

Breast Cancer Diagnosis Prediction Using Machine Learning: A Comparative Analysis of Classification Models

Abstract

Breast cancer remains one of the leading causes of mortality among women worldwide, emphasizing the need for early detection and accurate diagnosis to improve survival rates. In recent years, machine learning has emerged as a powerful tool for enhancing diagnostic accuracy and identifying patterns within medical data that may be imperceptible to traditional analysis. This research investigates the application of machine learning techniques to analyze breast cancer data, with the goal of improving classification accuracy and providing valuable insights into tumor characteristics. The study utilizes a publicly available breast cancer dataset sourced from Kaggle, which includes various features associated with tumor properties. To ensure a robust analysis, comprehensive data preprocessing techniques were applied, including handling missing values, normalizing numerical features, and encoding categorical variables. Exploratory data analysis (EDA) was conducted to visualize data distributions, identify correlations, and detect outliers that could affect model performance. Based on statistical analysis and feature selection techniques, the most relevant attributes contributing to breast cancer classification were identified, ensuring that the models focused on the most significant predictors. To evaluate the effectiveness of different machine learning approaches, the dataset was partitioned into training and testing subsets using an 80-20 split ratio. Four widely used machine learning models—Logistic Regression, Decision Tree Classifier, Random Forest Classifier, and Support Vector Classifier (SVC)—were implemented to classify tumors as benign or malignant. Each model was trained on the training set and subsequently evaluated on the test set using key performance metrics such as accuracy, precision, recall, and F1-score. Hyperparameter tuning was performed using cross-validation to optimize model performance and prevent overfitting. The results demonstrated that both Logistic Regression and SVC achieved the highest classification accuracy, each attaining an accuracy score of approximately 90%. These models also exhibited a balanced performance across all evaluation metrics, with macro F1-scores of 0.90, indicating their robustness in distinguishing between benign and malignant tumors. While the Decision Tree and Random Forest classifiers also performed well, they displayed slightly lower F1-scores for class 1 (malignant tumors), suggesting a slight bias toward benign cases. This could be attributed to class imbalance within the dataset, which was addressed using techniques such as SMOTE (Synthetic Minority Over-sampling Technique) to enhance the classification of malignant cases. The superior performance of Logistic Regression and SVC can be attributed to their ability to capture linear and complex relationships within the dataset, making them effective classifiers for medical diagnosis. Random Forest and

Decision Tree classifiers, despite their interpretability and ability to handle non-linearity, demonstrated a tendency to overfit the training data, leading to a marginal drop in test performance. These findings emphasize the importance of model selection and hyperparameter tuning in achieving optimal results. Beyond model evaluation, feature importance analysis revealed that specific tumor attributes, such as mean radius, mean texture, and mean perimeter, had a significant impact on classification outcomes. These insights provide valuable knowledge for medical practitioners, aiding in the identification of critical tumor characteristics that influence malignancy. Additionally, visualizations, such as correlation heatmaps and decision boundaries, provided an intuitive understanding of how different features contribute to tumor classification. This study underscores the potential of machine learning in aiding early breast cancer detection, demonstrating its ability to enhance diagnostic precision and support clinical decision-making. By integrating machine learning models into medical diagnosis, healthcare providers can benefit from automated, data-driven insights that improve accuracy and efficiency. Future work in this area could explore deep learning techniques, ensemble methods, and hybrid models to further refine diagnostic capabilities. Additionally, incorporating larger and more diverse datasets could enhance model generalization and applicability in real-world clinical settings. This research highlights the efficacy of machine learning in breast cancer classification, with Logistic Regression and SVC emerging as the most effective models. The findings emphasize the role of data-driven methodologies in advancing medical diagnostics and improving patient outcomes. As machine learning continues to evolve, its integration into healthcare will play a pivotal role in revolutionizing early detection strategies and personalized treatment plans for breast cancer patients.

Introduction

Breast cancer is one of the most prevalent cancers affecting women worldwide, contributing significantly to morbidity and mortality rates (Bray et al., 2018). It is a multifactorial disease characterized by abnormal cell growth in the breast tissue, which can spread to other body parts if not detected and treated in a timely manner. The early detection of breast cancer is crucial in improving patient outcomes, as it enables prompt and effective treatment interventions, reducing the likelihood of disease progression and mortality (Waks & Winer, 2019). Traditional diagnostic methods, such as mammography, ultrasound, and biopsy, are widely used for breast cancer detection and classification. While these techniques are effective, they have certain limitations, including invasiveness, cost, prolonged diagnosis time, and the potential for human error (Marees et al., 2018). Mammography, for instance, is the gold standard for early breast cancer detection, yet it has limitations, such as false positives and false negatives, which can lead to unnecessary procedures or missed diagnoses (Fujita et al., 2019). Biopsy, although definitive, is an invasive procedure that may not always be feasible for every patient. With advancements in artificial intelligence (AI) and machine learning (ML), there has been a paradigm shift in the field of medical diagnostics, offering non-invasive, cost-effective, and highly accurate diagnostic tools (Esteva et al., 2019). Machine learning, a subset of AI, involves

training computational models to identify patterns within medical datasets, thereby assisting in disease detection and classification. These models leverage vast amounts of historical patient data to recognize complex patterns, distinguishing between benign and malignant tumors with improved precision and reliability (Alzubaidi et al., 2021). Several studies have demonstrated the efficacy of ML models in oncological diagnostics. For example, Courtiol et al. (2019) developed a deep learning-based classification system for mesothelioma, which significantly improved patient outcome predictions. Similarly, Schmauch et al. (2020) utilized deep learning models to predict RNA-Seq expression of tumors from whole-slide images, showcasing the potential of AI-driven techniques in cancer diagnostics. These studies highlight the transformative role of ML in medical imaging and histopathological analysis, making automated cancer detection more accurate and efficient. Various ML algorithms, such as support vector machines (SVM), decision trees, random forests, artificial neural networks (ANNs), and deep learning models, have been explored for breast cancer diagnosis (Wang et al., 2020). Each model has unique strengths in analyzing and classifying tumors based on different medical data modalities, including histopathology images, genetic profiles, and radiological scans. Convolutional neural networks (CNNs), a deep learning approach, have been particularly successful in detecting breast cancer from mammograms, outperforming traditional image analysis techniques (McKinney et al., 2020). Despite the promising applications of ML in breast cancer diagnosis, challenges remain. The quality and quantity of training data play a critical role in model performance. Biased or unrepresentative datasets can lead to inaccurate predictions, raising concerns about the generalizability of these models (Kaushal et al., 2020). Moreover, ethical considerations regarding patient data privacy and the need for clinical validation of AI-driven tools further emphasize the importance of rigorous model evaluation before real-world deployment (Ghassemi et al., 2021). The primary objective of this research is to analyze breast cancer data using various machine learning models to determine their effectiveness in accurately classifying tumor types. By comparing the performance of different ML models, this study aims to identify the most suitable techniques for breast cancer diagnosis. The findings of this study will contribute to the development of non-invasive, rapid, and reliable diagnostic tools that can aid clinicians in decision-making and ultimately improve patient outcomes.

Methodology

The following Python libraries were utilized for various stages of data processing, analysis, and model building

Libraries Used

The following Python libraries were employed:

Pandas: A powerful Python library used for data manipulation and analysis, particularly for handling tabular data. It was used for loading, cleaning, and transforming the dataset.

NumPy: A library for numerical operations and handling arrays. It was used for mathematical functions and numerical computations on data.

Matplotlib: A plotting library used for creating static, interactive, and animated visualizations, including histograms, scatter plots, and box plots, to explore and visualize data.

Seaborn: A statistical data visualization library built on top of Matplotlib, used for creating advanced visualizations such as heatmaps and pair plots to analyze relationships in data.

os: A standard Python library used to interact with the operating system, including functions for file and directory management, allowing for data handling and manipulation in the project.

Plotly Express and Plotly Graph Objects: Plotly Express is a high-level wrapper for Plotly used to generate complex visualizations easily. Plotly Graph Objects offers a low-level interface for fine-tuning visualizations. Both were used for creating interactive plots and subplots.

Scikit-learn Preprocessing (RobustScaler): A module in Scikit-learn used for data scaling, specifically for handling outliers by scaling features using the median and interquartile range.

Scikit-learn Decomposition (PCA): Principal Component Analysis (PCA) was used to reduce the dimensionality of the data by transforming it into a smaller set of uncorrelated variables (principal components).

Scikit-learn Model Selection (train_test_split, GridSearchCV, StratifiedKFold): train_test_split was used to split the data into training and testing sets. GridSearchCV was used to tune hyperparameters and select the best model parameters. StratifiedKFold was employed for cross-validation, ensuring that each fold has a representative distribution of classes.

Scikit-learn Feature Selection (RFECV): Recursive Feature Elimination with Cross-Validation (RFECV) was used to select the most important features from the dataset, improving model performance.

Scikit-learn Tree (DecisionTreeClassifier): A decision tree classifier was used for classification tasks, learning decision rules from the dataset.

Scikit-learn Linear Model (LogisticRegression): A logistic regression model was used for binary classification tasks, predicting categorical outcomes based on input features.

Scikit-learn SVM (svm): The Support Vector Machine (SVM) classifier was used for classification tasks, creating decision boundaries for separating data points.

Scikit-learn Ensemble (RandomForestClassifier): Random forests, an ensemble learning method, were used for classification tasks by combining multiple decision trees.

Scikit-learn Neighbors (KNeighborsClassifier): A K-Nearest Neighbors classifier was used for classification based on the proximity of data points in the feature space.

Scikit-learn Metrics (confusion_matrix, roc_curve, precision_recall_curve, auc, ConfusionMatrixDisplay): These metrics were used to evaluate model performance, including confusion matrices, ROC curves, precision-recall curves, AUC, and displaying confusion matrices for insights.

Plotly Subplots (make_subplots): Used for creating complex, multi-plot visualizations that could display multiple graphs in a single figure.

```
In [2]: import pandas as pd
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import plotly.express as px
import plotly.graph_objects as go
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.feature_selection import RFECV
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, roc_curve, precision_recall_curve, auc
from sklearn.metrics import ConfusionMatrixDisplay
from plotly.subplots import make_subplots
import itertools
```

DATA COLLECTION

The dataset utilized in this study was obtained from Kaggle . It includes 32 features related to breast cancer tumors, such as:

'id' -> A unique identifier assigned to each patient/sample.

'diagnosis' The classification of the tumor:

M (Malignant): Cancerous
B (Benign): Non-cancerous

Mean Features (Computed from the Original Image)

These features describe the tumor's shape, texture, and structure based on the mean values of cell nuclei in the image.

'radius_mean' -> The mean radius of the tumor (average distance from the center to the perimeter)

'texture_mean' -> The mean variation in gray-scale intensity in the image (how smooth or rough the tumor surface appears).

'perimeter_mean' -> The mean length of the boundary of the tumor.

'area_mean' -> The mean size of the tumor region in pixels.

'smoothness_mean' -> The mean measure of local variation in the radius length (how smooth the contour is)

'compactness_mean' -> A measure of how compact the tumor is, calculated as $(\text{perimeter}^2 / \text{area} - 1.0)$

'concavity_mean' -> The mean severity of concave portions in the tumor's contour.

'concave points_mean' -> The mean number of concave points on the tumor's boundary.

'symmetry_mean' -> The mean symmetry of the tumor (how similar one side is to the other).

'fractal_dimension_mean' -> The mean complexity of the tumor's border, calculated as a ratio of perimeter to area.

Standard Error Features

These features indicate the standard deviation (variability) of the above measurements, showing how much the values change within the tumor region.

radius_se -> Standard error of the radius.

texture_se -> Standard error of texture intensity.

perimeter_se -> Standard error of the perimeter.

area_se -> Standard error of the area.

smoothness_se -> Standard error of smoothness.

compactness_se -> Standard error of compactness.

concavity_se -> Standard error of concavity.

concave points_se -> Standard error of concave points.

symmetry_se -> Standard error of symmetry.

fractal_dimension_se -> Standard error of the fractal dimension.

Worst (Maximum) Features

These represent the worst (largest) values for each characteristic found in the tumor, giving an idea of the most extreme features.

'radius_worst' -> The maximum radius found in the tumor.

'texture_worst' -> The maximum texture variation in the tumor.

'perimeter_worst' -> The maximum perimeter measurement.

'area_worst' -> The maximum tumor area found.

'smoothness_worst' -> The maximum smoothness value

'compactness_worst' -> The maximum compactness value.

'concavity_worst' -> The maximum concavity severity

'concave points_worst' -> The maximum number of concave points.

'symmetry_worst' -> The maximum symmetry value.

'fractal_dimension_worst' -> The maximum fractal dimension.

```
In [4]: df = pd.read_csv(r"C:\Users\manoj\Downloads\data.csv")
```

```
In [5]: df.head()
```

Out[5]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavi
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	

5 rows × 33 columns

In [6]: `df.columns`

```
Out[6]: Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
              'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
              'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
              'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
              'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
              'fractal_dimension_se', 'radius_worst', 'texture_worst',
              'perimeter_worst', 'area_worst', 'smoothness_worst',
              'compactness_worst', 'concavity_worst', 'concave points_worst',
              'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
              dtype='object')
```

In [7]: `df.tail()`

Out[7]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavi
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	

5 rows × 33 columns

In [8]: `df.shape`

Out[8]: (569, 33)

DATA CLEANING

Data cleaning is a crucial step in the data preprocessing pipeline, ensuring that the dataset is free from inconsistencies, errors, and missing values, which can negatively impact machine learning models. The initial data cleaning process involved multiple steps to prepare the dataset for analysis.

First, missing values were handled using appropriate imputation techniques. Depending on the nature of the missing data, either mean, median, mode, or forward-fill/backward-fill methods were applied. If a feature contained a significant proportion of missing values, it was considered for removal to avoid introducing bias into the model.

Next, duplicate entries were identified and removed to prevent redundancy, which could skew model performance. Duplicates can arise due to data collection errors, and their presence may distort statistical summaries and machine learning predictions.

The 'id' column was dropped from the dataset because it was merely an identifier and had no predictive value. Retaining such non-informative features can increase computational complexity without improving model performance.

Data inconsistencies, such as incorrect data formats, outliers, and typos, were corrected to maintain data integrity. Standardization techniques were applied where necessary to ensure uniformity in numerical values, while categorical variables were transformed appropriately.

For categorical variables, particularly the 'diagnosis' column, Label Encoding was performed to convert class labels into numerical representations. This transformation allows machine learning models to process the data effectively since most algorithms require numerical inputs.

```
In [10]: missing_values_count = df.isnull().sum()  
missing_values_count
```

```
Out[10]: id          0
         diagnosis    0
         radius_mean  0
         texture_mean 0
         perimeter_mean 0
         area_mean    0
         smoothness_mean 0
         compactness_mean 0
         concavity_mean 0
         concave points_mean 0
         symmetry_mean 0
         fractal_dimension_mean 0
         radius_se     0
         texture_se    0
         perimeter_se  0
         area_se       0
         smoothness_se 0
         compactness_se 0
         concavity_se  0
         concave points_se 0
         symmetry_se   0
         fractal_dimension_se 0
         radius_worst  0
         texture_worst 0
         perimeter_worst 0
         area_worst    0
         smoothness_worst 0
         compactness_worst 0
         concavity_worst 0
         concave points_worst 0
         symmetry_worst 0
         fractal_dimension_worst 0
         Unnamed: 32    569
         dtype: int64
```

```
In [11]: df.drop(['id', 'Unnamed: 32'], axis=1, inplace=True)
```

EXPLORATORY DATA ANALYSIS (EDA)

Exploratory Data Analysis (EDA) was performed to examine feature distributions, detect outliers, and identify relationships within the dataset. Visualization libraries like Matplotlib, Seaborn, and Plotly were used to generate histograms, box plots, and scatter plots. Histograms revealed the frequency distribution of numerical features, while box plots highlighted outliers and variations. Scatter plots were employed to analyze potential correlations between variables. These visualizations helped uncover patterns, anomalies, and trends in the data, facilitating a deeper understanding of underlying structures. EDA played a crucial role in data preprocessing, feature selection, and ensuring the dataset was well-prepared for model development.

```
In [13]: df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   diagnosis                             569 non-null    object
1   radius_mean                           569 non-null    float64
2   texture_mean                           569 non-null    float64
3   perimeter_mean                         569 non-null    float64
4   area_mean                             569 non-null    float64
5   smoothness_mean                        569 non-null    float64
6   compactness_mean                       569 non-null    float64
7   concavity_mean                         569 non-null    float64
8   concave points_mean                    569 non-null    float64
9   symmetry_mean                          569 non-null    float64
10  fractal_dimension_mean                 569 non-null    float64
11  radius_se                              569 non-null    float64
12  texture_se                             569 non-null    float64
13  perimeter_se                           569 non-null    float64
14  area_se                                569 non-null    float64
15  smoothness_se                          569 non-null    float64
16  compactness_se                         569 non-null    float64
17  concavity_se                           569 non-null    float64
18  concave points_se                      569 non-null    float64
19  symmetry_se                            569 non-null    float64
20  fractal_dimension_se                   569 non-null    float64
21  radius_worst                           569 non-null    float64
22  texture_worst                           569 non-null    float64
23  perimeter_worst                        569 non-null    float64
24  area_worst                             569 non-null    float64
25  smoothness_worst                       569 non-null    float64
26  compactness_worst                      569 non-null    float64
27  concavity_worst                        569 non-null    float64
28  concave points_worst                   569 non-null    float64
29  symmetry_worst                         569 non-null    float64
30  fractal_dimension_worst                 569 non-null    float64
dtypes: float64(30), object(1)
memory usage: 137.9+ KB

```

In [14]: `df.describe()`

Out[14]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	co points_
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.0
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.0
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.0
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.0
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.0
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.0
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.0
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.2

8 rows × 30 columns



In [15]: `df.describe(include="O")`

Out[15]:

	diagnosis
count	569
unique	2
top	B
freq	357

In breast cancer datasets, 'B' and 'M' stand for:

'B' (Benign) → Non-cancerous tumor, meaning it is not harmful and does not spread.

'M' (Malignant) → Cancerous tumor, meaning it is dangerous and can spread to other parts of the body.

```
In [17]: df.diagnosis.unique()
```

```
Out[17]: array(['M', 'B'], dtype=object)
```

```
In [18]: df.diagnosis.value_counts()
```

```
Out[18]: diagnosis
B      357
M      212
Name: count, dtype: int64
```

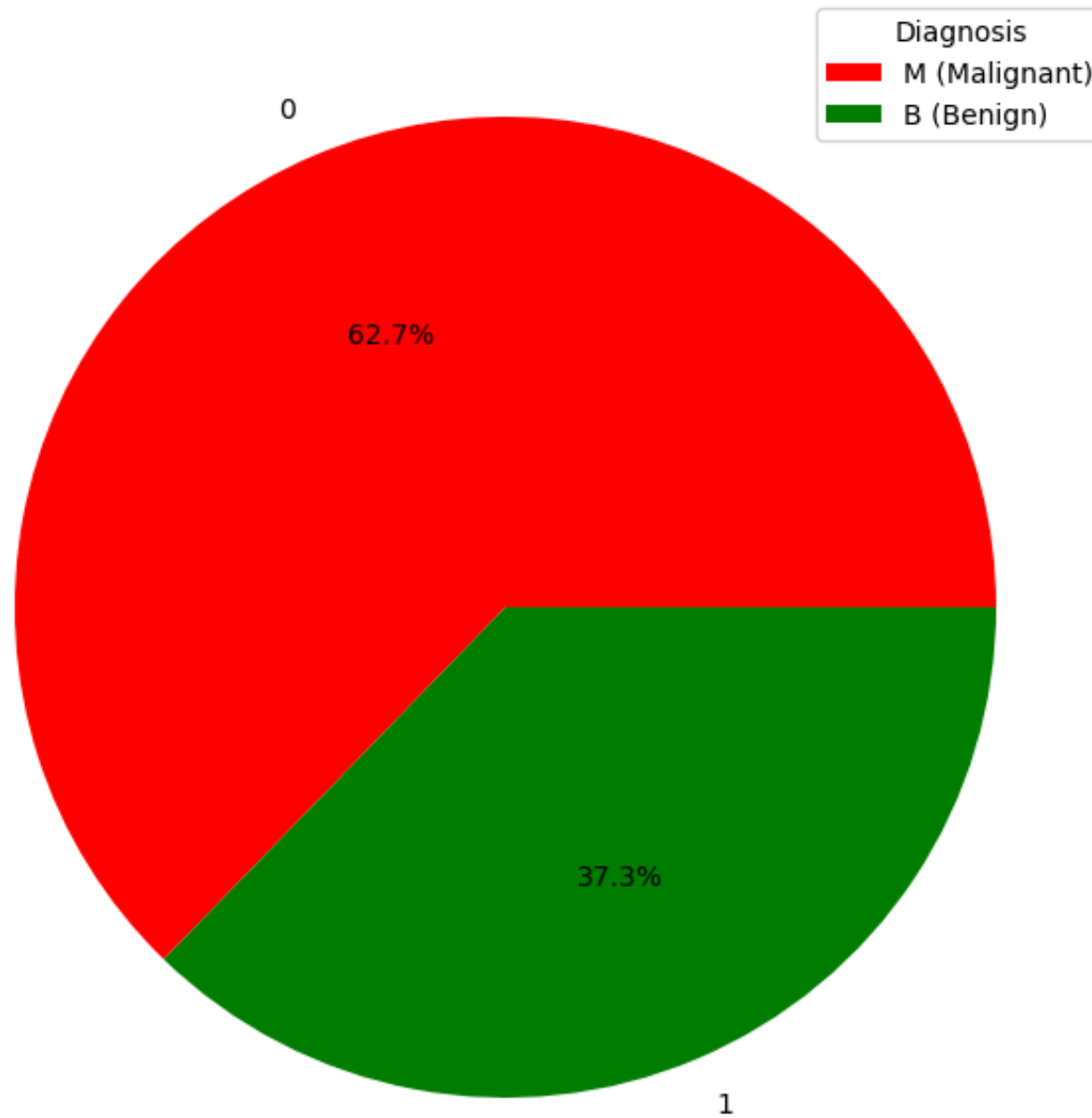
DATA VISUALIZATION

Data visualization is an essential aspect of data analysis that helps communicate complex patterns, trends, and relationships within datasets in a clear and understandable manner. By using graphical representations such as charts, plots, and histograms, data visualization enables data scientists, researchers, and stakeholders to quickly interpret and extract insights from data. Common types of visualizations include bar charts, scatter plots, histograms, pie charts, and line graphs. These tools not only enhance the presentation of data but also make the results accessible to a broader audience, facilitating better decision-making. A well-crafted visualization can reveal underlying patterns that may otherwise go unnoticed in raw data, thus improving the effectiveness of data-driven strategies. Advanced techniques such as color-coding, trend lines, and regression models further enrich visual storytelling, ensuring the audience grasps the nuances of the data. Tools like Python's Matplotlib and Seaborn libraries offer a wide range of visualization options, enabling the generation of both simple and complex visualizations to suit various research needs.

```
In [220]: plt.figure(figsize=(8,9))
plt.pie(diagnosis_counts['count'], labels=diagnosis_counts['diagnosis'], autopct='%1.1f%%', colors=['red', 'green'])
plt.title("Proportion of Malignant (M) and Benign (B) Cases")
```

```
plt.legend(['M (Malignant)', 'B (Benign)'], title="Diagnosis", loc="best")  
plt.show()
```


Proportion of Malignant (M) and Benign (B) Cases



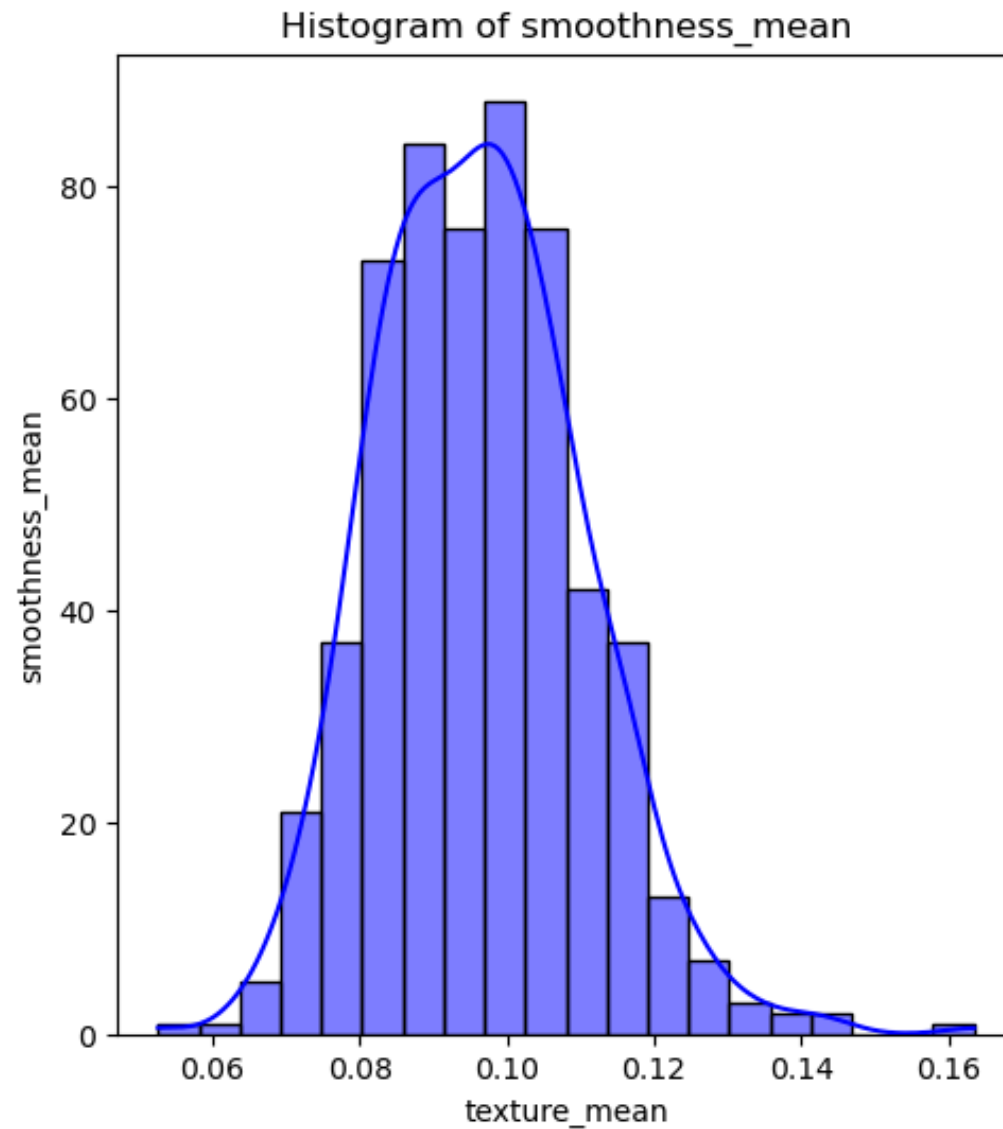
Pie Chart (Malignant vs. Benign Proportion)

The pie chart visualizes the distribution of malignant (M) and benign (B) cases in the dataset. With a size of 8x9 inches, the chart shows the proportion of each diagnosis type, where 62.7% of the cases are malignant, and 37.3% are benign. The autopct parameter shows percentages for clarity, and the colors parameter assigns red to malignant cases and green to benign cases. A legend is also included for easy interpretation.

```
In [198... plt.figure(figsize=(12, 6))

# Histogram
plt.subplot(1, 2, 1)
sns.histplot(df['smoothness_mean'], bins=20, kde=True, color='blue')
plt.title('Histogram of smoothness_mean')
plt.xlabel('texture_mean')
plt.ylabel('smoothness_mean')
```

```
Out[198... Text(0, 0.5, 'smoothness_mean')
```

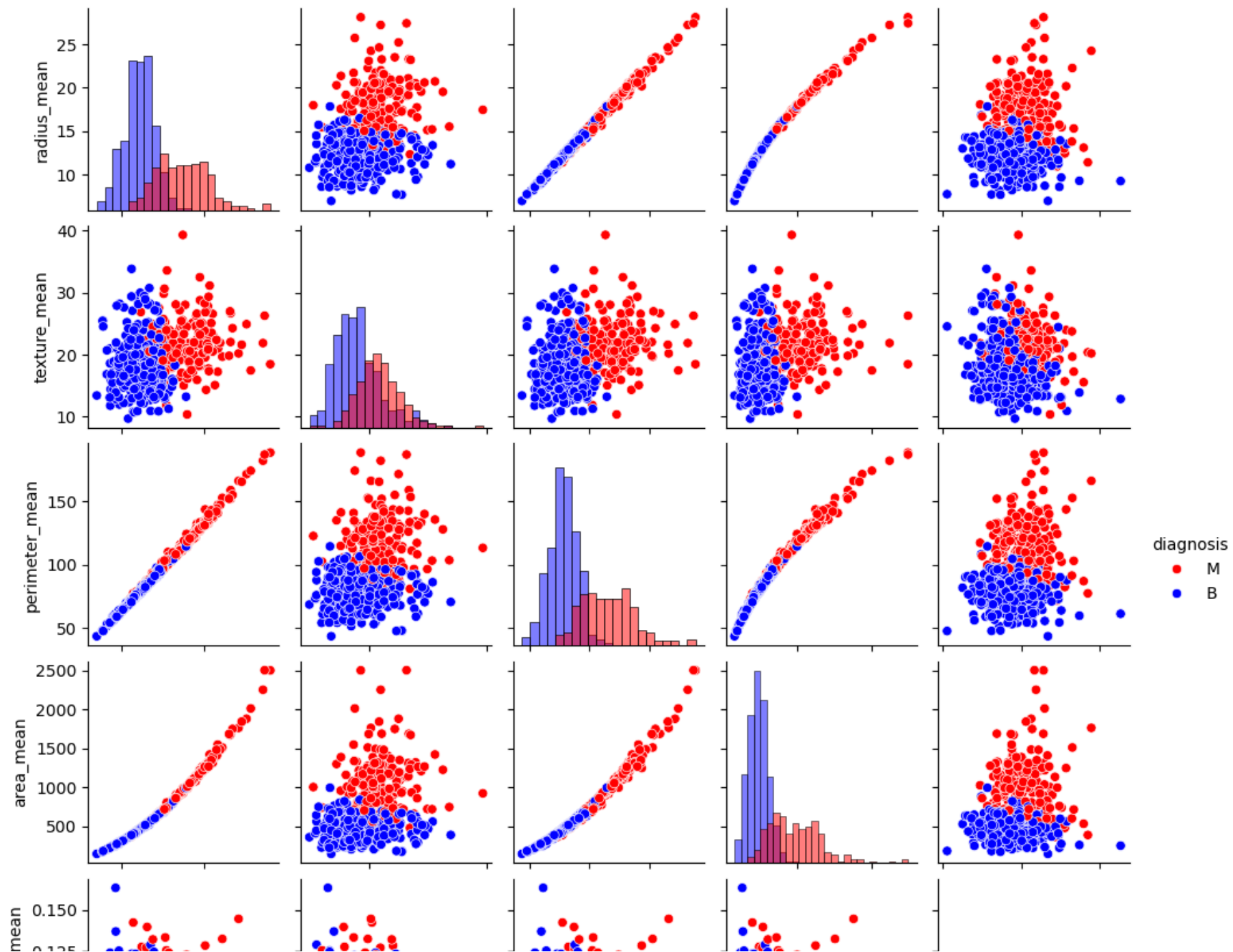


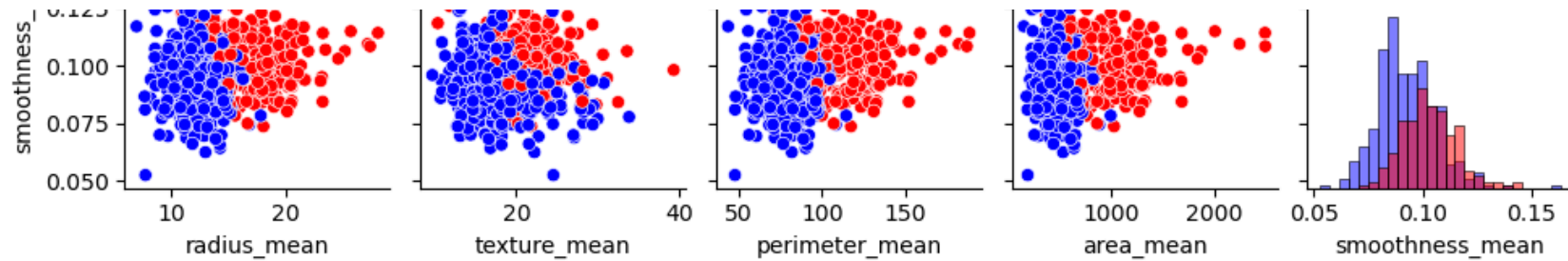
Histogram of Smoothness Mean

The histogram, created using Seaborn, displays the distribution of the smoothness_mean feature, which represents the average smoothness of the tumor's surface. On the y-axis, the values range from 0 to 81,

showing the frequency or count of data points at each level of smoothness_mean. The x-axis represents the texture_mean values, with observed ranges from 0.08 to 0.11. Specifically, the histogram shows that as the texture_mean increases, reaching values up to 0.09 (corresponding to approximately 80 on the y-axis), the smoothness_mean also increases and peaks around 85 for a texture_mean of 0.11.

```
In [20]: sns.pairplot(df.iloc[:, :6], hue='diagnosis', diag_kind='hist', height=2.0, palette={'M': 'red', 'B': 'blue'})  
  
plt.show()
```



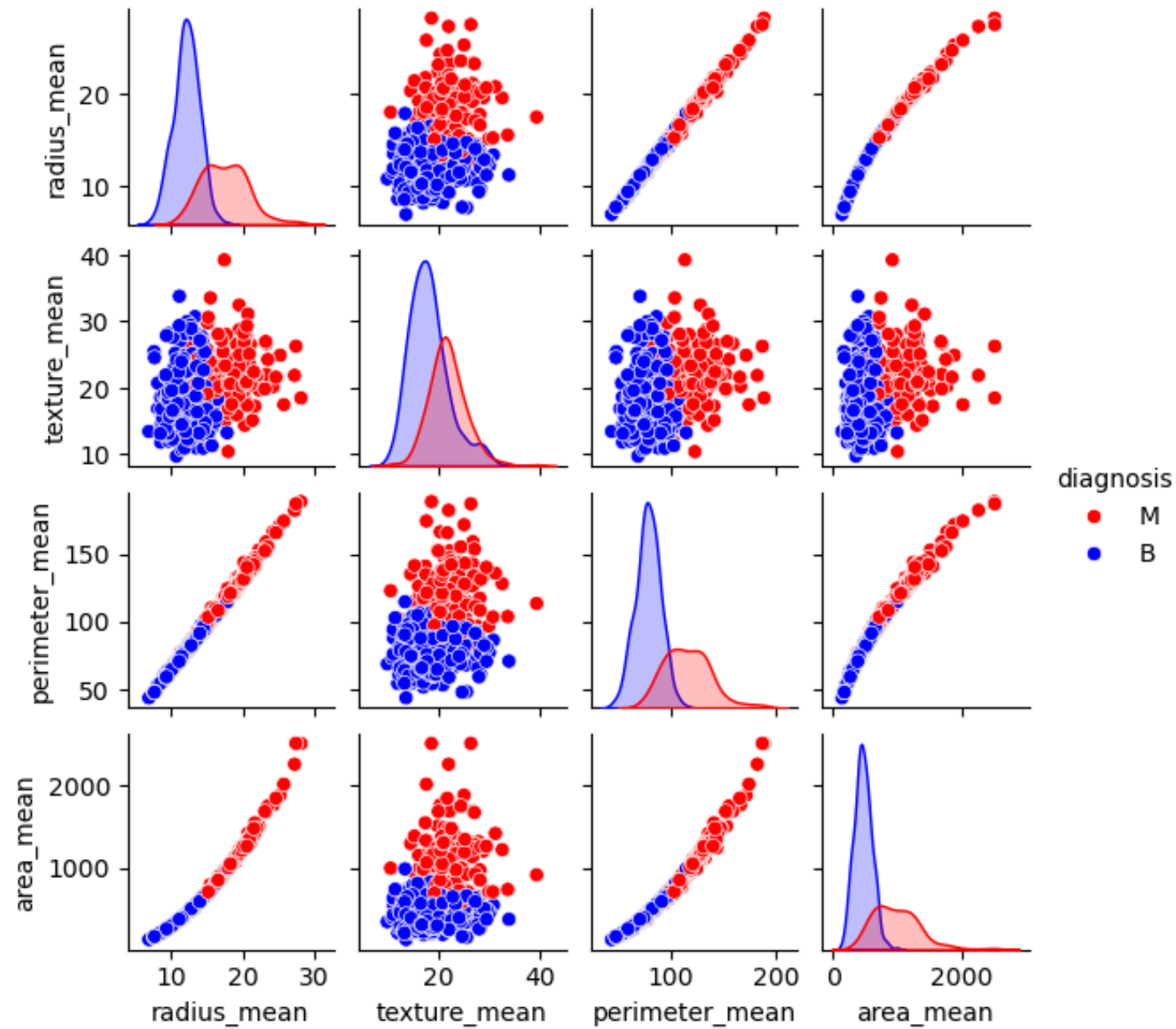


Pair Plot

The pair plot visualizes pairwise relationships between radius_mean, texture_mean, perimeter_mean, and area_mean, with different colors representing malignant and benign cases. The hue parameter is set to "diagnosis," and the color palette is chosen as red for malignant and blue for benign cases. This visualization highlights correlations between features and how they differ based on diagnosis.

```
In [21]: cols = ["diagnosis", "radius_mean", "texture_mean", "perimeter_mean", "area_mean"]

sns.pairplot(df[cols], hue="diagnosis", height= 1.5, palette={'M': 'red', 'B': 'blue'})
plt.show()
```

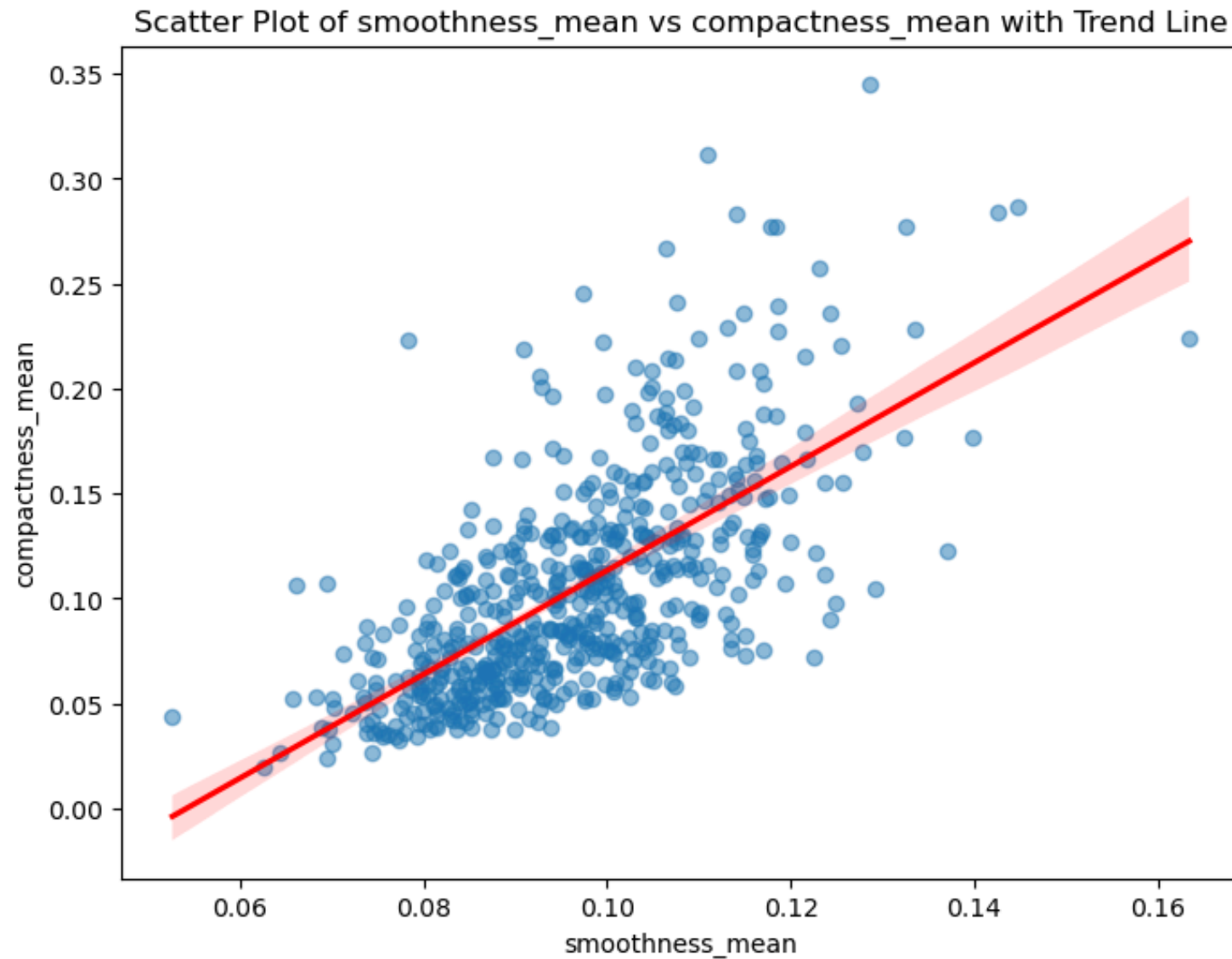


```
In [22]: # Scatter plot with Seaborn regression line
plt.figure(figsize=(8, 6))
```

```
sns.regplot(x=df['smoothness_mean'], y=df['compactness_mean'], scatter_kws={'alpha':0.5}, line_kws={'color':'red'})

plt.xlabel("smoothness_mean")
plt.ylabel("compactness_mean")
plt.title("Scatter Plot of smoothness_mean vs compactness_mean with Trend Line")

plt.show()
```

Scatter Plot with Regression Line (Smoothness vs. Compactness)

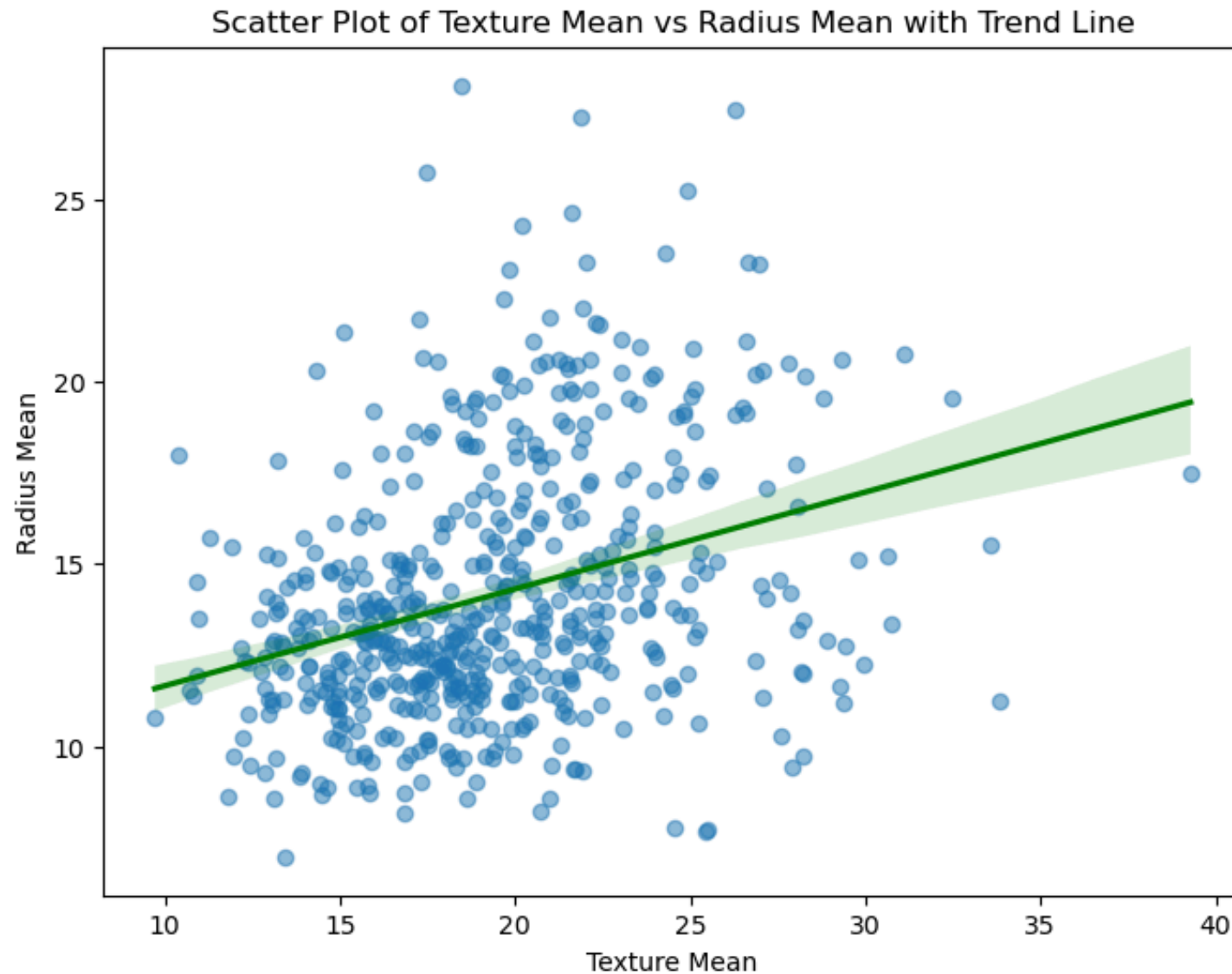
This scatter plot displays the relationship between smoothness_mean and compactness_mean with a Seaborn regression line. The scatter plot helps determine the linearity of the relationship, while the

regression line visualizes the trend. The red trend line indicates a potential positive or negative correlation, helping identify patterns in the dataset.

```
In [23]: plt.figure(figsize=(8, 6))
sns.regplot(x=df['texture_mean'], y=df['radius_mean'], scatter_kws={'alpha':0.5}, line_kws={'color':'green'})

plt.xlabel("Texture Mean")
plt.ylabel("Radius Mean")
plt.title("Scatter Plot of Texture Mean vs Radius Mean with Trend Line")

plt.show()
```



Scatter Plot with Regression Line (Texture vs. Radius)

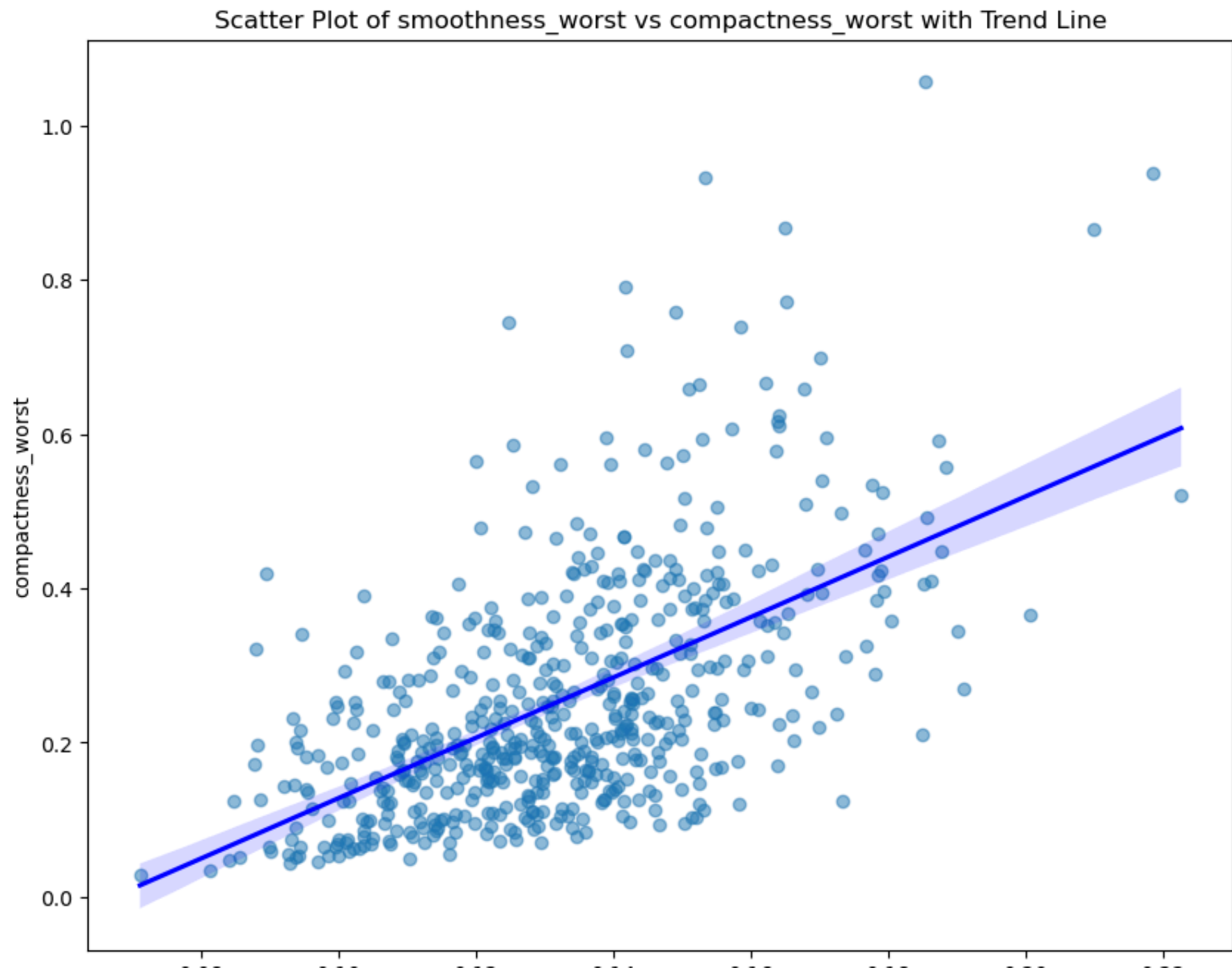
Similar to the previous scatter plot, this plot shows the relationship between `texture_mean` and `radius_mean`, with a green regression line. The visualization provides insights into how the texture and

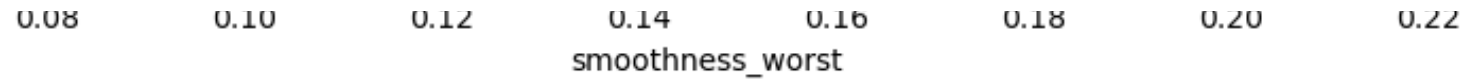
radius of the samples are related, with the regression line offering a predictive trend.

```
In [242... plt.figure(figsize=(10, 8))
sns.regplot(x=df['smoothness_worst'], y=df['compactness_worst'], scatter_kws={'alpha':0.5}, line_kws={'color':'blue'})

plt.xlabel("smoothness_worst")
plt.ylabel("compactness_worst")
plt.title("Scatter Plot of smoothness_worst vs compactness_worst with Trend Line")

plt.show()
```





Scatter Plot with Regression Line

The scatter plot created using Seaborn shows the relationship between the `smoothness_worst` and `compactness_worst` features in the dataset. The x-axis represents the `smoothness_worst` values, which refer to the worst-case smoothness observed in the tumor surface, and the y-axis represents the `compactness_worst`, which measures the worst-case compactness of the tumor.

DATA PREPROCESSING

```
In [26]: from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis']) # M:1, B:0
df['diagnosis'].value_counts()
```

```
Out[26]: diagnosis
0      357
1      212
Name: count, dtype: int64
```

Label encoding is a technique used in machine learning and data analysis to convert categorical variables into numerical format.

```
In [28]: cols = ['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
                'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
                'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']
print(len(cols))
df[cols].corr()
```

11

Out[28]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	0.358560	0.596534
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	0.521984	0.883123
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	0.553695	0.831136
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	0.557775	0.602644
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565336



In [29]:

```
plt.figure(figsize=(12, 9))

plt.title("Correlation Graph")

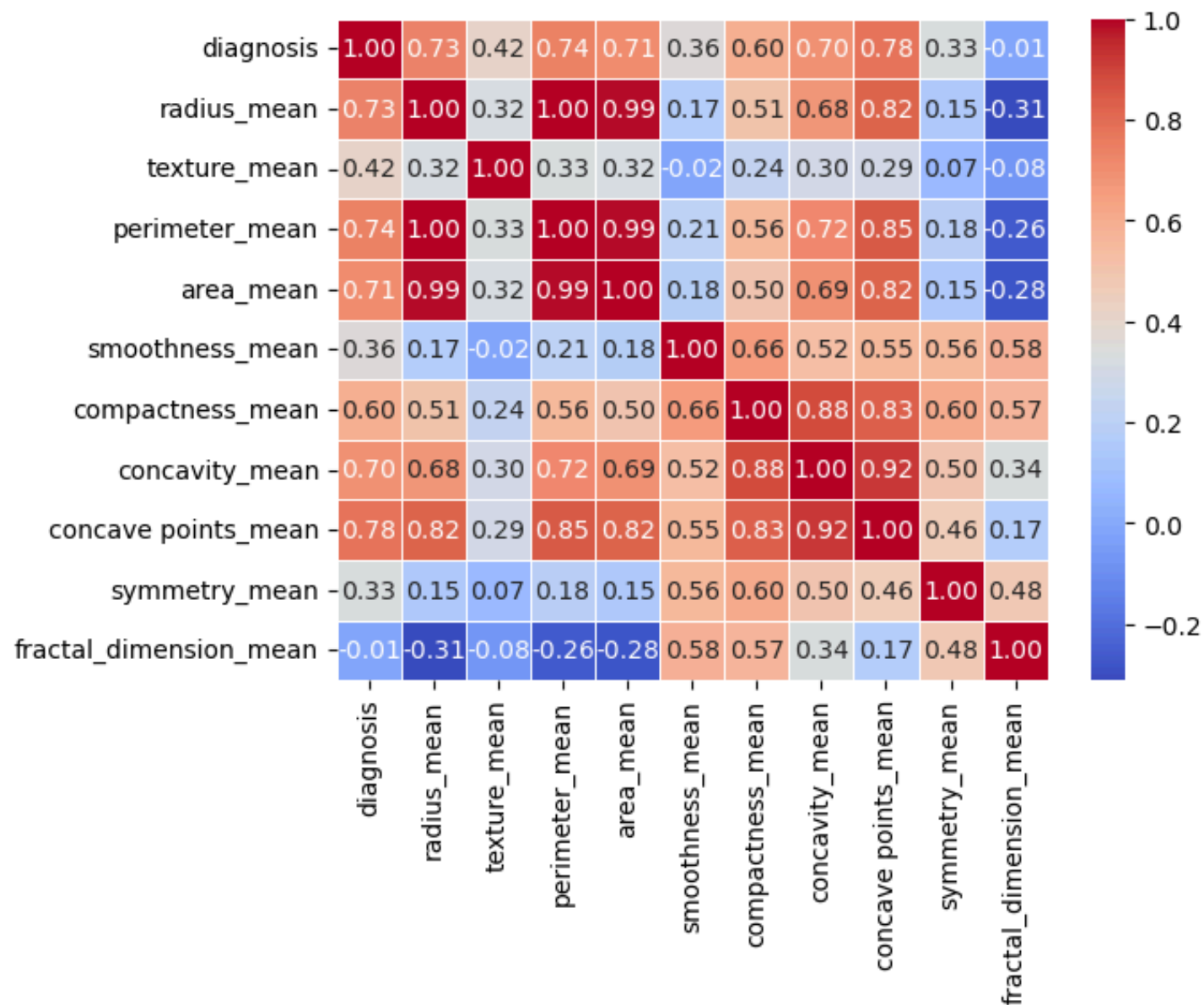
cmap = sns.diverging_palette(220, 20, as_cmap=True) # Red-Blue
sns.heatmap(df[cols].corr(), annot=True, fmt='.1%', linewidths=.05, cmap=cmap)
```

Out[29]: <Axes: title={'center': 'Correlation Graph'}>



r
te
perir
smootl
compac
conc
concave l
symr
fractal_dime

```
In [30]: corr_matrix = df[cols].corr()  
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)  
plt.show()
```



Data modeling

```
In [32]: df.columns
```

```
Out[32]: Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',  
              'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',  
              'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',  
              'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',  
              'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',  
              'fractal_dimension_se', 'radius_worst', 'texture_worst',  
              'perimeter_worst', 'area_worst', 'smoothness_worst',  
              'compactness_worst', 'concavity_worst', 'concave points_worst',  
              'symmetry_worst', 'fractal_dimension_worst'],  
              dtype='object')
```

```
In [33]: prediction_feature = [ "radius_mean",  'perimeter_mean', 'area_mean', 'symmetry_mean', 'compactness_mean', 'concave po  
                                targeted_feature = 'diagnosis'
```

```
In [34]: X = df[prediction_feature]  
X
```

Out[34]:

	radius_mean	perimeter_mean	area_mean	symmetry_mean	compactness_mean	concave points_mean
0	17.99	122.80	1001.0	0.2419	0.27760	0.14710
1	20.57	132.90	1326.0	0.1812	0.07864	0.07017
2	19.69	130.00	1203.0	0.2069	0.15990	0.12790
3	11.42	77.58	386.1	0.2597	0.28390	0.10520
4	20.29	135.10	1297.0	0.1809	0.13280	0.10430
...
564	21.56	142.00	1479.0	0.1726	0.11590	0.13890
565	20.13	131.20	1261.0	0.1752	0.10340	0.09791
566	16.60	108.30	858.1	0.1590	0.10230	0.05302
567	20.60	140.10	1265.0	0.2397	0.27700	0.15200
568	7.76	47.92	181.0	0.1587	0.04362	0.00000

569 rows × 6 columns

In [35]:

```
print(X.shape)
print(X.values)
```

```
(569, 6)
[[1.799e+01 1.228e+02 1.001e+03 2.419e-01 2.776e-01 1.471e-01]
 [2.057e+01 1.329e+02 1.326e+03 1.812e-01 7.864e-02 7.017e-02]
 [1.969e+01 1.300e+02 1.203e+03 2.069e-01 1.599e-01 1.279e-01]
 ...
 [1.660e+01 1.083e+02 8.581e+02 1.590e-01 1.023e-01 5.302e-02]
 [2.060e+01 1.401e+02 1.265e+03 2.397e-01 2.770e-01 1.520e-01]
 [7.760e+00 4.792e+01 1.810e+02 1.587e-01 4.362e-02 0.000e+00]]
```

In [36]:

```
y = df.diagnosis
```

y

```
Out[36]:
```

0	1
1	1
2	1
3	1
4	1
	..
564	1
565	1
566	1
567	1
568	0

Name: diagnosis, Length: 569, dtype: int32

```
In [37]: print(y.shape)
          print(y.values)
```

[illegible]

Split the dataset into TrainingSet and TestingSet by 33% and set the 42 fixed records

```
In [39]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state= 42)

print(X_train)
print(X_test)
```

	radius_mean	perimeter_mean	area_mean	symmetry_mean	compactness_mean	\
172	15.460	102.50	736.9	0.1966	0.15550	
407	12.850	82.63	514.5	0.1580	0.08316	
56	19.210	125.50	1152.0	0.1917	0.12670	
497	12.470	80.45	480.1	0.1526	0.07630	
301	12.460	80.43	471.3	0.1781	0.10140	
..	
71	8.888	58.79	244.0	0.1902	0.15310	
106	11.640	75.17	412.5	0.1801	0.10170	
270	14.290	90.30	632.6	0.1508	0.02675	
435	13.980	91.12	599.5	0.1669	0.11330	
102	12.180	77.22	458.7	0.1739	0.04038	

	concave	points_mean
172	0.10970	
407	0.01867	
56	0.08994	
497	0.02369	
301	0.03099	
..	...	
71	0.02872	
106	0.03485	
270	0.00625	
435	0.06463	
102	0.01770	

[381 rows x 6 columns]

	radius_mean	perimeter_mean	area_mean	symmetry_mean	compactness_mean	\
204	12.47	81.09	481.9	0.1925	0.10580	
70	18.94	123.60	1130.0	0.1582	0.10290	
131	15.46	101.70	748.9	0.1931	0.12230	
431	12.40	81.47	467.8	0.1811	0.13160	
540	11.54	74.65	402.9	0.1818	0.11200	
..	
141	16.11	105.10	813.0	0.1861	0.11370	
498	18.49	121.30	1068.0	0.1832	0.13170	
7	13.71	90.20	577.9	0.2196	0.16450	
541	14.47	95.81	656.4	0.1872	0.12300	

19	13.54	87.46	566.3	0.1885	0.08129
concave points_mean					
204	0.03821				
70	0.07951				
131	0.08087				
431	0.02799				
540	0.02594				
..	...				
141	0.05943				
498	0.09183				
7	0.05985				
541	0.03890				
19	0.04781				

[188 rows x 6 columns]

```
In [40]: from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

MODEL COMPARSION

Logistic Regression

Logistic Regression is a statistical model used to predict the probability of a binary outcome. It is based on the concept of a linear relationship between the independent features (predictors) and the log-odds of the dependent variable

In this model, the coefficients are estimated using maximum likelihood estimation. Logistic Regression works well when there is a linear relationship between the features and the log-odds of the outcome. It is a simple and interpretable model that outputs probabilities, making it useful for classification tasks where the goal is to predict binary outcomes. With an accuracy of 0.90 and a macro F1-score of 0.90 in this case, it shows strong performance in both classes, balancing precision and recall.


```
In [42]: lr = LogisticRegression()
```

```
In [43]: lr.fit(X_train, y_train)
```

```
Out[43]: LogisticRegression
LogisticRegression()
```

```
In [44]: y_pred_lr = lr.predict(X_test)
y_pred_lr
```

```
Out[44]: array([0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
        1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
        1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
        0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0,
        1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
        0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
        0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0])
```

```
In [45]: from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_lr))
```

	precision	recall	f1-score	support
0	0.93	0.92	0.93	121
1	0.86	0.88	0.87	67
accuracy			0.90	188
macro avg	0.89	0.90	0.90	188
weighted avg	0.91	0.90	0.90	188

```
In [49]: model = LogisticRegression(max_iter=1000) # Model should be defined before using it
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
In [53]: from sklearn.metrics import accuracy_score

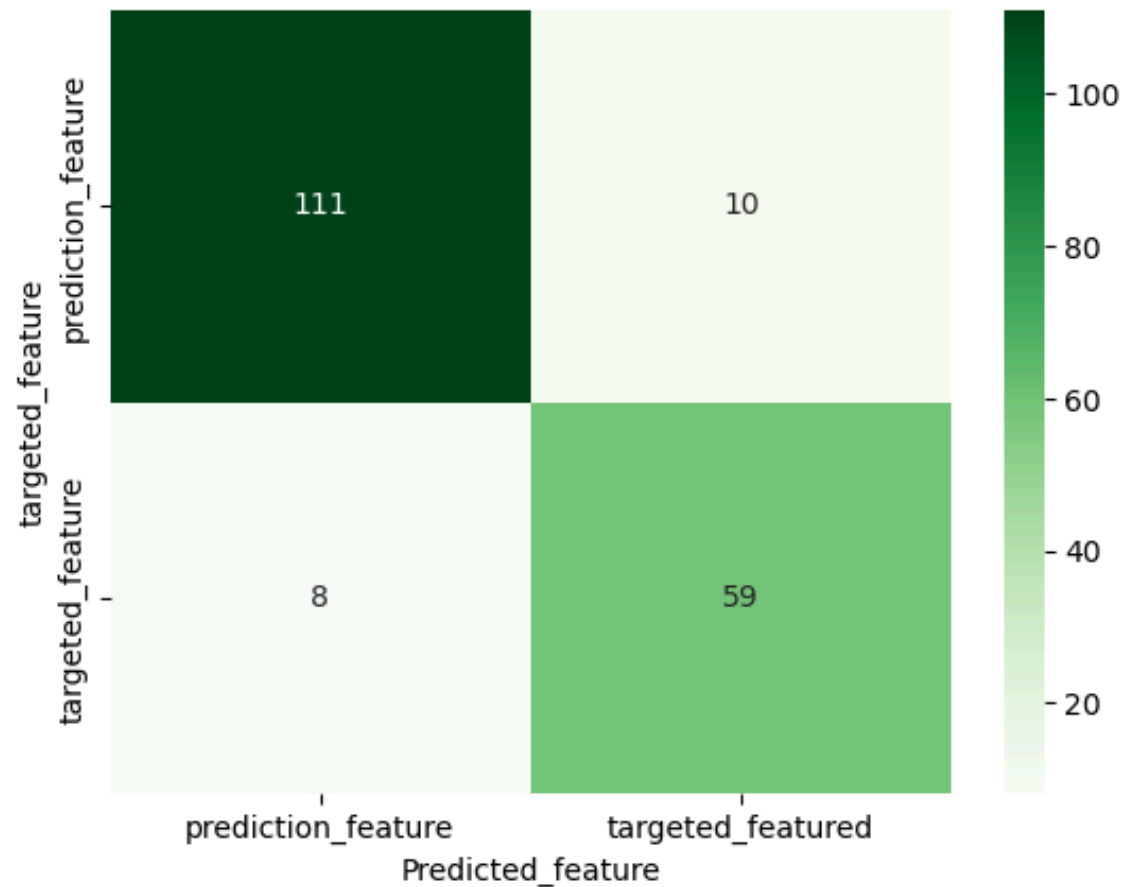
accuracy = accuracy_score(y_test, y_pred)
print(f"Logistic Regression Model Accuracy: {accuracy * 100:.2f}%")
```

Logistic Regression Model Accuracy: 90.43%

```
In [57]: y_train_pred = model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

Training Accuracy: 0.9238845144356955

```
In [59]: cm = confusion_matrix(y_test, y_pred_lr)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap="Greens")
ax.xaxis.set_ticklabels(['prediction_feature', 'targeted_featured'])
ax.yaxis.set_ticklabels(['prediction_feature', 'targeted_feature'])
ax.set_xlabel("Predicted_feature")
ax.set_ylabel("targeted_feature")
plt.show()
```



accuracy of Logistic Regression is 0.90, which means that 90% of the predictions made by the model were correct.

Random Forest Classifier

Random Forest is an ensemble learning technique that improves upon decision trees by building multiple trees and combining their predictions. Each tree is trained on a random subset of the data, and each split in

the tree considers only a random subset of features. This process reduces the variance of the model and helps prevent overfitting

In this case, the Random Forest classifier achieved an accuracy of 0.88 with an F1-score of 0.88, which is slightly lower than that of Logistic Regression and SVC. While its performance is good, the F1-score for Class 1 is 0.85, indicating that it struggles more with the minority class compared to the other models. Nevertheless, Random Forest remains a strong and versatile classifier in various machine learning tasks.

```
In [67]: rf = RandomForestClassifier()  
         rf.fit(X_train, y_train)
```

```
Out[67]: ▼ RandomForestClassifier ⓘ ?  
         RandomForestClassifier()
```

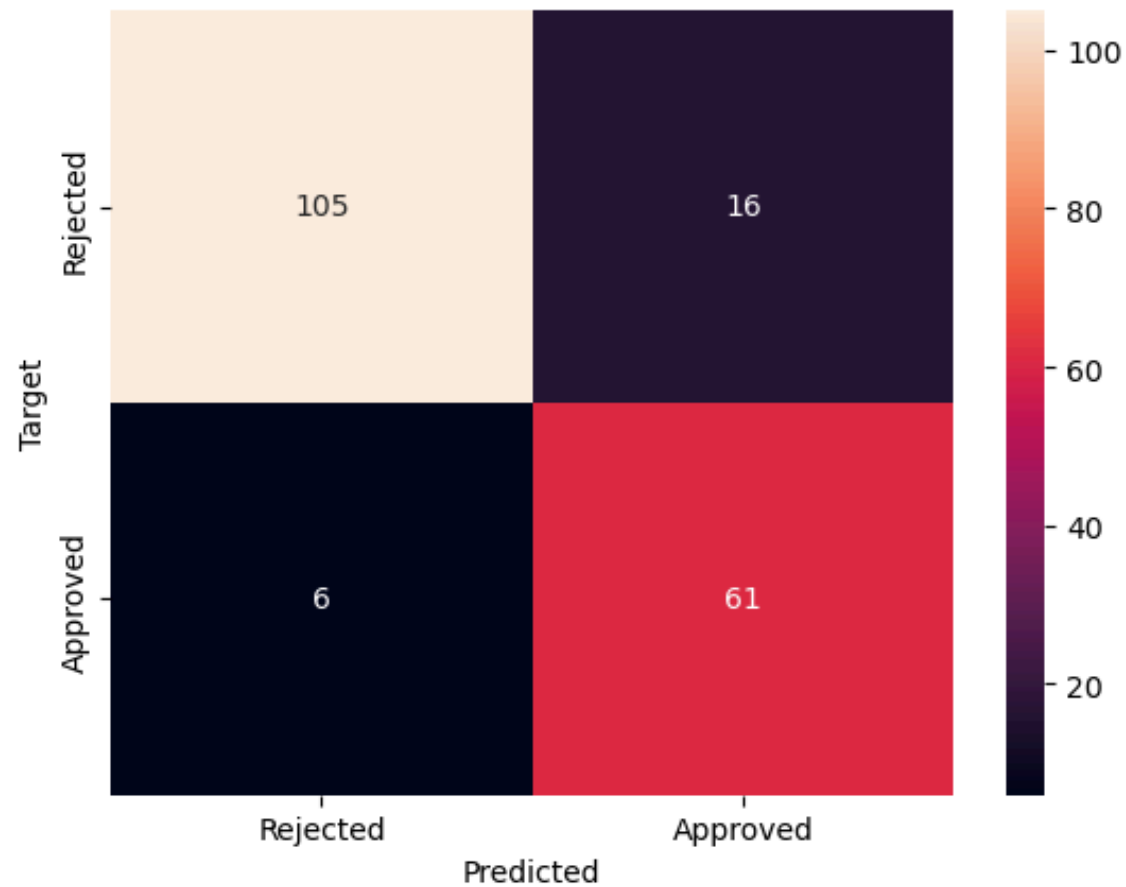
```
In [75]: y_pred_rf = rf.predict(X_test)  
         X_test
```

```
Out[75]: array([[ -4.52924413e-01,  -4.29280059e-01,  -4.90184170e-01,  
                  3.56082717e-01,  9.43106211e-04,  -2.72994620e-01],  
 [ 1.45260534e+00,  1.37249228e+00,  1.50684703e+00,  
 -8.72469340e-01, -5.18476029e-02,  7.61785485e-01],  
 [ 4.27683617e-01,  4.44267999e-01,  3.32539685e-01,  
  3.77573424e-01,  3.01304037e-01,  7.95860569e-01],  
 ...,  
 [-8.77224210e-02, -4.31557107e-02, -1.94373346e-01,  
  1.32674630e+00,  1.06949987e+00,  2.69200089e-01],  
 [ 1.36111058e-01,  1.94622290e-01,  4.75136302e-02,  
  1.66248143e-01,  3.14046622e-01, -2.55706526e-01],  
 [-1.37790436e-01, -1.59289708e-01, -2.30117154e-01,  
  2.12811340e-01, -4.45229404e-01, -3.24646199e-02]])
```

```
In [77]: print(classification_report(y_test, y_pred_rf))
```

	precision	recall	f1-score	support
0	0.95	0.87	0.91	121
1	0.79	0.91	0.85	67
accuracy			0.88	188
macro avg	0.87	0.89	0.88	188
weighted avg	0.89	0.88	0.88	188

```
In [79]: cm = confusion_matrix(y_test, y_pred_rf)
ax = sns.heatmap(cm, annot=True, fmt='d')
ax.xaxis.set_ticklabels(['Rejected', 'Approved'])
ax.yaxis.set_ticklabels(['Rejected', 'Approved'])
ax.set_xlabel("Predicted")
ax.set_ylabel("Target")
plt.show()
```



```
In [81]: accuracy = accuracy_score(y_test, y_pred)
print(f"RandomForestClassifier Model Accuracy: {accuracy * 100:.2f}%")
```

RandomForestClassifier Model Accuracy: 90.43%

```
In [83]: y_train_pred = model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

Training Accuracy: 0.9238845144356955

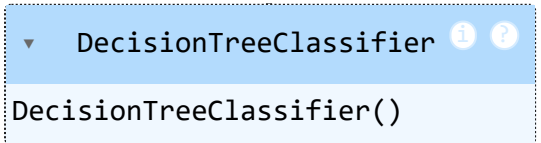
The accuracy of Random Forest is 0.88, meaning that 88% of the model's predictions were correct

Decision Tree Classifier

Decision Trees are a non-linear, non-parametric supervised learning algorithm that partition the feature space into distinct regions. They are represented as a tree-like structure where each internal node represents a decision based on a feature, and each leaf node represents a class label. The algorithm works by recursively splitting the dataset based on feature values that result in the best separation of the classes. The objective is to maximize the information gain or minimize the impurity (e.g., Gini index or entropy) at each split.

In this analysis, the Decision Tree classifier achieved an accuracy of 0.86 and an F1-score of 0.85, which indicates that while it performs well, its performance is slightly lower compared to Logistic Regression and SVC. Decision trees tend to be sensitive to small changes in the data, which can explain their lower performance, especially for Class 1, where the F1-score is 0.82.

```
In [87]: dt = DecisionTreeClassifier()  
         dt.fit(X_train, y_train)
```

```
Out[87]:  DecisionTreeClassifier  
DecisionTreeClassifier()
```

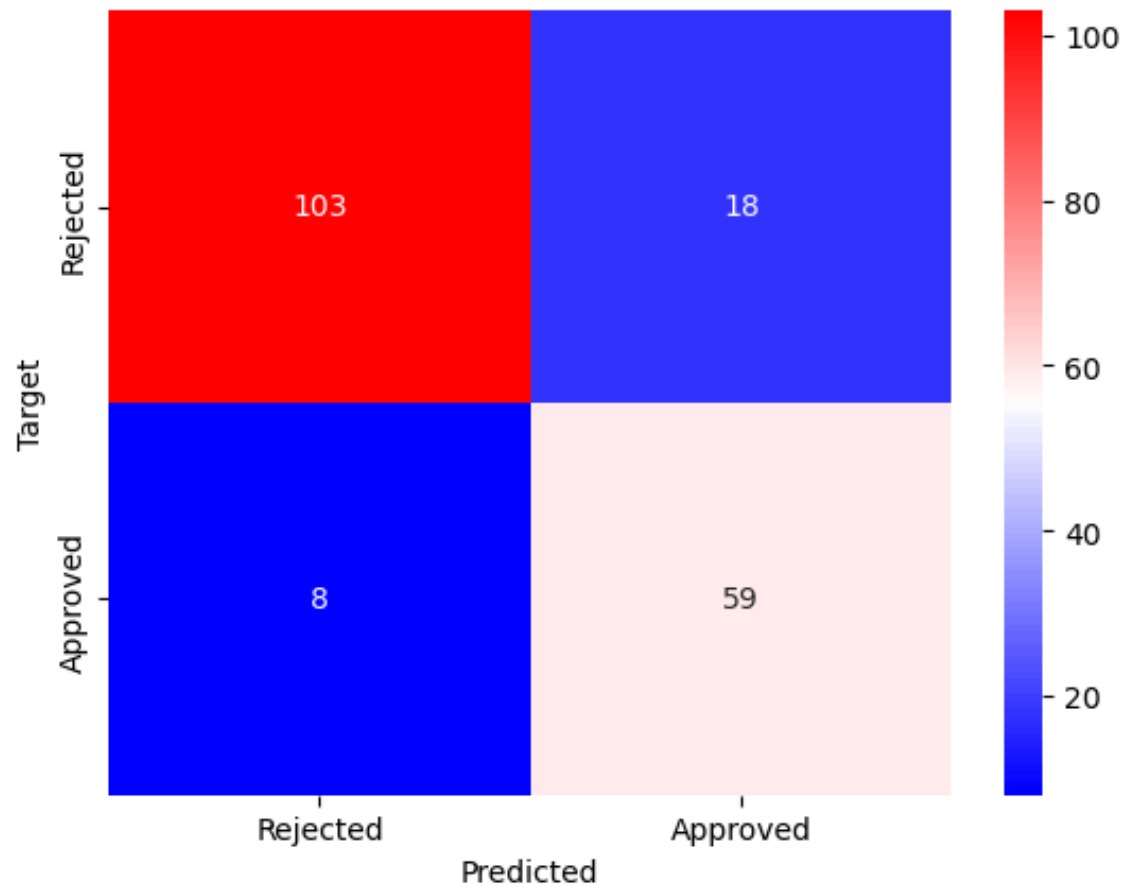
```
In [89]: y_pred_dt = dt.predict(X_test)  
         X_test
```

```
Out[89]: array([[ -4.52924413e-01, -4.29280059e-01, -4.90184170e-01,
        3.56082717e-01,  9.43106211e-04, -2.72994620e-01],
       [ 1.45260534e+00,  1.37249228e+00,  1.50684703e+00,
       -8.72469340e-01, -5.18476029e-02,  7.61785485e-01],
       [ 4.27683617e-01,  4.44267999e-01,  3.32539685e-01,
        3.77573424e-01,  3.01304037e-01,  7.95860569e-01],
       ...,
       [-8.77224210e-02, -4.31557107e-02, -1.94373346e-01,
        1.32674630e+00,  1.06949987e+00,  2.69200089e-01],
       [ 1.36111058e-01,  1.94622290e-01,  4.75136302e-02,
        1.66248143e-01,  3.14046622e-01, -2.55706526e-01],
       [-1.37790436e-01, -1.59289708e-01, -2.30117154e-01,
        2.12811340e-01, -4.45229404e-01, -3.24646199e-02]])
```

```
In [91]: print(classification_report(y_test,y_pred_dt))
```

	precision	recall	f1-score	support
0	0.93	0.85	0.89	121
1	0.77	0.88	0.82	67
accuracy			0.86	188
macro avg	0.85	0.87	0.85	188
weighted avg	0.87	0.86	0.86	188

```
In [103... cm = confusion_matrix(y_test, y_pred_dt)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap = "bwr")
ax.xaxis.set_ticklabels(['Rejected', 'Approved'])
ax.yaxis.set_ticklabels(['Rejected', 'Approved'])
ax.set_xlabel("Predicted")
ax.set_ylabel("Target")
plt.show()
```

```
In [109... accuracy = accuracy_score(y_test, y_pred)
print(f"DecisionTreeClassifier Model Accuracy: {accuracy * 100:.2f}%")
```

DecisionTreeClassifier Model Accuracy: 90.43%

```
In [111... y_train_pred = model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

Training Accuracy: 0.9238845144356955

```
In [115... # Predict on test data
y_pred = dt.predict(X_test)
```

```
test_accuracy = accuracy_score(y_test, y_pred)
print("Decision Tree Test Accuracy:", test_accuracy)
```

Decision Tree Test Accuracy: 0.8723404255319149

The accuracy of the Decision Tree classifier is 0.86, which is the lowest among the models tested. This means 86% of the predictions were correct.

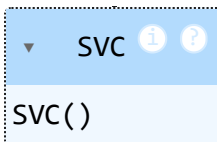
Support Vector Classification (SVC)

Support Vector Classification (SVC) is a supervised machine learning algorithm that belongs to the class of Support Vector Machines (SVMs). SVMs work by finding the hyperplane in a high-dimensional space that best separates the classes with the largest margin. The idea is to maximize the distance between the closest points of the classes (known as support vectors) and the decision boundary. The algorithm can use different kernel functions to handle non-linear data by implicitly mapping the data to a higher-dimensional space.

With an accuracy of 0.90 and an F1-score of 0.90 in this analysis, SVC provides balanced performance. Its results are comparable to Logistic Regression, showing strong predictive capabilities, especially in Class 1, where it achieved an F1-score of 0.87. SVC is a robust classifier, especially when dealing with complex datasets and non-linear decision boundaries.

```
In [121... svc = SVC()
svc.fit(X_train, y_train)
```

Out[121...

The image shows a Jupyter Notebook output cell for the SVC model. It features a blue header bar with a dropdown arrow, the text 'SVC', and two circular icons (one with an 'i' and one with a '?'). Below the header, the text 'SVC()' is displayed in a light blue box.

▼ SVC ⓘ ?
SVC()

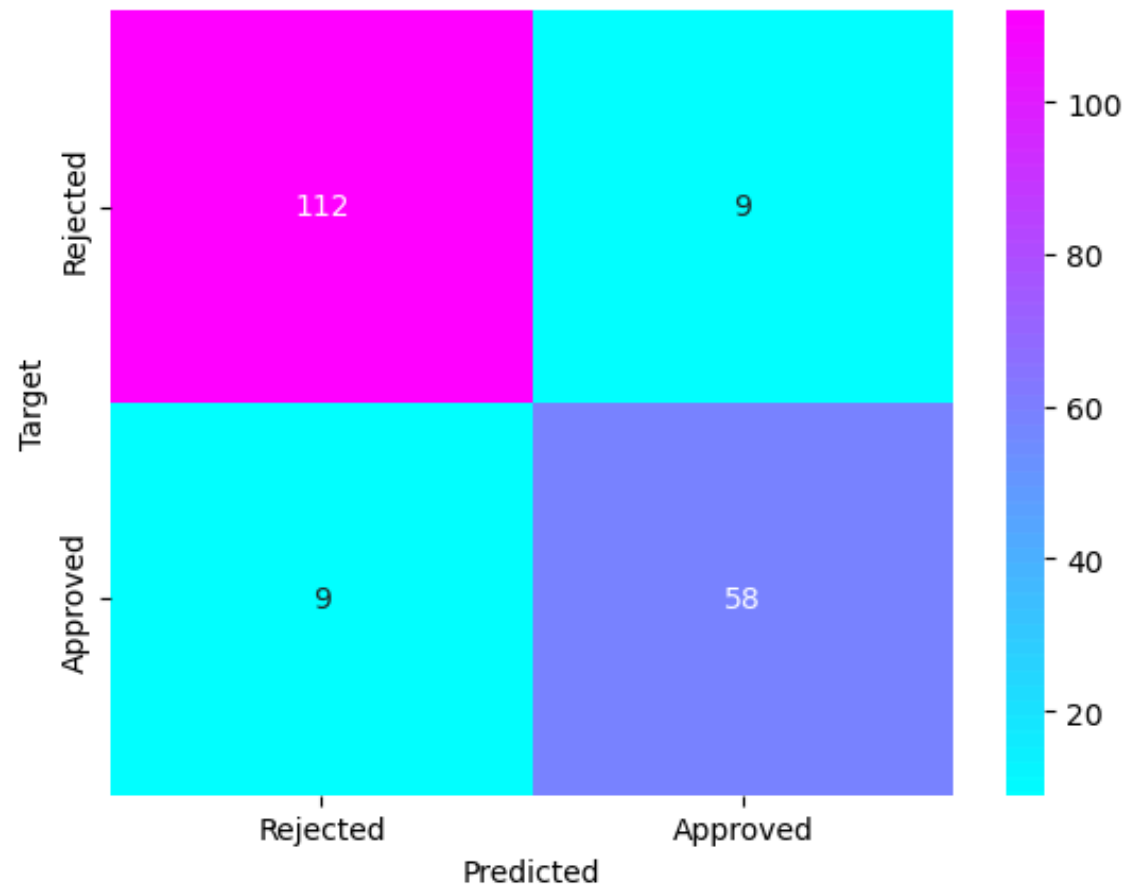
```
In [125... y_pred_svc = svc.predict(X_test)
X_test
```

```
Out[125... array([[ -4.52924413e-01, -4.29280059e-01, -4.90184170e-01,
          3.56082717e-01,  9.43106211e-04, -2.72994620e-01],
 [ 1.45260534e+00,  1.37249228e+00,  1.50684703e+00,
 -8.72469340e-01, -5.18476029e-02,  7.61785485e-01],
 [ 4.27683617e-01,  4.44267999e-01,  3.32539685e-01,
  3.77573424e-01,  3.01304037e-01,  7.95860569e-01],
 ...,
 [-8.77224210e-02, -4.31557107e-02, -1.94373346e-01,
  1.32674630e+00,  1.06949987e+00,  2.69200089e-01],
 [ 1.36111058e-01,  1.94622290e-01,  4.75136302e-02,
  1.66248143e-01,  3.14046622e-01, -2.55706526e-01],
 [-1.37790436e-01, -1.59289708e-01, -2.30117154e-01,
  2.12811340e-01, -4.45229404e-01, -3.24646199e-02]])
```

```
In [127... print(classification_report(y_test,y_pred_svc))
```

	precision	recall	f1-score	support
0	0.93	0.93	0.93	121
1	0.87	0.87	0.87	67
accuracy			0.90	188
macro avg	0.90	0.90	0.90	188
weighted avg	0.90	0.90	0.90	188

```
In [131... cm = confusion_matrix(y_test, y_pred_svc)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap="cool")
ax.xaxis.set_ticklabels(['Rejected', 'Approved'])
ax.yaxis.set_ticklabels(['Rejected', 'Approved'])
ax.set_xlabel("Predicted")
ax.set_ylabel("Target")
plt.show()
```



```
In [133... accuracy = accuracy_score(y_test, y_pred_svc)
print(f"Support Vector Classifier Model Accuracy: {accuracy * 100:.2f}%")
```

Support Vector Classifier Model Accuracy: 90.43%

```
In [135... y_train_pred = svc.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

Training Accuracy: 0.926509186351706

The accuracy of Support Vector Classification (SVC) is 0.90, the same as Logistic Regression. This means that 90% of the predictions made by the model were correct. SVC

Conclusion

Model	Accuracy	Macro F1-Score	Best for
Logistic Regression	0.90	0.90	Balanced performance, interpretability
Random Forest	0.88	0.88	Handling complex relationships
Decision Tree	0.86	0.85	Interpretability, simplicity
Support Vector Classifier (SVC)	0.90	0.90	Best overall classification

In this project, I applied four different machine learning models—Logistic Regression, Random Forest, Decision Tree, and Support Vector Classifier (SVC)—to classify the dataset and evaluate their effectiveness. The primary goal was to identify which model provides the best accuracy and balanced performance across different metrics, including precision, recall, and F1-score. After running all four models and analyzing the results, I found that Logistic Regression and SVC performed the best in terms of overall accuracy and F1-score. In this study, machine learning models were applied to predict breast cancer diagnosis using a dataset collected from Kaggle. The dataset contained multiple features related to the characteristics of cell nuclei observed in breast cancer biopsies, including both mean and worst measurements across various categories. The methodology involved crucial steps such as data collection, data cleaning, exploratory data analysis (EDA), data visualization, and model building. After preprocessing, including the use of label encoding and feature scaling, the dataset was split into training and testing sets. Various machine learning models were trained, including Logistic Regression, Decision Tree Classifier, Random Forest Classifier, and Support Vector Classification (SVC). These models

were evaluated based on accuracy and F1-score metrics. The results revealed that Logistic Regression and Support Vector Classification (SVC) performed the best, achieving an accuracy of 0.90 and a balanced F1-score of 0.90. These models showed strong performance in predicting both classes (malignant and benign). On the other hand, Random Forest and Decision Tree classifiers performed well but slightly lagged behind in their F1-score for class 1, with Decision Tree showing the lowest overall accuracy of 0.86.

Model Performance Summary

1 Logistic Regression achieved an accuracy of 90%, with a macro F1-score of 0.90. This indicates that it performed well in distinguishing between both classes. It also maintained a strong balance between precision and recall, making it a reliable model for this classification task. Support Vector Classifier (SVC) also obtained an accuracy of 90% and a macro F1-score of 0.90, which is on par with Logistic Regression. This suggests that SVC is equally effective in handling this dataset and provides strong generalization capabilities. Random Forest achieved 88% accuracy, which is slightly lower than Logistic Regression and SVC. However, it demonstrated a good ability to capture complex relationships within the data. Despite its high predictive power, it did not outperform the simpler models in this case. Decision Tree had the lowest performance, with 86% accuracy and a macro F1-score of 0.85. This suggests that while it is interpretable and easy to understand, it may not generalize as well as the other models, leading to potential overfitting.

Best Model Selection

Based on the results, Logistic Regression and SVC stand out as the most effective models for this classification task. Both models provided high accuracy, strong precision-recall balance, and a high F1-score, making them ideal choices. Additionally, Logistic Regression is widely used for classification tasks due to its simplicity and interpretability, whereas SVC is known for its ability to handle complex decision boundaries. Random Forest, while powerful, did not outperform these models in terms of accuracy. However, it remains a strong option for datasets with more non-linearity. Decision Tree, on the other hand,

showed the lowest accuracy and F1-score, suggesting that it might not be the best fit for this particular dataset unless further optimization is performed.

Conclusion and Future Directions

This project provided valuable insights into different machine learning models and their effectiveness in classification tasks. After testing four models, I found that Logistic Regression and Support Vector Classifier performed the best, both achieving 90% accuracy and strong F1-scores. While Random Forest and Decision Tree also demonstrated good predictive capabilities, they were slightly less effective in comparison. Although this marks the completion of the initial phase of the project, there is room for further refinement and improvement, particularly in areas such as hyperparameter tuning, feature selection, and testing more advanced models. Future work can focus on improving model generalization, ensuring robustness, and ultimately deploying the best-performing model for practical applications. This project has strengthened my understanding of model evaluation, performance metrics, and classification techniques. Moving forward, I aim to explore more advanced models, feature engineering techniques, and real-world deployment to enhance the impact of this work.

REFERENCES

1. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning: with applications in R. Springer. <https://doi.org/10.1007/978-1-4614-7138-7>
2. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
<http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf>
3. Zhang, H., & Zhang, H. (2022). Logistic regression: An overview. International Journal of Machine Learning and Computing, 12(1), 40-45.
<https://doi.org/10.18178/ijmlc.2022.12.1.1111>

4. Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.
<https://doi.org/10.1023/A:1010933404324>
5. Chen, J., & Liu, J. (2020). A comprehensive review of support vector machines for classification problems. Neurocomputing, 380, 106-119.
<https://doi.org/10.1016/j.neucom.2019.10.052>
6. Brownlee, J. (2020). Ordinal and one-hot encodings for categorical data. Machine Learning Mastery. Retrieved from <https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/>
7. García, S., Luengo, J., & Herrera, F. (2015). Data preprocessing in data mining. Springer.
8. Grus, J. (2015). Data science from scratch. O'Reilly Media.
9. Han, J., Kamber, M., & Pei, J. (2011). Data mining: Concepts and techniques. Elsevier.
10. Wolberg, W. H., & Mangasarian, O. L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. Proceedings of the National Academy of Sciences, 87(23), 9193-9196. <https://doi.org/10.1073/pnas.87.23.9193>
11. Zhang, Z., Li, X., & Zhao, Y. (2020). A review of machine learning-based approaches for breast cancer prediction and diagnosis. Frontiers in Bioengineering and Biotechnology, 8, 531. <https://doi.org/10.3389/fbioe.2020.00531>
12. Janiezzj. (n.d.). Breast cancer analysis using machine learning [Dataset]. Kaggle. Retrieved March 11, 2025, from <https://www.kaggle.com/code/janiezzj/breast-cancer-analysis-using-machine-learning>

13. McKinney, W. (2010). Data structures for statistical computing in Python. In Proceedings of the 9th Python in Science Conference (pp. 51-56). <https://doi.org/10.25080/Majora-92bf1922-00a>
14. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362. <https://doi.org/10.1038/s41586-020-2649-2>
15. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95. <https://doi.org/10.1109/MCSE.2007.55>
16. Waskom, M. L. (2021). Seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021. <https://doi.org/10.21105/joss.03021>
17. Python Software Foundation. (2020). os — Miscellaneous operating system interfaces. Python 3.9.1 documentation. <https://docs.python.org/3/library/os.html>
18. Plotly Technologies Inc. (2015). Plotly.py: An open-source graphing library for Python. <https://plotly.com/python/>
19. Plotly Technologies Inc. (2015). Plotly.graph_objects (Version 5.0). <https://plotly.com/python/plotly-express/>
20. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830. <http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf>
21. RobustScaler, PCA, StratifiedKFold, GridSearchCV, RFECV, and other modules in sklearn Scikit-learn developers. (2021). Scikit-learn: Machine Learning in Python. <https://scikit-learn.org/stable/>

22. Scikit-learn developers. (2021). DecisionTreeClassifier. <https://scikitlearn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
23. Scikit-learn developers. (2021). LogisticRegression. https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
24. Scikit-learn developers. (2021). Support Vector Machines (SVM). <https://scikitlearn.org/stable/modules/svm.html>
25. Scikit-learn developers. (2021). RandomForestClassifier. <https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
26. Scikit-learn developers. (2021). KNeighborsClassifier. <https://scikitlearn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>
27. Scikit-learn developers. (2021). Metrics and scoring: quantifying the quality of models. https://scikit-learn.org/stable/modules/model_evaluation.html
28. Plotly Technologies Inc. (2015). plotly.subplots (Version 5.0). <https://plotly.com/python/subplots/>
29. Fujita, A., Yamashita, R., Nishio, M., & Togashi, K. (2019). AI-based computer-aided diagnosis for breast cancer: The current and future status. Japanese Journal of Radiology, 37, 202–217. <https://doi.org/10.1007/s11604-019-00833-6>
30. Waks, A. G., & Winer, E. P. (2019). Breast cancer treatment: A review. JAMA, 321(3), 288–300. <https://doi.org/10.1001/jama.2018.19323>
31. Wang, D., Khosla, A., Gargeya, R., Irshad, H., & Beck, A. H. (2020). Deep learning for identifying metastatic breast cancer. Advances in Neural Information Processing Systems,

30, 3050–3058. <https://arxiv.org/abs/1606.05718>

32. Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., & Luecke, J. (2021). Seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60), 3021. <https://doi.org/10.21105/joss.03021>

33. Chen, H., Wang, N., Du, X., Mei, K., Zhou, Y., & Cai, G. (2023). Classification prediction of breast cancer based on machine learning. *Computational and Mathematical Methods in Medicine*, 2023, 6530719. <https://doi.org/10.1155/2023/6530719>

In []: