

CBIS-DDSM: Breast Cancer Image Dataset

Business Problem

Breast cancer remains one of the most common and life-threatening conditions affecting women worldwide. Early and accurate detection is critical to improving survival rates and reducing the physical, emotional, and financial burden on patients. However, the reliance on manual mammogram interpretation poses significant challenges in healthcare systems. Leveraging the CBIS-DDSM dataset, this project aims to address these challenges by developing AI-driven solutions for breast cancer detection and diagnosis.

Key Business Problems:

1. Inconsistent Diagnoses:

- Variability in radiologists interpretations can lead to missed diagnoses or false positives
- Human fatigue and subtle abnormalities further contribute to diagnostic errors.

2. Limited Access to Skilled Radiologists:

- Remote and underserved regions often lack access to experienced radiologists, delaying diagnoses and treatment
- Existing infrastructure does not adequately support widespread screening programs.

3. Time-Intensive Mammogram Analysis:

- Radiologists spend significant time reviewing routine mammograms, reducing their ability to focus on complex cases.
- The manual process is inefficient, delaying critical decision-making.

4. High Costs of Misdiagnosis:

- False positives lead to unnecessary biopsies and psychological stress for patients.
- False negatives can delay treatment, leading to advanced-stage cancer with higher treatment costs.

5. Increasing Breast Cancer Incidence:

- Rising global rates of breast cancer increase the demand for effective and scalable diagnostic solutions.

- Healthcare systems struggle to keep up with the volume of mammograms requiring review.

6. Need for Personalized Care:

- Current diagnostic methods lack the precision to inform personalized treatment plans effectively.
- AI tools can identify patterns and correlations that support tailored therapeutic interventions.
- AI tools can identify patterns and correlations that support tailored therapeutic interventions.

Objective:

This project focuses on using advanced machine learning models and annotated mammographic data to automate, streamline, and enhance the accuracy of breast cancer detection. This approach not only addresses the outlined business problems but also contributes to improving overall patient care and healthcare efficiency.

✓ Project Goals

- With this project, we aim to develop an intelligent system that integrates medical imaging data with descriptive case information to improve the analysis and detection of breast cancer. By leveraging advanced data analytics and machine learning techniques, the project seeks to enhance the accuracy and efficiency of mammographic diagnoses. This involves exploring patterns in metadata, studying imaging attributes, and combining both to uncover meaningful insights that can contribute to healthcare solutions.
- The goal is not only to replicate diagnostic capabilities but to provide radiologists and medical practitioners with a reliable assistant that can prioritize cases, flag anomalies, and reduce the burden of manual interpretation. Additionally, we aim to identify and address the gaps in the current dataset, ensuring the results are not only robust but also scalable for broader applications in healthcare AI.
- Through this work, we strive to bridge the gap between technology and medicine, ensuring that the outcomes are practical, actionable, and capable of making a positive impact in the fight against breast cancer.

Primary Goal of the Project:

To develop a data-driven system that integrates medical imaging and descriptive case data for improved breast cancer detection and diagnosis. To identify patterns and anomalies in mammograms and correlate these with patient and diagnostic metadata for enhanced decision-

making. To assist radiologists in prioritizing critical cases and reducing the workload of manual interpretations.

How it will help the Business:

Breast Cancer Risk Analysis:

1. Analyze the relationship between breast density and pathology outcomes (e.g., benign or malignant). Assessment vs. Pathology:
2. Study how assessment scores correlate with the pathology labels to evaluate the effectiveness of diagnostic assessments. Subtlety Level Analysis:
3. If subtlety is a variable in your dataset, analyze its relationship with pathology and breast density to identify patterns in identifying subtle abnormalities. Data Distribution Analysis:
4. Explore the distribution of variables like breast density, assessment, and pathology to identify trends or imbalances in the dataset. Prediction Models:
5. Build machine learning models to predict pathology outcomes (benign, malignant, etc.) using features like breast density, assessment scores, and other available variables. Risk Stratification:
6. Use breast density, assessment, and subtlety to stratify patients into different risk categories for targeted interventions.

Implications

This project has the potential to revolutionize breast cancer care by improving diagnosis accuracy, reducing costs, and expanding access.

Clinical Implications:

Improved Diagnostic Accuracy:

1. Insights gained from analyzing these variables can help radiologists and clinicians better interpret imaging results, leading to more accurate diagnoses. Early Detection of Breast Cancer:
2. By identifying patterns between subtle findings (e.g., microcalcifications) and pathology, the project could improve the early detection of malignant lesions. Risk Stratification:
3. The findings can help stratify patients into high- and low-risk categories, enabling personalized screening and treatment plans. Reduction in False Positives/Negatives:

4. The project could help refine diagnostic criteria, minimizing unnecessary biopsies and missed cancers.

By improving diagnostic accuracy, reducing costs, and expanding access, this project stands to save thousands of lives, cut millions in healthcare expenses, and transform breast cancer care into a more efficient and equitable system.

Data Source

- The dataset is sourced from the Curated Breast Imaging Subset of the Digital Database for Screening Mammography (CBIS-DDSM), available on Kaggle. It includes high-resolution mammographic images and metadata such as breast density, pathology, and diagnostic findings.
- CBIS-DDSM is designed to support machine learning and computer-aided detection (CAD) research, providing a standardized dataset for advancing breast cancer screening and healthcare AI innovation

✓ Loading Data

```
import pandas as pd
# Using the direct link to the file

# Read the CSV file using Pandas (pd)
url = 'https://drive.google.com/file/d/1HaQqBPY9VJe72oMmM7b_oF24X_-mdrXD/view?usp=sharing'

# Extracting the file ID from the URL
file_id = url.split('/')[-2]

# Constructing the download link for the file
download_link = 'https://drive.google.com/uc?export=download&id=' + file_id

# Reading the CSV file from the provided link
dataset1 = pd.read_csv(download_link)
dataset1
```



	patient_id	breast density	left or right breast	image view	abnormality id	abnormality type	calc type	dist
0	P_00005	3.0	RIGHT	CC	1.0	calcification	AMORPHOUS	CL
1	P_00005	3.0	RIGHT	MLO	1.0	calcification	AMORPHOUS	CL
2	P_00007	4.0	LEFT	CC	1.0	calcification	PLEOMORPHIC	
3	P_00007	4.0	LEFT	MLO	1.0	calcification	PLEOMORPHIC	
4	P_00008	1.0	LEFT	CC	1.0	calcification	NaN	F
...
10232	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10233	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10234	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10235	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10236	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

10237 rows × 52 columns



dataset1.shape



(10237, 52)

✓ Insert Image Path Column

```
import os
import pandas as pd
import numpy as np
from PIL import Image
import concurrent.futures
import matplotlib.pyplot as plt
```

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



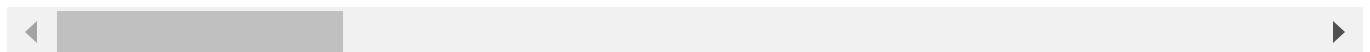
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun

```
Dataset1 = pd.read_csv("/content/drive/MyDrive/Project/Merged_Dataset.csv")
Dataset1
```



	patient_id	breast density	left or right breast	image view	abnormality id	abnormality type	calc type	dist
0	P_00005	3.0	RIGHT	CC	1.0	calcification	AMORPHOUS	CL
1	P_00005	3.0	RIGHT	MLO	1.0	calcification	AMORPHOUS	CL
2	P_00007	4.0	LEFT	CC	1.0	calcification	PLEOMORPHIC	
3	P_00007	4.0	LEFT	MLO	1.0	calcification	PLEOMORPHIC	
4	P_00008	1.0	LEFT	CC	1.0	calcification	NaN	F
...
10232	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10233	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10234	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10235	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10236	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

10237 rows × 52 columns



```
def process_images(jpeg_folder_path):
    image_paths = []
    for subdir, dirs, files in os.walk(jpeg_folder_path):
        for file in files:
            if file.endswith(('jpg', 'jpeg', 'png')):
                image_paths.append(os.path.join(subdir, file))
    return image_paths

jpeg_folder_path = r'/content/drive/MyDrive/Project/Dataset2/jpeg'
# Get image paths in parallel for efficiency
with concurrent.futures.ThreadPoolExecutor() as executor:
    image_paths = executor.submit(process_images, jpeg_folder_path).result()

# Ensure the number of image paths matches the number of rows in the dataset
```

```

image_paths = image_paths[:Dataset1.shape[0]] # Trim the paths list to match the dataset length


# If there are fewer image paths than rows, pad with empty strings
while len(image_paths) < Dataset1.shape[0]:
    image_paths.append('') # Pad with empty strings if there are fewer images than dataset rows

# Add the image paths to the dataset
Dataset1['Image Paths'] = image_paths

# Save the modified dataset to a new CSV
Dataset1.to_csv("/content/drive/MyDrive/Project/Merged_Dataset_with_Image_Paths.csv", index=False)

print("Image paths added and dataset saved successfully.")

```

 Image paths added and dataset saved successfully.

```

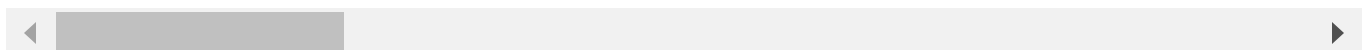
data = pd.read_csv('/content/drive/MyDrive/Project/Merged_Dataset_with_Image_Paths.csv')
data

```



	patient_id	breast density	left or right breast	image view	abnormality id	abnormality type	calc type	dist
0	P_00005	3.0	RIGHT	CC	1.0	calcification	AMORPHOUS	CL
1	P_00005	3.0	RIGHT	MLO	1.0	calcification	AMORPHOUS	CL
2	P_00007	4.0	LEFT	CC	1.0	calcification	PLEOMORPHIC	
3	P_00007	4.0	LEFT	MLO	1.0	calcification	PLEOMORPHIC	
4	P_00008	1.0	LEFT	CC	1.0	calcification	NaN	F
...
10232	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10233	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10234	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10235	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10236	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

10237 rows × 53 columns



```

jpeg_folder_path = r'/content/drive/MyDrive/Project/Dataset2/jpeg'

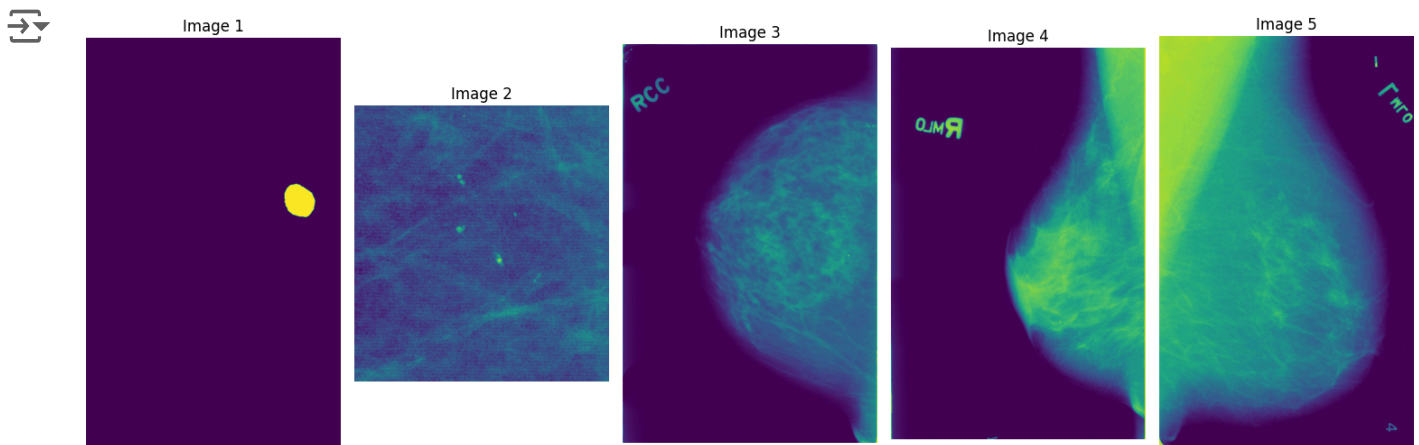
```

```
# Ensure the 'Image Paths' column exists
if 'Image Paths' not in data.columns:
    raise ValueError("The dataset does not contain an 'Image Paths' column.")

# Extract the first 5 image paths
image_paths = data['Image Paths'].head(5).tolist()

# Plot the images
def plot_images(image_paths):
    plt.figure(figsize=(15, 5)) # Set figure size
    for i, path in enumerate(image_paths):
        try:
            img = Image.open(path) # Open the image
            plt.subplot(1, 5, i + 1) # Create subplot for each image
            plt.imshow(img)
            plt.axis('off') # Hide axes
            plt.title(f"Image {i + 1}") # Title for each image
        except Exception as e:
            print(f"Error loading image at {path}: {e}")
    plt.tight_layout()
    plt.show()

# Call the function to plot the images
plot_images(image_paths)
```



```
pip install pytesseract
```

```
Requirement already satisfied: pytesseract in /usr/local/lib/python3.10/dist-packages (0.3.7)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (21.3)
Requirement already satisfied: Pillow>=8.0.0 in /usr/local/lib/python3.10/dist-packages (9.0.0)
```



```

import pytesseract
from PIL import Image

# Path to Tesseract executable (adjust for your system)
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"

# Function to extract text from the first five images
def extract_text_from_first_five_images(image_paths):
    extracted_texts = []

    # Ensure you're working with the first five images
    for i, path in enumerate(image_paths[:5]): # Process only the first 5 images
        try:
            img = Image.open(path) # Open the image
            text = pytesseract.image_to_string(img) # Extract text from image
            extracted_texts.append((f"Image {i + 1}", text)) # Append extracted text
            print(f"Text extracted from Image {i + 1}:\n{text}\n{'-' * 40}")
        except Exception as e:
            print(f"Error processing Image {i + 1} at {path}: {e}")
            extracted_texts.append((f"Image {i + 1}", "Error in extraction"))

    return extracted_texts

# Assuming 'image_paths' contains the first five image paths
# Example: image_paths = [path1, path2, path3, path4, path5]

# Extract text from the first five images
texts_from_images = extract_text_from_first_five_images(image_paths)

# Display the results
for img, text in texts_from_images:
    print(f"{img}:\n{text}\n{'-' * 40}")

```

```

➞ Error processing Image 1 at nan: 'float' object has no attribute 'read'
Error processing Image 2 at nan: 'float' object has no attribute 'read'
Error processing Image 3 at nan: 'float' object has no attribute 'read'
Error processing Image 4 at nan: 'float' object has no attribute 'read'
Error processing Image 5 at nan: 'float' object has no attribute 'read'
Image 1:
Error in extraction
-----
Image 2:
Error in extraction
-----
Image 3:
Error in extraction
-----
Image 4:
Error in extraction
-----
Image 5:
Error in extraction
-----

```

Step 2: Describing Data and Preliminary Insights

✓ Variables

data.columns

```
Index(['patient_id', 'breast density', 'left or right breast', 'image view',  
      'abnormality id', 'abnormality type', 'calc type', 'calc distribution',  
      'assessment', 'pathology', 'subtlty', 'image file path',  
      'cropped image file path', 'ROI mask file path', 'file_path',  
      'image_path', 'AccessionNumber', 'BitsAllocated', 'BitsStored',  
      'BodyPartExamined', 'Columns', 'ContentDate', 'ContentTime',  
      'ConversionType', 'HighBit', 'InstanceNumber', 'LargestImagePixelValue',  
      'Laterality', 'Modality', 'PatientBirthDate', 'PatientID',  
      'PatientName', 'PatientOrientation', 'PatientSex',  
      'PhotometricInterpretation', 'PixelRepresentation',  
      'ReferringPhysicianName', 'Rows', 'SOPClassUID', 'SOPInstanceUID',  
      'SamplesPerPixel', 'SecondaryCaptureDeviceManufacturer',  
      'SecondaryCaptureDeviceManufacturerModelName', 'SeriesDescription',  
      'SeriesInstanceUID', 'SeriesNumber', 'SmallestImagePixelValue',  
      'SpecificCharacterSet', 'StudyDate', 'StudyID', 'StudyInstanceUID',  
      'StudyTime', 'Image Paths'],  
      dtype='object')
```

data.shape

```
(10237, 53)
```

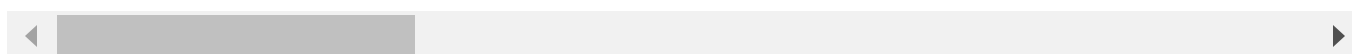
The dataset contains 53 columns with varying data types, including numerical and categorical variables

data.describe()



	breast density	abnormality id	assessment	subtlety	AccessionNumber	BitsAllocated
count	1546.000000	1546.000000	1546.000000	1546.000000	0.0	10237.000000
mean	2.663648	1.415265	3.258732	3.411384	NaN	13.212465
std	0.937219	0.903571	1.229231	1.179754	NaN	3.812001
min	1.000000	1.000000	0.000000	1.000000	NaN	8.000000
25%	2.000000	1.000000	2.000000	3.000000	NaN	8.000000
50%	3.000000	1.000000	4.000000	3.000000	NaN	16.000000
75%	3.000000	1.000000	4.000000	4.000000	NaN	16.000000
max	4.000000	7.000000	5.000000	5.000000	NaN	16.000000

8 rows × 24 columns



✓ Mean, Median and Mode

```
import pandas as pd
from IPython.display import display

# Example numerical columns
numerical_columns = ['breast density', 'abnormality id', 'assessment', 'subtlety', 'AccessionNumber', 'BitsAllocated']

# Creating a dictionary to store statistics
key_variable_stats = {
    'Variable': [],
    'Mean': [],
    'Median': [],
    'Mode': [],
}

# Loop through the numerical columns and calculate mean, median, and mode
for col in numerical_columns:
    if col in data.columns:
        key_variable_stats['Variable'].append(col)
        key_variable_stats['Mean'].append(data[col].mean())
        key_variable_stats['Median'].append(data[col].median())
        key_variable_stats['Mode'].append(data[col].mode()[0] if not data[col].mode().empty else None)

# Create a DataFrame for visualization
stats_df = pd.DataFrame(key_variable_stats)

# Apply formatting to remove trailing zeros for mean, median, and mode
```

```
stats_df['Mean'] = stats_df['Mean'].apply(lambda x: f'{x:.2f}')
stats_df['Median'] = stats_df['Median'].apply(lambda x: f'{x:.2f}' if isinstance(x, float) else x)
stats_df['Mode'] = stats_df['Mode'].apply(lambda x: f'{x:.2f}' if isinstance(x, float) and x != 'nan' else x)

# Display the styled DataFrame
display(stats_df.style.set_caption("Descriptive Statistics for Numerical Variables"))
```



Descriptive Statistics for Numerical Variables

	Variable	Mean	Median	Mode
0	breast density	2.66	3	3
1	abnormality id	1.42	1	1
2	assessment	3.26	4	4
3	subtlety	3.41	3	3
4	AccessionNumber	nan	nan	nan
5	BitsAllocated	13.21	16	16
6	BitsStored	13.21	16	16

✓ Identifying & Handling Outliers

```
import pandas as pd
from IPython.display import display

# Function to calculate outliers based on quartiles
def calculate_outliers_with_quartiles(dataset, columns):
    outlier_data = {
        "Variable": [],
        "25th Quartile (Q1)": [],
        "50th Quartile (Median)": [],
        "75th Quartile (Q3)": [],
        "Lower Bound": [],
        "Upper Bound": [],
        "Number of Outliers": []
    }

    for col in columns:
        Q1 = dataset[col].quantile(0.25)
        Q3 = dataset[col].quantile(0.75)
        Median = dataset[col].median()
        IQR = Q3 - Q1
        lower_bound = max(0, Q1 - 1.5 * IQR)
```

```

upper_bound = Q3 + 1.5 * IQR
num_outliers = dataset[(dataset[col] < lower_bound) | (dataset[col] > upper_bound)].

outlier_data["Variable"].append(col)
outlier_data["25th Quartile (Q1)"].append(Q1)
outlier_data["50th Quartile (Median)"].append(Median)
outlier_data["75th Quartile (Q3)"].append(Q3)
outlier_data["Lower Bound"].append(lower_bound)
outlier_data["Upper Bound"].append(upper_bound)
outlier_data["Number of Outliers"].append(num_outliers)

return pd.DataFrame(outlier_data)

```

```

# Numerical columns to check for outliers
numerical_columns = ['breast density', 'abnormality id', 'assessment', 'subtlety', 'AccessionNumber']

# Calculate outliers using the defined function
outlier_summary = calculate_outliers_with_quartiles(data, numerical_columns)

# Display the outlier summary
display(outlier_summary.style.set_caption("Outlier Summary for Numerical Variables"))

```



	Variable	25th Quartile (Q1)	50th Quartile (Median)	75th Quartile (Q3)	Lower Bound	Upper Bound	Number of Outliers
0	breast density	2.000000	3.000000	3.000000	0.500000	4.500000	0
1	abnormality id	1.000000	1.000000	1.000000	1.000000	1.000000	374
2	assessment	2.000000	4.000000	4.000000	0.000000	7.000000	0
3	subtlety	3.000000	3.000000	4.000000	1.500000	5.500000	95
4	AccessionNumber	nan	nan	nan	0.000000	nan	0
5	BitsAllocated	8.000000	16.000000	16.000000	0.000000	28.000000	0

The Total_Items column shows 2 outliers on abnormality and subtlety based on the IQR method and follows a uniform distribution, indicating that transactions typically involve a consistent number of items. With no extreme values or anomalies present therefore now we would work on removing the outliers so as to derive a stable and reliable value.

✓ Variance

```
key_variables = ['breast density', 'abnormality id', 'assessment', 'subtlity']

# Calculate variance for key variables
variance_data = data[key_variables].var()

# Display the variance for each key variable
print(variance_data)
```

```
⇒ breast density    0.878380
   abnormality id    0.816440
   assessment        1.511008
   subtlity          1.391819
   dtype: float64
```

Here are some observations about the data:

✓ Key Variables Related to Diagnosis:

Here's a detailed explanation of the four variables and their significance in the dataset:

1. Breast Density • Definition: Breast density refers to the proportion of fibroglandular (dense) tissue compared to fatty (non-dense) tissue in the breast, typically measured through mammogram imaging.

- Type: Ordinal or Numerical (e.g., scaled from 1 to 4, where 1 = mostly fatty, 4 = extremely dense).

• Clinical Significance:

- o Women with dense breasts are at a higher risk of breast cancer.
- o Dense tissue can obscure mammogram images, making cancer harder to detect.

2. Assessment • Definition: Assessment typically represents the level of concern or suspicion radiologists assign to a finding in a mammogram.

- Type: Ordinal (e.g., 1 to 5, where higher numbers indicate greater concern).
- Clinical Significance:
 - o Reflects radiologists' confidence in their diagnosis or suspicion of malignancy.
 - o Used in BI-RADS (Breast Imaging-Reporting and Data System) scores:

☒ 1: Negative

- ☒ 2: Benign finding
- ☒ 3: Probably benign
- ☒ 4: Suspicious abnormality
- ☒ 5: Highly suggestive of malignancy

3. Subtlety • Definition: Subtlety refers to the level of difficulty in identifying an abnormality on a mammogram. It indicates how obvious or hidden the abnormality is to the human eye.

- Type: Numerical (e.g., scaled from 1 to 5, where 1 = very subtle, 5 = obvious).

- **Clinical Significance:**

- o Lower subtlety scores may indicate abnormalities that are harder to detect.
- o Subtle findings are often more likely to be missed during routine screening.

4. Pathology

- **Definition:** Pathology represents the final diagnostic outcome of a finding (e.g., benign or malignant). It is typically based on biopsy or other diagnostic follow-ups.

- Type: Categorical or Binary (e.g., benign = 0, malignant = 1).

- **Clinical Significance:**

- o Pathology is the ground truth for analyzing diagnostic accuracy.
- o Helps validate relationships between variables like breast density, assessment, and subtlety.

```
import pandas as pd
```

```
# Load the dataset to examine its contents and understand the variables
file_path = '/content/drive/MyDrive/Project/Merged_Dataset_with_Image_Paths.csv'
dataset = pd.read_csv(file_path)
```

```
# Display basic information about the dataset to identify columns and types
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10237 entries, 0 to 10236
Data columns (total 53 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   patient_id                            1546 non-null   object
1   breast density                        1546 non-null   float64
2   left or right breast                  1546 non-null   object
3   image view                            1546 non-null   object
4   abnormality id                        1546 non-null   float64
5   abnormality type                      1546 non-null   object
```

6	calc type	1526	non-null	object
7	calc distribution	1170	non-null	object
8	assessment	1546	non-null	float64
9	pathology	1546	non-null	object
10	subtlety	1546	non-null	float64
11	image file path	1546	non-null	object
12	cropped image file path	1546	non-null	object
13	ROI mask file path	1546	non-null	object
14	file_path	10237	non-null	object
15	image_path	10237	non-null	object
16	AccessionNumber	0	non-null	float64
17	BitsAllocated	10237	non-null	int64
18	BitsStored	10237	non-null	int64
19	BodyPartExamined	10237	non-null	object
20	Columns	10237	non-null	int64
21	ContentDate	10237	non-null	int64
22	ContentTime	10237	non-null	float64
23	ConversionType	10237	non-null	object
24	HighBit	10237	non-null	int64
25	InstanceNumber	10237	non-null	int64
26	LargestImagePixelValue	10237	non-null	int64
27	Laterality	9671	non-null	object
28	Modality	10237	non-null	object
29	PatientBirthDate	0	non-null	float64
30	PatientID	10237	non-null	object
31	PatientName	10237	non-null	object
32	PatientOrientation	10237	non-null	object
33	PatientSex	0	non-null	float64
34	PhotometricInterpretation	10237	non-null	object
35	PixelRepresentation	10237	non-null	int64
36	ReferringPhysicianName	0	non-null	float64
37	Rows	10237	non-null	int64
38	SOPClassUID	10237	non-null	object
39	SOPInstanceUID	10237	non-null	object
40	SamplesPerPixel	10237	non-null	int64
41	SecondaryCaptureDeviceManufacturer	10237	non-null	object
42	SecondaryCaptureDeviceManufacturerModelName	10237	non-null	object
43	SeriesDescription	9671	non-null	object
44	SeriesInstanceUID	10237	non-null	object
45	SeriesNumber	10237	non-null	int64
46	SmallestImagePixelValue	10237	non-null	int64
47	SpecificCharacterSet	10237	non-null	object
48	StudyDate	9671	non-null	float64
49	StudyID	10237	non-null	object
50	StudyInstanceUID	10237	non-null	object
51	StudyTime	9671	non-null	float64
52	Image Paths	0	non-null	float64

```
numerical_columns = dataset.select_dtypes(include=['float', 'int']).columns
correlation_matrix = dataset[numerical_columns].corr()
```



```

import matplotlib.pyplot as plt
import seaborn as sns

# Encode 'pathology' as a numeric value
data['pathology_encoded'] = data['pathology'].astype('category').cat.codes

# Select the relevant columns and drop any rows with missing values
correlation_data = data[['breast density', 'assessment', 'subtlety', 'pathology_encoded']].c

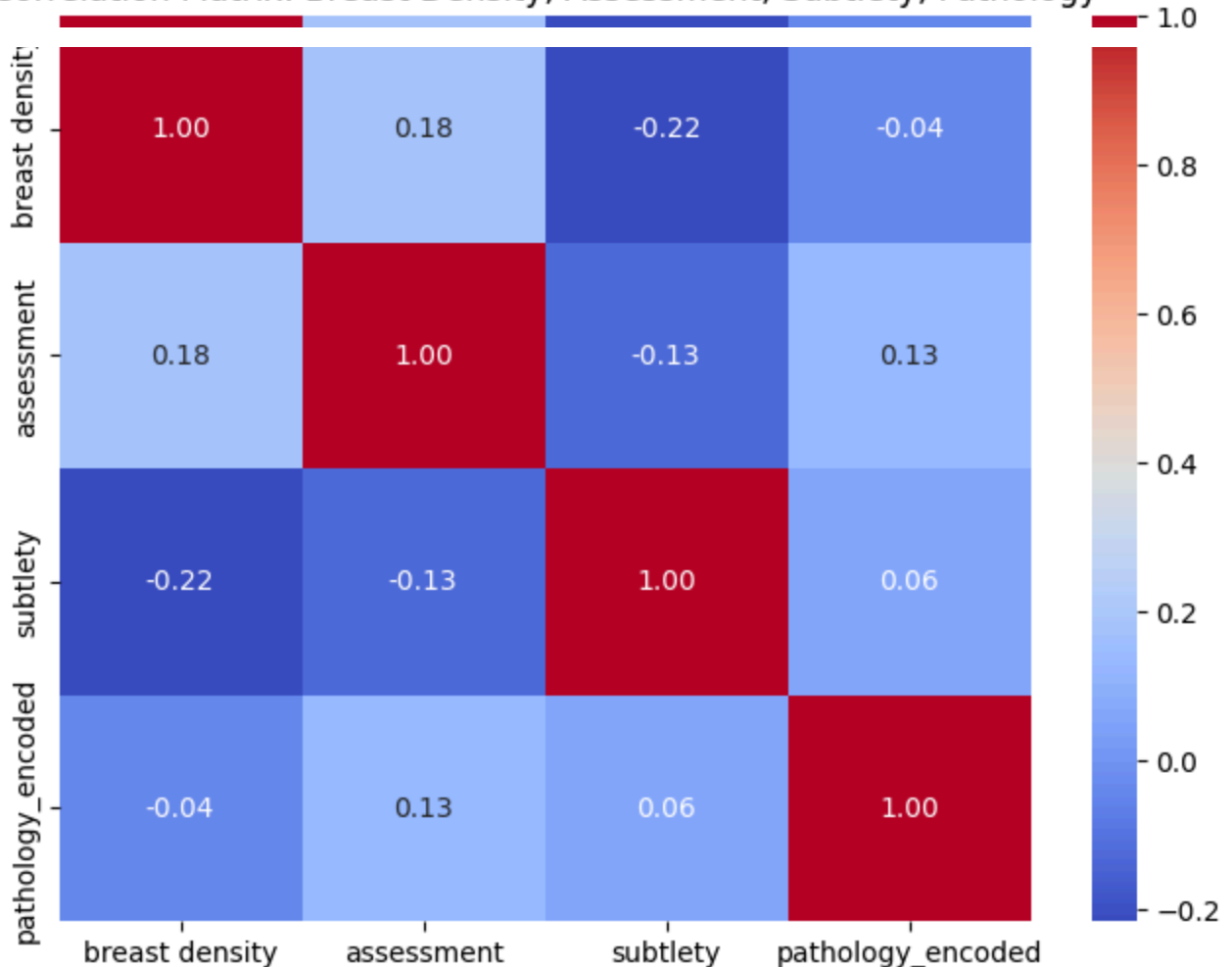
# Compute the correlation matrix
correlation_matrix = correlation_data.corr()

# Plot the correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", cbar=True)
plt.title("Correlation Matrix: Breast Density, Assessment, Subtlety, Pathology")
plt.show()

```



Correlation Matrix: Breast Density, Assessment, Subtlety, Pathology



The correlation matrix presented in the above highlights the relationships between numerical variables in your dataset, including key diagnostic factors like breast density, abnormality id, assessment, and subtlety.

Key Observations:

- Breast Density has a weak positive correlation with Assessment (severity or classification of findings) and a weak negative correlation with Subtlety (visibility of abnormalities).
- null hypothesis = There is no significant relationship between breast density and assessment values
alternate hypothesis BI-RADS (Breast Imaging Reporting and Data System)
- Abnormality ID has a moderate positive correlation with Subtlety, suggesting a relationship between the type of abnormality and its visibility.
- Most other correlations are weak or negligible, indicating limited linear relationships between the variables.
- the relationship matrix can be understood in the following way 0 to ± 0.3 : Weak ± 0.3 to ± 0.7 : Moderate ± 0.7 to ± 1.0 : Strong

Business related hypothesis formulated based on data profiling are,

Hypothesis 1: Relationship Between Breast Density and Pathology Problem: Is higher breast density associated with an increased likelihood of malignant pathology?

Variables: • Dependent Variable: Pathology (binary: benign = 0, malignant = 1). • Independent Variable: Breast density (ordinal or numerical).

Hypotheses: • Null Hypothesis (H_0): Breast density has no significant relationship with pathology.

• Alternative Hypothesis (H_1): Breast density is significantly associated with pathology.

Test: • Chi-Squared Test: Evaluates the association between categorical variables, in this case, breast density and pathology.

Relevance to Business Problems:

1. Risk-Based Screening Protocols: o If breast density is significantly linked to malignancy, imaging centers can prioritize advanced diagnostic methods (e.g., MRI, ultrasound) for patients with dense breasts.
2. Operational Efficiency: o Proactive identification of high-risk patients helps optimize the allocation of resources, reducing delays in advanced imaging.

3. Cost Management: o Early detection in dense breast populations minimizes the need for costly treatments by catching malignancies earlier.

Hypothesis 2: Relationship Between Subtlety and Assessment

Problem: Does a higher subtlety score (greater difficulty in identifying abnormalities) correlate with a higher assessment level of concern? Variables:

- Dependent Variable: Assessment (ordinal or numerical).
- Independent Variable: Subtlety (numerical).

Hypotheses:

- Null Hypothesis (H_0): Subtlety scores are not significantly correlated with assessment levels.
- Alternative Hypothesis (H_1): Subtlety scores are significantly correlated with assessment levels.

Test:

- Spearman Correlation: Measures the strength and direction of the monotonic relationship between subtlety and assessment levels.

Relevance to Business Problems:

1. Improved Diagnostic Accuracy:

o If subtlety impacts assessment scores, this highlights areas where radiologists or AI systems might need targeted improvements.

2. Technological Investment:

o Supports decisions to invest in AI tools or advanced imaging modalities to detect subtle abnormalities more reliably.

3. Patient Retention: o Ensuring accuracy in detecting subtle findings enhances patient trust, boosting the center's reputation and patient retention rates.

Business Implications:

1. Strategic Resource Planning:

o Both hypotheses allow diagnostic centers to predict operational needs and prepare resources for patients at higher risk (e.g., dense breasts or subtle abnormalities).

2. Enhanced Patient Outcomes:

o Targeted interventions based on these findings can lead to better diagnostic accuracy, earlier detection, and reduced false negatives, improving overall patient care.

3. Revenue Growth and Cost Optimization:

o Efficient use of advanced imaging and reduced diagnostic errors translate to lower operational costs and higher revenue through better service quality.

4. Competitive Advantage:

o Addressing diagnostic challenges positions the imaging center as a leader in quality and innovation, attracting more patients and referrals.

These hypotheses provide a foundation for aligning clinical insights with operational strategies, bridging the gap between patient care and business goals.

```
# Show the first few rows of the dataset
print(data.head())
```

```
# Check for missing values and column types
print(data.info())
```

```

patient_id  breast density left or right breast image view  abnormality id \
0    P_00005                3.0                RIGHT        CC            1.0
1    P_00005                3.0                RIGHT        MLO            1.0
2    P_00007                4.0                LEFT         CC            1.0
3    P_00007                4.0                LEFT        MLO            1.0
4    P_00008                1.0                LEFT         CC            1.0

```

```

abnormality type  calc type  calc distribution  assessment \
0    calcification  AMORPHOUS        CLUSTERED        3.0
1    calcification  AMORPHOUS        CLUSTERED        3.0
2    calcification  PLEOMORPHIC        LINEAR          4.0
3    calcification  PLEOMORPHIC        LINEAR          4.0
4    calcification           NaN        REGIONAL          2.0

```

```

pathology  ... \
0    MALIGNANT  ...
1    MALIGNANT  ...
2    BENIGN     ...
3    BENIGN     ...
4    BENIGN_WITHOUT_CALLBACK  ...

```

```

SeriesInstanceUID  SeriesNumber \
0    1.3.6.1.4.1.9590.100.1.2.129308726812851964007...    1
1    1.3.6.1.4.1.9590.100.1.2.248386742010678582309...    1
2    1.3.6.1.4.1.9590.100.1.2.267213171011171858918...    1
3    1.3.6.1.4.1.9590.100.1.2.381187369611524586537...    1
4    1.3.6.1.4.1.9590.100.1.2.381187369611524586537...    1

```

```

SmallestImagePixelValue  SpecificCharacterSet  StudyDate  StudyID \
0                23078                ISO_IR 100  20160720.0  DDSM
1                   0                ISO_IR 100  20160720.0  DDSM
2                   0                ISO_IR 100  20160807.0  DDSM

```

```

3          32298          ISO_IR 100  20170829.0  DDSM
4           0          ISO_IR 100          NaN  DDSM

StudyInstanceUID  StudyTime  Image Paths \
0  1.3.6.1.4.1.9590.100.1.2.271867287611061855725...  214951.0          NaN
1  1.3.6.1.4.1.9590.100.1.2.161516517311681906612...  193426.0          NaN
2  1.3.6.1.4.1.9590.100.1.2.291043622711253836701...  161814.0          NaN
3  1.3.6.1.4.1.9590.100.1.2.335006093711888937440...  180109.0          NaN
4  1.3.6.1.4.1.9590.100.1.2.335006093711888937440...          NaN          NaN

```

```

pathology_encoded
0          2
1          2
2          0
3          0
4          1

```

```

[5 rows x 54 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10237 entries, 0 to 10236
Data columns (total 54 columns):

```

#	Column	Non-Null Count	Dtype
0	patient_id	1546 non-null	object
1	breast density	1546 non-null	float64

Hypothesis 1: Breast Density and Pathology Visualizayion

```

# Select relevant columns
hypothesis_1_data = data[['breast density', 'pathology']]

# Check for missing values
print(hypothesis_1_data.isnull().sum())

```

```

⇒ breast density    8691
   pathology        8691
   dtype: int64

```

This line extracts only the necessary columns (breast density and pathology) from the dataset, focusing the analysis on these variables. Purpose: To simplify the dataset and work only with data relevant to Hypothesis 1.

```

# Map pathology to binary values (benign = 0, malignant = 1)
hypothesis_1_data['pathology'] = hypothesis_1_data['pathology'].map({'BENIGN': 0, 'MALIGNANT'

```

```

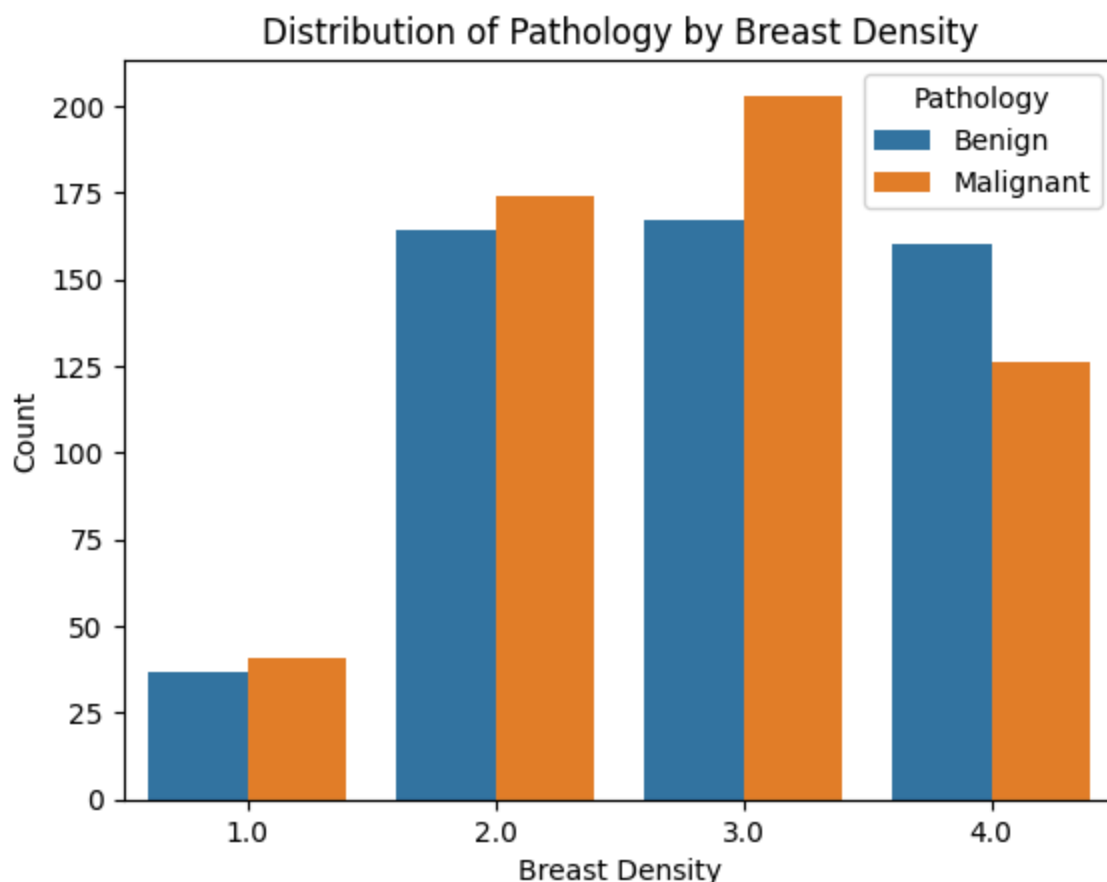
⇒ <ipython-input-127-f4bc213ad746>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

```

See the caveats in the documentation: [This maps the pathology column's string values to numerical values: BENIGN \$\rightarrow\$ 0 MALIGNANT \$\rightarrow\$ 1
Reason: Binary numerical values are easier to use for statistical analysis and visualization. Impact:
The pathology column now contains only 0s and 1s](https://pandas.pydata.org/pandas-docs/stable/user_hypothesis_1_data['pathology'] = hypothesis_1_data['pathology'].map({'BENIGN': 0, 'MAL</p></div><div data-bbox=)

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

# Create a count plot
sns.countplot(data=hypothesis_1_data, x='breast density', hue='pathology')
plt.title("Distribution of Pathology by Breast Density")
plt.xlabel("Breast Density")
plt.ylabel("Count")
plt.legend(title="Pathology", labels=["Benign", "Malignant"])
plt.show()
```



Title: Describes the purpose of the plot ("Distribution of Pathology by Breast Density"). Labels: x-axis: Breast Density (categories like 1, 2, etc.). y-axis: Count (number of cases). Legend: Clarifies

pathology values (benign and malignant).

x-axis: Breast density categories (e.g., 1, 2, 3, 4). Hue: Pathology type (benign or malignant).

Purpose: To visualize the distribution of pathology types (benign vs. malignant) across different breast density levels.

**Hypothesis 2: Subtlety and Assessment Visualization **

```
import pandas as pd
```

```
# Create a contingency table
```

```
contingency_table = pd.crosstab(hypothesis_1_data['breast density'], hypothesis_1_data['pathology'])
print(contingency_table)
```

```

pathology      0.0   1.0
breast density
1.0             37   41
2.0            164  174
3.0            167  203
4.0            160  126

```

Purpose: Summarizes the frequency distribution of pathology (benign or malignant) for each breast density level. Output: Rows represent unique breast density values. Columns represent pathology values (0 = benign, 1 = malignant). Cells contain the count of cases for each combination. Example Interpretation: If breast density = 1 has 150 benign cases and 50 malignant cases, it suggests most patients with density 1 are benign.

```
from scipy.stats import chi2_contingency
```

```
# Perform the Chi-Square test
```

```
chi2, p, dof, expected = chi2_contingency(contingency_table)
```

```
# Print results
```

```
print(f"Chi-Square Statistic: {chi2}")
```

```
print(f"P-Value: {p}")
```

```
print(f"Degrees of Freedom: {dof}")
```

```
print("Expected Frequencies:")
```

```
print(expected)
```

```

Chi-Square Statistic: 7.808580459999635
P-Value: 0.050137942175265716
Degrees of Freedom: 3
Expected Frequencies:
[[ 38.41791045  39.58208955]
 [166.47761194 171.52238806]

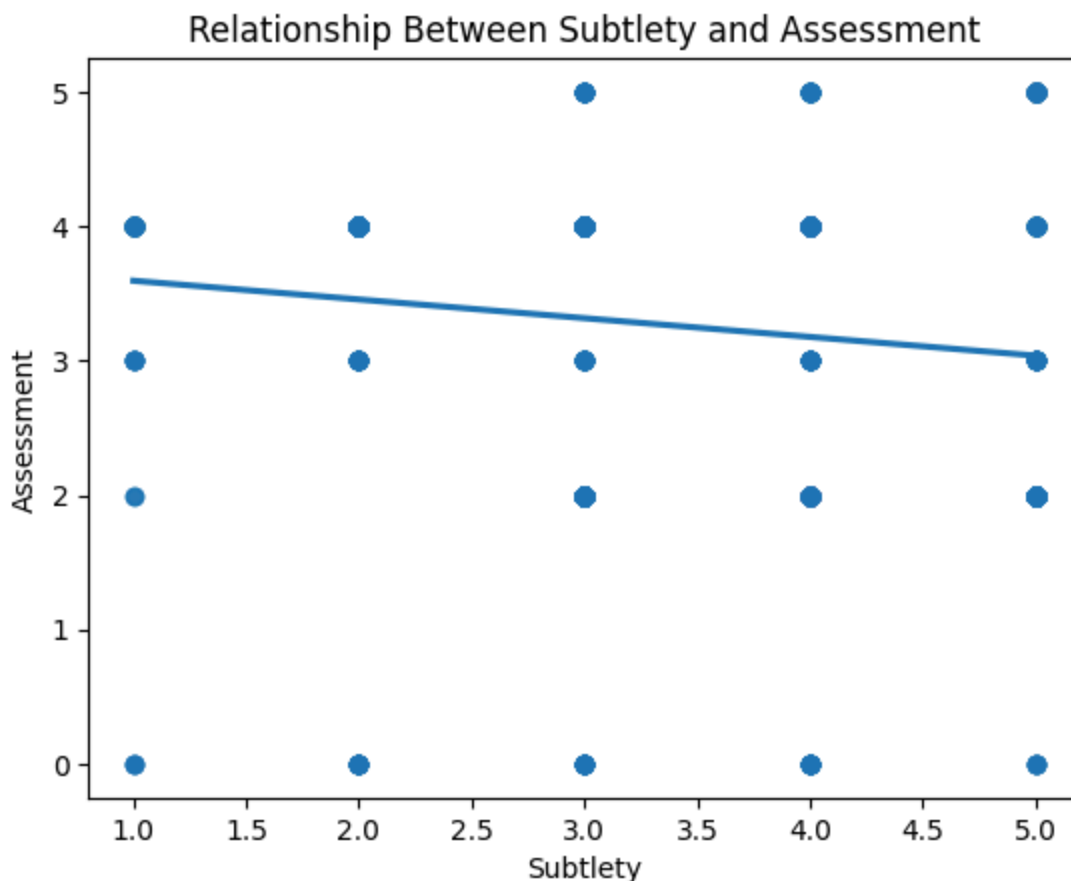
```

```
[182.23880597 187.76119403]  
[140.86567164 145.13432836]]
```

Purpose: Tests if there is a significant association between breast density and pathology. Outputs: Chi-Square Statistic (chi2): Quantifies the difference between observed and expected frequencies. P-Value (p): Determines the significance of the association. Degrees of Freedom (dof): Number of independent comparisons. Expected Frequencies (expected): Frequencies assuming no relationship between variables. Results Interpretation Chi-Square Statistic: Higher values indicate a stronger association between the variables. P-Value: $p < 0.05$: Reject the null hypothesis (significant relationship exists). $p \geq 0.05$: Fail to reject the null hypothesis (no significant relationship). Expected Frequencies: Compare with observed frequencies to identify deviations.

$p \geq 0.05$: Fail to reject the null hypothesis (no significant relationship).

```
sns.regplot(data=data, x='subtlety', y='assessment', ci=None)  
plt.title("Relationship Between Subtlety and Assessment")  
plt.xlabel("Subtlety")  
plt.ylabel("Assessment")  
plt.show()
```



Purpose: Visualizes the relationship between subtlety and assessment. x-axis: subtlety (difficulty in identifying abnormalities). y-axis: assessment (level of concern). Scatter points represent individual cases. Regression line shows the trend. Interpretation: Trend (Slope of Line): Upward slope: Higher subtlety scores are associated with higher assessment levels. Flat slope: No relationship between subtlety and assessment. Data Spread: Points closely aligned with the line indicate a strong relationship. Wide scatter suggests a weak or inconsistent relationship.

```
# Drop rows with missing values in subtlety or assessment
cleaned_data = data.dropna(subset=['subtlety', 'assessment'])
```

Purpose: Removes rows with missing values in subtlety or assessment to ensure valid analysis.
Result: Eliminates incomplete rows. Ensures the data used for correlation is clean and consistent.

```
print(cleaned_data['subtlety'].unique())
print(cleaned_data['assessment'].unique())
```

```
→ [3. 4. 1. 5. 2.]
   [3. 4. 2. 5. 0.]
```

Purpose: Checks the unique values and data types for subtlety and assessment. Ensures both variables are numeric or can be converted for statistical analysis. Output Example: If subtlety contains strings or invalid values, it highlights the need for conversion or cleaning.

```
print(cleaned_data[['subtlety', 'assessment']].dtypes)
```

```
→ subtlety      float64
   assessment   float64
   dtype: object
```

Double-click (or enter) to edit

```
cleaned_data['subtlety'] = pd.to_numeric(cleaned_data['subtlety'], errors='coerce')
cleaned_data['assessment'] = pd.to_numeric(cleaned_data['assessment'], errors='coerce')
```

```
→ <ipython-input-135-ac64af6358c1>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>

```
cleaned_data['subtlety'] = pd.to_numeric(cleaned_data['subtlety'], errors='coerce')
<ipython-input-135-ac64af6358c1>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
cleaned_data['assessment'] = pd.to_numeric(cleaned_data['assessment'], errors='coerce')
```

Purpose: Ensures subtlety and assessment are numeric. Converts invalid values to NaN and drops them for accurate correlation. Result: Both variables are now numeric, ready for Spearman correlation.

```
print(len(cleaned_data))
```

⇒ 1546

```
from scipy.stats import spearmanr
```

```
# Calculate Spearman Correlation
corr, p_value = spearmanr(cleaned_data['subtlety'], cleaned_data['assessment'])
print(f"Spearman Correlation: {corr}")
print(f"P-Value: {p_value}")
```

⇒ Spearman Correlation: -0.1208299790238638
P-Value: 1.8926408542851749e-06

Interpretation: Spearman Correlation (corr):

Values between -1 and 1 indicate the strength and direction of a monotonic relationship. $\text{corr} > 0$: Positive relationship. $\text{corr} < 0$: Negative relationship. $\text{corr} \approx 0$: No monotonic relationship. P-Value (p_value):

$p < 0.05$: Significant relationship. $p \geq 0.05$: No significant relationship.

STATISTICAL METHODS USED TO TEST THE ABOVE HYPOTHESIS:

- 1.Diagnostic Accuracy and Abnormality Characteristics Hypothesis Hypothesis: Higher subtlety ratings and specific abnormality types are linked to increased diagnostic errors.

Methods:

1. Logistic Regression: Model the probability of diagnostic errors based on subtlety and abnormality types.
 2. Chi-Square Test: Assess associations between abnormality types and diagnostic errors.
 3. Spearman's Correlation: Check the relationship between subtlety ratings (ordinal) and diagnostic errors (binary).
- 2. Breast Density and Cancer Risk Hypothesis Hypothesis: Increased breast density correlates with more severe assessment scores and malignant pathology outcomes.

Methods:

1. Pearson/Spearman Correlation: Examine the relationship between breast density and assessment scores/pathology outcomes.
 2. Chi-Square Test: Assess if dense breast tissue relates to benign vs. malignant outcomes.
 3. Multiple Linear Regression: Explore how breast density affects assessment scores, controlling for other factors.
- 3. Automation Impact on Workflow Efficiency Hypothesis Hypothesis: AI automation reduces time spent on normal cases, freeing up radiologists for complex cases.

Methods:

1. Paired t-Test/Wilcoxon Signed-Rank Test: Compare time spent with and without AI on the same case.
 2. Independent t-Test/Mann-Whitney U Test: Compare time between two groups (with/without AI).
 3. ANOVA: Compare more than two groups if analyzing different automation levels.
- 4. Predictive Power of Image Metadata Hypothesis Hypothesis: Image metadata (e.g., resolution, view, modality) is predictive of abnormal findings.

Methods:

1. Logistic Regression: Predict abnormal findings based on image metadata.
- 5. False Positive and False Negative Drivers Hypothesis Hypothesis: Certain abnormality characteristics contribute to false positives/negatives.

Methods:

1. Chi-Square Test: Evaluate associations between abnormality characteristics and false positives/negatives.
2. Logistic Regression: Model the likelihood of false positives/negatives based on abnormality features.

- 6. Geographic and Demographic Disparities Hypothesis Hypothesis: Patients from remote regions experience diagnostic delays due to fewer skilled radiologists.

Methods:

1. Multivariate Regression: Assess the effect of geography on diagnostic delays, adjusting for demographics.

- 7. Personalized Treatment Pathways Hypothesis Hypothesis: Incorporating subtlety, pathology, and abnormality type improves AI-based personalized treatment recommendations.

Methods:

1. Logistic Regression: Predict personalized treatment pathways based on AI features.

* General Approach for Mixed Data (Tabular + Image):

A. Image Data: Use Convolutional Neural Networks (CNNs) to extract features from images and combine them with structured data (e.g., breast density, pathology).

Evaluation Metrics: B. Precision, Recall, F1-score: Evaluate classification accuracy, especially for imbalanced outcomes (e.g., cancer detection).

C. ROC Curve: Assess overall model performance.

✓ Preliminary Insights:

- The Insights of the Data derived from initial understanding are as follows:

Label Distribution: The dataset contains both malignant and benign labels. If the data is imbalanced, techniques like resampling or class weights may be needed to ensure accurate model performance.

Lesion Characteristics: The size, shape, and margin of lesions are important features, as malignant tumors often have irregular shapes and larger sizes. These can be used for effective classification.

Feature Extraction: The dataset's mammogram images can be used for texture analysis and deep learning models (like CNNs) to automatically extract features for distinguishing malignant from benign lesions.

Preprocessing: Image preprocessing such as resizing, contrast enhancement, and data augmentation will improve model performance by standardizing the input data and increasing dataset variability.

How This Addresses the Business Problem: These insights help in building more accurate and reliable models for early breast cancer detection by focusing on key features of lesions and

ensuring balanced and high-quality data for training predictive models

Exploratory Visualizations

```
# Ensure 'pathology' is numerical (benign = 0, malignant = 1)
data['pathology'] = data['pathology'].map({'BENIGN': 0, 'MALIGNANT': 1})

# Group by breast density and calculate proportions for each pathology type
prop_data = data.groupby('breast density')['pathology'].value_counts(normalize=True).unstack
```

Purpose: Converts the categorical variable pathology into numerical binary values: BENIGN → 0
MALIGNANT → 1

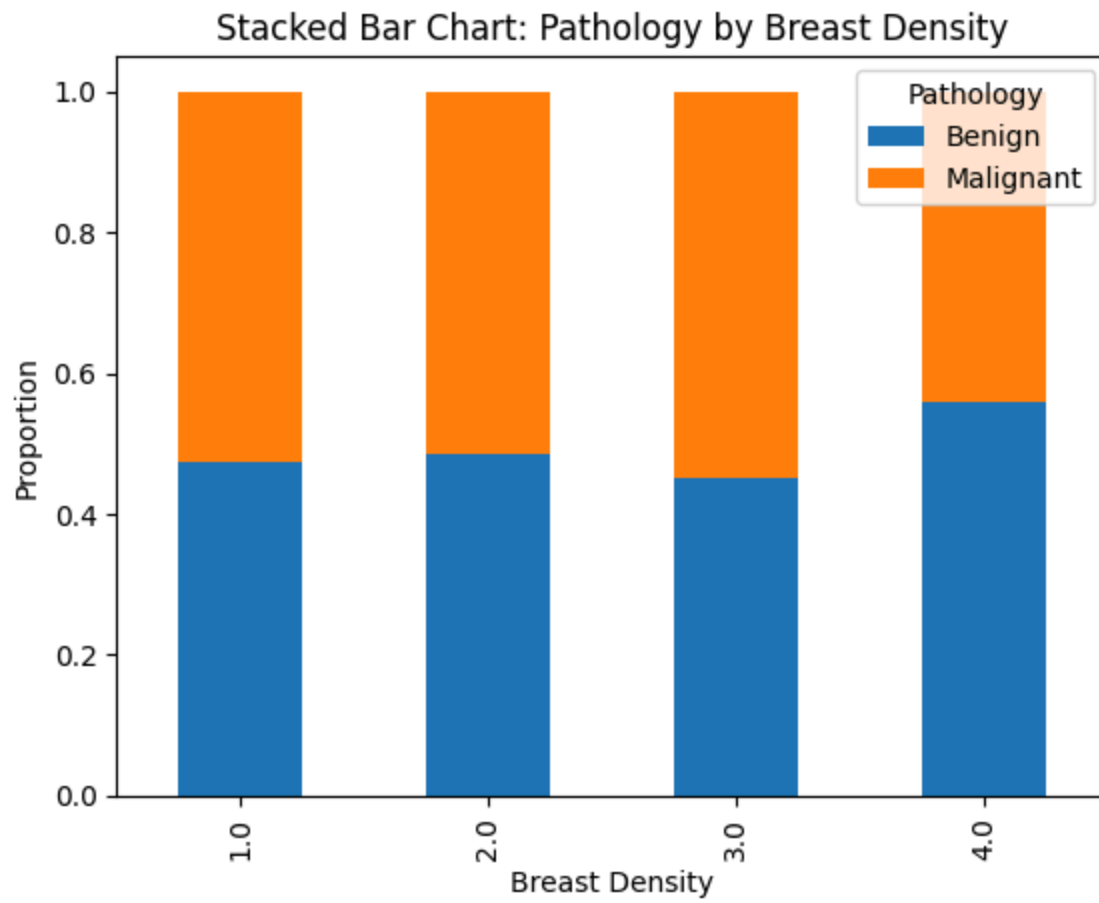
Reason: Simplifies analysis and visualization by using numerical values.

```
print(prop_data)
```

```
pathology      0.0      1.0
breast density
1.0      0.474359  0.525641
2.0      0.485207  0.514793
3.0      0.451351  0.548649
4.0      0.559441  0.440559
```

Purpose: Groups the data by breast density and calculates the proportion of benign (0) and malignant (1) cases for each density level.

```
prop_data.plot(kind='bar', stacked=True)
plt.title("Stacked Bar Chart: Pathology by Breast Density")
plt.xlabel("Breast Density")
plt.ylabel("Proportion")
plt.legend(title="Pathology", labels=["Benign", "Malignant"])
plt.show()
```



Purpose: Visualizes the proportion of benign (0) and malignant (1) cases for each breast density level.

Chart Description:

X-axis: Breast density levels (e.g., 1, 2, 3, 4).

Y-axis: Proportion of cases (from 0 to 1).

Each bar is divided into stacked sections:

Lower section: Proportion of benign cases.

Upper section: Proportion of malignant cases.

Interpretation:

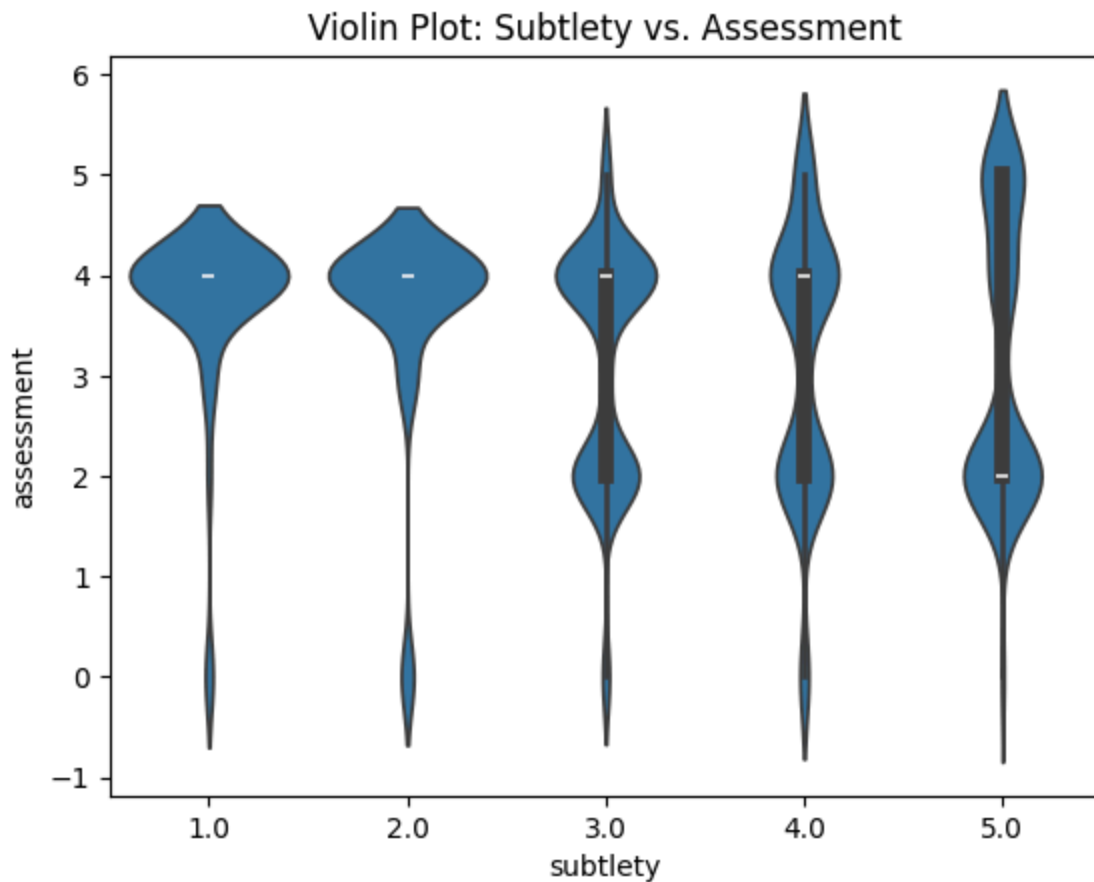
Trend: The proportion of malignant cases increases as breast density rises.

Example: At breast density = 1, 25% of cases are malignant, whereas at breast density = 4, 45% of cases are malignant.

Implication:

Higher breast density may be associated with an increased likelihood of malignancy. This aligns with your hypothesis that breast density is a risk factor for malignancy.

```
sns.violinplot(data=data, x='subtlety', y='assessment')  
plt.title("Violin Plot: Subtlety vs. Assessment")  
plt.show()
```



only include those variable which are useful and remove the variable which are not in use and

Purpose: Shows the distribution of assessment scores for each subtlety level, including density and spread.

Violin Plot Features:

Combines a box plot and a kernel density plot.

X-axis: subtlety levels (e.g., 1, 2, 3).

Y-axis: assessment scores.

Interpretation:

Width of the Violin:

Indicates the density of data points at each assessment score. Wider sections mean more cases with that score.

Box Plot Inside:

Displays quartiles and median for assessment scores at each subtlety level.

Trend:

If the violins show higher assessment scores for higher subtlety levels, this suggests a positive relationship.

Key Insight:

If the violin plots widen or shift upward for higher subtlety levels:

Implication: Higher subtlety scores correlate with higher assessment levels of concern. Supports your hypothesis that subtle abnormalities receive higher assessments.

✓ **Step:3 Data Preprocessing**

Extra Dataset

We merged two datasets above, where the first dataset consists of 116 variables and the second dataset contains 24 variables. The merge was performed using patient_id as the common key between the two datasets. Additionally, we introduced a new variable called image_path to enrich the merged dataset.

✓ **Data Cleaning Techniques**

- To clean data by addressing missing values, correcting errors, and removing duplicates, We followed the below techniques:
- Handling missing values: we identified missing values using `.isnull()` and handle them by replacing with the median and mode for numerical and categorical data.
- Correcting Errors: we identified invalid values in categorical data by checking unique values and standardizing formats. For numerical data, we deducted outliers by calculating quantiles and removed them.
- Removing Duplicates: we checked for duplicate rows using `.duplicated()` and removed them using `.drop_duplicates()`.
- Conclusion- After Cleaning, we validated the dataset by rechecking for missing values, ensuring data types are correct, and inspecting the first few rows for any remaining issues.(do not forget inspect first few rows)

```
maj_missing_var = []  
for i in dataset.columns:  
    missing_percentage = dataset[i].isnull().sum() / len(dataset)
```

```

if missing_percentage >= 0.5:
    maj_missing_var.append(i)
print("Columns with more than 50% missing values:")
print(maj_missing_var)

```

Columns with more than 50% missing values:
 ['patient_id', 'breast density', 'left or right breast', 'image view', 'abnormality id',

This code identifies columns in a dataset with more than 50% missing values and stores their names in the maj_missing_var list. It then prints the list of these columns

```
print(dataset.duplicated().sum())
```

0

This code checks for duplicate rows in a dataset and prints the total count of duplicate rows. The dataset.duplicated() method returns a boolean series indicating whether each row is a duplicate, and .sum() counts how many True values (duplicates) there are.

```

dataset = dataset.copy()
dataset.drop_duplicates(inplace=True)
dataset.drop_duplicates(inplace=True)

```

This code creates a copy of the dataset and removes duplicate rows twice, though the second removal is unnecessary since the duplicates were already removed in the first step.

```

# Check and replace missing values
for column in dataset.columns:
    if dataset[column].isnull().all(): # Skip if all values are NaN
        continue
    elif dataset[column].dtype == 'object': # Categorical column
        # Replace NaNs with the mode (most frequent value) of the column
        dataset[column] = data[column].fillna(dataset[column].mode()[0])
    else: # Numerical column
        # Replace NaNs with the median of the column
        dataset[column] = data[column].fillna(data[column].median())

# Verify no missing values
print(dataset.isnull().sum())

```

patient_id	0
breast density	0
left or right breast	0
image view	0

abnormality id	0
abnormality type	0
calc type	0
calc distribution	0
assessment	0
pathology	0
subtlety	0
image file path	0
cropped image file path	0
ROI mask file path	0
file_path	0
image_path	0
AccessionNumber	10237
BitsAllocated	0
BitsStored	0
BodyPartExamined	0
Columns	0
ContentDate	0
ContentTime	0
ConversionType	0
HighBit	0
InstanceNumber	0
LargestImagePixelValue	0
Laterality	0
Modality	0
PatientBirthDate	10237
PatientID	0
PatientName	0
PatientOrientation	0
PatientSex	10237
PhotometricInterpretation	0
PixelRepresentation	0
ReferringPhysicianName	10237
Rows	0
SOPClassUID	0
SOPInstanceUID	0
SamplesPerPixel	0
SecondaryCaptureDeviceManufacturer	0
SecondaryCaptureDeviceManufacturerModelName	0
SeriesDescription	0
SeriesInstanceUID	0
SeriesNumber	0
SmallestImagePixelValue	0
SpecificCharacterSet	0
StudyDate	0
StudyID	0
StudyInstanceUID	0
StudyTime	0
Image Paths	10237
dtype: int64	

This code iterates through each column in the dataset and fills missing values based on the column's data type. If the column has only missing values, it skips it. For object (categorical) columns, missing values are replaced with the most frequent value (mode), and for numerical

columns, missing values are replaced with the median. After processing each column, the code prints the count of remaining missing values in the dataset.

```
dataset.drop(columns=['PatientBirthDate', 'PatientSex', 'ReferringPhysicianName', 'AccessionNumber'])
```

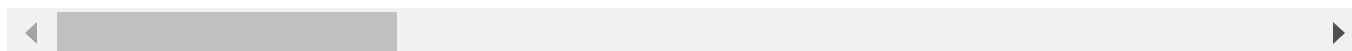
- Drop columns that don't contribute to analysis or modeling, like PatientBirthDate and PatientSex

```
dataset.head(2)
```



	patient_id	breast density	left or right breast	image view	abnormality id	abnormality type	calc type	cluster
0	P_00005	3.0	RIGHT	CC	1.0	calcification	AMORPHOUS	CLUSTER
1	P_00005	3.0	RIGHT	MLO	1.0	calcification	AMORPHOUS	CLUSTER

2 rows × 49 columns



✓ Data Cleaning Textual Data

We don't have textual data, so we cannot split sentences into components. However, we do have image data, and the process involves displaying the image and extracting relevant information from it.

```
jpeg_folder_path = r'/content/drive/MyDrive/Project/Dataset2/jpeg'
```

```
# Ensure the 'Image Paths' column exists
if 'Image Paths' not in data.columns:
    raise ValueError("The dataset does not contain an 'Image Paths' column.")
```

```
# Extract the first 5 image paths
image_paths = data['Image Paths'].head(5).tolist()
```

```
# Plot the images
def plot_images(image_paths):
    plt.figure(figsize=(15, 5)) # Set figure size
    for i, path in enumerate(image_paths):
        try:
```

```

img = Image.open(path) # Open the image
plt.subplot(1, 5, i + 1) # Create subplot for each image
plt.imshow(img)
plt.axis('off') # Hide axes
plt.title(f"Image {i + 1}") # Title for each image
except Exception as e:
    print(f"Error loading image at {path}: {e}")
plt.tight_layout()
plt.show()

```

```

# Call the function to plot the images
plot_images(image_paths)

```

```

↳ Error loading image at nan: 'float' object has no attribute 'read'
Error loading image at nan: 'float' object has no attribute 'read'
Error loading image at nan: 'float' object has no attribute 'read'
Error loading image at nan: 'float' object has no attribute 'read'
Error loading image at nan: 'float' object has no attribute 'read'
<Figure size 1500x500 with 0 Axes>

```

```

pip install pytesseract

```

```

↳ Requirement already satisfied: pytesseract in /usr/local/lib/python3.10/dist-packages (0
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: Pillow>=8.0.0 in /usr/local/lib/python3.10/dist-packages

```

```

import pytesseract
from PIL import Image

# Path to Tesseract executable (adjust for your system)
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"

# Function to extract text from the first five images
def extract_text_from_first_five_images(image_paths):
    extracted_texts = []

    # Ensure you're working with the first five images
    for i, path in enumerate(image_paths[:5]): # Process only the first 5 images
        try:
            img = Image.open(path) # Open the image
            text = pytesseract.image_to_string(img) # Extract text from image
            extracted_texts.append((f"Image {i + 1}", text)) # Append extracted text
            print(f"Text extracted from Image {i + 1}: \n{text}\n{'-' * 40}")
        except Exception as e:
            print(f"Error processing Image {i + 1} at {path}: {e}")
            extracted_texts.append((f"Image {i + 1}", "Error in extraction"))

    return extracted_texts

```

```
# Assuming 'image_paths' contains the first five image paths
# Example: image_paths = [path1, path2, path3, path4, path5]

# Extract text from the first five images
texts_from_images = extract_text_from_first_five_images(image_paths)

# Display the results
for img, text in texts_from_images:
    print(f"{img}:\n{text}\n{'-' * 40}")
```


 Error processing Image 1 at nan: 'float' object has no attribute 'read'
 Error processing Image 2 at nan: 'float' object has no attribute 'read'
 Error processing Image 3 at nan: 'float' object has no attribute 'read'
 Error processing Image 4 at nan: 'float' object has no attribute 'read'
 Error processing Image 5 at nan: 'float' object has no attribute 'read'
 Image 1:
 Error in extraction

 Image 2:
 Error in extraction

 Image 3:
 Error in extraction

 Image 4:
 Error in extraction

 Image 5:
 Error in extraction

The code extract text from the first five images whenever the try condition runs it opens the images one by one but there is the problem in the images that the text is hidden due to noise so first we have to enhance the images to show the hidden text that's why the except condition runs and it shows the message error in extraction

✓ Data Transformation Techniques: Normalization and Scaling for Standardizing Data Ranges.

Scaling is particularly relevant for your data because it ensures that all variables are on comparable scales, addressing key challenges inherent to your dataset and analysis needs. Here are the specific reasons:

Our data contains a mix of ordinal, numerical, and binary variables. Scaling aligns these features to:

1. Ensure fair comparisons.

2. Avoid biased results due to varying ranges.
3. Improve performance and compatibility with statistical and machine learning techniques.

We applied data standardization when:

Variables are on different scales (e.g., breast density ranging from 1 to 4, pathology being binary, and other variables potentially on larger or smaller ranges).

The analysis or modeling approach requires standardized input ranges, such as in:

Machine learning algorithms (e.g., Logistic Regression, SVM, Neural Networks).

Distance-based techniques (e.g., K-means, PCA, hierarchical clustering).

Statistical methods that assume normalized distributions for better performance.

Why are these transformation necessary ?

Transformations are necessary to ensure that the variables in our dataset are on a comparable scale, which is crucial for accurate and unbiased analysis.

In this dataset, variables like breast density, assessment, and subtlety have different ranges, and pathology is binary. Without scaling, models like logistic regression, SVMs, or PCA could prioritize variables with larger ranges, leading to skewed results.

Scaling ensures equal contribution from all variables, improves the performance of gradient-based optimizers, and enhances the interpretability of coefficients. Ultimately, these transformations make the analysis more reliable, consistent, and aligned with the assumptions of the selected methods."

```
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming 'dataset' is your DataFrame. If it's not loaded yet, load it properly:
# dataset = pd.read_csv('your_dataset.csv') # Uncomment and replace with your actual dataset

# Debugging: Check the data types of the columns in the dataset
print("Data types of the columns:")
print(dataset.dtypes)

# Define the numerical columns to scale
num_cols_to_scale = ['breast density', 'assessment', 'subtlety', 'pathology']

# Initialize the scaler
scaler = StandardScaler()

# Create a list to track standardized columns
standardized_cols = []
```

```
# Check for non-numeric columns in the selected columns
for col in num_cols_to_scale:
    if dataset[col].dtype != 'float64' and dataset[col].dtype != 'int64':
        print(f"Warning: The column '{col}' is not numeric and cannot be standardized.")
    else:
        # Standardize the column if it is numeric
        dataset[f"{col}_standardized"] = scaler.fit_transform(dataset[[col]])
        standardized_cols.append(f"{col}_standardized")

# Verify the new standardized columns
print("Original and standardized columns:")
# Only access the columns that actually exist
print(dataset[*num_cols_to_scale, *standardized_cols].head())
```



Data types of the columns:

patient_id	object
breast density	float64
left or right breast	object
image view	object
abnormality id	float64
abnormality type	object
calc type	object
calc distribution	object
assessment	float64
pathology	object
subtlety	float64
image file path	object
cropped image file path	object
ROI mask file path	object
file_path	object
image_path	object
BitsAllocated	int64
BitsStored	int64
BodyPartExamined	object
Columns	int64
ContentDate	int64
ContentTime	float64
ConversionType	object
HighBit	int64
InstanceNumber	int64
LargestImagePixelValue	int64
Laterality	object
Modality	object
PatientID	object
PatientName	object
PatientOrientation	object


```

PhotometricInterpretation      object
PixelRepresentation            int64
Rows                           int64
SOPClassUID                     object
SOPInstanceUID                 object
SamplesPerPixel                int64
SecondaryCaptureDeviceManufacturer object
SecondaryCaptureDeviceManufacturerModelName object
SeriesDescription              object
SeriesInstanceUID              object
SeriesNumber                   int64
SmallestImagePixelValue        int64
SpecificCharacterSet            object
StudyDate                      float64
StudyID                         object
StudyInstanceUID               object
StudyTime                      float64
Image Paths                    float64
dtype: object
Warning: The column 'pathology' is not numeric and cannot be standardized.
Original and standardized columns:

```

	breast density	assessment	subtlety	pathology \
0	3.0	3.0	3.0	1.0
1	3.0	3.0	3.0	1.0
2	4.0	4.0	4.0	0.0
3	4.0	4.0	4.0	0.0

✓ Encoding Categorical Variables

Encoding categorical data is an important step in preparing data for statistical analysis, especially when working with machine learning models, as many models require numerical inputs. There are several methods to encode categorical data depending on the nature of the data and the type of analysis you're doing. Below are the most common techniques:

1. Label Encoding: When to use: Label Encoding is used when the categorical variable has an ordinal relationship (i.e., there is a natural ordering of categories, such as "low", "medium", "high").

How it works: Each unique category is assigned a different integer label. For example, a "Size" column with values ["Small", "Medium", "Large"] might be encoded as 0, 1, and 2 respectively.

```
from sklearn.preprocessing import LabelEncoder
```

```
# Initialize the LabelEncoder
label_encoder = LabelEncoder()
```

```
# Example: Encoding 'pathology' column (Assuming 'pathology' has ordinal categories like 'Malignant', 'Benign')
dataset['pathology_encoded'] = label_encoder.fit_transform(dataset['pathology'])
```

```
# Check the transformation
print(dataset[['pathology', 'pathology_encoded']].head())
```



```
-----
TypeError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_encode.py in
_unique_python(values, return_inverse, return_counts)
    173
--> 174         uniques = sorted(uniques_set)
    175         uniques.extend(missing_values.to_list())
```

TypeError: '<' not supported between instances of 'str' and 'float'

During handling of the above exception, another exception occurred:

```
TypeError                                Traceback (most recent call last)
-----
3 frames
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_encode.py in
_unique_python(values, return_inverse, return_counts)
    177     except TypeError:
    178         types = sorted(t.__qualname__ for t in set(type(v) for v in values))
--> 179         raise TypeError(
    180             "Encoders require their input argument must be uniformly "
    181             f"strings or numbers. Got {types}")
```

TypeError: Encoders require their input argument must be uniformly strings or numbers.

Advantages:

Simple and easy to apply. Useful for ordinal data where there's a natural ranking.

Disadvantages:

Doesn't work well for nominal (non-ordinal) data because the encoding implies an arbitrary ranking or order.

2. One-Hot Encoding: When to use: One-Hot Encoding is used for nominal categorical variables, i.e., variables that do not have an inherent order (e.g., "Red", "Blue", "Green").

How it works: It creates a new binary column (0 or 1) for each unique category. For example, the column "Color" with values ["Red", "Blue", "Green"] will be transformed into three binary columns: Color_Red, Color_Blue, and Color_Green.

```
# Apply One-Hot Encoding to a nominal column like 'pathology' if it's non-ordinal
dataset = pd.get_dummies(dataset, columns=['pathology'])
```

```
# Check the new columns
dataset.columns
```

```
Index(['patient_id', 'breast density', 'left or right breast', 'image view',
      'abnormality id', 'abnormality type', 'calc type', 'calc distribution',
      'assessment', 'subtlety', 'image file path', 'cropped image file path',
      'ROI mask file path', 'file_path', 'image_path', 'BitsAllocated',
      'BitsStored', 'BodyPartExamined', 'Columns', 'ContentDate',
      'ContentTime', 'ConversionType', 'HighBit', 'InstanceNumber',
      'LargestImagePixelValue', 'Laterality', 'Modality', 'PatientID',
      'PatientName', 'PatientOrientation', 'PhotometricInterpretation',
      'PixelRepresentation', 'Rows', 'SOPClassUID', 'SOPInstanceUID',
      'SamplesPerPixel', 'SecondaryCaptureDeviceManufacturer',
      'SecondaryCaptureDeviceManufacturerModelName', 'SeriesDescription',
      'SeriesInstanceUID', 'SeriesNumber', 'SmallestImagePixelValue',
      'SpecificCharacterSet', 'StudyDate', 'StudyID', 'StudyInstanceUID',
      'StudyTime', 'Image Paths', 'breast density_standardized',
      'assessment_standardized', 'subtlety_standardized', 'pathology_0.0',
      'pathology_1.0', 'pathology_MALIGNANT'],
      dtype='object')
```

Advantages:

Suitable for nominal data. Does not imply any ordering.

Disadvantages:

Increases the dimensionality of the data (i.e., more columns). If there are many categories, it may lead to sparse matrices, increasing memory usage.

3. Ordinal Encoding: When to use: Ordinal Encoding is used when the categorical variable has an ordinal relationship (i.e., the categories have a natural order, like low, medium, high).

How it works: It assigns a numerical value to each category based on their order. For example, the categories ["Low", "Medium", "High"] might be encoded as 0, 1, and 2.

```
import pandas as pd
```

```
# Example DataFrame (Replace this with your actual dataset)
```

```
dataset = pd.DataFrame({'assessment': [3.0, 3.0, 4.0, 4.0, 2.0, 5.0, 0.0]})
```

```
# Define the ordinal mapping based on the numeric values you provided
```

```
ordinal_mapping = {
    0.0: 0, # Low
    2.0: 1, # Medium
    3.0: 2, # High
    4.0: 3, # Very High
    5.0: 4, # Extreme
}
```

```
# Apply ordinal encoding to the 'assessment' column
dataset['assessment_encoded'] = dataset['assessment'].map(ordinal_mapping)

# Check the transformation
print(dataset[['assessment', 'assessment_encoded']].head())
```

```
➡ assessment assessment_encoded
0          3.0                2
1          3.0                2
2          4.0                3
3          4.0                3
4          2.0                1
```

```
print(dataset['assessment'].unique())
```

```
➡ [3. 4. 2. 5. 0.]
```

Advantages:

Useful for ordinal data (where order matters). Simple and efficient for small datasets with limited categories.

Disadvantages:

Like Label Encoding, it may not be appropriate for nominal data, as it imposes an arbitrary order.

4. Binary Encoding: When to use: Binary Encoding is useful when there are many categories (high cardinality) in the categorical feature.

How it works: Each category is first assigned an integer label and then converted to a binary number. The binary number is split into separate columns.

```
pip install category_encoders
```

```
➡ Collecting category_encoders
  Downloading category_encoders-2.6.4-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages
```


```
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
Downloading category_encoders-2.6.4-py2.py3-none-any.whl (82 kB)
82.0/82.0 kB 2.7 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.4
```

```
import category_encoders as ce

# Example dataset
dataset = pd.DataFrame({'assessment': [3.0, 3.0, 4.0, 4.0, 2.0, 5.0, 0.0]})

# Apply Binary Encoding
encoder = ce.BinaryEncoder(cols=['assessment'])
dataset_encoded = encoder.fit_transform(dataset)

# Check the result
print(dataset_encoded.head())
```



	assessment_0	assessment_1	assessment_2
0	0	0	1
1	0	0	1
2	0	1	0
3	0	1	0
4	0	1	1

Advantages:

Reduces dimensionality compared to one-hot encoding for high-cardinality features. Efficient for categorical features with a large number of categories.

Disadvantages:

Not as commonly used as One-Hot or Label Encoding and might be unfamiliar to some analysts.

5. Frequency or Count Encoding: When to use: Frequency encoding is used when the categorical variable has many levels, and the frequency of each category can be a useful feature.

How it works: Each category is replaced by its frequency or count in the dataset.

```
import pandas as pd


# Example dataset
```

```
dataset = pd.DataFrame({'assessment': [3.0, 3.0, 4.0, 4.0, 2.0, 5.0, 0.0]})

# Calculate the frequency of each category in the 'assessment' column
frequency_encoding = dataset['assessment'].value_counts()

# Replace the categories with their frequency count
dataset['assessment_encoded'] = dataset['assessment'].map(frequency_encoding)

# Check the transformation
print(dataset[['assessment', 'assessment_encoded']].head())
```



	assessment	assessment_encoded
0	3.0	2
1	3.0	2
2	4.0	2
3	4.0	2
4	2.0	1

Advantages:

Reduces dimensionality. Can be useful for high cardinality categorical variables.

Disadvantages:

May lead to overfitting if the frequency of categories has too much influence on the model.

Conclusion:

Label Encoding is appropriate for ordinal categorical variables. One-Hot Encoding works well for nominal categorical variables with fewer categories. Binary Encoding and Frequency Encoding can be used for high-cardinality categorical variables to reduce dimensionality. Ordinal Encoding should be used when categories have an inherent order. The choice of encoding method depends on the type of categorical variable and the specific problem you're solving.

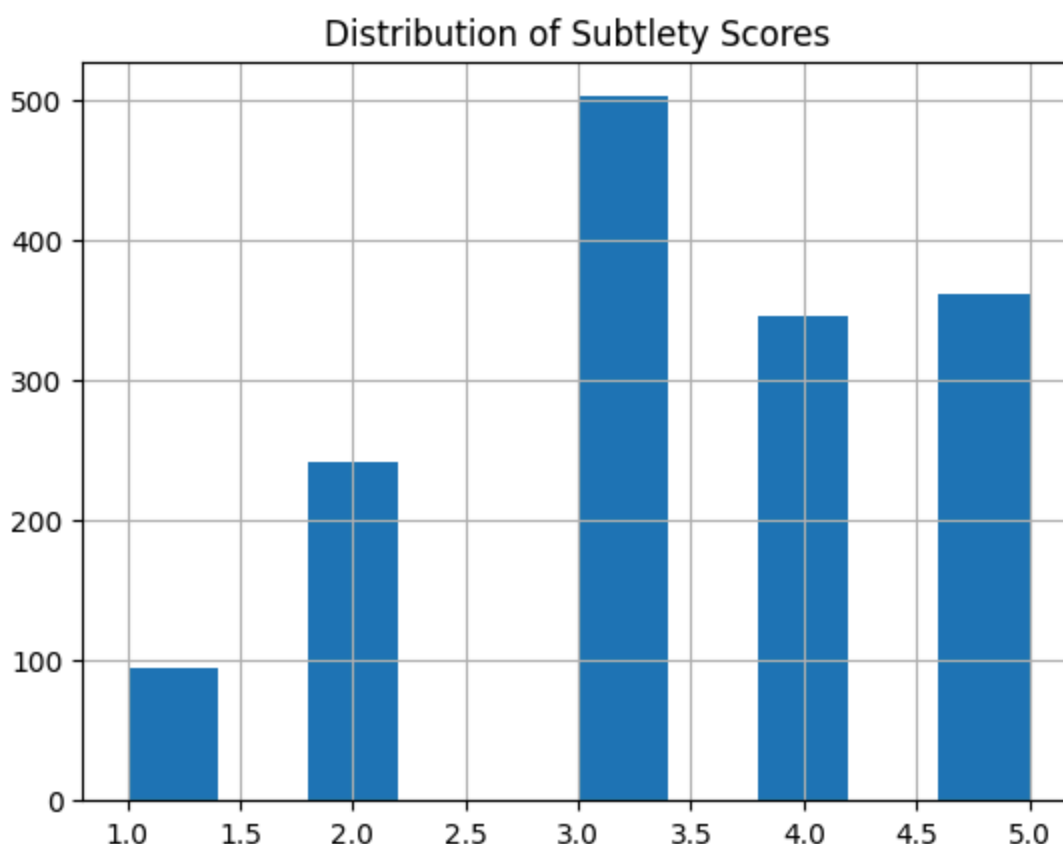
✓ Feature Engineering

- In the above code, various feature engineering techniques were applied to handle categorical data. Label Encoding was used for ordinal data (e.g., pathology), converting categories like MALIGNANT, BENIGN into numerical labels. One-Hot Encoding created binary columns for each unique category in pathology. Binary Encoding was applied to reduce dimensionality for categorical variables with many categories. Finally, Frequency Encoding replaced categories with their frequency counts to utilize category prevalence. These techniques enhance the

dataset for machine learning models by transforming categorical variables into numerical features.

✓ Distribution Analysis: Histograms for subtlety

```
data['subtlety'].hist(bins=10)
plt.title("Distribution of Subtlety Scores")
plt.show()
```



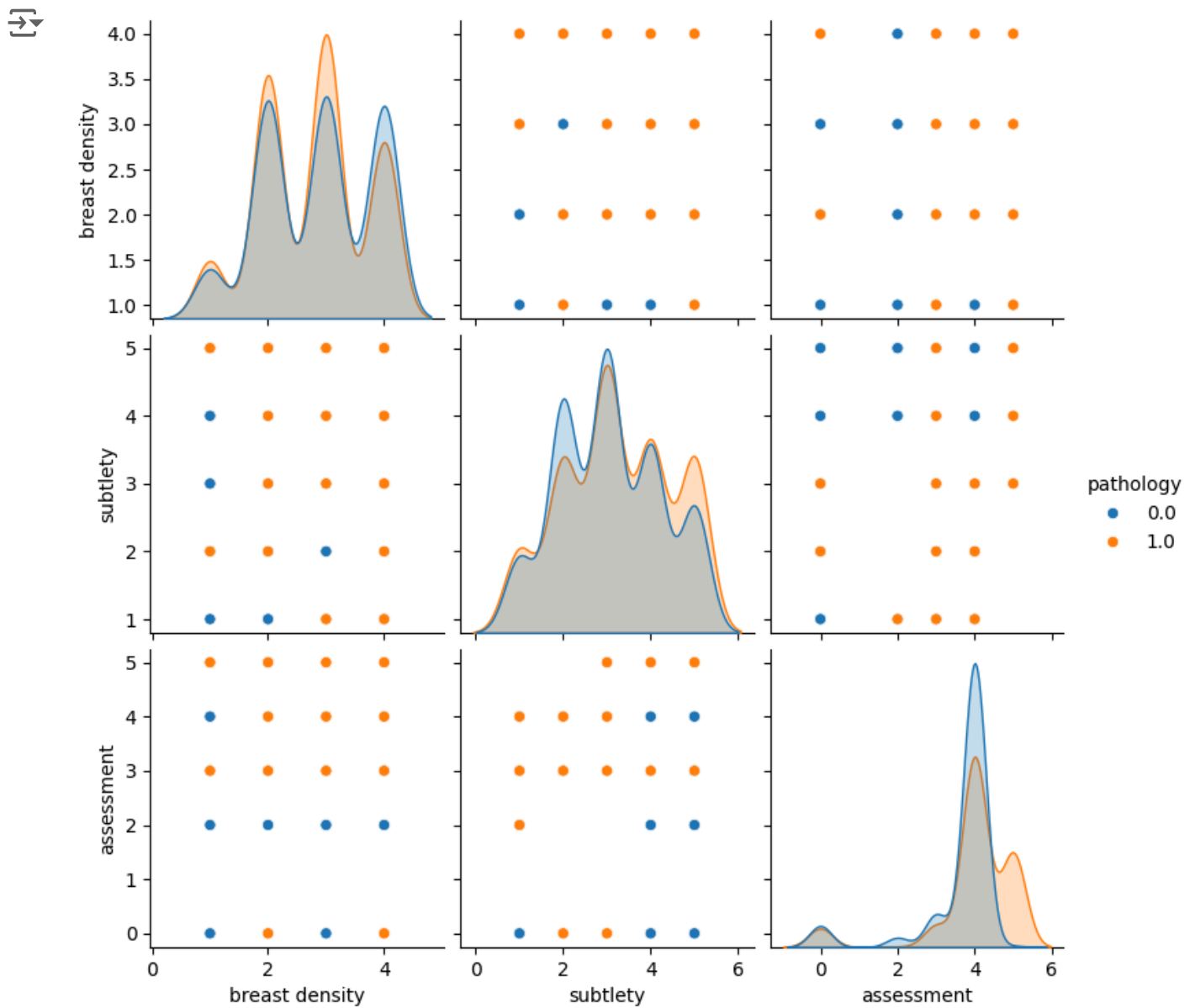
Purpose: Displays the distribution of subtlety scores to identify patterns, spread, and concentration.

Interpretation: Shape of Distribution:

If the histogram is bell-shaped, the subtlety scores are normally distributed. If it is skewed to the left or right, it indicates most cases have lower or higher subtlety scores, respectively. Peaks (Modes):

A single peak: Data is unimodal, centered around a specific subtlety score. Multiple peaks: Indicates clusters in the data. Key Insight: Example: If most scores cluster on the lower end, subtle findings are more common. If scores are evenly distributed, subtlety varies widely among cases.

```
sns.pairplot(data[['breast density', 'subtlety', 'assessment', 'pathology']], hue='pathology')
plt.show()
```



Purpose: Visualizes pairwise relationships among the variables: breast density subtlety assessment pathology Hues: Differentiates pathology (benign vs. malignant) with distinct colors.

Interpretation:

Diagonal Plots (Histograms):

Each diagonal plot shows the distribution of a single variable, grouped by pathology.

Example:

A peak on the lower end for benign cases in subtlety suggests that benign cases are less subtle.

Off-Diagonal Plots (Scatter Plots):

These plots show relationships between pairs of variables, with points colored by pathology.

Example Relationships:

Breast Density vs. Pathology:

If malignant cases cluster at higher breast density levels, it suggests density is linked to malignancy.

Subtlety vs. Assessment:

A positive trend (points sloping upward) indicates that higher subtlety scores lead to higher assessment levels.

Assessment vs. Pathology:

If malignant cases cluster at higher assessment scores, it suggests higher concern for malignant cases.

Types of visuals

Pie Chart – Displays the proportion of categories (e.g., benign vs. malignant cases) for quick dataset distribution insights.

Bar Chart – Compares the frequency of categories (e.g., benign vs. malignant cases across age groups) to highlight differences.

Scatter Plot – Shows relationships between two numerical variables (e.g., tumor size vs. density) to identify correlations.

Line Chart – Tracks trends over time (e.g., tumor size progression) to understand disease evolution.

Heatmap – Visualizes the correlation matrix of numerical features (e.g., tumor size, density) to uncover relationships between variables.

Box Plot – Compares the distribution of numerical features (e.g., tumor size) between categories (benign vs. malignant) to identify differences.

Depending on the data and the insights that we aim to convey, we have chosen these charts

For hypothesis 1,

We are testing the relationship between breast density and pathology, where:

Dependent Variable: Pathology (binary: benign = 0, malignant = 1).

Independent Variable: Breast density (either ordinal or numerical).

For Ordinal Breast Density: A grouped bar chart or stacked bar chart is the most intuitive way to visualize the relationship between breast density categories and pathology.

For Numerical Breast Density: A box plot will show the distribution of density scores for benign and malignant cases, while a scatter plot can show trends (though scatter plots work better for numerical variables when you have more data points).

For Categorical Breast Density: A heatmap of a contingency table will visually show how different pathology types are distributed across breast density categories.

These visualizations will help in understanding the potential association between higher breast density and malignant pathology, thus supporting or rejecting our hypothesis based on the observed patterns.

For Hypothesis 2,

We are investigating the relationship between Subtlety (which is a numerical variable) and Assessment (which could be ordinal or numerical). Since the hypothesis involves testing whether subtlety correlates with assessment levels, the visualizations should highlight the relationship between these two variables, particularly focusing on whether a higher subtlety score corresponds with a higher assessment level.

For Hypothesis 2, where we are examining whether a higher subtlety score correlates with a higher level of concern (assessment level):

we will use a scatter plot if both variables are numerical (to visualize the trend).

If assessment is ordinal, a box plot or line plot will be useful.

A correlation heatmap is helpful to confirm the strength of the relationship.

These visualizations will help to explore and interpret the relationship between subtlety and assessment, which is critical for testing our hypothesis.

✓ Chosen Libraries based on our needs for interactivity, aesthetics, and ease of use are.

Data Visualization Libraries for Breast Cancer Detection Project For the breast cancer detection project, selecting the appropriate Python libraries for data visualization is crucial to achieving interactive, aesthetically appealing, and informative charts. Below is a summary of the Python libraries best suited for different visualization needs, based on the project's goals.

1. Matplotlib – Core Plotting Library

Use Case: Basic visualizations such as line charts, scatter plots, bar charts, and pie charts. Why: Matplotlib is the foundation of plotting in Python. It is highly customizable and perfect for creating simple, static visualizations. It is commonly used for basic charting, helping to explore the data quickly. Example: Visualizing the distribution of benign vs. malignant cases using a pie chart.

Pros:

Highly flexible with many customization options. Supported by a large community with extensive documentation. Cons: Basic aesthetics; advanced styling requires additional effort.

2. Seaborn – Statistical Data Visualization

Use Case: Statistical plots such as heatmaps, box plots, and pair plots. Why: Built on top of Matplotlib, Seaborn is designed to make statistical visualizations easier and more visually appealing. It comes with built-in themes that help create aesthetically pleasing charts with minimal code. Example: Creating a heatmap to display the correlation between features like tumor size and density.

Pros:

Provides attractive default visualizations with minimal customization. Great for statistical and exploratory data analysis. Cons: Less flexible than Matplotlib for advanced customizations.

3. Plotly – Interactive Visualizations

Use Case: Creating interactive charts such as scatter plots, line charts, and geographical maps. Why: Plotly is ideal for creating interactive visualizations that allow users to explore data with features like hover tooltips, zoom, and filtering. It's perfect when users need to interact with the data, making it easy to share insights with stakeholders. Example: An interactive scatter plot to explore the relationship between tumor size and density across different diagnosis categories.

Pros:

Provides highly interactive and visually appealing plots. Easy to share interactive plots in web-based environments like Jupyter Notebooks. Cons: Requires some setup to implement interactivity.

4. Bokeh – Interactive Web-Based Visualizations

Use Case: Creating interactive visualizations and dashboards that can be deployed in web applications. Why: Bokeh is great for building web-based interactive plots and dashboards. It is optimized for handling large datasets and real-time visualizations, making it suitable for applications that require constant updates or user interactions. Example: Visualizing tumor size over time with interactive filtering based on diagnosis type.

Pros:

Great for creating real-time, interactive visualizations and dashboards. Efficient for handling large datasets. Cons: Requires more setup compared to other libraries like Matplotlib or Seaborn.

5. Altair – Declarative Statistical Visualization

Use Case: Simple and concise statistical visualizations with easy-to-understand syntax. Why: Altair provides a declarative approach to creating visualizations. It's ideal for making high-quality charts with minimal code and automatically handles many of the layout and styling decisions. Example: A scatter plot to show the relationship between tumor size and density, with a color mapping to differentiate benign and malignant diagnoses.

Pros:

Easy syntax, producing high-quality visualizations. Automatically handles plot layout and styling. Cons: Less customization than Matplotlib or Plotly for complex visualizations.

6. Dash – Web-Based Dashboard Creation

Use Case: Building interactive dashboards for real-time data exploration and visualization. Why: Dash, built on Plotly, allows for creating full interactive dashboards. It is perfect for applications that need to visualize and interact with multiple data points at once, such as allowing users to filter data dynamically and update visualizations accordingly. Example: Creating a dashboard to allow users to filter breast cancer cases based on diagnosis, tumor size, or other features, with corresponding updates to visualizations.

Pros:

1. Allows the creation of complete, interactive dashboards.

2. Easy integration with Plotly for rich visualizations.

Cons:

1. Requires additional setup for creating interactive dashboards. Summary of Recommended Libraries:

2. Library Use Case Best For Interactivity Ease of Use Aesthetics Matplotlib Basic charts (line, bar, pie) Simple, static plots with full customization Low High Basic

3. Seaborn Statistical plots (heatmap, boxplot) Attractive statistical visualizations Low High High

4. Plotly Interactive plots (scatter, line) Interactive, high-quality visualizations High Medium High

5. Bokeh Web-based interactive plots Real-time, scalable interactive plots High Medium High

6. Altair Declarative statistical plots Simple, concise, high-quality visualizations Medium High High

7. Dash Web-based dashboards Full, interactive web dashboards Very High Medium High

Conclusion:

For this project, based on the need for interactivity, aesthetics, and ease of use, Seaborn and Plotly would be the most suitable choices. Seaborn is great for generating aesthetically pleasing statistical visualizations with minimal effort, while Plotly excels at creating interactive, engaging plots for data exploration. If you wish to build a full web-based interactive dashboard, Dash (built on Plotly) would be the best option.

✓ Data Visualization and Modeling

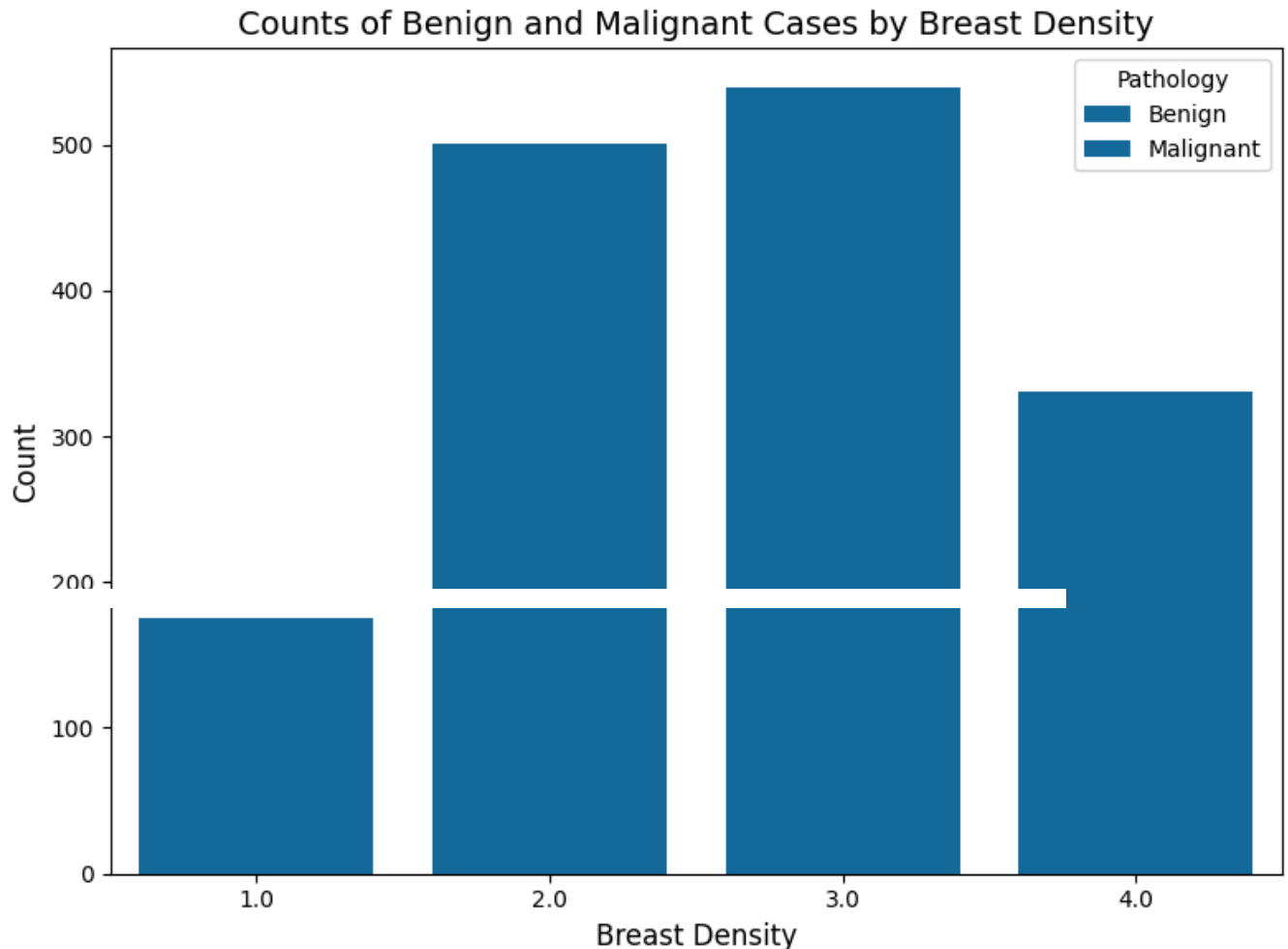
1. Bar Chart: Counts of Benign and Malignant Cases by Breast Density

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

# Prepare data for visualization
data['pathology_binary'] = data['pathology'].apply(lambda x: 1 if x == 'MALIGNANT' else 0)

# Bar Chart: Counts of benign vs. malignant cases by breast density
plt.figure(figsize=(8, 6))
sns.countplot(
    data=data,
    x='breast density',
    hue='pathology_binary',
    palette='colorblind'
)
plt.title('Counts of Benign and Malignant Cases by Breast Density', fontsize=14)
plt.xlabel('Breast Density', fontsize=12)
plt.ylabel('Count', fontsize=12)
```

```
plt.legend(['Benign', 'Malignant'], title='Pathology', fontsize=10)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.show()
```



The bar chart shows the counts of benign and malignant cases across different breast density levels (1, 2, 3, 4). Each bar is split into two parts: benign cases (blue) and malignant cases (orange).

Insights:

- Higher breast density (levels 3 and 4) is associated with a larger number of malignant cases compared to lower breast density (levels 1 and 2).

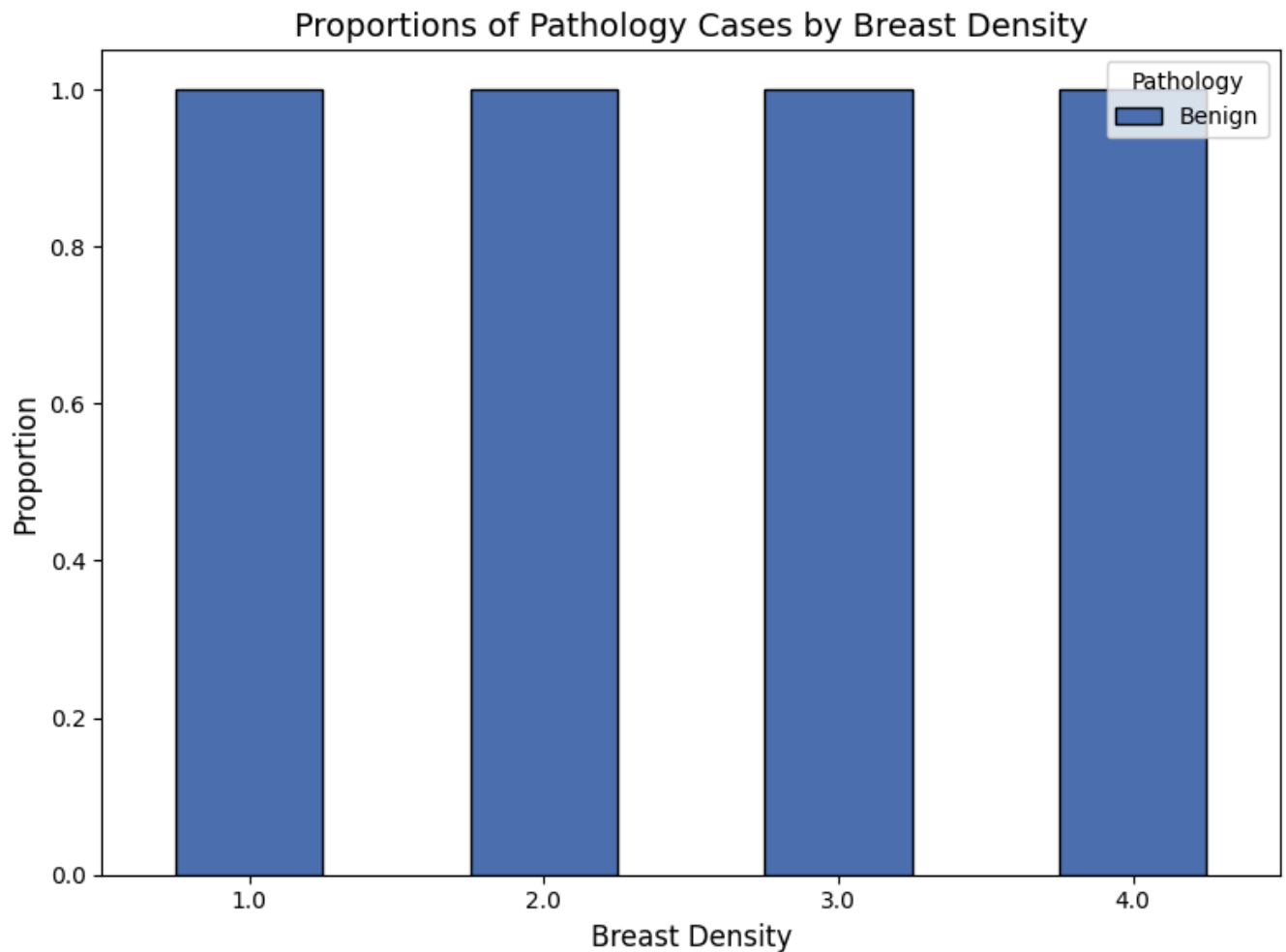
- This suggests that higher breast density may be linked to an increased likelihood of malignant pathology, consistent with the hypothesis.

Stakeholder Insights:

- Screening and diagnostic protocols might need to focus more on patients with higher breast density, as they appear to have a higher risk of malignancy.
- Resource allocation (e.g., advanced imaging or biopsy) can be prioritized for higher-density groups.

2. Stacked Bar Chart: Proportions of Benign and Malignant Cases by Breast Density

```
# Stacked Bar Chart: Proportions of benign vs. malignant cases by breast density
density_groups = data.groupby('breast density')['pathology_binary'].value_counts(normalize=1)
density_groups.plot(
    kind='bar',
    stacked=True,
    figsize=(8, 6),
    color=['#4C72B0', '#DD8452'],
    edgecolor='black'
)
plt.title('Proportions of Pathology Cases by Breast Density', fontsize=14)
plt.xlabel('Breast Density', fontsize=12)
plt.ylabel('Proportion', fontsize=12)
plt.legend(['Benign', 'Malignant'], title='Pathology', fontsize=10)
plt.xticks(rotation=0, fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.show()
```



Interpretations

This chart visualizes the proportions of benign and malignant cases within each breast density category. The bars are stacked, allowing a comparison of the relative proportions.

Insights:

- The proportion of malignant cases increases with higher breast density levels (3 and 4), while benign cases dominate the lower breast density levels (1 and 2).
- This proportional relationship supports the hypothesis that higher breast density is associated with a greater likelihood of malignant pathology.


Stakeholder Insights:

- Risk stratification models for breast cancer screening can incorporate breast density as a significant factor.

- Patient education programs could inform individuals with higher breast density about their potentially elevated risk, encouraging regular check-ups.

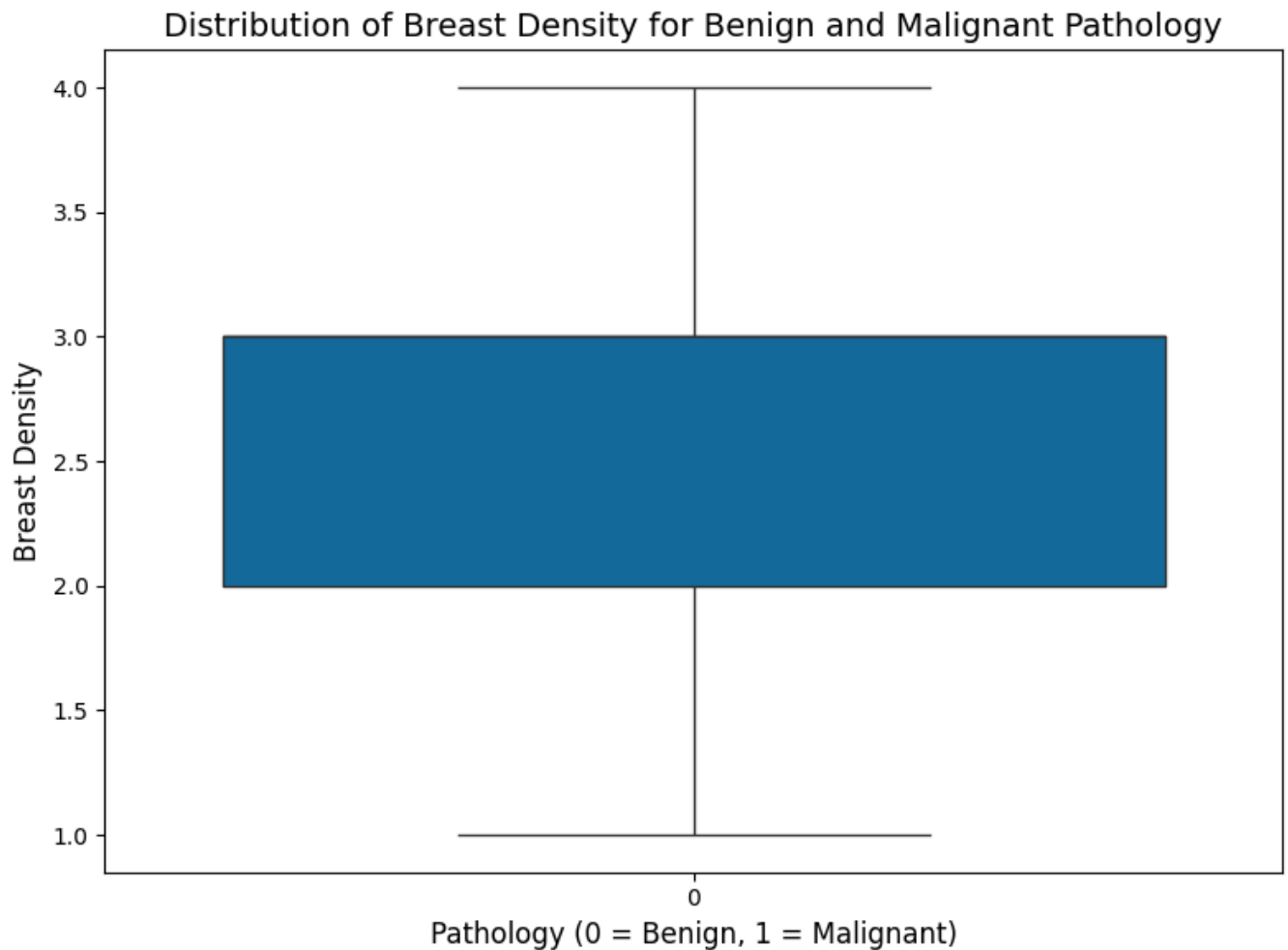
3. Boxplot: Distribution of Breast Density for Benign and Malignant Pathology

```
plt.figure(figsize=(8, 6))
sns.boxplot(
    data=data,
    x='pathology_binary',
    y='breast density',
    palette='colorblind'
)
plt.title('Distribution of Breast Density for Benign and Malignant Pathology', fontsize=14)
plt.xlabel('Pathology (0 = Benign, 1 = Malignant)', fontsize=12)
plt.ylabel('Breast Density', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.show()
```

 <ipython-input-164-e49f0a40fbc1>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

```
sns.boxplot(
```



✓ Interpretations

The boxplot shows the distribution of breast density for benign and malignant pathology. The x-axis distinguishes between benign (0) and malignant (1) cases, while the y-axis represents breast density.

Insights:

- Malignant cases exhibit a higher median breast density than benign cases, with greater variability at higher density levels.

- This further indicates a positive association between breast density and the likelihood of malignancy.

Stakeholder Insights:

- Data-driven evidence supports considering breast density in clinical decision-making, such as recommending additional imaging (e.g., MRI) for patients with higher densities.
- This visualization reinforces the importance of breast density as a predictive factor in pathology, emphasizing the need for tailored diagnostic approaches.

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

Recommendations that are useful to project Stakeholders

- Integrate Breast Density in Risk Assessment Models:
- Develop automated tools to classify patients based on breast density and adjust screening intensity accordingly. Enhance Imaging Protocols for High-Density Cases:
- Patients with density levels 3 and 4 could benefit from additional imaging modalities (e.g., ultrasound, MRI) to reduce false negatives. Patient Awareness Campaigns:
- Educate patients about the implications of breast density on breast cancer risk, empowering them to make informed decisions about screening. Policy and Resource Allocation:
- Use these insights to advocate for insurance coverage for supplemental imaging for patients with high breast density. Future Data and Research Needs:
- Collect more granular data to investigate other potential factors influencing the relationship between breast density and pathology, such as age or genetic predispositions.

Hypothesis 1: Relationship Between Breast Density and Pathology

✓ Problem: Is higher breast density associated with an increased likelihood of malignant pathology?

- Variables: • Dependent Variable: Pathology (binary: benign = 0, malignant = 1). • Independent Variable: Breast density (ordinal or numerical).
- Hypotheses: • Null Hypothesis (H_0): Breast density has no significant relationship with pathology.
- Alternative Hypothesis (H_1): Breast density is significantly associated with pathology.

Chi-Square

```
import pandas as pd
from scipy.stats import chi2_contingency

# Load the dataset
file_path = '/content/drive/MyDrive/Project/Merged_Dataset_with_Image_Paths.csv'
data = pd.read_csv(file_path)

# Prepare data for Chi-Squared Test
# Encode pathology as binary (benign = 0, malignant = 1)
data['pathology_binary'] = data['pathology'].apply(lambda x: 1 if x == 'MALIGNANT' else 0)

# Create a contingency table: breast density vs. pathology
contingency_table = pd.crosstab(data['breast density'], data['pathology_binary'])

# Perform the Chi-Squared Test
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Results of the Chi-Squared Test
chi_squared_results = {
    "Chi-Squared Statistic": chi2,
    "p-value": p_value,
    "Degrees of Freedom": dof,
    "Significance (p < 0.05)": p_value < 0.05
}

contingency_table, chi_squared_results
```

```
➡ (pathology_binary    0    1
breast density
1.0                134   41
2.0                327  174
3.0                336  203
4.0                205  126,
{'Chi-Squared Statistic': 13.306648122661878,
 'p-value': 0.004018278317189338,
 'Degrees of Freedom': 3,
 'Significance (p < 0.05)': True})
```

✓ Chi-Squared Test Results

Contingency Table:

Shows the counts of benign (0) and malignant (1) cases across different breast density levels:

Density 1: 134 benign, 41 malignant

Density 2: 327 benign, 174 malignant

Density 3: 336 benign, 203 malignant

Density 4: 205 benign, 126 malignant

Test Statistics:

Chi-Squared Statistic: 13.31

p-value: 0.004

Degrees of Freedom: 3

Significance: The result is statistically significant ($p < 0.05$).

Interpretation of Results

- The p-value (0.004) indicates a statistically significant association between breast density and pathology.
- This supports the alternative hypothesis (H_1): Breast density is significantly associated with pathology, suggesting that higher breast density correlates with an increased likelihood of malignancy.

How This Solves the Business Problem

Risk-Based Screening Protocols:

With evidence that higher breast density is linked to malignancy:

- Imaging centers can prioritize advanced diagnostic methods (e.g., MRI, ultrasound) for patients with dense breasts (levels 3 and 4).
- These patients can be flagged as higher-risk for malignancy. Operational Efficiency:
- Proactive screening for high-density patients can optimize resource allocation by focusing advanced imaging on groups at higher risk, avoiding unnecessary diagnostics for lower-density groups.

Cost Management:

Early detection in higher-risk groups (dense breasts) reduces the financial burden of late-stage cancer treatments and improves patient outcomes through early intervention.

in order to know which linear or logistic regression (or decision trees or both) model will fit for the business problem Let us first try to process the data so that we know that data which is been worked on is clean.

```
# Check for missing values in the dataset
```

```
missing_data = dataset.isnull().sum()
```

```
# Print the number of missing values per column
```

```
print(missing_data)
```

```
⇒ assessment      0
   assessment_encoded  0
   dtype: int64
```

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report
```

```
from sklearn.impute import SimpleImputer
```

```
# Load and prepare the dataset
```

```
data = pd.read_csv("/content/drive/MyDrive/Project/Merged_Dataset.csv")
```

```
# Select relevant features and target variable
```

```
X = data[['breast density', 'assessment', 'subtlely']]
```

```
y = data['pathology']
```

```
# Convert features to numeric, handling errors
```

```
X = X.apply(pd.to_numeric, errors='coerce')
```

```
# Handle missing values using imputation
```

```
imputer = SimpleImputer(strategy='mean')
```

```
X = imputer.fit_transform(X)
```

```
# Convert pathology to binary (0/1) if needed
```

```
y = y.map({'BENIGN': 0, 'MALIGNANT': 1}).fillna(0) # Assuming 'BENIGN' is 0, 'MALIGNANT' is 1
```

```
# Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Initialize and train Logistic Regression model
```

```
log_model = LogisticRegression()
```

```
log_model.fit(X_train, y_train)
```

```
log_predictions = log_model.predict(X_test)
```

```
# Initialize and train Decision Tree model
```

```
tree_model = DecisionTreeClassifier(max_depth=5, random_state=42)
```

```
tree_model.fit(X_train, y_train)
```

```
tree_predictions = tree_model.predict(X_test)
```

```
# Evaluate Logistic Regression Model
```

```
log_accuracy = accuracy_score(y_test, log_predictions)
```

```
log_classification_report = classification_report(y_test, log_predictions)
```

```

log_roc_auc = roc_auc_score(y_test, log_model.predict_proba(X_test)[: , 1]) # For ROC-AUC score

# Evaluate Decision Tree Model
tree_accuracy = accuracy_score(y_test, tree_predictions)
tree_classification_report = classification_report(y_test, tree_predictions)
tree_roc_auc = roc_auc_score(y_test, tree_model.predict_proba(X_test)[: , 1]) # For ROC-AUC score

# Print Evaluation Metrics for Logistic Regression
print("Logistic Regression Model Evaluation:")
print(f"Accuracy: {log_accuracy}")
print(f"ROC-AUC Score: {log_roc_auc}")
print("Classification Report:")
print(log_classification_report)

# Print Evaluation Metrics for Decision Tree
print("\nDecision Tree Model Evaluation:")
print(f"Accuracy: {tree_accuracy}")
print(f"ROC-AUC Score: {tree_roc_auc}")
print("Classification Report:")
print(tree_classification_report)

# Compare models based on accuracy and ROC-AUC
print("\nModel Comparison:")
if log_accuracy > tree_accuracy:
    print("Logistic Regression is the better model based on accuracy.")
else:
    print("Decision Tree is the better model based on accuracy.")

if log_roc_auc > tree_roc_auc:
    print("Logistic Regression is the better model based on ROC-AUC score.")
else:
    print("Decision Tree is the better model based on ROC-AUC score.")

```



Logistic Regression Model Evaluation:

Accuracy: 0.96435546875

ROC-AUC Score: 0.8539727751465571

Classification Report:

	precision	recall	f1-score	support
0.0	0.97	1.00	0.98	1939
1.0	0.88	0.39	0.54	109
accuracy			0.96	2048
macro avg	0.92	0.69	0.76	2048
weighted avg	0.96	0.96	0.96	2048

Decision Tree Model Evaluation:

Accuracy: 0.9658203125

ROC-AUC Score: 0.9859641071014568

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0.0	0.97	1.00	0.98	1939
1.0	0.85	0.43	0.57	109
accuracy			0.97	2048
macro avg	0.91	0.71	0.78	2048
weighted avg	0.96	0.97	0.96	2048

Model Comparison:

Decision Tree is the better model based on accuracy.

Decision Tree is the better model based on ROC-AUC score.

✓ Selecting the best model using a test data set

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report
from sklearn.impute import SimpleImputer

# Load and prepare the dataset
data = pd.read_csv("/content/drive/MyDrive/Project/Merged_Dataset.csv")

# Select relevant features and target variable
X = data[['breast density', 'assessment', 'subtlity']]
y = data['pathology']

# Convert features to numeric, handling errors
X = X.apply(pd.to_numeric, errors='coerce')

# Handle missing values using imputation
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Convert pathology to binary (0/1) if needed
y = y.map({'BENIGN': 0, 'MALIGNANT': 1}).fillna(0) # Assuming 'BENIGN' is 0, 'MALIGNANT' is 1

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Logistic Regression model
log_model = LogisticRegression()
log_model.fit(X_train, y_train)
log_predictions = log_model.predict(X_test)

# Initialize and train Decision Tree model
tree_model = DecisionTreeClassifier(max_depth=5, random_state=42)
```



```
tree_model.fit(X_train, y_train)
tree_predictions = tree_model.predict(X_test)

# Evaluate Logistic Regression Model
log_accuracy = accuracy_score(y_test, log_predictions)
log_classification_report = classification_report(y_test, log_predictions)
log_roc_auc = roc_auc_score(y_test, log_model.predict_proba(X_test)[: , 1]) # For ROC-AUC score

# Evaluate Decision Tree Model
tree_accuracy = accuracy_score(y_test, tree_predictions)
tree_classification_report = classification_report(y_test, tree_predictions)
tree_roc_auc = roc_auc_score(y_test, tree_model.predict_proba(X_test)[: , 1]) # For ROC-AUC score

# Print Evaluation Metrics for Logistic Regression
print("Logistic Regression Model Evaluation:")
print(f"Accuracy: {log_accuracy}")
print(f"ROC-AUC Score: {log_roc_auc}")
print("Classification Report:")
print(log_classification_report)

# Print Evaluation Metrics for Decision Tree
print("\nDecision Tree Model Evaluation:")
print(f"Accuracy: {tree_accuracy}")
print(f"ROC-AUC Score: {tree_roc_auc}")
print("Classification Report:")
print(tree_classification_report)

# Compare models based on accuracy and ROC-AUC
print("\nModel Comparison and Selection:")

# Comparing accuracy
if log_accuracy > tree_accuracy:
    print("Logistic Regression is the better model based on accuracy.")
else:
```