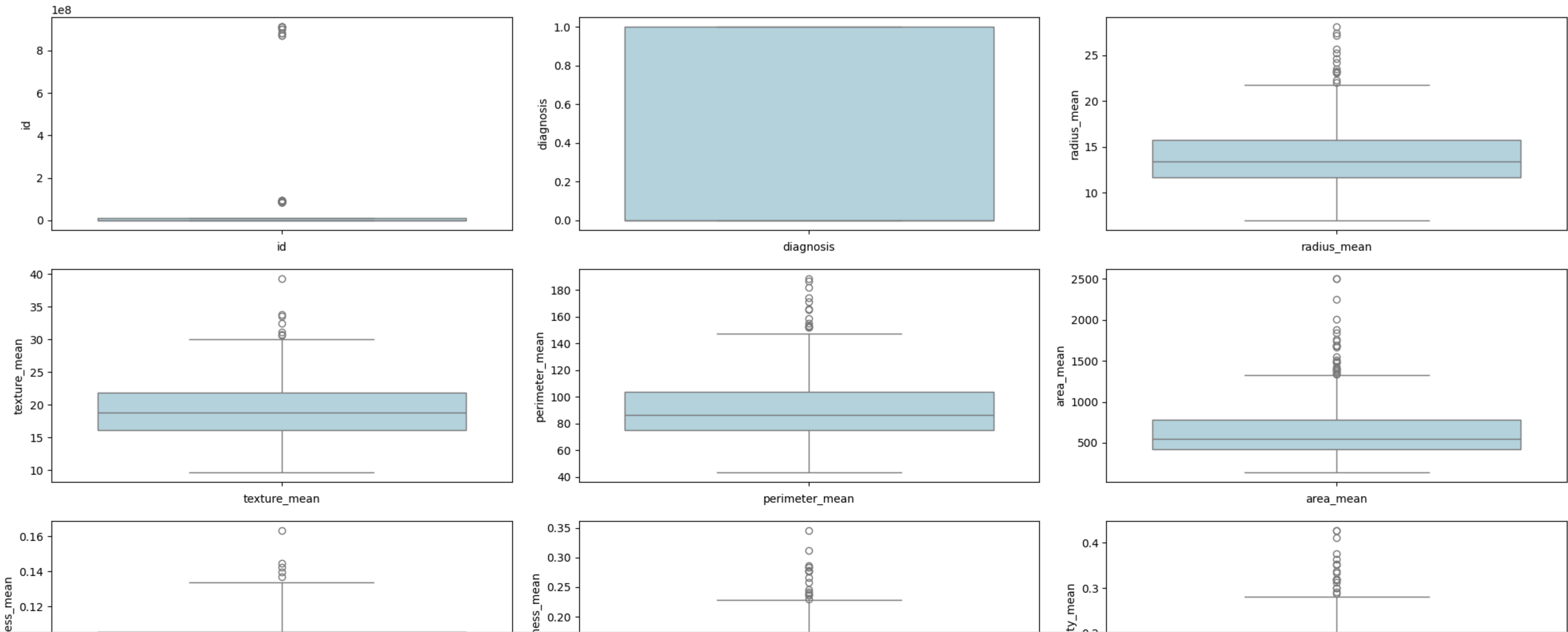
# لكل فئة من area_mean رسم توزيع classification
g = sns.FacetGrid(df, col='diagnosis', height=5, aspect=1)
g.map(sns.histplot, 'area_mean', bins=25, kde=True, color='salmon')
g.set_axis_labels("Area Mean", "Count")
g.set_titles("Classification: {col_name}")
plt.show()
```



```
plt.figure(figsize=(20, 35))
plotnumber = 1
num_cols = [col for col in df.columns if df[col].dtype != 'object']

for column in num_cols:
    ax = plt.subplot(11, 3, plotnumber) # Changed 10 to 11 to accommodate more plots
    sns.boxplot(y=df[column], color='lightblue')
    plt.xlabel(column)
    plotnumber += 1

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

```
colors = ['lightgreen', 'lightcoral']
labels = ['Area_Mean (0)', 'Area_worst (1)']

order = df['diagnosis'].value_counts().index

plt.figure(figsize=(16, 8))
plt.suptitle('diagnosis Distribution', fontweight='bold', fontsize=16)

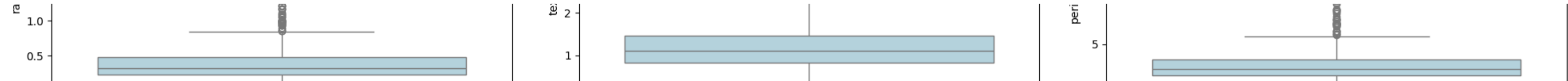
plt.subplot(1, 2, 1)
plt.title('Pie Chart', fontsize=14)
plt.pie(df['diagnosis'].value_counts(), labels=labels, colors=colors,
        autopct='%0.2f%%', startangle=90)
centre = plt.Circle((0, 0), 0.45, color='white')
plt.gca().add_artist(centre)

plt.subplot(1, 2, 2)
plt.title('Bar Chart', fontsize=14)
ax = sns.countplot(x='diagnosis', data=df, palette=colors, order=order)

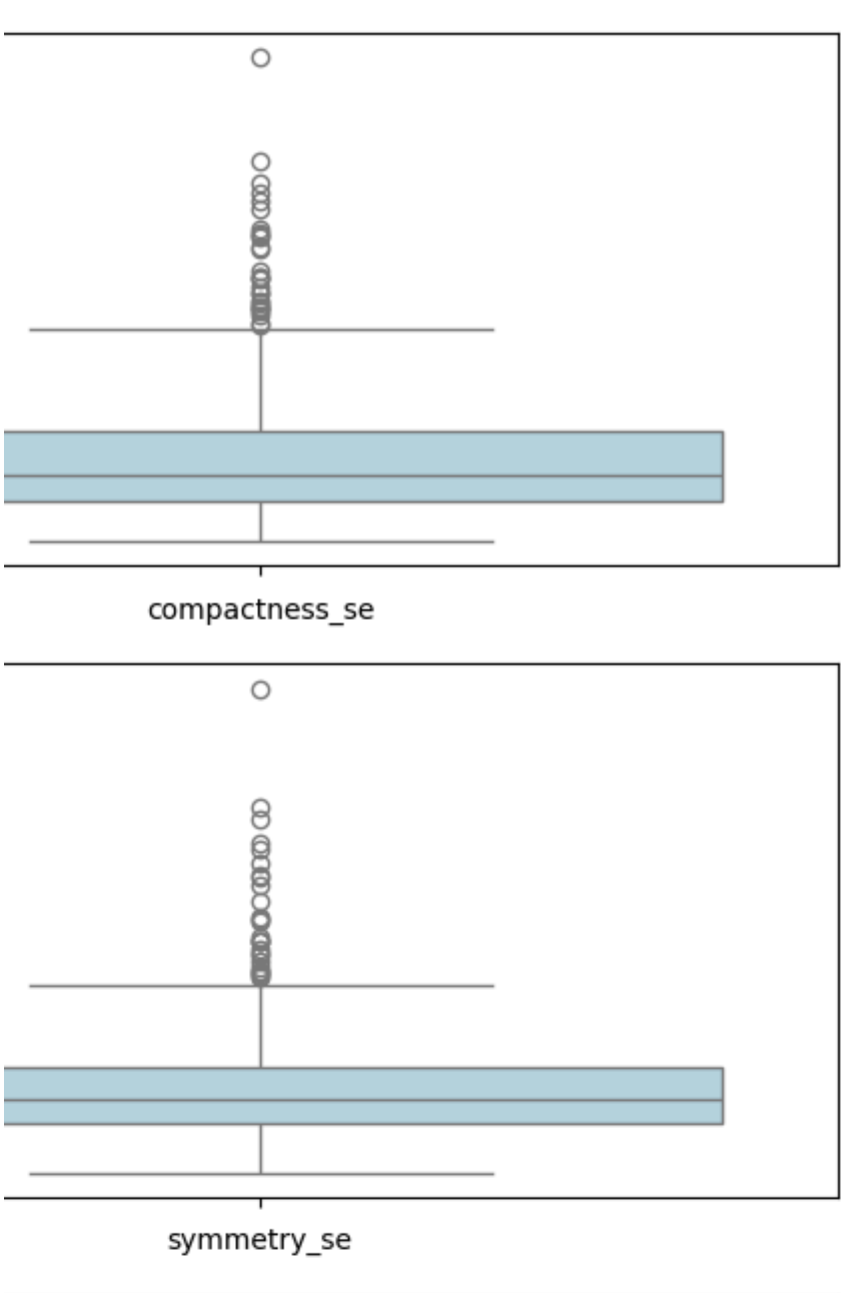
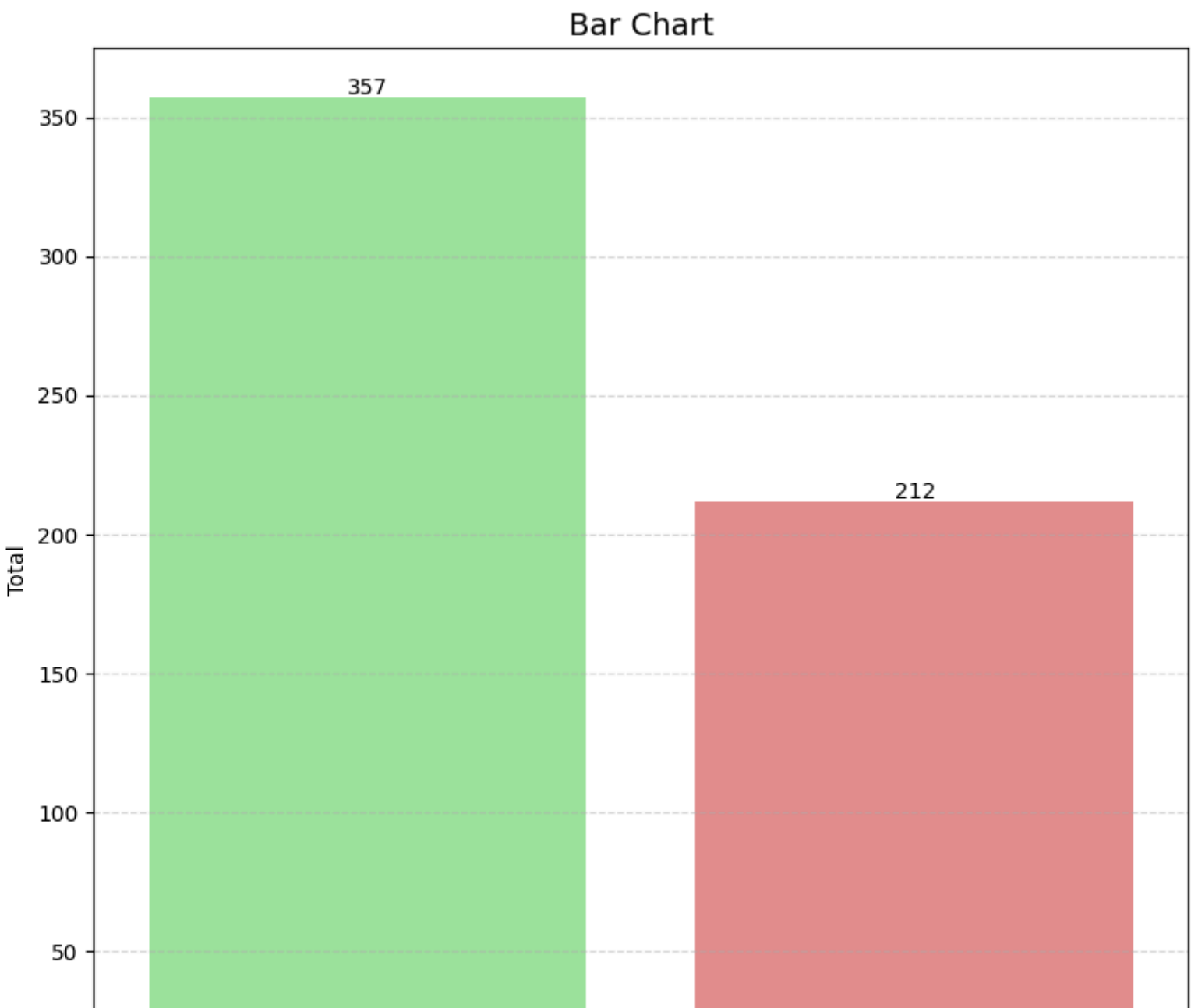
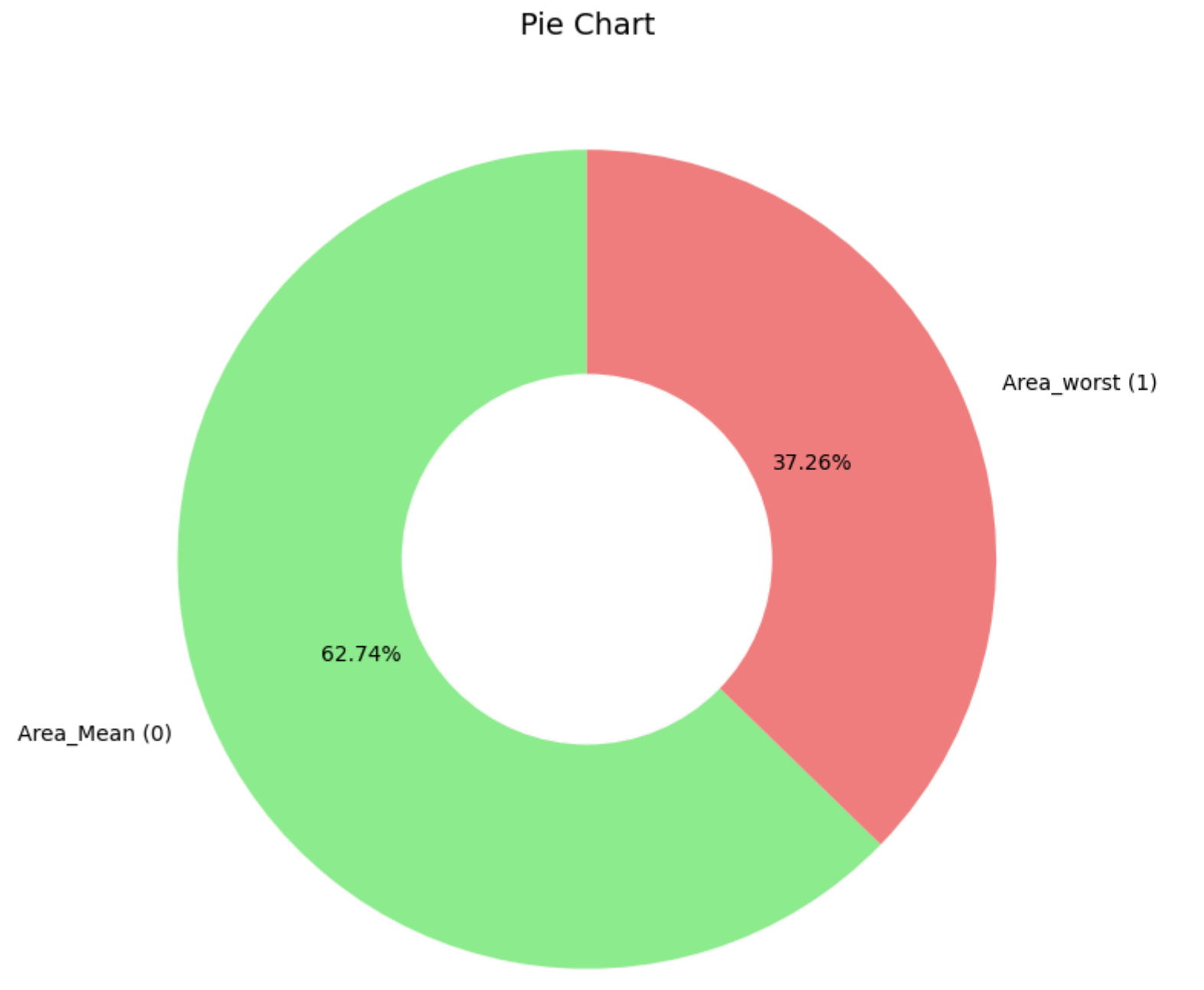
for bar in ax.patches:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height + 1, int(height),
            ha='center', fontsize=10)

plt.xlabel('diagnosis')
plt.ylabel('Total')
plt.xticks([0, 1], labels)
plt.grid(axis='y', linestyle='--', alpha=0.5)

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



diagnosis Distribution



```
colors = ['lightgreen', 'lightcoral']
labels = ['smoothness_mean (0)', 'smoothness_worst (1)']
order = df['diagnosis'].value_counts().index

plt.figure(figsize=(16, 8))
plt.suptitle('Smoothness tissue type', fontsize=16, fontweight='bold')

plt.subplot(1, 2, 1)
plt.title('Pie Chart', fontsize=14)
plt.pie(df['diagnosis'].value_counts(), labels=labels, colors=colors, autopct='%.2f%%', startangle=90)
centre = plt.Circle((0, 0), 0.45, color='white')
plt.gca().add_artist(centre)

plt.subplot(1, 2, 2)
plt.title('Histogram', fontsize=14)
ax = sns.countplot(x='diagnosis', data=df, palette=colors, order=order)

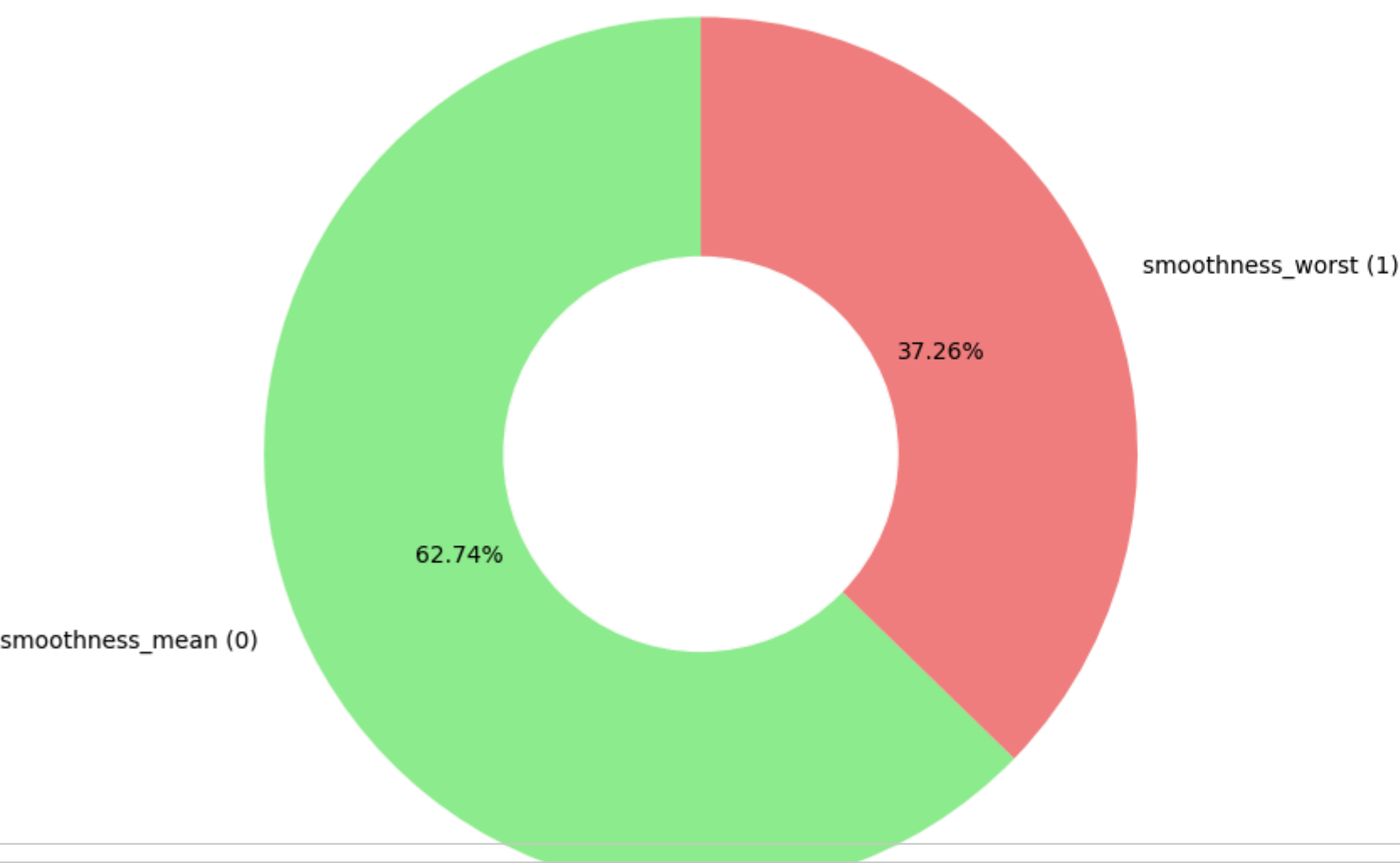
for bar in ax.patches:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width() / 2, height + 1, int(height),
            ha='center', fontsize=10)

plt.xlabel('Diagnosis')
plt.ylabel('Total')
plt.xticks([0, 1], labels)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()

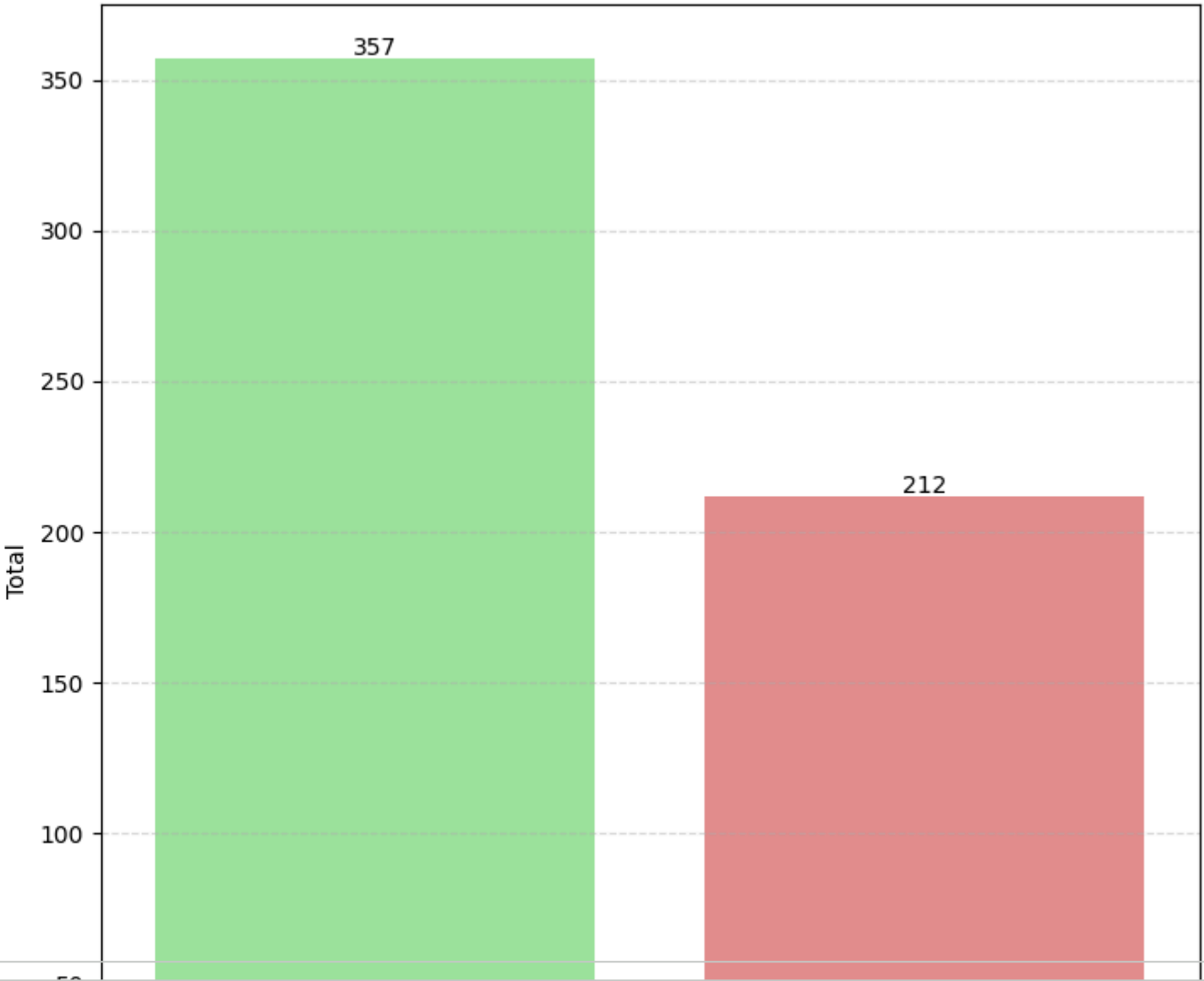
print('* * 30)
print('Diagnosis Total')
print('* * 30)
print(df['diagnosis'].value_counts(dropna=False))
```

Smoothness tissue type

Pie Chart



Histogram



```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.scatterplot(
    x='smoothness_mean',
    y='smoothness_worst'
```