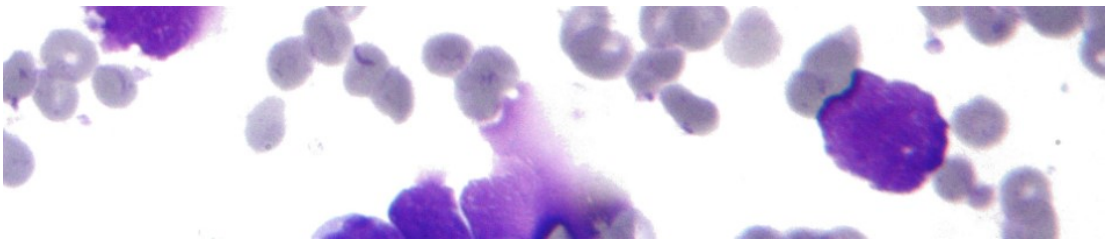
Data Visualization with Seaborn (Part 2): Feature Selection and Classification



This project is second in a series focused on data visualization with Seaborn. You can find the first project Exploratory Data Analysis with Seaborn on Coursera (Hands-on projects).

Task 1: Loading Libraries and Data

- Exploratory Data Analysis (Separate Target from Features, Plot Diagnosis Distributions)
- Data Visualization '+Visualizing Standardized Data with Seaborn '+Violin Plots and Box Plots '+Using Joint Plots for Feature Comparison '+Observing the Distribution of Values and their Variance with Swarm Plots '+Observing all Pair-wise Correlations

Task 2: Dropping Correlated Columns from Feature Matrix

Task 3: Classification using XGBoost (minimal feature selection)

Task 4: Univariate Feature Selection and XGBoost

Task 5: Recursive Feature Elimination with Cross-Validation

Task 6: Feature Extraction using Principal Component Analysis

About the Dataset:

The Breast Cancer Diagnostic data is available on the UCI Machine Learning Repository. This database is also available through the UW CS ftp server.

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

Attribute Information:

- ID number
- Diagnosis (M = malignant, B = benign) 3-32)

Ten real-valued features are computed for each cell nucleus:

1. radius (mean of distances from center to points on the perimeter)
2. texture (standard deviation of gray-scale values)
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter^2 / area - 1.0)
7. concavity (severity of concave portions of the contour)
8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

All feature values are recoded with four significant digits.

Missing attribute values: none

Class distribution: 357 benign, 212 malignant

Task 1: Loading Libraries and Data

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('seaborn')
import time
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_3612\322581383.py:5: MatplotlibDeprec
ationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6,
as they no longer correspond to the styles shipped by seaborn. However, they will
remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use the seabo
rn API instead.
  plt.style.use('seaborn')
```

In [ ]:
```
data = pd.read_csv('data.csv')
```

Exploratory Data Analysis

Separate Target from Features

In [ ]:
```
data.head()
```

Out[ ]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoo
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 [ ]:
```
col = data.columns
print(col)
```

```
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 [ ]:
```
data.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 [ ]:
```
data.shape
```

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

In [ ]:
```
y = data.diagnosis

# drop_cols = ['id', 'diagnosis', 'Unnamed: 32']
drop_cols = ['id', 'Unnamed: 32']

data_new = data.drop(drop_cols, axis=1)      #axis=1 because dropping columns, i
data_new
```

Out[ ]:

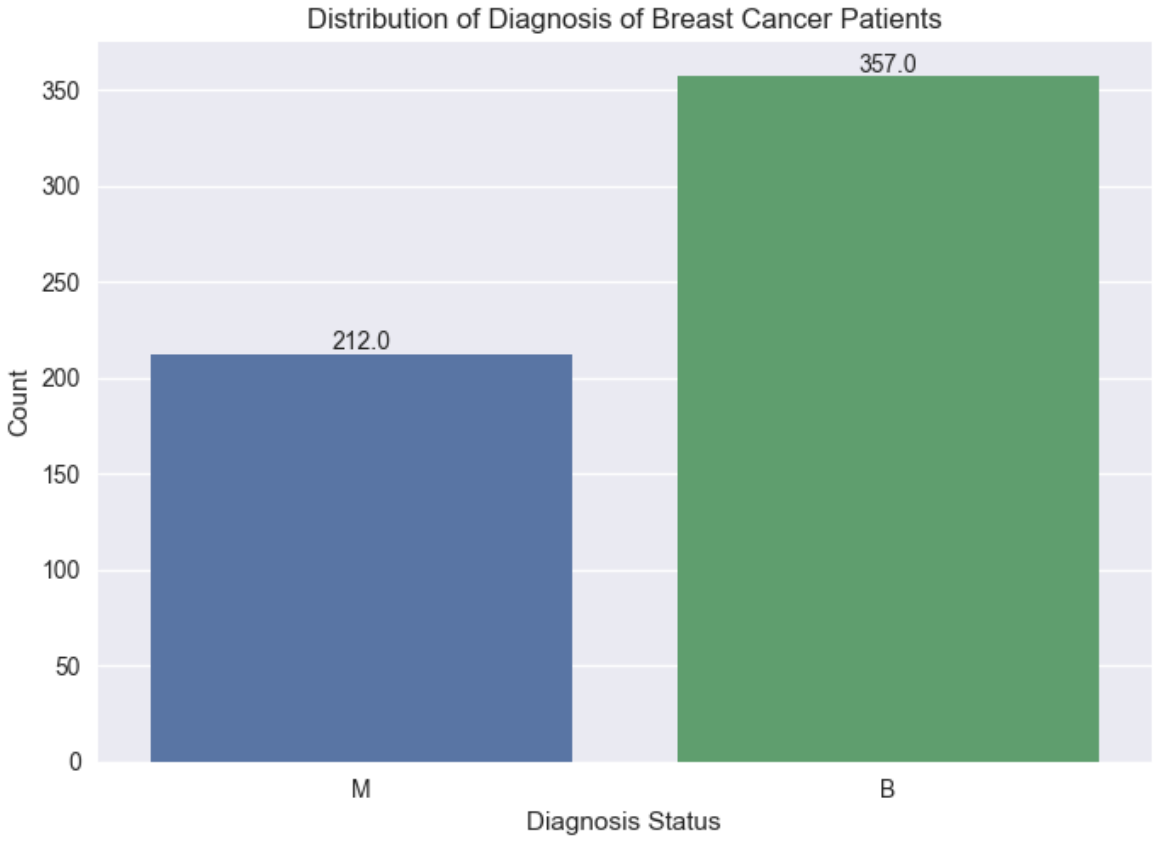
	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
0	M	17.99	10.38	122.80	1001.0	0.1
1	M	20.57	17.77	132.90	1326.0	0.0
2	M	19.69	21.25	130.00	1203.0	0.1
3	M	11.42	20.38	77.58	386.1	0.1
4	M	20.29	14.34	135.10	1297.0	0.1
...	...	...	...	...	...	
564	M	21.56	22.39	142.00	1479.0	0.1
565	M	20.13	28.25	131.20	1261.0	0.0
566	M	16.60	28.08	108.30	858.1	0.0
567	M	20.60	29.33	140.10	1265.0	0.1
568	B	7.76	24.54	47.92	181.0	0.0

569 rows × 31 columns

Plot Diagnosis Distributions

In [ ]:

```
# Create a DataFrame from the sample data
import pandas as pd
df = pd.DataFrame(data_new)

# Create the count plot
ax = sns.countplot(x='diagnosis', data=df)

# Add labels to the bars
for p in ax.patches:
   ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_heigh

# Add a title and labels
plt.title('Distribution of Diagnosis of Breast Cancer Patients')
plt.xlabel('Diagnosis Status')
plt.ylabel('Count')

# Display the plot
plt.show()
```



In [ ]:
```
x= df.drop('diagnosis', axis=1)       #axis=1 because dropping columns, if droppi
x.describe()
```

Out[ ]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cc
count	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	0.096360	
std	3.524049	4.301036	24.298981	351.914129	0.014064	
min	6.981000	9.710000	43.790000	143.500000	0.052630	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	

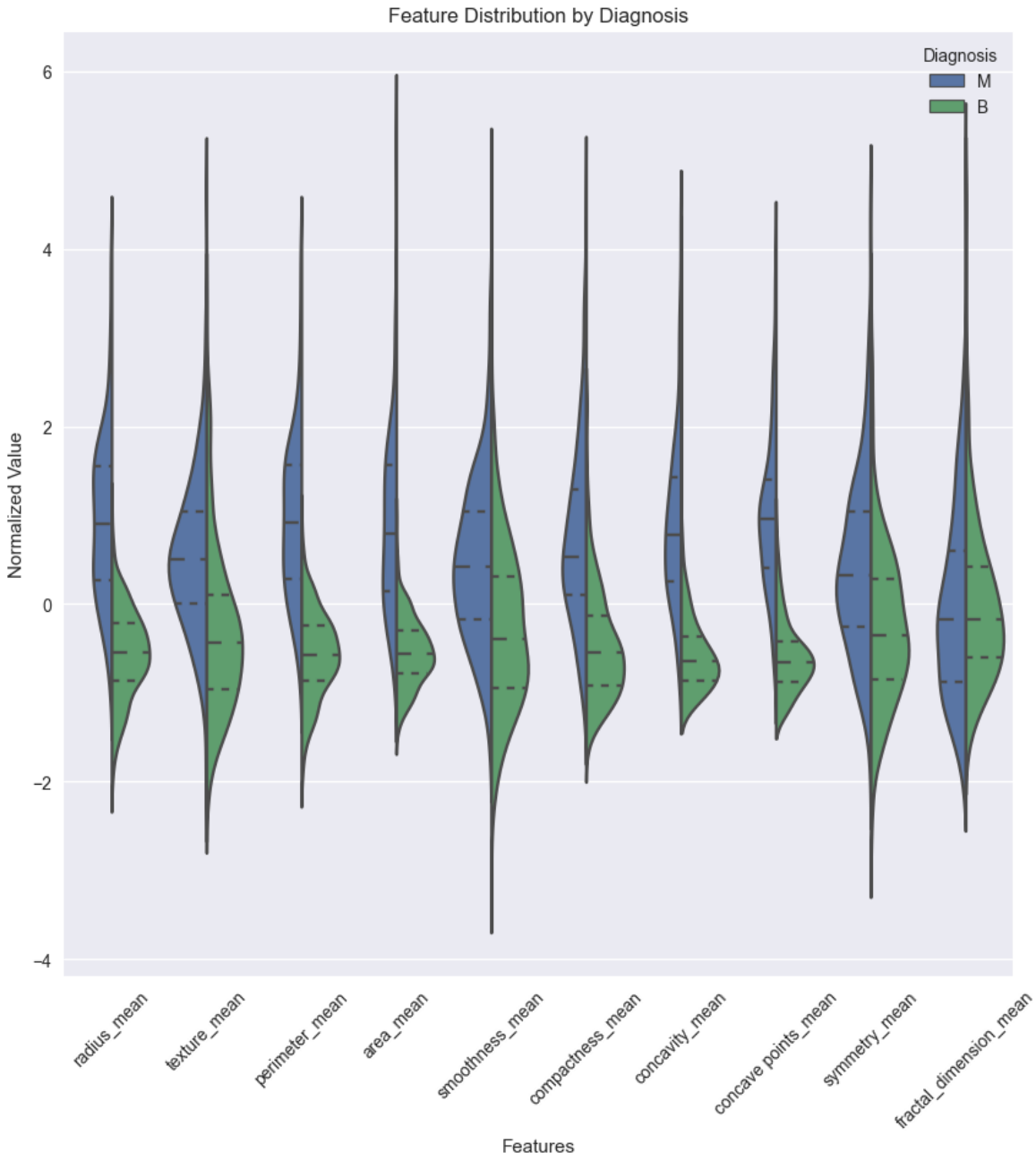
8 rows × 30 columns

Data Visualization

Visualizing Standardized Data with Seaborn

In [ ]:

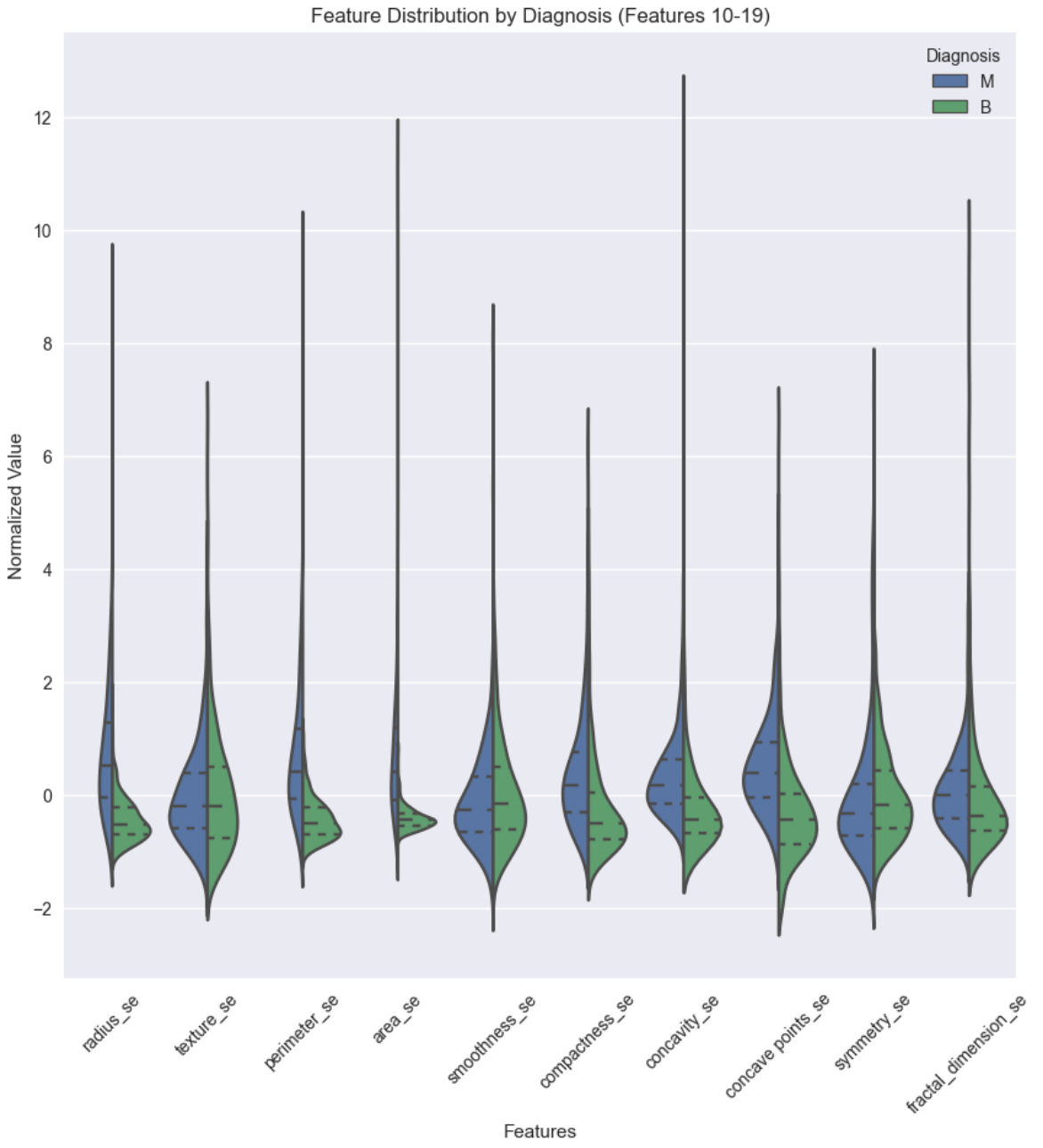
```
# Step 1: Assign diagnosis data to 'data_dia'
data_dia = y

# Step 2: Assign feature data to 'data'
data = x

# Step 3: Standardize features using Z-score normalization
data_n_2 = (data - data.mean()) / data.std()

# Step 4: Combine diagnosis data with the first ten standardized features
data = pd.concat([y, data_n_2.iloc[:, 0:10]], axis=1)

# Step 5: Reshape the DataFrame into a long format
data = pd.melt(data, id_vars="diagnosis", var_name="features", value_name='value

# Step 6: Create a new figure for plotting
plt.figure(figsize=(10, 10))

# Step 7: Create a violin plot for visualizing the data distribution
sns.violinplot(x="features", y="value", hue="diagnosis", data=data, split=True,

# Step 8: Rotate x-axis tick labels for better readability
plt.xticks(rotation=45)

# Add titles, labels, and legends to make the plot more informative
plt.title("Feature Distribution by Diagnosis")
plt.xlabel("Features")
plt.ylabel("Normalized Value")
plt.legend(title="Diagnosis")

# Display the plot
plt.show()
```



Violin Plots and Box Plots

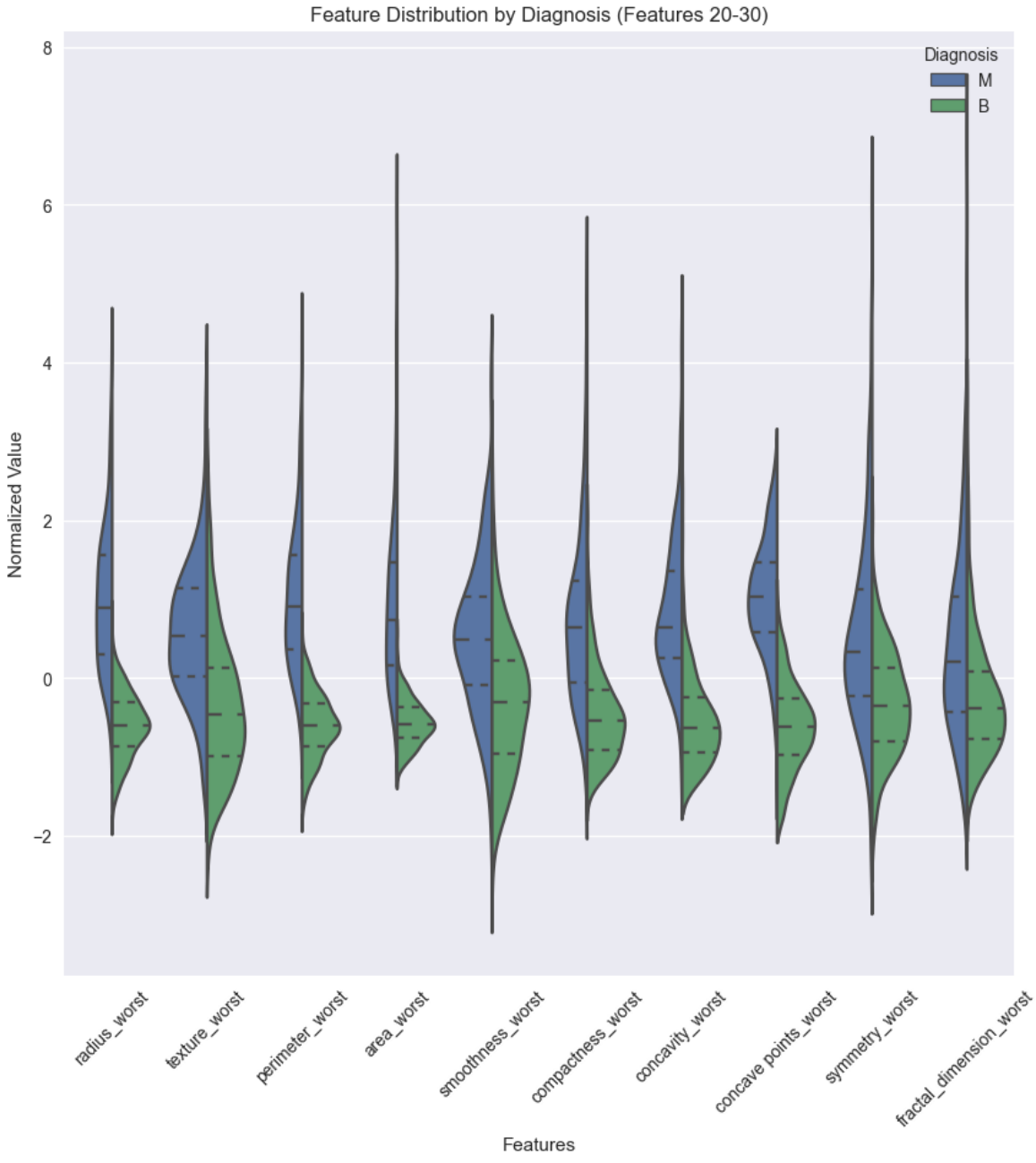
In [ ]:
```
# Combine diagnosis data with standardized features from 10th to 19th
data = pd.concat([y, data_n_2.iloc[:, 10:20]], axis=1)

# Reshape the DataFrame into a long format
data = pd.melt(data, id_vars="diagnosis", var_name="features", value_name='value

# Create a new figure for plotting
plt.figure(figsize=(10, 10))

# Create a violin plot for visualizing the data distribution
sns.violinplot(x="features", y="value", hue="diagnosis", data=data, split=True,

# Rotate x-axis tick labels for better readability
plt.xticks(rotation=45)

# Add titles, labels, and legends to make the plot more informative
```

```
plt.title("Feature Distribution by Diagnosis (Features 10-19)")
plt.xlabel("Features")
plt.ylabel("Normalized Value")
plt.legend(title="Diagnosis")

# Display the plot
plt.show()
```



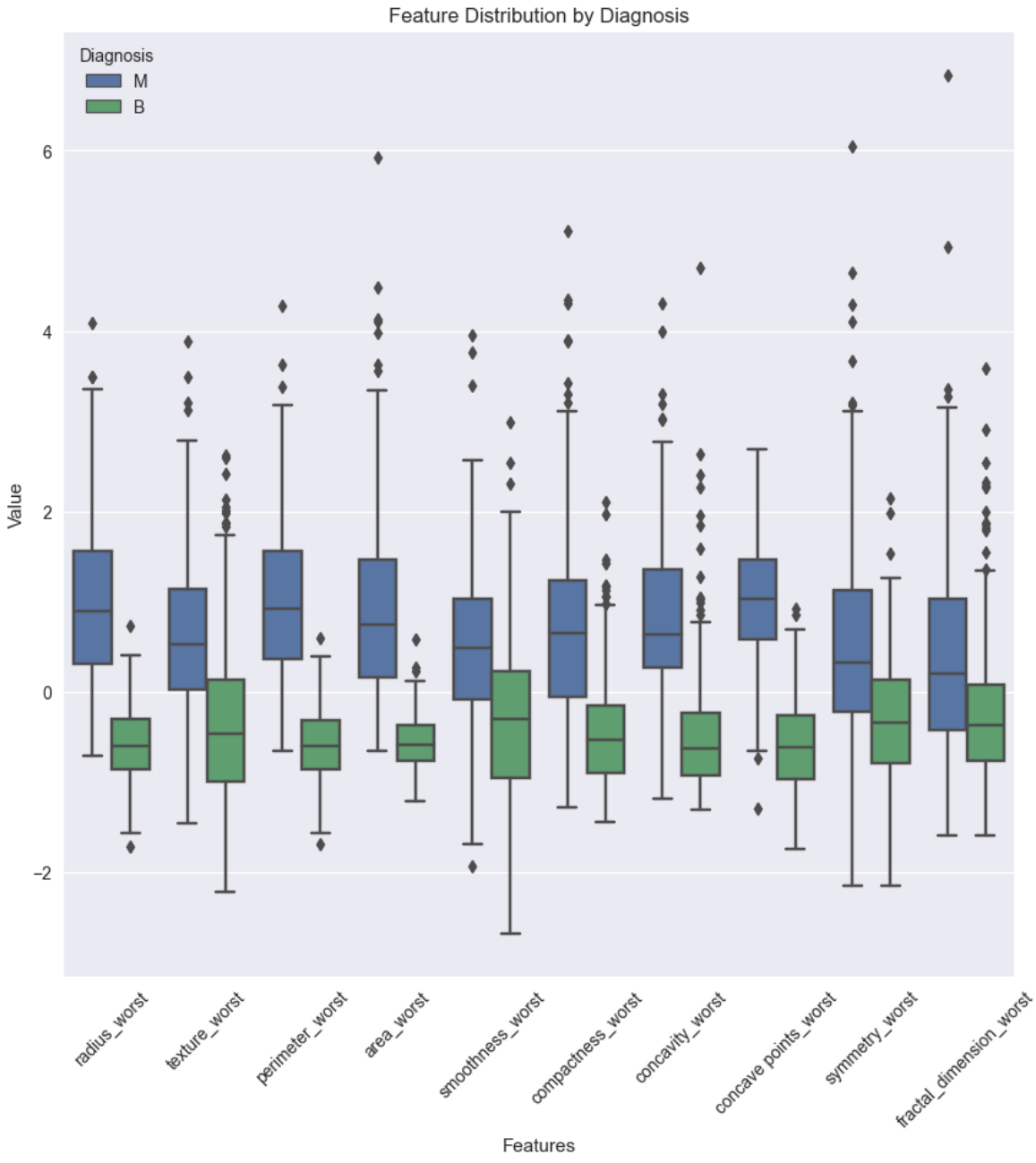
In [ ]:
```
# Combine diagnosis data with standardized features from 20th to 30th
data = pd.concat([y, data_n_2.iloc[:, 20:31]], axis=1)

# Reshape the DataFrame into a long format
data = pd.melt(data, id_vars="diagnosis", var_name="features", value_name='value

# Create a new figure for plotting
plt.figure(figsize=(10, 10))

# Create a violin plot for visualizing the data distribution
sns.violinplot(x="features", y="value", hue="diagnosis", data=data, split=True,

# Rotate x-axis tick labels for better readability
```

```
plt.xticks(rotation=45)

# Add titles, labels, and legends to make the plot more informative
plt.title("Feature Distribution by Diagnosis (Features 20-30)")
plt.xlabel("Features")
plt.ylabel("Normalized Value")
plt.legend(title="Diagnosis")

# Display the plot
plt.show()
```



In [ ]:
```
# Create a new figure for plotting
plt.figure(figsize=(10, 10))

# Create a box plot for comparing data distributions
sns.boxplot(x="features", y="value", hue="diagnosis", data=data)

# Rotate x-axis tick labels for better readability
plt.xticks(rotation=45)

# Add titles, labels, and legends to make the plot more informative
```

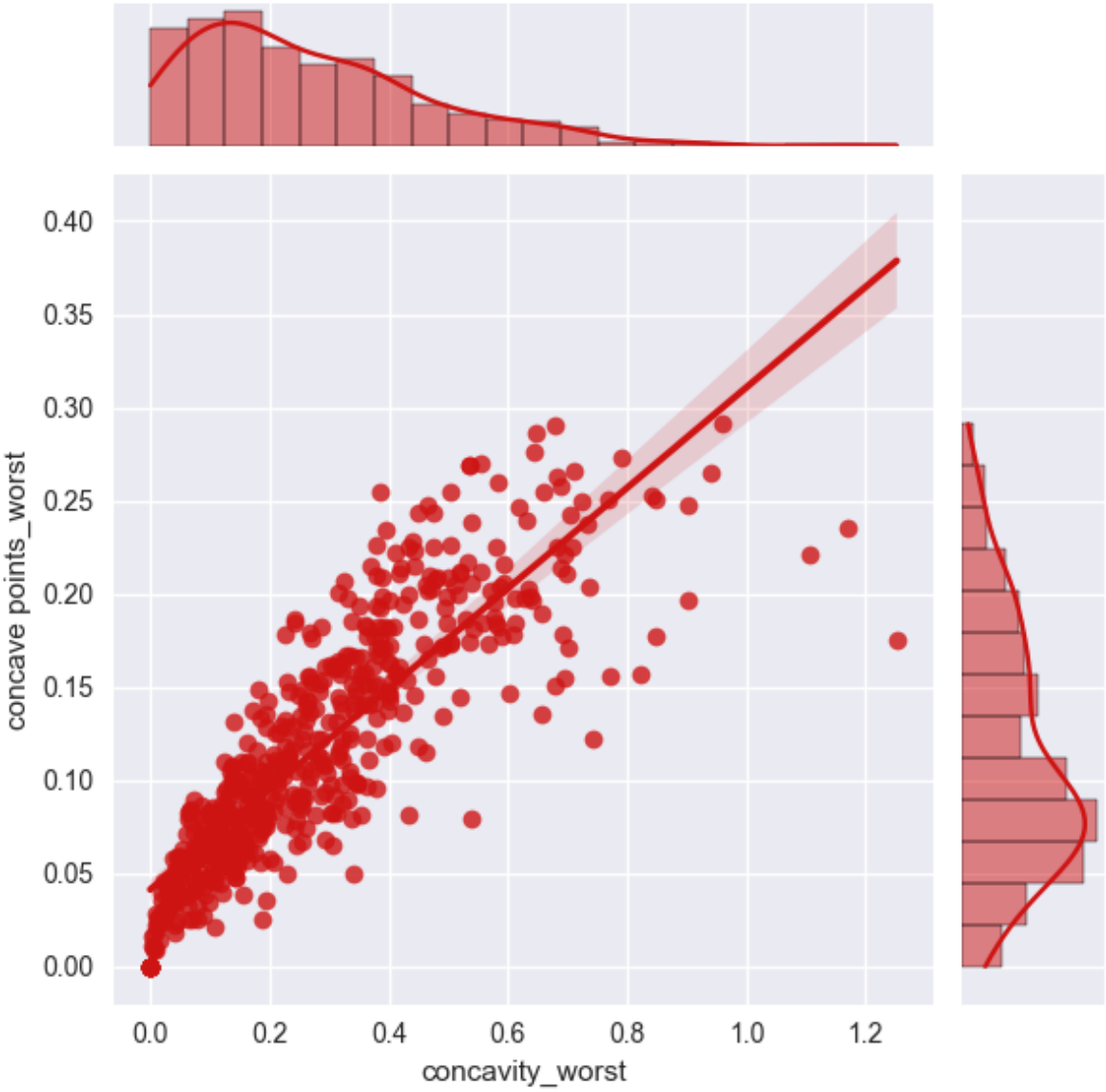
```
plt.title("Feature Distribution by Diagnosis")
plt.xlabel("Features")
plt.ylabel("Value")
plt.legend(title="Diagnosis")

# Display the plot
plt.show()
```



Using Joint Plots for Feature Comparison

In [ ]:
```
sns.jointplot(
    x=x.loc[:, 'concavity_worst'],        # Data for the x-axis (all rows of the
    y=x.loc[:, 'concave points_worst'],   # Data for the y-axis (all rows of the
    kind='reg',                           # Regression plot with scatter points
    color='#ce1413'                       # Color of the plot (you can specify a
)
```

Out[ ]:
```
<seaborn.axisgrid.JointGrid at 0x1d41af9bbb0>
```



Observing the Distribution of Values and their Variance with Swarm Plots

In [ ]:
```
# Combine diagnosis data with the first ten standardized features
data = pd.concat([y, data_n_2.iloc[:, 0:10]], axis=1)  # axis=1: concatenating t

# Reshape the DataFrame into a long format
data = pd.melt(data, id_vars="diagnosis", var_name="features", value_name='value

# Create a new figure for plotting
plt.figure(figsize=(10, 10))

# Create a swarm plot for comparing data distributions
sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)

# Rotate x-axis tick labels for better readability
plt.xticks(rotation=45)

# Add titles, labels, and legends to make the plot more informative
plt.title("Feature Distribution by Diagnosis")
plt.xlabel("Features")
plt.ylabel("Value")
plt.legend(title="Diagnosis")
```

```
# Display the plot
plt.show()
```

```
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 40.6% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 40.2% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 45.5% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 38.3% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 43.9% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 48.5% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 45.9% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 40.4% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 42.2% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 41.3% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 41.1% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 46.0% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 38.7% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 44.1% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 42.9% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
d:\Data science & Python  2022\0. Data Analyst_2023\Portfolio_Thach\I. Beginner level\Statistical Data Visualization with Seaborn_\Statistical Data Visualization with Seaborn.ipynb Cell 38 in 2
     <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/Statistical%20Data%20Visualization%20with%20Seaborn_/Statistical%20Data%20Visualization%20with%20Seaborn.ipynb#Y141sZmlsZQ%3D%3D?line=19'>20</a> plt.legend(title="Diagnosis")
     <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/Statistical%20Data%20Visualization%20with%20Seaborn_/Statistical%20Data%20Visualization%20with%20Seaborn.ipynb#Y141sZmlsZQ%3D%3D?line=21'>22</a> # Display the plot
---> <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/Statistical%20Data%20Visualization%20with%20Seaborn_/Statistical%20Data%20Visualization%20with%20Seaborn.ipynb#Y141sZmlsZQ%3D%3D?line=22'>23</a> plt.show()

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\pyplot.py:446, in show(*args, **kwargs)
    402 """
    403 Display all open figures.
    404
   (...)
    443 explicitly there.
    444 """
    445 _warn_if_gui_out_of_main_thread()
--> 446 return _get_backend_mod().show(*args, **kwargs)

File ~\AppData\Roaming\Python\Python310\site-packages\matplotlib_inline\backend_inline.py:90, in show(close, block)
     88 try:
     89     for figure_manager in Gcf.get_all_fig_managers():
---> 90         display(
     91             figure_manager.canvas.figure,
     92             metadata=_fetch_figure_metadata(figure_manager.canvas.figure)
     93         )
     94 finally:
     95     show._to_draw = []

File ~\AppData\Roaming\Python\Python310\site-packages\IPython\core\display_functions.py:298, in display(include, exclude, metadata, transient, display_id, raw, clear, *objs, **kwargs)
    296     publish_display_data(data=obj, metadata=metadata, **kwargs)
    297 else:
--> 298     format_dict, md_dict = format(obj, include=include, exclude=exclude)
    299     if not format_dict:
    300         # nothing to display (e.g. _ipython_display_ took over)
    301         continue

File ~\AppData\Roaming\Python\Python310\site-packages\IPython\core\formatters.py:178, in DisplayFormatter.format(self, obj, include, exclude)
    176 md = None
    177 try:
--> 178     data = formatter(obj)
    179 except:
    180     # FIXME: log the exception
    181     raise

File ~\AppData\Roaming\Python\Python310\site-packages\decorator.py:232, in decora
```

```
te.<locals>.fun(*args, **kw)
    230 if not kwsyntax:
    231     args, kw = fix(args, kw, sig)
--> 232 return caller(func, *(extras + args), **kw)

File ~\AppData\Roaming\Python\Python310\site-packages\IPython\core\formatters.py:
222, in catch_format_error(method, self, *args, **kwargs)
    220 """show traceback on failed format call"""
    221 try:
--> 222     r = method(self, *args, **kwargs)
    223 except NotImplementedError:
    224     # don't warn on NotImplementedErrors
    225     return self._check_return(None, args[0])

File ~\AppData\Roaming\Python\Python310\site-packages\IPython\core\formatters.py:
339, in BaseFormatter.__call__(self, obj)
    337     pass
    338 else:
--> 339     return printer(obj)
    340 # Finally look for special method names
    341 method = get_real_method(obj, self.print_method)

File ~\AppData\Roaming\Python\Python310\site-packages\IPython\core\pylabtools.py:
151, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    148     from matplotlib.backend_bases import FigureCanvasBase
    149     FigureCanvasBase(fig)
--> 151 fig.canvas.print_figure(bytes_io, **kw)
    152 data = bytes_io.getvalue()
    153 if fmt == 'svg':

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\backend_bases.py:2366, in FigureCanvasBase.print_figure(self, filename, d
pi, facecolor, edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extr
a_artists, backend, **kwargs)
   2362 try:
   2363     # _get_renderer may change the figure dpi (as vector formats
   2364     # force the figure dpi to 72), so we need to set it again here.
   2365     with cbook._setattr_cm(self.figure, dpi=dpi):
-> 2366         result = print_method(
   2367             filename,
   2368             facecolor=facecolor,
   2369             edgecolor=edgecolor,
   2370             orientation=orientation,
   2371             bbox_inches_restore=_bbox_inches_restore,
   2372             **kwargs)
   2373 finally:
   2374     if bbox_inches and restore_bbox:

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\backend_bases.py:2232, in FigureCanvasBase._switch_canvas_and_return_prin
t_method.<locals>.<lambda>(*args, **kwargs)
   2228     optional_kws = {  # Passed by print_figure for other renderers.
   2229         "dpi", "facecolor", "edgecolor", "orientation",
   2230         "bbox_inches_restore"}
   2231     skip = optional_kws - {*inspect.signature(meth).parameters}
-> 2232     print_method = functools.wraps(meth)(lambda *args, **kwargs: meth(
   2233         *args, **{k: v for k, v in kwargs.items() if k not in skip}))
   2234 else:  # Let third-parties do as they see fit.
   2235     print_method = meth
```

```
File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\backends\backend_agg.py:509, in FigureCanvasAgg.print_png(self, filename_
or_obj, metadata, pil_kwargs)
    462 def print_png(self, filename_or_obj, *, metadata=None, pil_kwargs=None):
    463     """
    464     Write the figure to a PNG file.
    465
   (...)
    507         *metadata*, including the default 'Software' key.
    508     """
--> 509     self._print_pil(filename_or_obj, "png", pil_kwargs, metadata)

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\backends\backend_agg.py:457, in FigureCanvasAgg._print_pil(self, filename
_or_obj, fmt, pil_kwargs, metadata)
    452 def _print_pil(self, filename_or_obj, fmt, pil_kwargs, metadata=None):
    453     """
    454     Draw the canvas, then save it using `.image.imsave` (to which
    455     *pil_kwargs* and *metadata* are forwarded).
    456     """
--> 457     FigureCanvasAgg.draw(self)
    458     mpl.image.imsave(
    459         filename_or_obj, self.buffer_rgba(), format=fmt, origin="upper",
    460         dpi=self.figure.dpi, metadata=metadata, pil_kwargs=pil_kwargs)

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\backends\backend_agg.py:400, in FigureCanvasAgg.draw(self)
    396 # Acquire a lock on the shared font cache.
    397 with RendererAgg.lock, \
    398      (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    399       else nullcontext()):
--> 400     self.figure.draw(self.renderer)
    401     # A GUI class may be need to update a window using this draw, so
    402     # don't forget to call the superclass.
    403     super().draw()

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\artist.py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, re
nderer, *args, **kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\artist.py:72, in allow_rasterization.<locals>.draw_wrapper(artist, render
er)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\figure.py:3175, in Figure.draw(self, renderer)
   3172         # ValueError can occur when resizing a window.
   3174 self.patch.draw(renderer)
-> 3175 mimage._draw_list_compositing_images(
   3176     renderer, self, artists, self.suppressComposite)
```

```
   3178 for sfig in self.subfigs:
   3179     sfig.draw(renderer)

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\image.py:131, in _draw_list_compositing_images(renderer, parent, artists, suppress_composite)
    129 if not_composite or not has_images:
    130     for a in artists:
--> 131         a.draw(renderer)
    132 else:
    133     # Composite any adjacent images together
    134     image_group = []

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\artist.py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\axes\_base.py:3064, in _AxesBase.draw(self, renderer)
   3061 if artists_rasterized:
   3062     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3064 mimage._draw_list_compositing_images(
   3065     renderer, self, artists, self.figure.suppressComposite)
   3067 renderer.close_group('axes')
   3068 self.stale = False

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\image.py:131, in _draw_list_compositing_images(renderer, parent, artists, suppress_composite)
    129 if not_composite or not has_images:
    130     for a in artists:
--> 131         a.draw(renderer)
    132 else:
    133     # Composite any adjacent images together
    134     image_group = []

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\categorical.py:377, in _CategoricalPlotterNew.plot_swarms.<locals>.draw(points, renderer, center)
    375 def draw(points, renderer, *, center=center):
--> 377     beeswarm(points, center)
    379     if self.orient == "h":
    380         scalex = False

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\categorical.py:3400, in Beeswarm.__call__(self, points, center)
   3398 # Adjust points along the categorical axis to prevent overlaps
   3399 new_xyr = np.empty_like(orig_xyr)
-> 3400 new_xyr[sorter] = self.beeswarm(orig_xyr)
   3402 # Transform the point coordinates back to data coordinates
   3403 if self.orient == "h":

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\categorical.py:3438, in Beeswarm.beeswarm(self, orig_xyr)
   3433 # Loop over the remaining points
```

```
   3434 for xyr_i in orig_xyr[1:]:
   3435
   3436     # Find the points in the swarm that could possibly
   3437     # overlap with the point we are currently placing
-> 3438     neighbors = self.could_overlap(xyr_i, swarm)
   3440     # Find positions that would be valid individually
   3441     # with respect to each of the swarm neighbors
   3442     candidates = self.position_candidates(xyr_i, neighbors)

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\categorical.py:3468, in Beeswarm.could_overlap(self, xyr_i, swarm)
   3466     else:
   3467         break
-> 3468 return np.array(neighbors)[::-1]

KeyboardInterrupt:
```

In [ ]:

```python
# Combine diagnosis data with standardized features from 10th to 19th
data = pd.concat([y, data_n_2.iloc[:, 10:20]], axis=1)

# Reshape the DataFrame into a long format
data = pd.melt(data, id_vars="diagnosis", var_name="features", value_name='value

# Create a new figure for plotting
plt.figure(figsize=(10, 10))

# Create a swarm plot for comparing data distributions
sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)

# Rotate x-axis tick labels for better readability
plt.xticks(rotation=45)

# Add titles, labels, and legends to make the plot more informative
plt.title("Feature Distribution by Diagnosis (Features 10-19)")
plt.xlabel("Features")
plt.ylabel("Value")
plt.legend(title="Diagnosis")

# Display the plot
plt.show()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
d:\Data science & Python  2022\0. Data Analyst_2023\Portfolio_Thach\I. Beginner level\Statistical Data Visualization with Seaborn_\Statistical Data Visualization with Seaborn.ipynb Cell 39 in 1
      <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/Statistical%20Data%20Visualization%20with%20Seaborn_/Statistical%20Data%20Visualization%20with%20Seaborn.ipynb#Y142sZmlsZQ%3D%3D?line=7'>8</a> plt.figure(figsize=(10, 10))
     <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/Statistical%20Data%20Visualization%20with%20Seaborn_/Statistical%20Data%20Visualization%20with%20Seaborn.ipynb#Y142sZmlsZQ%3D%3D?line=9'>10</a> # Create a swarm plot for comparing data distributions
---> <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/Statistical%20Data%20Visualization%20with%20Seaborn_/Statistical%20Data%20Visualization%20with%20Seaborn.ipynb#Y142sZmlsZQ%3D%3D?line=10'>11</a> sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)
     <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/Statistical%20Data%20Visualization%20with%20Seaborn_/Statistical%20Data%20Visualization%20with%20Seaborn.ipynb#Y142sZmlsZQ%3D%3D?line=12'>13</a> # Rotate x-axis tick labels for better readability
     <a href='vscode-notebook-cell:/d%3A/Data%20science%20%26%20Python%20%202022/0.%20Data%20Analyst_2023/Portfolio_Thach/I.%20Beginner%20level/Statistical%20Data%20Visualization%20with%20Seaborn_/Statistical%20Data%20Visualization%20with%20Seaborn.ipynb#Y142sZmlsZQ%3D%3D?line=13'>14</a> plt.xticks(rotation=45)

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\categorical.py:2664, in swarmplot(data, x, y, hue, order, hue_order, dodge, orient, color, palette, size, edgecolor, linewidth, hue_norm, native_scale, formatter, legend, warn_thresh, ax, **kwargs)
   2657     linewidth = size / 10
   2659 kwargs.update(dict(
   2660     s=size ** 2,
   2661     linewidth=linewidth,
   2662 ))
-> 2664 p.plot_swarms(
   2665     dodge=dodge,
   2666     color=color,
   2667     edgecolor=edgecolor,
   2668     warn_thresh=warn_thresh,
   2669     plot_kws=kwargs,
   2670 )
   2672 p._add_axis_labels(ax)
   2673 p._adjust_cat_axis(ax, axis=p.cat_axis)

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\categorical.py:399, in _CategoricalPlotterNew.plot_swarms(self, dodge, color, edgecolor, warn_thresh, plot_kws)
    395             super(points.__class__, points).draw(renderer)
    397         points.draw = draw.__get__(points)
--> 399 _draw_figure(ax.figure)
    401 # Finalize the axes details
    402 if self.legend == "auto":

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\utils.py:80, in _draw_figure(fig)
     78 """Force draw of a matplotlib figure, accounting for back-compat."""
```

```
     79 # See https://github.com/matplotlib/matplotlib/issues/19197 for context
---> 80 fig.canvas.draw()
     81 if fig.stale:
     82     try:

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\backends\backend_agg.py:400, in FigureCanvasAgg.draw(self)
    396 # Acquire a lock on the shared font cache.
    397 with RendererAgg.lock, \
    398      (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    399       else nullcontext()):
--> 400     self.figure.draw(self.renderer)
    401     # A GUI class may be need to update a window using this draw, so
    402     # don't forget to call the superclass.
    403     super().draw()

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\artist.py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, re
nderer, *args, **kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\artist.py:72, in allow_rasterization.<locals>.draw_wrapper(artist, render
er)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\figure.py:3175, in Figure.draw(self, renderer)
   3172         # ValueError can occur when resizing a window.
   3174 self.patch.draw(renderer)
-> 3175 mimage._draw_list_compositing_images(
   3176     renderer, self, artists, self.suppressComposite)
   3178 for sfig in self.subfigs:
   3179     sfig.draw(renderer)

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\image.py:131, in _draw_list_compositing_images(renderer, parent, artists,
suppress_composite)
    129 if not_composite or not has_images:
    130     for a in artists:
--> 131         a.draw(renderer)
    132 else:
    133     # Composite any adjacent images together
    134     image_group = []

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\artist.py:72, in allow_rasterization.<locals>.draw_wrapper(artist, render
er)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
```

```
     74     if artist.get_agg_filter() is not None:

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\axes\_base.py:3064, in _AxesBase.draw(self, renderer)
   3061 if artists_rasterized:
   3062     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3064 mimage._draw_list_compositing_images(
   3065     renderer, self, artists, self.figure.suppressComposite)
   3067 renderer.close_group('axes')
   3068 self.stale = False

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\mat
plotlib\image.py:131, in _draw_list_compositing_images(renderer, parent, artists,
suppress_composite)
    129 if not_composite or not has_images:
    130     for a in artists:
--> 131         a.draw(renderer)
    132 else:
    133     # Composite any adjacent images together
    134     image_group = []

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\sea
born\categorical.py:377, in _CategoricalPlotterNew.plot_swarms.<locals>.draw(poin
ts, renderer, center)
    375 def draw(points, renderer, *, center=center):
--> 377     beeswarm(points, center)
    379     if self.orient == "h":
    380         scalex = False

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\sea
born\categorical.py:3400, in Beeswarm.__call__(self, points, center)
   3398 # Adjust points along the categorical axis to prevent overlaps
   3399 new_xyr = np.empty_like(orig_xyr)
-> 3400 new_xyr[sorter] = self.beeswarm(orig_xyr)
   3402 # Transform the point coordinates back to data coordinates
   3403 if self.orient == "h":

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\sea
born\categorical.py:3449, in Beeswarm.beeswarm(self, orig_xyr)
   3446 candidates = candidates[np.argsort(offsets)]
   3448 # Find the first candidate that does not overlap any neighbors
-> 3449 new_xyr_i = self.first_non_overlapping_candidate(candidates, neighbors)
   3451 # Place it into the swarm
   3452 swarm = np.vstack([swarm, new_xyr_i])

File c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\sea
born\categorical.py:3504, in Beeswarm.first_non_overlapping_candidate(self, candi
dates, neighbors)
   3502 dx = neighbors_x - x_i
   3503 dy = neighbors_y - y_i
-> 3504 sq_distances = np.square(dx) + np.square(dy)
   3506 sep_needed = np.square(neighbors_r + r_i)
   3508 # Good candidate does not overlap any of neighbors which means that
   3509 # squared distance between candidate and any of the neighbors has
   3510 # to be at least square of the summed radii

KeyboardInterrupt:
```

```
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 64.7% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 61.2% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 66.4% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 71.2% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 61.7% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

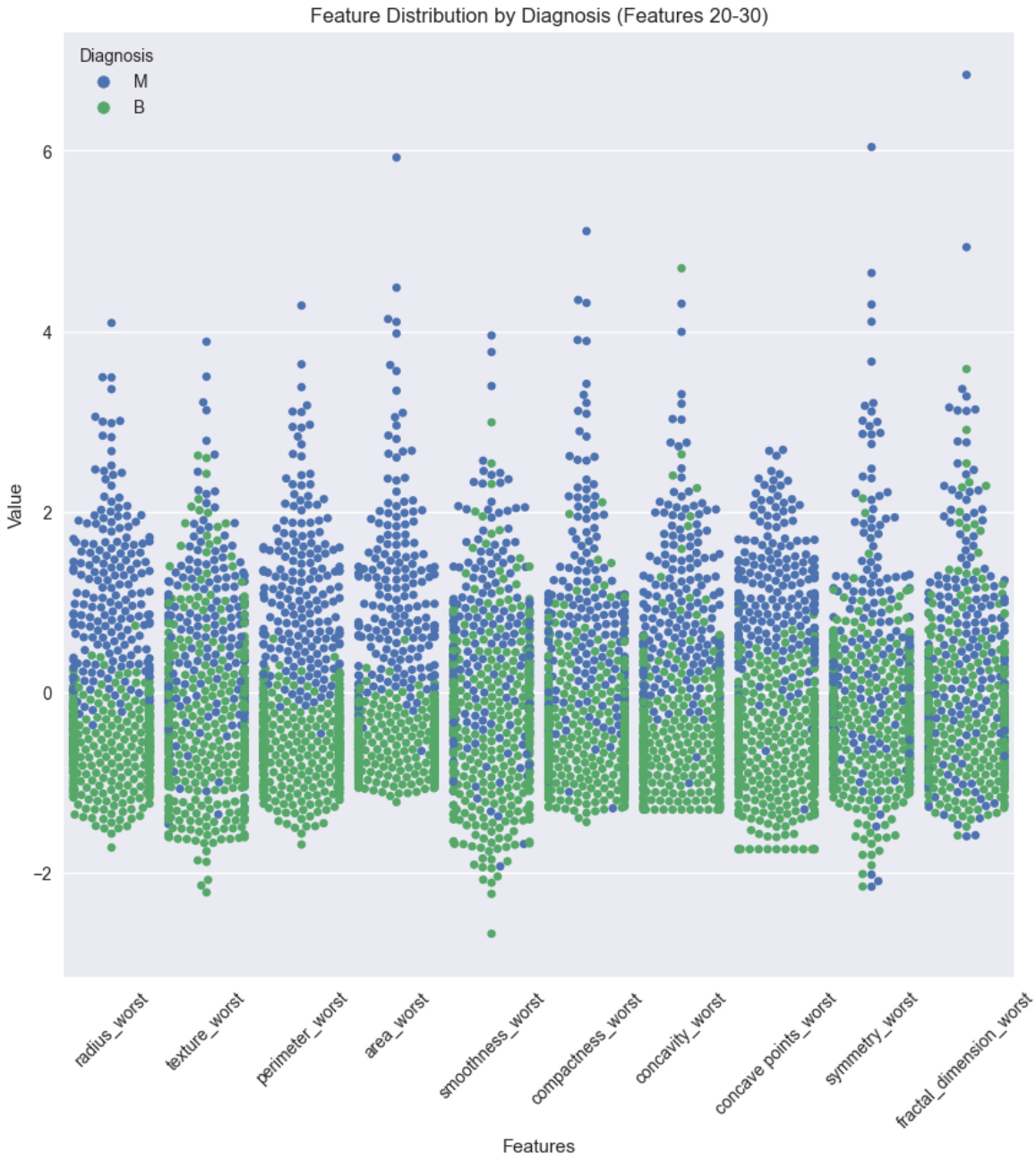
In [ ]:

```python
# Combine diagnosis data with standardized features from 20th to 30th
data = pd.concat([y, data_n_2.iloc[:, 20:31]], axis=1)

# Reshape the DataFrame into a long format
data = pd.melt(data, id_vars="diagnosis", var_name="features", value_name='value

# Create a new figure for plotting
plt.figure(figsize=(10, 10))

# Create a swarm plot for comparing data distributions
sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)

# Rotate x-axis tick labels for better readability
plt.xticks(rotation=45)

# Add titles, labels, and legends to make the plot more informative
plt.title("Feature Distribution by Diagnosis (Features 20-30)")
plt.xlabel("Features")
plt.ylabel("Value")
plt.legend(title="Diagnosis")
```

```
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 46.4% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 43.8% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 46.6% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 53.4% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 42.9% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 49.7% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 48.7% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 43.9% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 49.0% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 48.0% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

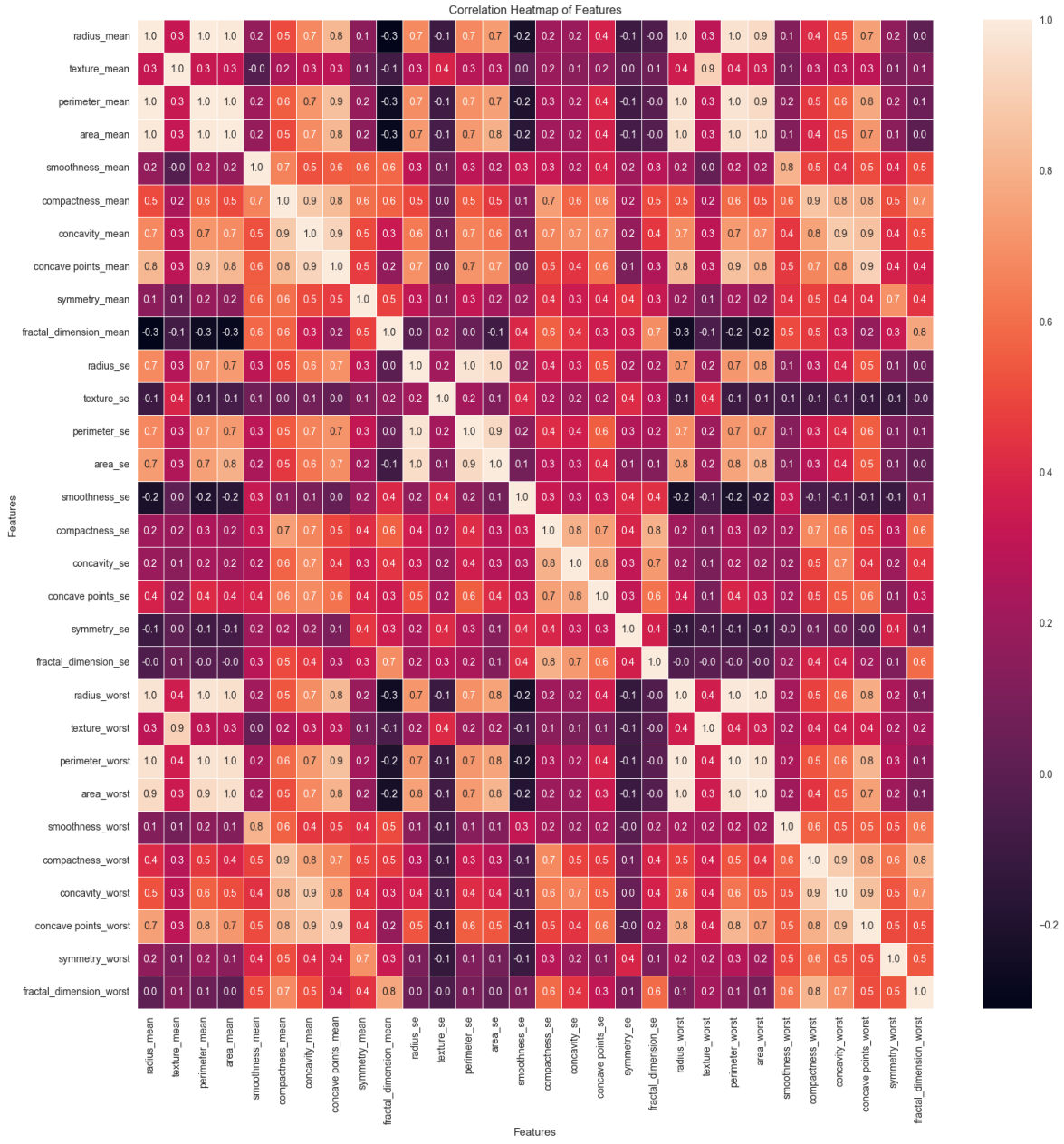
Out[ ]: <matplotlib.legend.Legend at 0x1d86a06f910>

```
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 46.7% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 44.3% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 53.8% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 43.4% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 50.1% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\categorical.py:3544: UserWarning: 49.9% of the points cannot be placed; you may
want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



Observing all Pair-wise Correlations

In [ ]:
```
# Create a new figure for plotting
f, ax = plt.subplots(figsize=(18, 18))

# Generate a correlation heatmap using a seaborn heatmap
sns.heatmap(x.corr(), annot=True, linewidths=.5, fmt='.1f', ax=ax)

# Add titles and labels for better understanding
plt.title("Correlation Heatmap of Features")
plt.xlabel("Features")
plt.ylabel("Features")

# Display the heatmap
plt.show()
```



Task 2: Dropping Correlated Columns from Feature Matrix

Note: If you are starting the notebook from this task, you can run cells from all the previous tasks in the kernel by going to the top menu and Kernel > Restart and Run All\n",

---

In [ ]:

```
# List of columns to be dropped
drop_cols = ['perimeter_mean', 'radius_mean', 'compactness_mean',
             'concave points_mean', 'radius_se', 'perimeter_se',
             'radius_worst', 'perimeter_worst', 'compactness_worst',
             'concave points_worst', 'compactness_se', 'concave points_se',
             'texture_worst', 'area_worst']

# Create a new DataFrame by dropping the specified columns
df = x.drop(drop_cols, axis=1)
```

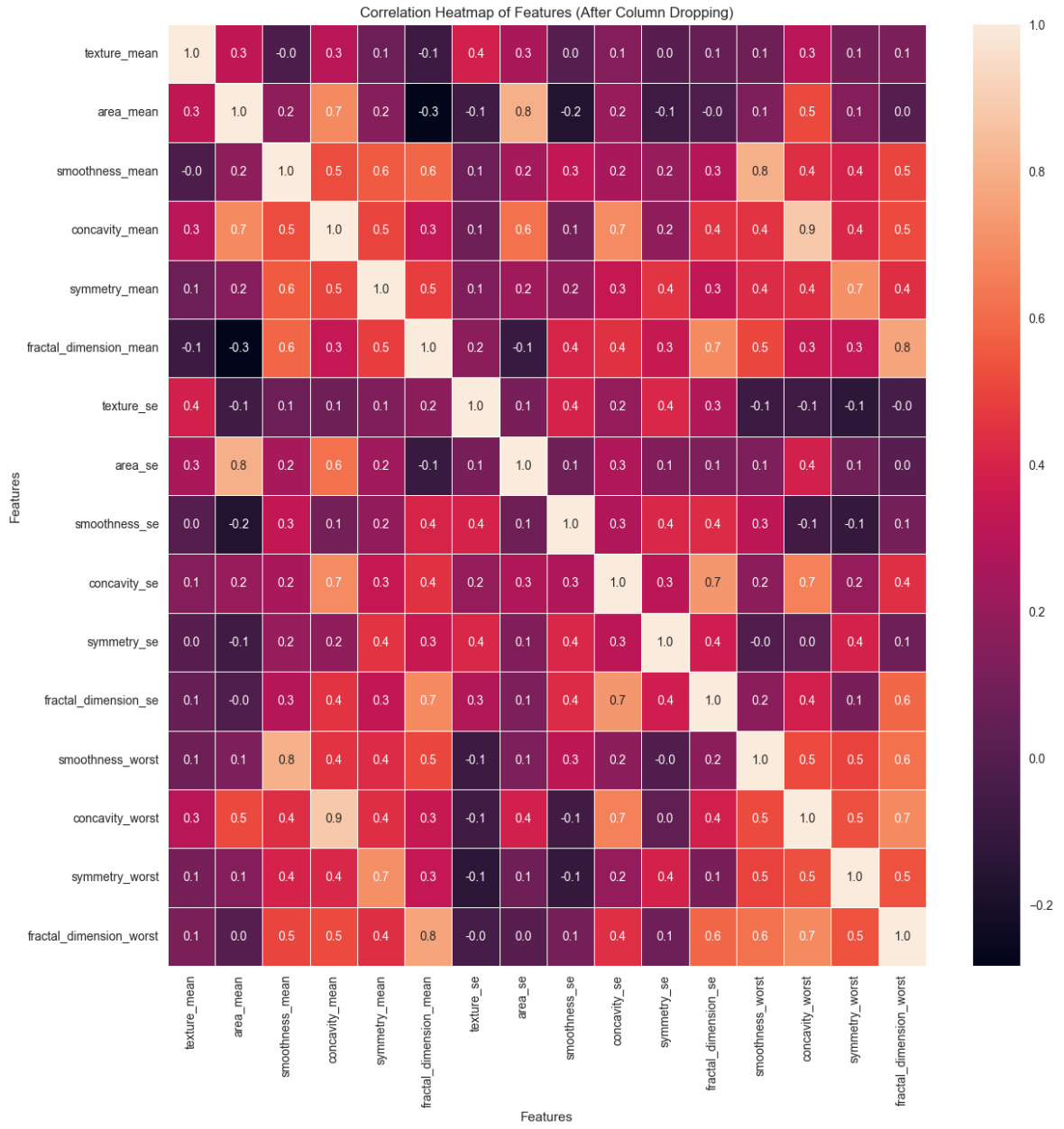
```
# Display the first few rows of the new DataFrame
df.head()
```

Out[ ]:

	texture_mean	area_mean	smoothness_mean	concavity_mean	symmetry_mean	fract
0	10.38	1001.0	0.11840	0.3001	0.2419	
1	17.77	1326.0	0.08474	0.0869	0.1812	
2	21.25	1203.0	0.10960	0.1974	0.2069	
3	20.38	386.1	0.14250	0.2414	0.2597	
4	14.34	1297.0	0.10030	0.1980	0.1809	

In [ ]:

```
# Create a new figure for plotting
f, ax = plt.subplots(figsize=(14, 14))

# Generate a correlation heatmap using a seaborn heatmap
sns.heatmap(df.corr(), annot=True, linewidths=.5, fmt='.1f', ax=ax)

# Add titles and labels for better understanding
plt.title("Correlation Heatmap of Features (After Column Dropping)")
plt.xlabel("Features")
plt.ylabel("Features")

# Display the heatmap
plt.show()
```



Task 3: Classification using XGBoost (minimal feature selection)

In [ ]:
```
# pip install scikit-learn
```

In [ ]:
```
# pip install xgboost
```

In [ ]:
```
# Import necessary libraries
from sklearn.model_selection import train_test_split
import xgboost as xgb

# Import necessary metric functions
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
```

In [ ]:
```
# Convert class labels to numerical format
y = y.apply(lambda label: 0 if label == 'B' else 1)

# Convert class labels to numerical format
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
```

```
y_encoded = label_encoder.fit_transform(y)

# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(df, y, test_size=0.3, random

# Creating an XGBoost classifier model
clf = xgb.XGBClassifier(random_state=42)

# Training the model on the training data
clf.fit(x_train, y_train)

# Making predictions on the testing data
y_pred = clf.predict(x_test)
```

In [ ]:
```
y_pred.shape
```

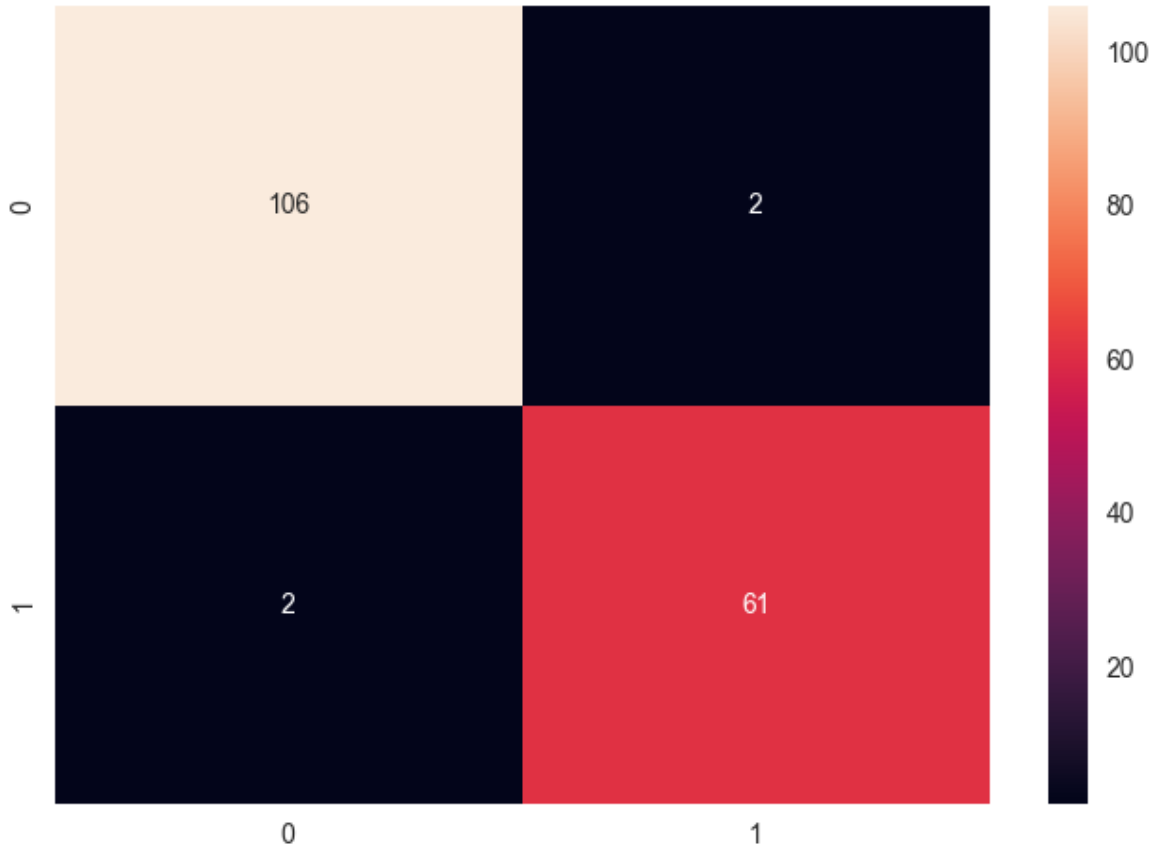
Out[ ]: (171,)

Accuracy = Correct predictions/ Total = (TN +TP)/Total

In [ ]:
```
# Calculate and print accuracy
accuracy = accuracy_score(y_test, clf_l.predict(x_test))
print('Accuracy = (TP+TN)/Total = (106+61)/171 is:', round(accuracy*100,2), '%')

# Calculate confusion matrix
cn = confusion_matrix(y_test, clf_l.predict(x_test))

# Create a heatmap of the confusion matrix, with label value (annot=True)
sns.heatmap(cn, annot=True, fmt='d')
```

```
Accuracy = (TP+TN)/Total = (106+61)/171 is: 97.66 %
```

Out[ ]: <Axes: >



Task 4: Univariate Feature Selection and XGBoost

In [ ]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2 #for the chi-squared test
```

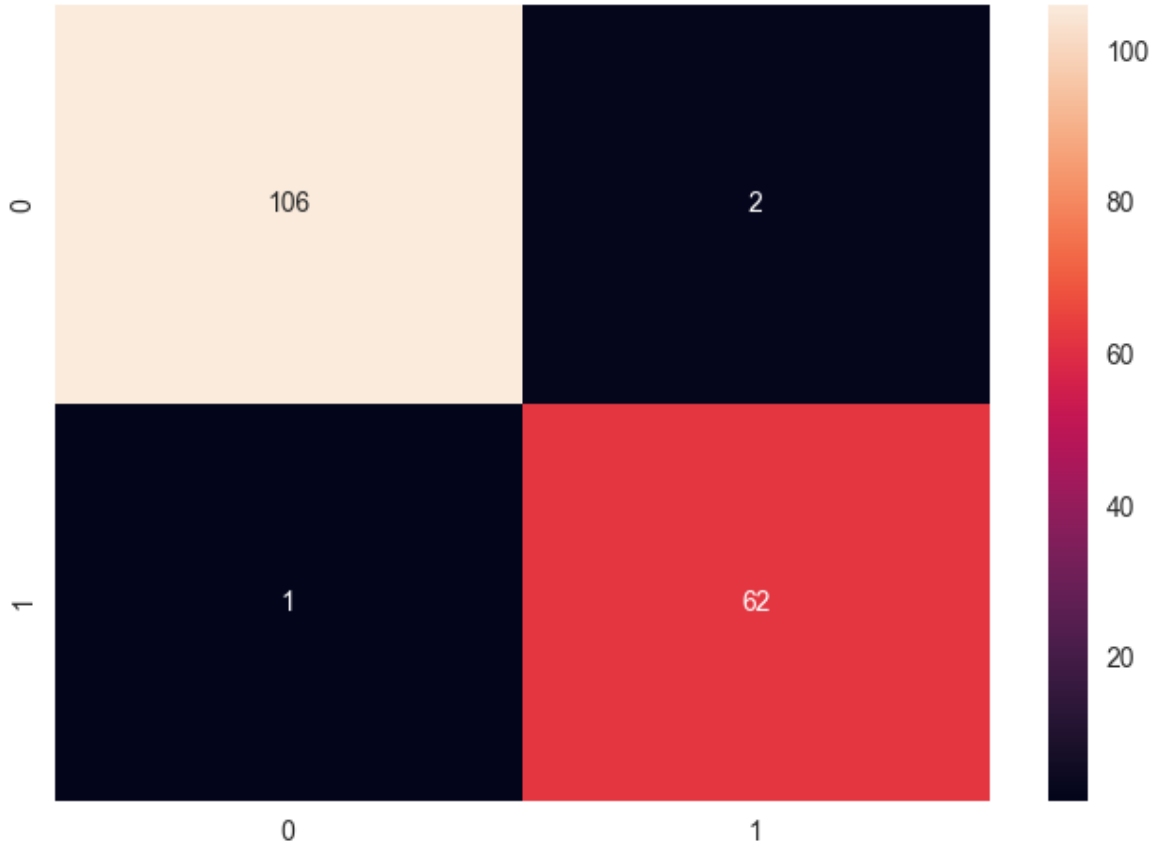
In [ ]:
```
# Create the SelectKBest object with chi-squared test and k=10
select_feature = SelectKBest(chi2, k=10).fit(x_train, y_train)

# Retrieve the scores and feature names
feature_scores = select_feature.scores_
selected_features = x_train.columns

# Print the scores and corresponding feature names
print('Score List:', feature_scores)
print('Feature List:', selected_features)
```

```
Score List: [6.06916433e+01 3.66899557e+04 1.00015175e-01 1.30547650e+01
 1.95982847e-01 3.42575072e-04 4.07131026e-02 6.12741067e+03
 1.32470372e-03 6.92896719e-01 1.39557806e-03 2.65927071e-03
 2.63226314e-01 2.58858117e+01 1.00635138e+00 1.23087347e-01]
Feature List: Index(['texture_mean', 'area_mean', 'smoothness_mean', 'concavity_m
ean',
       'symmetry_mean', 'fractal_dimension_mean', 'texture_se', 'area_se',
       'smoothness_se', 'concavity_se', 'symmetry_se', 'fractal_dimension_se',
       'smoothness_worst', 'concavity_worst', 'symmetry_worst',
       'fractal_dimension_worst'],
      dtype='object')
```

In [ ]:
```
# Transform the original training and testing features using the selected featur
x_train_2 = select_feature.transform(x_train)
x_test_2 = select_feature.transform(x_test)

# Create a new XGBoost classifier (clf_2) and train it on the transformed traini
clf_2 = xgb.XGBClassifier().fit(x_train_2, y_train)

# Calculate and print accuracy
accuracy = accuracy_score(y_test, clf_2.predict(x_test_2))
print('Accuracy of the new XGBoost classifier (clf_2) is:', accuracy)

# Calculate confusion matrix
cn_2 = confusion_matrix(y_test, clf_2.predict(x_test_2))

# Create a heatmap of the confusion matrix
sns.heatmap(cn_2, annot=True, fmt='d')
```

```
Accuracy of the new XGBoost classifier (clf_2) is: 0.9824561403508771
```

Out[ ]: <Axes: >



Task 5: Recursive Feature Elimination with Cross-Validation

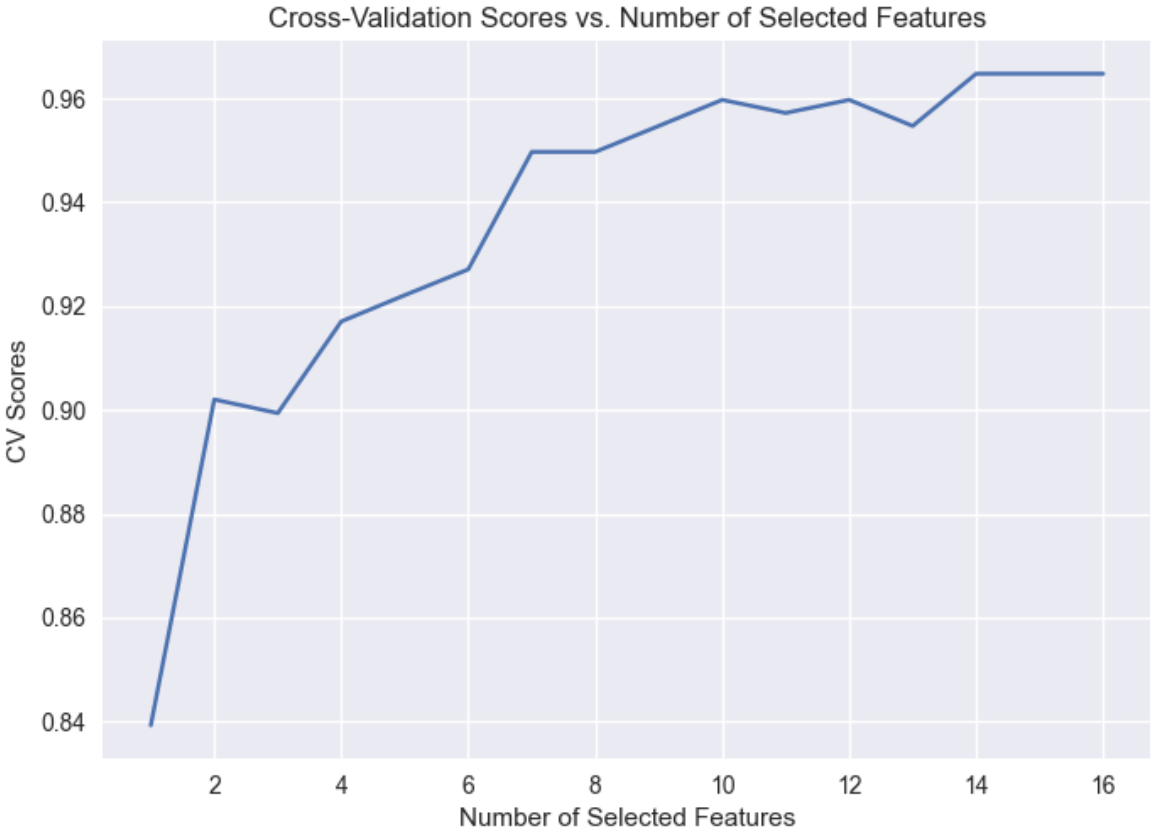
In [ ]:
```
# using Recursive Feature Elimination with Cross-Validation (RFECV)
# to determine the optimal number of features
# and identify the best features for your XGBoost classifier.
# Create an XGBoost classifier
clf_3 = xgb.XGBClassifier()

# Create the RFECV object
rfecv = RFECV(estimator=clf_3, step=1, cv=5, scoring='accuracy', n_jobs=-1).fit(
# estimator: The classifier (clf_3) to use during feature selection.
# step: The number of features to remove in each iteration (1 in this case).
# cv: The number of cross-validation folds (5 in this case).
# scoring: The scoring metric to evaluate feature subsets (accuracy in this case
# n_jobs: Number of CPU cores to use for parallel computation (-1 indicates usin


# Print the results
print('Optimal number of features:', rfecv.n_features_)
print('Best features:', x_train.columns[rfecv.support_])
```

```
Optimal number of features: 14
Best features: Index(['texture_mean', 'area_mean', 'smoothness_mean', 'concavity_
mean',
       'symmetry_mean', 'texture_se', 'area_se', 'smoothness_se',
       'concavity_se', 'symmetry_se', 'fractal_dimension_se',
       'smoothness_worst', 'concavity_worst', 'symmetry_worst'],
      dtype='object')
```

In [ ]:
```
print('Accuracy is: ', accuracy_score(y_test, rfecv.predict(x_test)))
```

```
Accuracy is:  0.9824561403508771
```

In [ ]:
```
# Extract the cross-validation results
cv_results = rfecv.cv_results_  #using the cv_results_ attribute from the RFECV

num_features = list(range(1, len(cv_results['mean_test_score']) + 1))
cv_scores = cv_results['mean_test_score']


# Create a line plot using Seaborn
ax = sns.lineplot(x=num_features, y=cv_scores)

# Set plot labels and titles
ax.set(xlabel='Number of Selected Features', ylabel='CV Scores')
plt.title('Cross-Validation Scores vs. Number of Selected Features')

# Display the plot
plt.show()
```



Task 6: Feature Extraction using Principal Component Analysis

In [ ]:
```
from sklearn.decomposition import PCA

# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_

# Normalize the training and testing data
x_train_norm = (x_train - x_train.mean()) / (x_train.max() - x_train.min())
x_test_norm = (x_test - x_test.mean()) / (x_test.max() - x_test.min())

# Create a PCA object
pca = PCA()
```

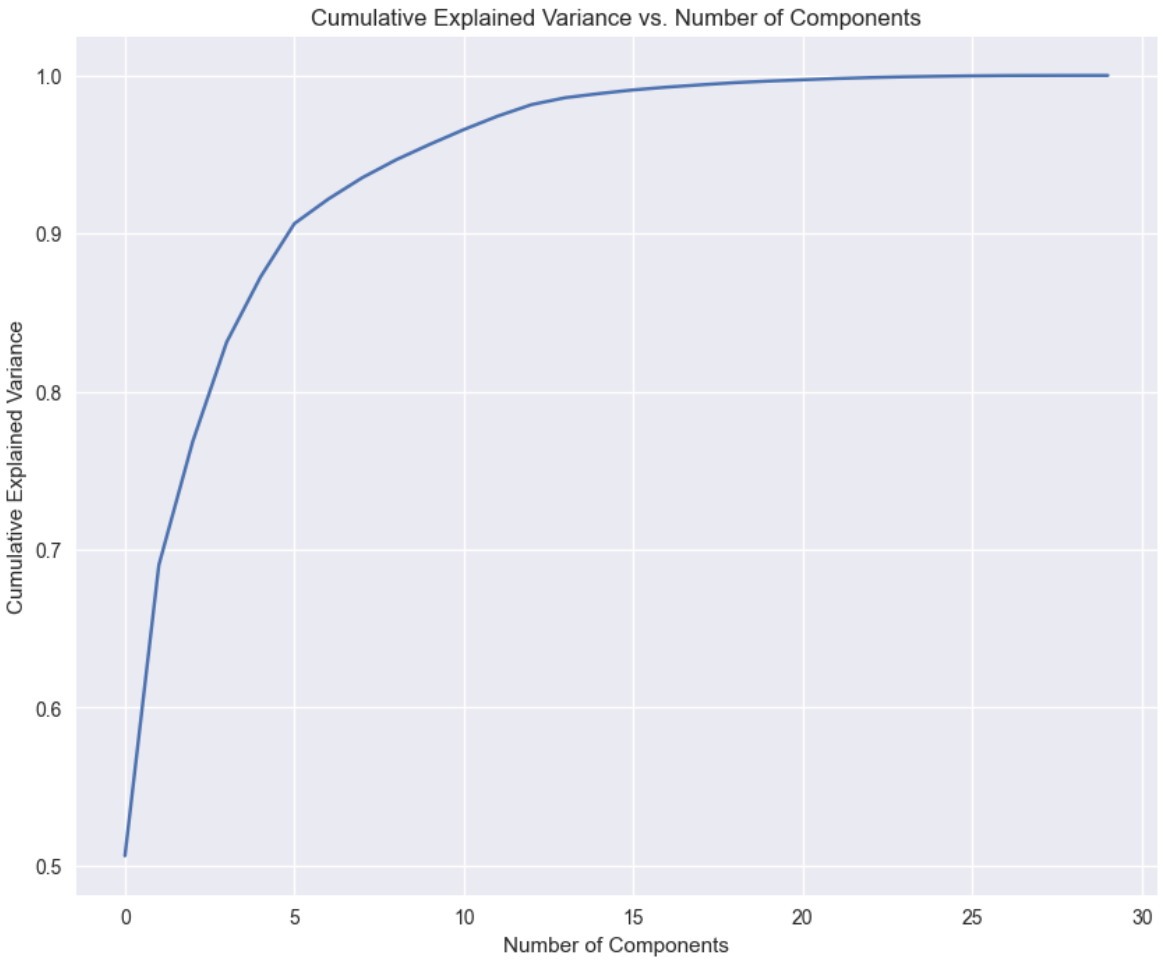
```
# Fit PCA on the normalized training data
pca.fit(x_train_norm)
```

Out[ ]:

```
PCA()
```

In [ ]:

```
# Calculate the cumulative explained variance ratio
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)

# Create a line plot using Seaborn
plt.figure(figsize=(10, 8))
sns.lineplot(data=cumulative_variance_ratio)

# Set plot labels and titles
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance vs. Number of Components')

# Display the plot
plt.show()
```



Many thanks to MargoSolo