Breast Cancer Wisconsin (Diagnostic) Data Set

Breast Cancer Wisconsin (Diagnostic) Data Set

The Breast Cancer Wisconsin (Diagnostic) Data Set is a dataset used for classification tasks, specifically for diagnosing breast cancer. The dataset contains information about various features computed for cell nuclei found in breast cancer biopsies. The goal is to predict whether a cell nucleus is malignant (indicating cancerous cells) or benign (indicating non-cancerous cells).

Attribute Information:

1. **ID number:** Unique identifier for each biopsy sample.
2. **Diagnosis:** The target variable, indicating the diagnosis. Malignant (M) represents cancerous cells, and Benign (B) represents non-cancerous cells. 3-12. **Ten real-valued features:** These features are computed for each cell nucleus: a) **Radius:** The mean of distances from the center to points on the perimeter. b) **Texture:** The standard deviation of gray-scale values. c) **Perimeter:** The perimeter of the cell nucleus. d) **Area:** The area of the cell nucleus. e) **Smoothness:** The local variation in radius lengths. f) **Compactness:** Calculated as (perimeter^2 / area - 1.0). g) **Concavity:** Indicates the severity of concave portions of the contour. h) **Concave Points:** The number of concave portions of the contour. i) **Symmetry:** A measure of symmetry in the cell nucleus. j) **Fractal Dimension:** Represents the "coastline approximation" - 1.

Data Description:

The dataset contains a total of 30 features. For each feature, there are three values recorded for each image: mean, standard error, and "worst" or largest (mean of the three largest values). Therefore, there are a total of 30 * 3 = 90 columns, but you may use only the mean values

(fields 3-12) for your analysis if needed.

Missing Attribute Values:

There are no missing attribute values in this dataset.

Class Distribution:

The dataset is imbalanced, with the following class distribution:

- Benign (B): 357 samples
- Malignant (M): 212 samples

Usage:

This dataset is widely used in machine learning and data analysis for tasks related to breast cancer classification. The goal is to build a model that can accurately distinguish between benign and malignant cells based on the given features.

References:

1. W.N. Street, W.H. Wolberg, and O.L. Mangasarian. "Nuclear feature extraction for breast tumor diagnosis." IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.
2. Dataset source: UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29 (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29))

Model Overview:

In this notebook, we used several machine learning algorithms to classify the Breast Cancer Wisconsin (Diagnostic) Data Set. The goal was to build predictive models that can accurately distinguish between benign and malignant breast cancer cells based on the provided features.

1. Logistic Regression: Logistic Regression is a simple and widely used classification algorithm. It models the probability of a binary outcome (in this case, benign or malignant) using a logistic function. It is suitable for problems with a linear decision boundary. Logistic Regression is interpretable and computationally efficient.

2. Decision Tree: Decision Trees are non-linear models that recursively partition the data into subsets based on the feature values. At each node, the algorithm selects the feature that best splits the data and creates branches accordingly. Decision Trees are easy to interpret, but they can be prone to overfitting.

3. Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. It builds several decision trees using bootstrap samples of the data and random feature subsets. The final prediction is obtained by averaging the predictions of individual trees, reducing overfitting and improving accuracy.

4. Bagging Algorithm (Bootstrap Aggregating): Bagging is an ensemble learning technique that reduces variance by creating multiple copies of the training dataset through bootstrapping. It trains each model on a different subset of the data and then combines their predictions.

Bagging works well with unstable models and can improve accuracy.

5. K-Nearest Neighbors (KNN) Algorithm: K-Nearest Neighbors is a simple and intuitive classification algorithm. It assigns a data point to the class based on the majority class of its k-nearest neighbors. The value of k is a hyperparameter that affects the model's performance. KNN is lazy learning and requires minimal training, but it can be computationally expensive during prediction.

6. Support Vector Machine (SVM): Support Vector Machine is a powerful classification algorithm used for both linear and non-linear data. It aims to find the optimal hyperplane that maximizes the margin between different classes. SVM can handle high-dimensional data well and is effective for binary classification tasks.

Importing Libraries

In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')


pd.set_option("display.max_rows", 10000)
pd.set_option('display.max_columns',15)
pd.set_option('display.width', 10000)
```

In [2]:
```
dataset = pd.read_csv('breast-cancer.csv')
dataset
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smooth
0	842302	M	17.990	10.38	122.80	1001.0	
1	842517	M	20.570	17.77	132.90	1326.0	
2	84300903	M	19.690	21.25	130.00	1203.0	
3	84348301	M	11.420	20.38	77.58	386.1	
4	84358402	M	20.290	14.34	135.10	1297.0	
5	843786	M	12.450	15.70	82.57	477.1	
6	844359	M	18.250	19.98	119.60	1040.0	
7	84458202	M	13.710	20.83	90.20	577.9	
8	844981	M	13.000	21.82	87.50	519.8	
9	84501001	M	12.460	24.04	83.97	475.9	

In [3]:
```
dataset.describe()
```

Out[3]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.0000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.0963
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.0140
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.0526
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.0863
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.0958
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.1053
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.1634

8 rows × 31 columns

In [4]:
```
dataset.columns
```

Out[4]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'a
rea_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave
points_mean', 'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'textur
e_se', 'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se', 'concavi
ty_se', 'concave points_se', 'symmetry_se', 'fractal_dimension_se', 'radius_w
orst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst',
'compactness_worst', 'concavity_worst', 'concave points_worst', 'symmetry_wor
st', 'fractal_dimension_worst'], dtype='object')
```

Data Preprocessing

In [5]:
```
dataset.isnull().sum()
```

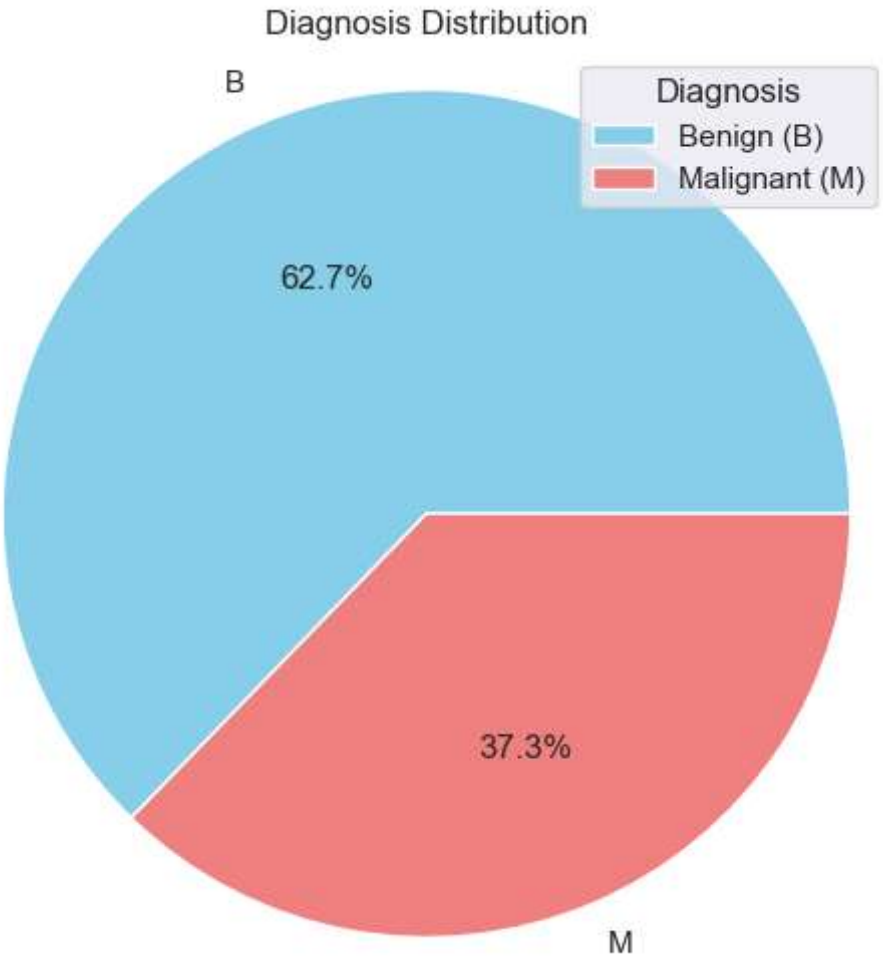
Out[5]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

In [6]:
```
dataset.info()
```

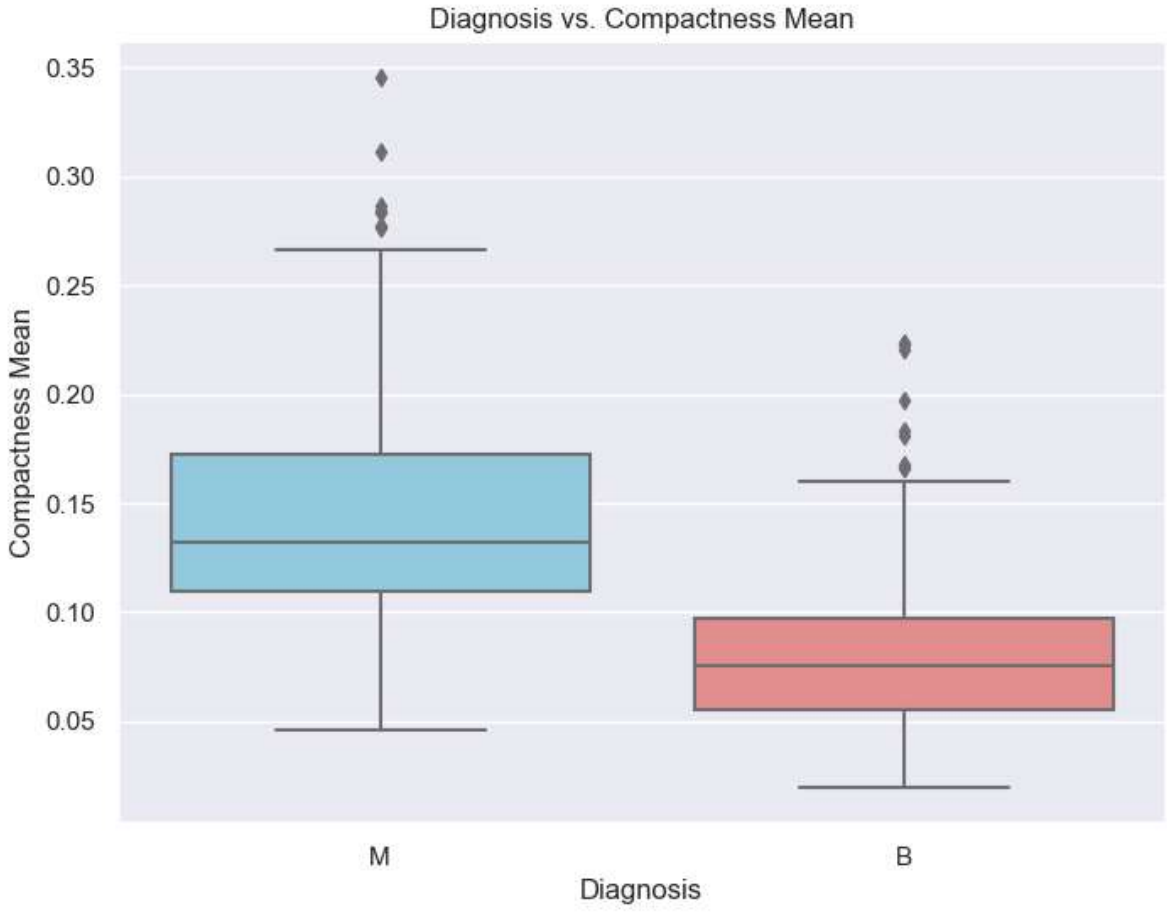
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

Visualization

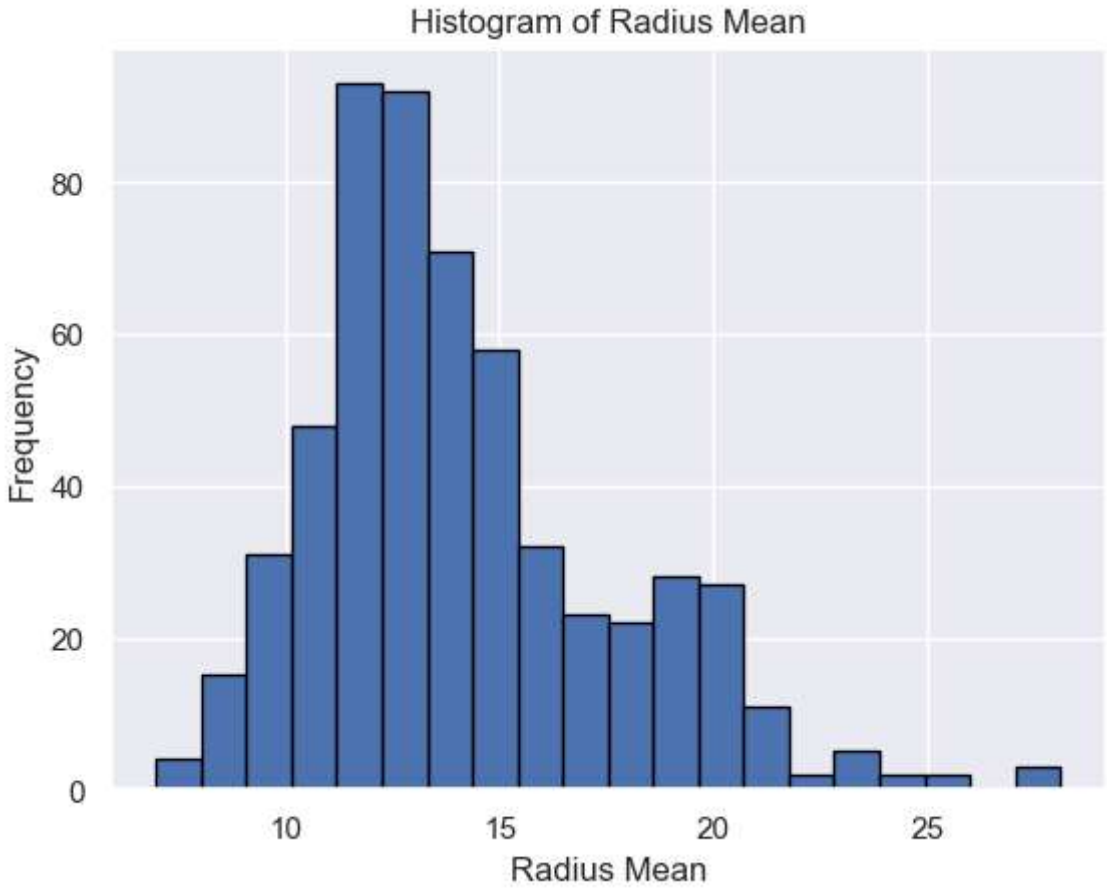
In [7]:
```
diagnosis_counts = dataset['diagnosis'].value_counts()
plt.figure(figsize=(6, 6))
plt.pie(diagnosis_counts, labels=diagnosis_counts.index, autopct='%1.1f%%', cc
plt.title('Diagnosis Distribution')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.legend(title='Diagnosis', labels=['Benign (B)', 'Malignant (M)'], loc='upp


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



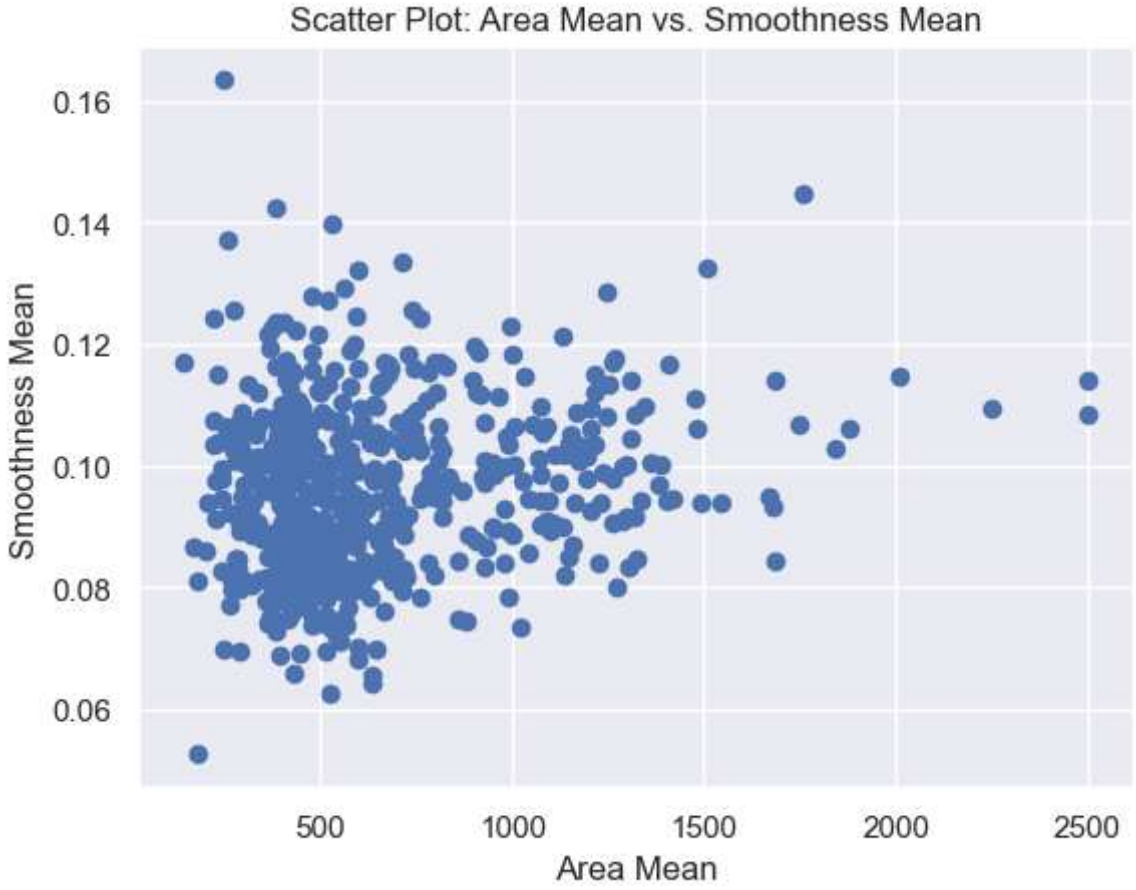
In [8]:
```
plt.figure(figsize=(8, 6))
sns.boxplot(x='diagnosis', y='compactness_mean', data=dataset, palette=['skybl
plt.title('Diagnosis vs. Compactness Mean')
plt.xlabel('Diagnosis')
plt.ylabel('Compactness Mean')

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



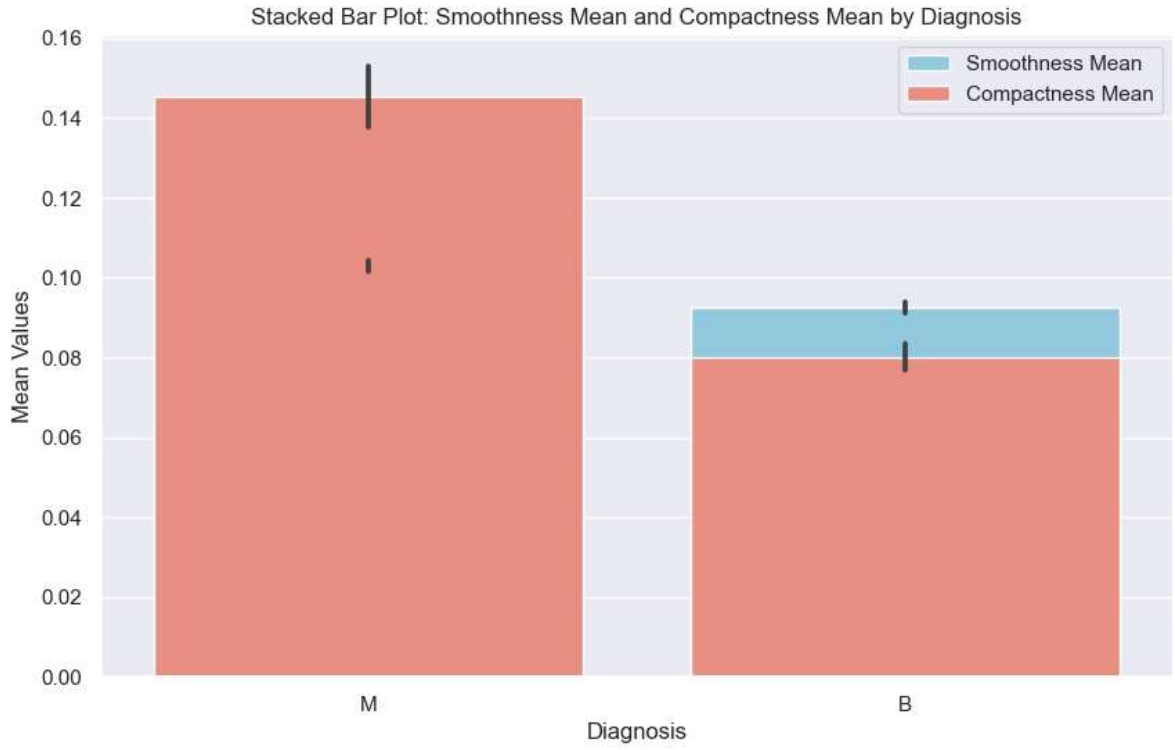
In [9]:
```
import matplotlib.pyplot as plt

plt.hist(dataset['radius_mean'], bins=20, edgecolor='black')
plt.xlabel('Radius Mean')
plt.ylabel('Frequency')
plt.title('Histogram of Radius Mean')
plt.show()
```



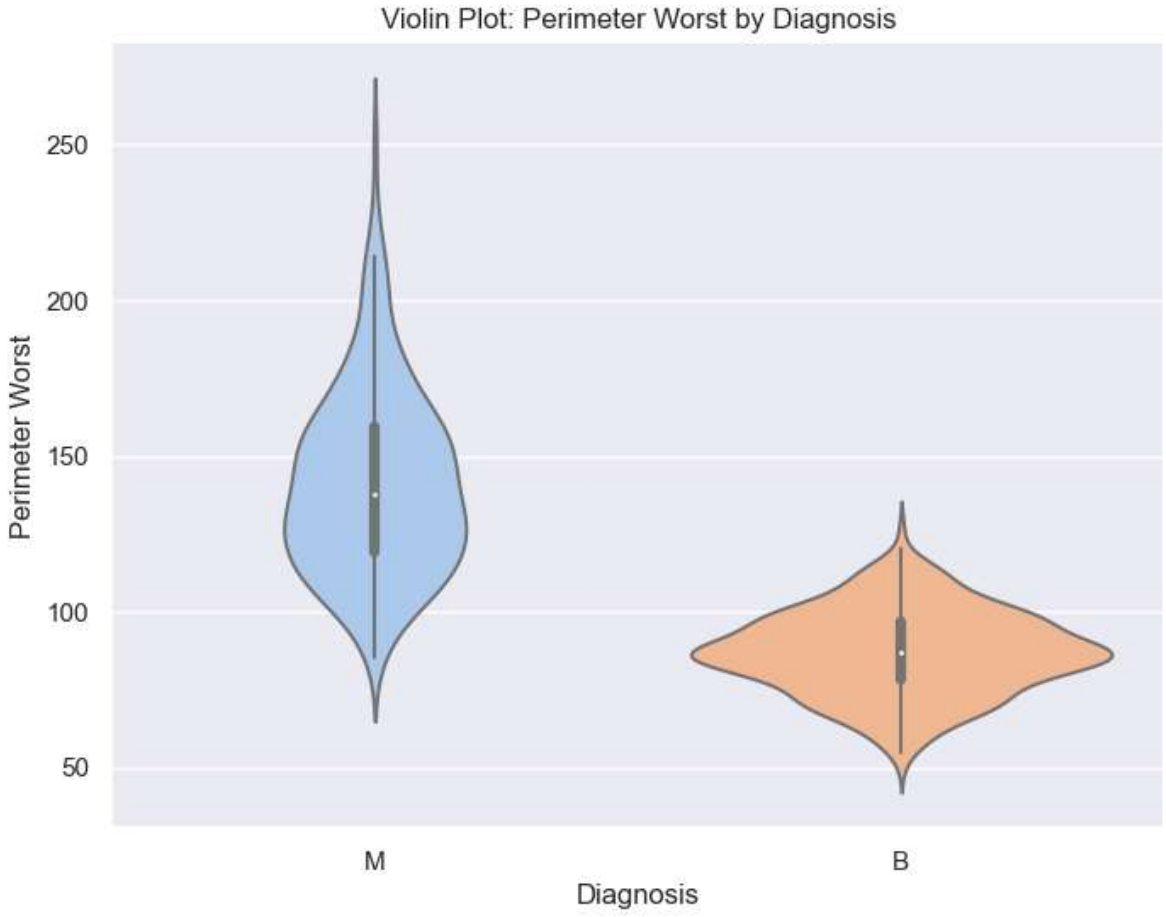
In [10]:
```
plt.scatter(dataset['area_mean'], dataset['smoothness_mean'])
plt.xlabel('Area Mean')
plt.ylabel('Smoothness Mean')
plt.title('Scatter Plot: Area Mean vs. Smoothness Mean')
plt.show()
```



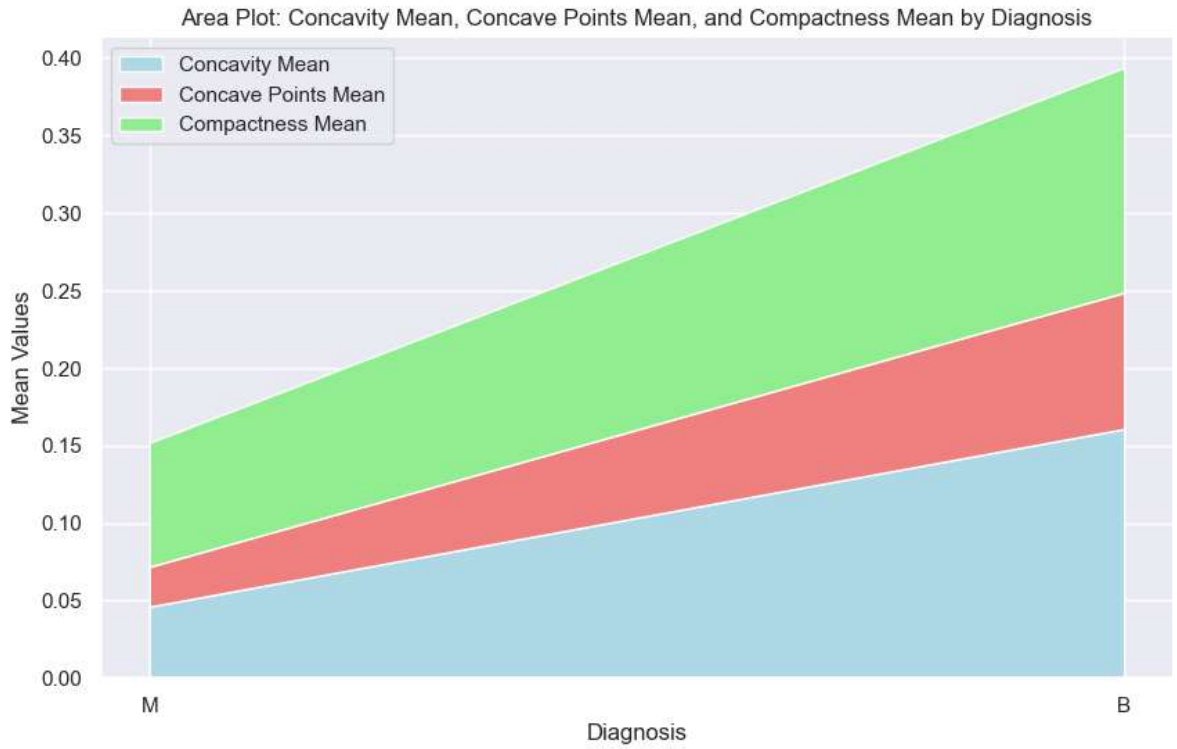
In [11]:
```
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.barplot(x='diagnosis', y='smoothness_mean', data=dataset, color='skyblue',
sns.barplot(x='diagnosis', y='compactness_mean', data=dataset, color='salmon',
plt.xlabel('Diagnosis')
plt.ylabel('Mean Values')
plt.title('Stacked Bar Plot: Smoothness Mean and Compactness Mean by Diagnosis
plt.legend()
plt.show()
```



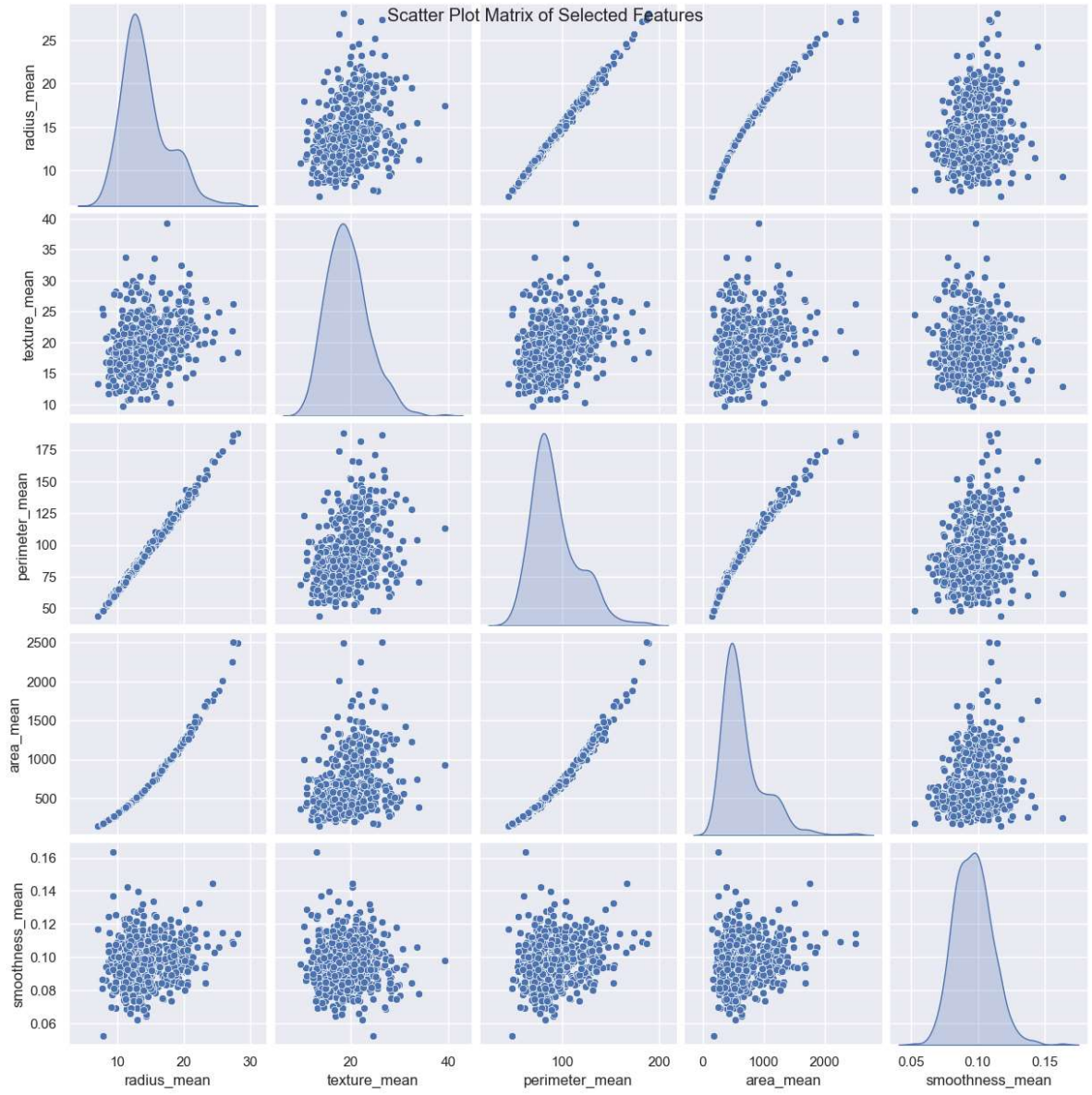
In [12]:
```
plt.figure(figsize=(8, 6))
sns.violinplot(x='diagnosis', y='perimeter_worst', data=dataset, palette='past
plt.xlabel('Diagnosis')
plt.ylabel('Perimeter Worst')
plt.title('Violin Plot: Perimeter Worst by Diagnosis')
plt.show()
```



In [13]:
```
plt.figure(figsize=(10, 6))
colors = ['lightblue', 'lightcoral', 'lightgreen']
plt.stackplot(dataset['diagnosis'].unique(),
              dataset.groupby('diagnosis')['concavity_mean'].mean(),
              dataset.groupby('diagnosis')['concave points_mean'].mean(),
              dataset.groupby('diagnosis')['compactness_mean'].mean(),
              labels=['Concavity Mean', 'Concave Points Mean', 'Compactness Me
              colors=colors)
plt.xlabel('Diagnosis')
plt.ylabel('Mean Values')
plt.title('Area Plot: Concavity Mean, Concave Points Mean, and Compactness Mea
plt.legend(loc='upper left')
plt.show()
```



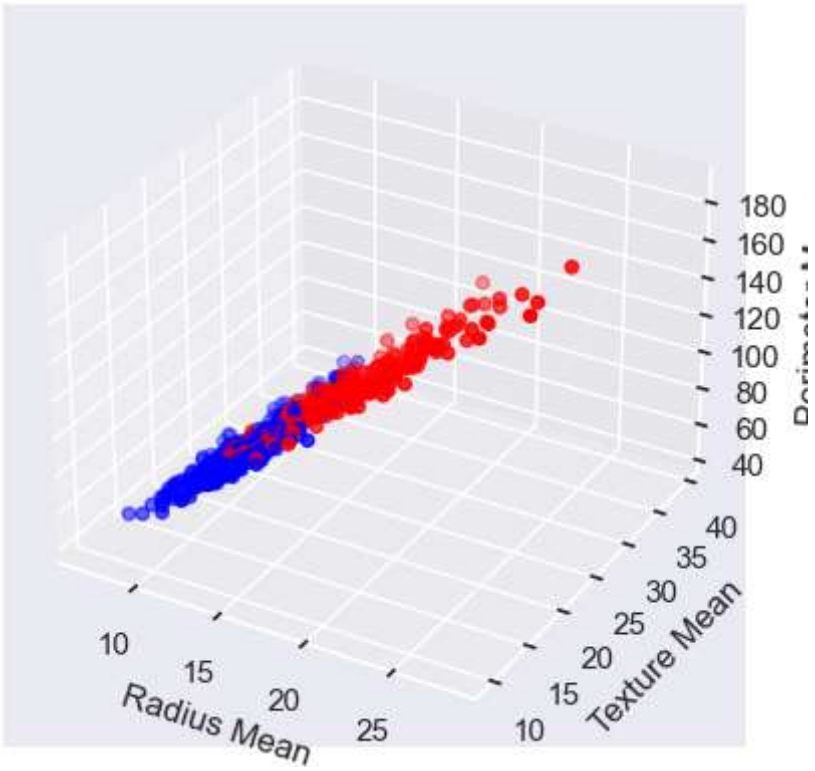
In [14]:
```
selected_features = ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_me
sns.pairplot(dataset[selected_features], diag_kind='kde', palette='viridis')
plt.suptitle('Scatter Plot Matrix of Selected Features')
plt.show()
```



In [15]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

In [ ]:
```

```

In [16]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Assign colors to benign (B) and malignant (M) data points
colors = {'B': 'blue', 'M': 'red'}
dataset['color'] = dataset['diagnosis'].map(colors)

fig = plt.figure(figsize=(10, 8))
ax.scatter(dataset['radius_mean'], dataset['texture_mean'], dataset['perimeter
ax.set_xlabel('Radius Mean')
ax.set_ylabel('Texture Mean')
ax.set_zlabel('Perimeter Mean')
# Adjust the layout
plt.tight_layout()
plt.show()
```




```
<Figure size 1000x800 with 0 Axes>
```

Encoding

In [17]:
```
dataset = dataset.iloc[:,1:]
dataset.head()
```

Out[17]:

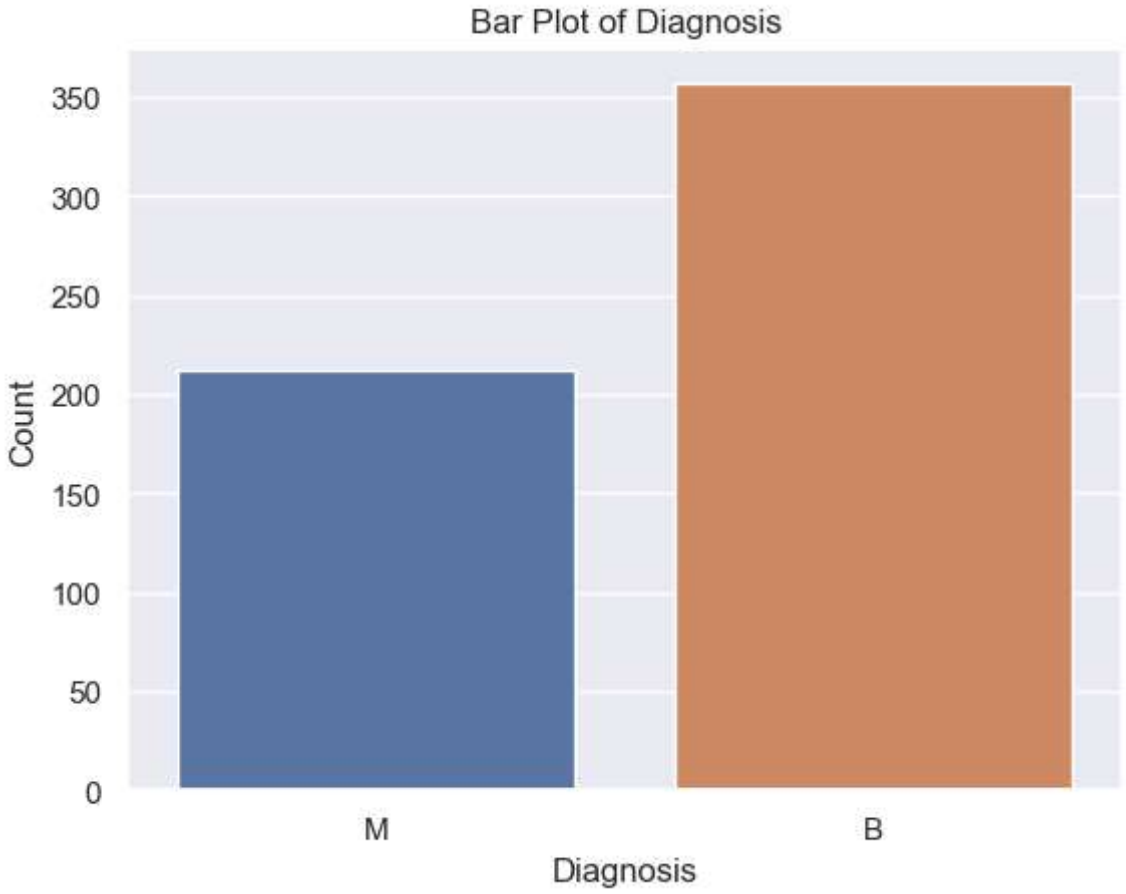
	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 32 columns

In [18]:
```
dataset['diagnosis'].value_counts()
```

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

In [19]:
```
sns.countplot(x='diagnosis', data=dataset)
plt.xlabel('Diagnosis')
plt.ylabel('Count')
plt.title('Bar Plot of Diagnosis')
plt.show()
```



In [20]:
```
dataset['diagnosis'] = dataset['diagnosis'].astype('category')
dataset['diagnosis'] = dataset['diagnosis'].cat.codes
```

In [21]:
```
dataset.head()
```

Out[21]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
0	1	17.99	10.38	122.80	1001.0	0.11840	
1	1	20.57	17.77	132.90	1326.0	0.08474	
2	1	19.69	21.25	130.00	1203.0	0.10960	
3	1	11.42	20.38	77.58	386.1	0.14250	
4	1	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 32 columns

In [22]:
```
for i in dataset.columns:
    print("**********************************", i ,
          "*****************************************")
    print()
    print(set(dataset[i].tolist()))
    print()
```

```
********************************** diagnosis ****************************
**************

{0, 1}

********************************** radius_mean ***************************
****************

{6.981, 7.729, 8.196, 9.504, 10.95, 11.42, 12.45, 13.53, 13.73, 14.54, 14.
68, 13.71, 13.0, 12.46, 16.02, 17.99, 18.25, 19.69, 20.57, 20.29, 19.17, 1
9.81, 21.16, 20.18, 25.22, 22.27, 24.25, 27.22, 28.11, 27.42, 9.738, 9.0,
9.268, 10.75, 10.25, 11.75, 11.5, 11.25, 12.0, 12.25, 12.75, 13.75, 13.5,
14.25, 14.5, 15.75, 15.0, 15.5, 16.78, 16.03, 16.5, 16.25, 17.47, 17.75, 1
8.22, 18.03, 19.0, 19.53, 20.47, 21.75, 8.726, 8.571, 9.72, 9.333, 10.44,
10.94, 10.97, 11.22, 11.47, 11.94, 11.69, 12.19, 12.47, 12.72, 12.94, 12.2
2, 13.44, 13.69, 13.47, 13.94, 14.44, 14.97, 14.22, 14.19, 14.69, 15.19, 1
4.47, 15.22, 9.029, 9.904, 10.66, 10.91, 10.16, 11.41, 11.66, 11.16, 12.9
1, 12.16, 13.66, 13.16, 14.41, 15.66, 16.13, 16.16, 16.69, 17.19, 17.91, 1
8.63, 18.94, 18.66, 19.16, 19.19, 19.44, 20.16, 20.94, 20.44, 20.13, 24.6
```

In [23]:
```
# Split the data into ind variable and dep variable
x = dataset.drop(['diagnosis', 'color'], axis=1)
y = dataset[['diagnosis']]
```

In [24]:
```
x.head()
```

Out[24]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_me
0	17.99	10.38	122.80	1001.0	0.11840	0.277
1	20.57	17.77	132.90	1326.0	0.08474	0.078
2	19.69	21.25	130.00	1203.0	0.10960	0.159
3	11.42	20.38	77.58	386.1	0.14250	0.283
4	20.29	14.34	135.10	1297.0	0.10030	0.132

5 rows × 30 columns

In [25]:
```
y.head()
```

Out[25]:

	diagnosis
0	1
1	1
2	1
3	1
4	1

Feature Scaling

In [26]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x1 = sc.fit_transform(x)
```

In [27]:
```
# Check imbalance dataset
y.value_counts()
```

Out[27]:
```
diagnosis
0            357
1            212
dtype: int64
```

In [28]:
```
dataset.corr()
```

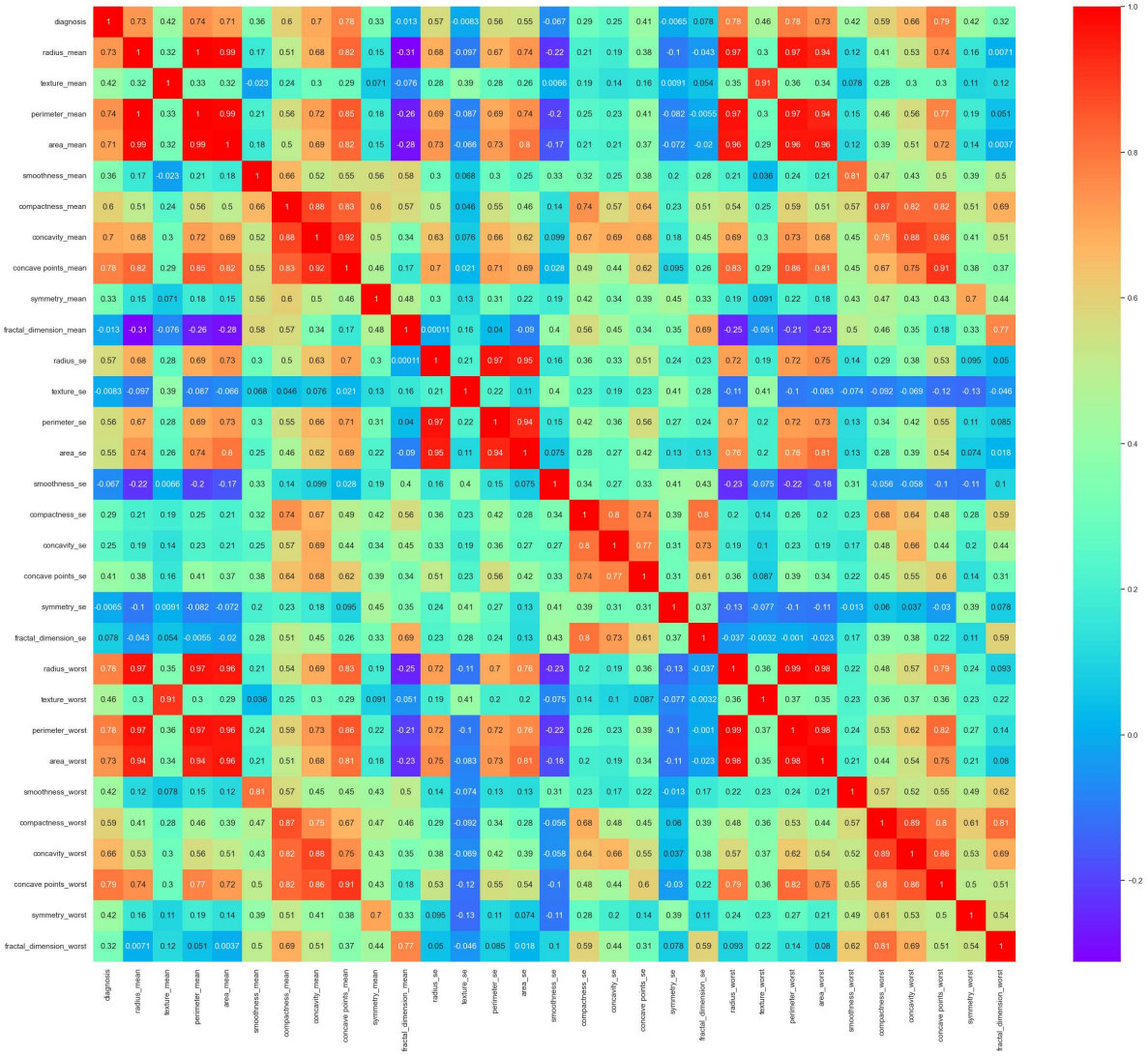
Out[28]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	sm
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	
radius_se	0.567134	0.679090	0.275869	0.691765	0.732562	
texture_se	-0.008303	-0.097317	0.386358	-0.086761	-0.066280	
perimeter_se	0.556141	0.674172	0.281673	0.693135	0.726628	
area_se	0.548236	0.735864	0.259845	0.744983	0.800086	
smoothness_se	-0.067016	-0.222600	0.006614	-0.202694	-0.166777	
compactness_se	0.292999	0.206000	0.191975	0.250744	0.212583	
concavity_se	0.253730	0.194204	0.143293	0.228082	0.207660	
concave points_se	0.408042	0.376169	0.163851	0.407217	0.372320	
symmetry_se	-0.006522	-0.104321	0.009127	-0.081629	-0.072497	
fractal_dimension_se	0.077972	-0.042641	0.054458	-0.005523	-0.019887	
radius_worst	0.776454	0.969539	0.352573	0.969476	0.962746	
texture_worst	0.456903	0.297008	0.912045	0.303038	0.287489	
perimeter_worst	0.782914	0.965137	0.358040	0.970387	0.959120	
area_worst	0.733825	0.941082	0.343546	0.941550	0.959213	
smoothness_worst	0.421465	0.119616	0.077503	0.150549	0.123523	
compactness_worst	0.590998	0.413463	0.277830	0.455774	0.390410	
concavity_worst	0.659610	0.526911	0.301025	0.563879	0.512606	
concave points_worst	0.793566	0.744214	0.295316	0.771241	0.722017	
symmetry_worst	0.416294	0.163953	0.105008	0.189115	0.143570	
fractal_dimension_worst	0.323872	0.007066	0.119205	0.051019	0.003738	

31 rows × 31 columns

In [29]:
```
plt.figure(figsize=(30,25))
sns.heatmap(dataset.corr(), annot=True, cmap='rainbow')
```

Out[29]: <Axes: >



Split the Train and Test Data

In [30]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
from sklearn.tree import export_text
```

In [31]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train,y_test = train_test_split(x, y, train_size=0.75, rand
```

Logistic Regression Model

In [32]:
```
from sklearn.linear_model import LogisticRegression
```

In [33]:
```
logit = LogisticRegression(multi_class='multinomial')
logit.fit(x_train, y_train)
```

Out[33]:
```
LogisticRegression(multi_class='multinomial')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [34]:
```
y_pred_train = logit.predict(x_train)
y_pred_test = logit.predict(x_test)
```

In [35]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_m
```

In [36]:
```
print( classification_report(y_train, y_pred_train))
print("****************"*10)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       264
           1       0.96      0.93      0.94       162

    accuracy                           0.96       426
   macro avg       0.96      0.95      0.95       426
weighted avg       0.96      0.96      0.96       426

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

           0       0.97      0.98      0.97        93
           1       0.96      0.94      0.95        50

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.96      0.97      0.96       143
```

In [37]:
```
print( confusion_matrix(y_train, y_pred_train))
print("****************"*10)
print(confusion_matrix(y_test, y_pred_test))
```

```
[[258   6]
 [ 12 150]]
********************************************************************************
********************************************************************************
******
[[91  2]
 [ 3 47]]
```

In [38]:
```
print("Training Accuracy :", accuracy_score(y_train, y_pred_train))
print("****************"*5)
print("Test Accuracy :", accuracy_score(y_test, y_pred_test))
```

```
Training Accuracy : 0.9577464788732394
********************************************************************************
***
Test Accuracy : 0.965034965034965
```

In [39]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(logit, x_train, y_train, cv=10)
test_accuracy = cross_val_score(logit, x_test, y_test, cv=10)
print("Training Accuracy after CV :", training_accuracy.mean())
print("**********************"*5)
print("Test Accuracy after CV :", test_accuracy.mean())
```

```
Training Accuracy after CV : 0.9342746400885936
********************************************************************************
**********************************
Test Accuracy after CV : 0.950952380952381
```

In [40]:
```
from sklearn.metrics import roc_auc_score
logit_roc_auc = roc_auc_score(y_test, y_pred_test)
logit_roc_auc
```

Out[40]:
```
0.959247311827957
```

In [41]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_test)
display(fpr[:10])
display(tpr[:10])
display(thresholds[:10])
```

```
array([0.        , 0.02150538, 1.        ])

array([0.  , 0.94, 1.  ])

array([2, 1, 0], dtype=int8)
```

In [42]:
```
tpr
```

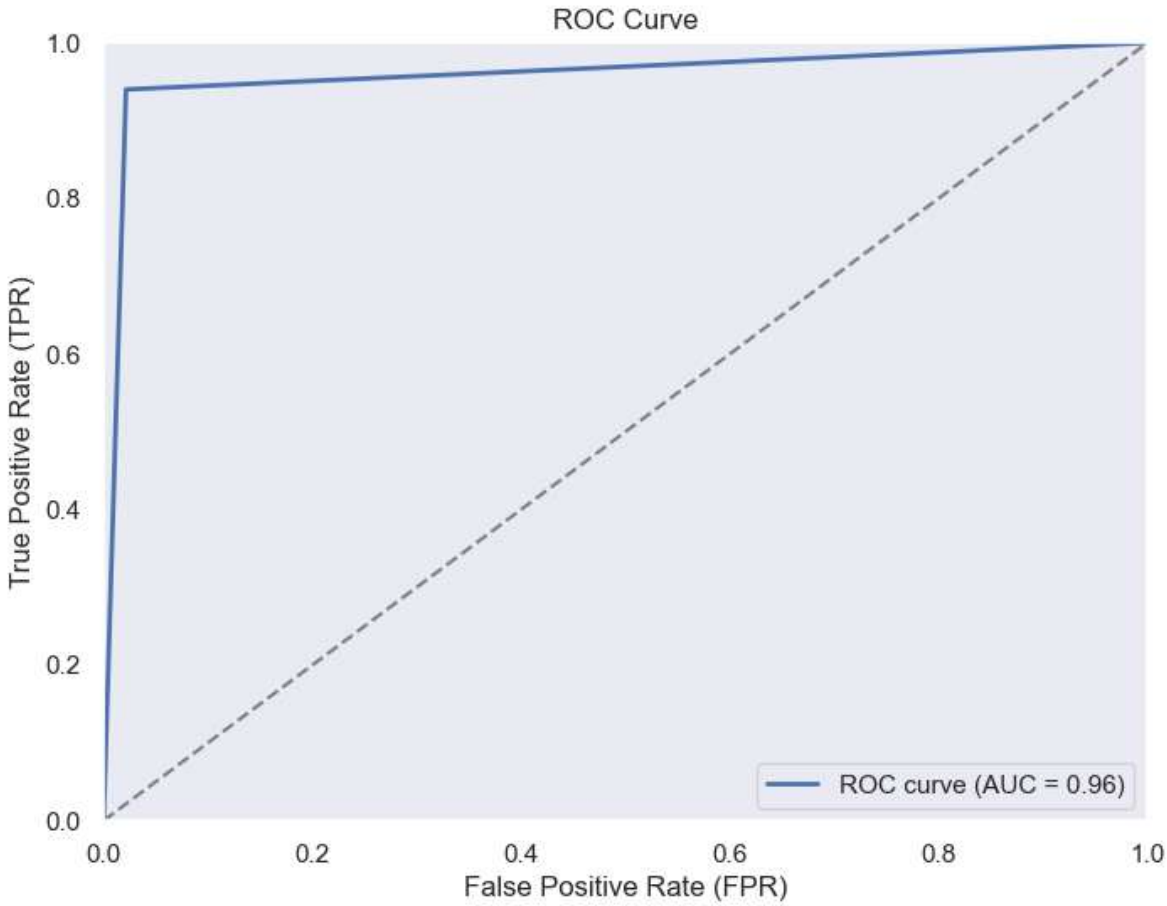
Out[42]:
```
array([0.  , 0.94, 1.  ])
```

In [43]:
```
thresholds
```

Out[43]:
```
array([2, 1, 0], dtype=int8)
```

In [44]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_test)
auc_score = roc_auc_score(y_test, y_pred_test)
```

In [45]:
```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='b', lw=2, label=f'ROC curve (AUC = {auc_score:.2f})'
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Building DecisionTree Classifier Model

Criterion = 'gini'

Criterion = 'entropy'

In [46]:
```
from sklearn.tree import DecisionTreeClassifier
dt1 = DecisionTreeClassifier(criterion='gini')
dt1.fit(x_train, y_train)

dt2 = DecisionTreeClassifier(criterion='entropy')
dt2.fit(x_train, y_train)
```

Out[46]:
```
DecisionTreeClassifier(criterion='entropy')
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [47]:
```
# Predict
# Gini
y_pred_train_dt1 = dt1.predict(x_train)
y_pred_test_dt1 = dt1.predict(x_test)

# Entropy
y_pred_train_dt2 = dt2.predict(x_train)
y_pred_test_dt2 = dt2.predict(x_test)
```

In [48]:
```
# Evaluation
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

In [49]:
```
# Gini
print(classification_report(y_train, y_pred_train_dt1))
print()
print(classification_report(y_test, y_pred_test_dt1))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       264
           1       1.00      1.00      1.00       162

    accuracy                           1.00       426
   macro avg       1.00      1.00      1.00       426
weighted avg       1.00      1.00      1.00       426


              precision    recall  f1-score   support

           0       0.94      0.95      0.94        93
           1       0.90      0.88      0.89        50

    accuracy                           0.92       143
   macro avg       0.92      0.91      0.92       143
weighted avg       0.92      0.92      0.92       143
```

In [50]:
```
# Entropy
print(classification_report(y_train, y_pred_train_dt2))
print()
print(classification_report(y_test, y_pred_test_dt2))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       264
           1       1.00      1.00      1.00       162

    accuracy                           1.00       426
   macro avg       1.00      1.00      1.00       426
weighted avg       1.00      1.00      1.00       426


              precision    recall  f1-score   support

           0       0.97      0.98      0.97        93
           1       0.96      0.94      0.95        50

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.96      0.97      0.96       143
```

In [51]:
```
# Gini
print(confusion_matrix(y_train, y_pred_train_dt1))
print()
print(confusion_matrix(y_test, y_pred_test_dt1))
```

```
[[264   0]
 [  0 162]]

[[88  5]
 [ 6 44]]
```

In [52]:
```
# Entropy
print(confusion_matrix(y_train, y_pred_train_dt2))
print()
print(confusion_matrix(y_test, y_pred_test_dt2))
```

```
[[264   0]
 [  0 162]]

[[91  2]
 [ 3 47]]
```

In [53]:
```
# Gini
print(accuracy_score(y_train, y_pred_train_dt1))
print()
print(accuracy_score(y_test, y_pred_test_dt1))
```

```
1.0

0.9230769230769231
```

In [54]:
```
# Entropy
print(accuracy_score(y_train, y_pred_train_dt2))
print()
print(accuracy_score(y_test, y_pred_test_dt2))
```

```
1.0

0.965034965034965
```

Feature Importance

In [55]:
```
# Check Feature Importance
pd.DataFrame(index=x.columns, data=dt1.feature_importances_, columns=['Feature
```

Out[55]:

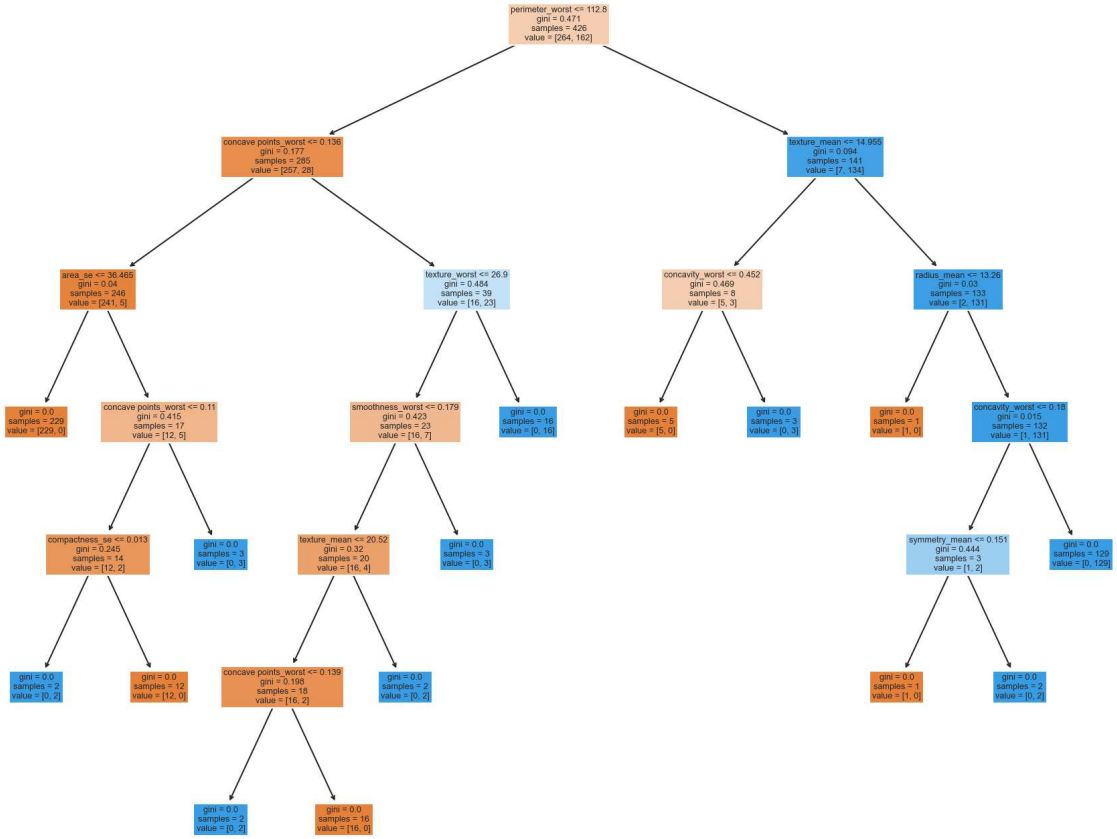
	Feature Importance
radius_mean	0.009737
texture_mean	0.042132
perimeter_mean	0.000000
area_mean	0.000000
smoothness_mean	0.000000
compactness_mean	0.000000
concavity_mean	0.000000
concave points_mean	0.000000
symmetry_mean	0.006640
fractal_dimension_mean	0.000000
radius_se	0.000000
texture_se	0.000000
perimeter_se	0.000000
area_se	0.013636
smoothness_se	0.000000
compactness_se	0.017076
concavity_se	0.000000
concave points_se	0.000000
symmetry_se	0.000000
fractal_dimension_se	0.000000
radius_worst	0.000000
texture_worst	0.045484
perimeter_worst	0.682237
area_worst	0.000000
smoothness_worst	0.016630
compactness_worst	0.000000
concavity_worst	0.021921
concave points_worst	0.144508
symmetry_worst	0.000000
fractal_dimension_worst	0.000000

In [56]:
```
pd.DataFrame(index=x.columns, data=dt2.feature_importances_, columns=['Feature
```

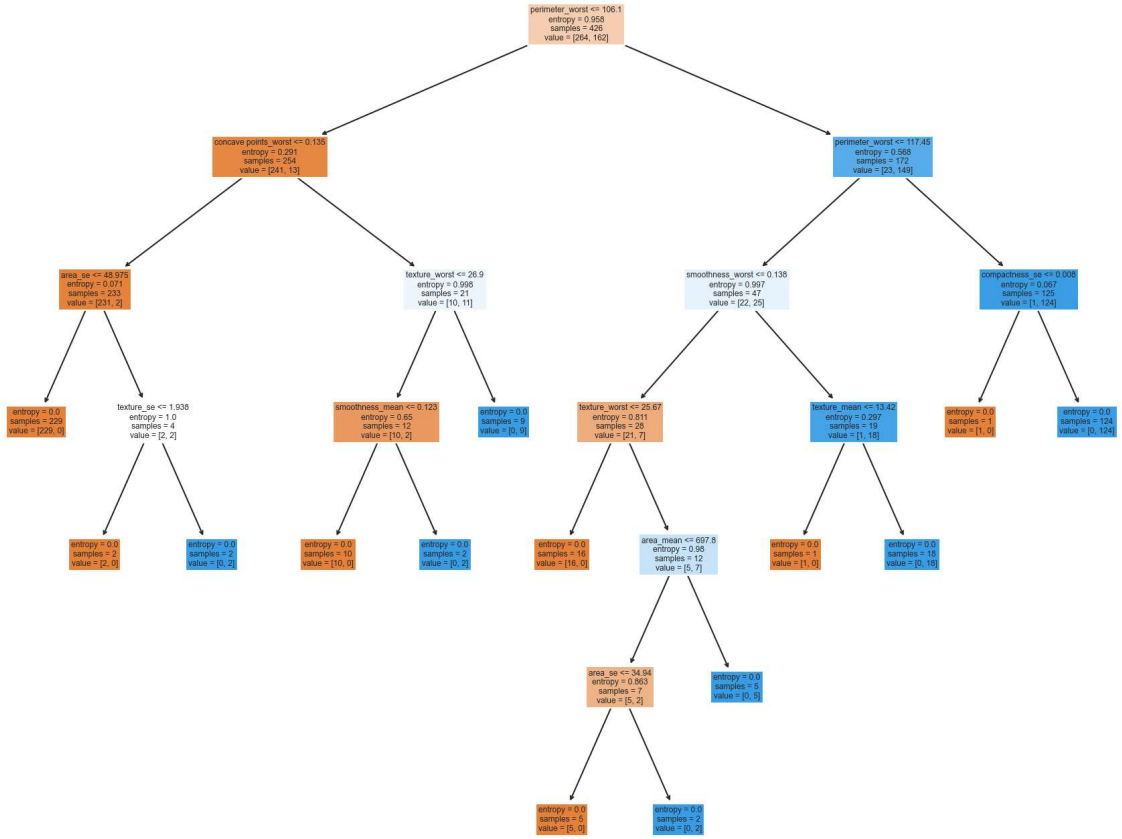
Out[56]:

	Feature Importance
radius_mean	0.000000
texture_mean	0.013846
perimeter_mean	0.000000
area_mean	0.014004
smoothness_mean	0.019108
compactness_mean	0.000000
concavity_mean	0.000000
concave points_mean	0.000000
symmetry_mean	0.000000
fractal_dimension_mean	0.000000
radius_se	0.000000
texture_se	0.009799
perimeter_se	0.000000
area_se	0.045671
smoothness_se	0.000000
compactness_se	0.020584
concavity_se	0.000000
concave points_se	0.000000
symmetry_se	0.000000
fractal_dimension_se	0.000000
radius_worst	0.000000
texture_worst	0.059094
perimeter_worst	0.683305
area_worst	0.000000
smoothness_worst	0.045305
compactness_worst	0.000000
concavity_worst	0.000000
concave points_worst	0.089285
symmetry_worst	0.000000
fractal_dimension_worst	0.000000

In [57]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(15,12), dpi=150)
plot_tree(dt1, filled=True, feature_names = x.columns)
plt.show()
```



In [58]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(15,12), dpi=150)
plot_tree(dt2, filled=True, feature_names = x.columns)
plt.show()
```



In [59]:
```
# Using Post-Prunning Method to handle overfitting problem
def report_model(model):
    model_preds = model.predict(x_test)
    print(classification_report(y_test, model_preds))
    print('\n')
    plt.figure(figsize=(15,12), dpi=150)
    plot_tree(model, filled=True, feature_names=x.columns)
```

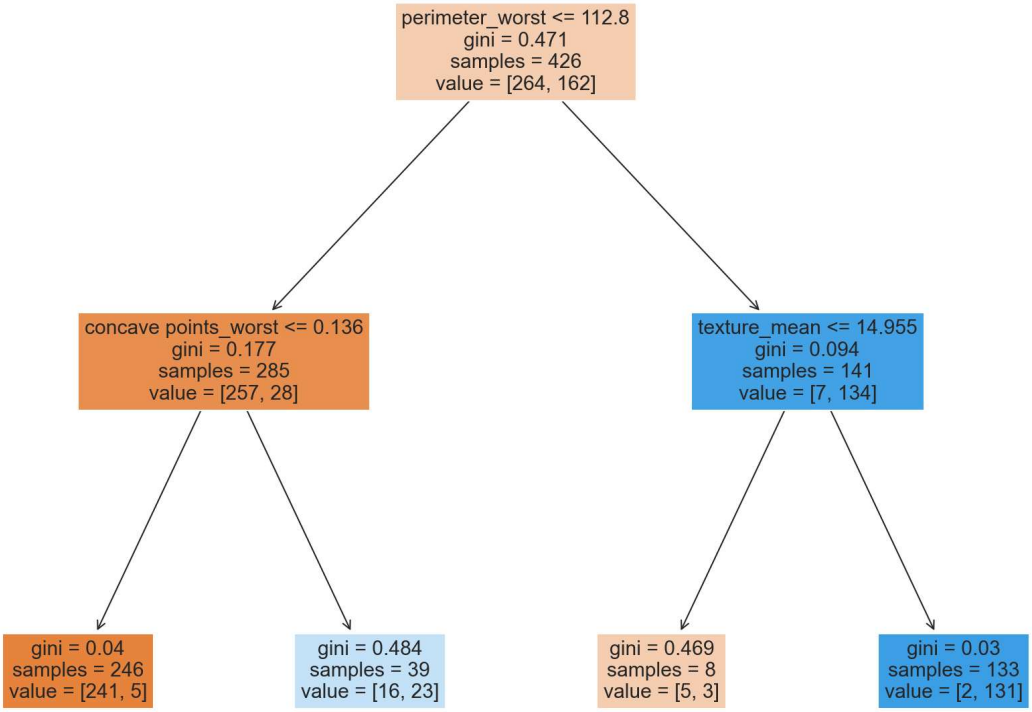
In [60]:
```
#prunned_dtree = DecisionTreeClassifier(max_depth=3, min_samples_leaf=3)
prunned_dtree = DecisionTreeClassifier(max_depth=2)
prunned_dtree.fit(x_train, y_train)
```

Out[60]:
```
DecisionTreeClassifier(max_depth=2)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [61]:
```
report_model(prunned_dtree)
```

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        93
           1       0.88      0.92      0.90        50

    accuracy                           0.93       143
   macro avg       0.92      0.93      0.92       143
weighted avg       0.93      0.93      0.93       143
```



In [62]:
```
y_pred_prunned_train = prunned_dtree.predict(x_train)
y_pred_prunned_test = prunned_dtree.predict(x_test)
```

In [63]:
```
print(accuracy_score(y_train, y_pred_prunned_train))
print()
print(accuracy_score(y_test, y_pred_prunned_test))
```

```
0.9389671361502347

0.9300699300699301
```

In [64]:
```
# Cross Validation approach
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(prunned_dtree, x_train, y_train, cv=10)
test_accuracy = cross_val_score(prunned_dtree, x_test, y_test, cv=10)
print(training_accuracy.mean())
print()
print(test_accuracy.mean())
```

```
0.8942414174972315

0.9019047619047619
```

Building Bagging Algorithm

In [65]:
```
from sklearn.ensemble import BaggingClassifier
bagging = BaggingClassifier()
bagging.fit(x_train, y_train)
```

Out[65]:
```
BaggingClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [66]:
```
# predict
y_pred_train_bgg = bagging.predict(x_train)
y_pred_test_bgg = bagging.predict(x_test)
```

In [67]:
```
# Evaluate
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

In [68]:
```
print(confusion_matrix(y_train, y_pred_train_bgg))
print()
print(confusion_matrix(y_test, y_pred_test_bgg))
```

```
[[264   0]
 [  1 161]]

[[90  3]
 [ 4 46]]
```

In [69]:
```
print(classification_report(y_train, y_pred_train_bgg))
print()
print(classification_report(y_test, y_pred_test_bgg))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       264
           1       1.00      0.99      1.00       162

    accuracy                           1.00       426
   macro avg       1.00      1.00      1.00       426
weighted avg       1.00      1.00      1.00       426


              precision    recall  f1-score   support

           0       0.96      0.97      0.96        93
           1       0.94      0.92      0.93        50

    accuracy                           0.95       143
   macro avg       0.95      0.94      0.95       143
weighted avg       0.95      0.95      0.95       143
```

In [70]:
```
print(accuracy_score(y_train, y_pred_train_bgg))
print()
print(accuracy_score(y_test, y_pred_test_bgg))
```

```
0.9976525821596244

0.951048951048951
```

In [71]:
```
# Cross Validation approach
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(bagging, x_train, y_train, cv=10)
test_accuracy = cross_val_score(bagging, x_test, y_test, cv=10)
print(training_accuracy.mean())
print()
print(test_accuracy.mean())
```

```
0.9293466223698783

0.9580952380952381
```

RandomForest Classifier Model

In [72]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=200, criterion='entropy',
                            bootstrap=True, oob_score=False)
rf.fit(x_train, y_train)
```

Out[72]:
```
RandomForestClassifier(criterion='entropy', n_estimators=200)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [73]:
```
# predict
y_pred_train_rf = rf.predict(x_train)
y_pred_test_rf = rf.predict(x_test)
```

In [74]:
```
print(confusion_matrix(y_train, y_pred_train_rf))
print()
print(confusion_matrix(y_test, y_pred_test_rf))
```

```
[[264   0]
 [  0 162]]

[[92  1]
 [ 3 47]]
```

In [75]:
```
print(classification_report(y_train, y_pred_train_rf))
print()
print(classification_report(y_test, y_pred_test_rf))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       264
           1       1.00      1.00      1.00       162

    accuracy                           1.00       426
   macro avg       1.00      1.00      1.00       426
weighted avg       1.00      1.00      1.00       426


              precision    recall  f1-score   support

           0       0.97      0.99      0.98        93
           1       0.98      0.94      0.96        50

    accuracy                           0.97       143
   macro avg       0.97      0.96      0.97       143
weighted avg       0.97      0.97      0.97       143
```

In [76]:
```
print(accuracy_score(y_train, y_pred_train_rf))
print()
print(accuracy_score(y_test, y_pred_test_rf))
```

```
1.0

0.972027972027972
```

In [77]:
```
# Cross Validation approach
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(rf, x_train, y_train, cv=10)
test_accuracy = cross_val_score(rf, x_test, y_test, cv=10)
print(training_accuracy.mean())
print()
print(test_accuracy.mean())
```

```
0.9647286821705426

0.9585714285714285
```

K-nearest neighbors algorithm

In [78]:
```
from sklearn.neighbors import KNeighborsClassifier
```

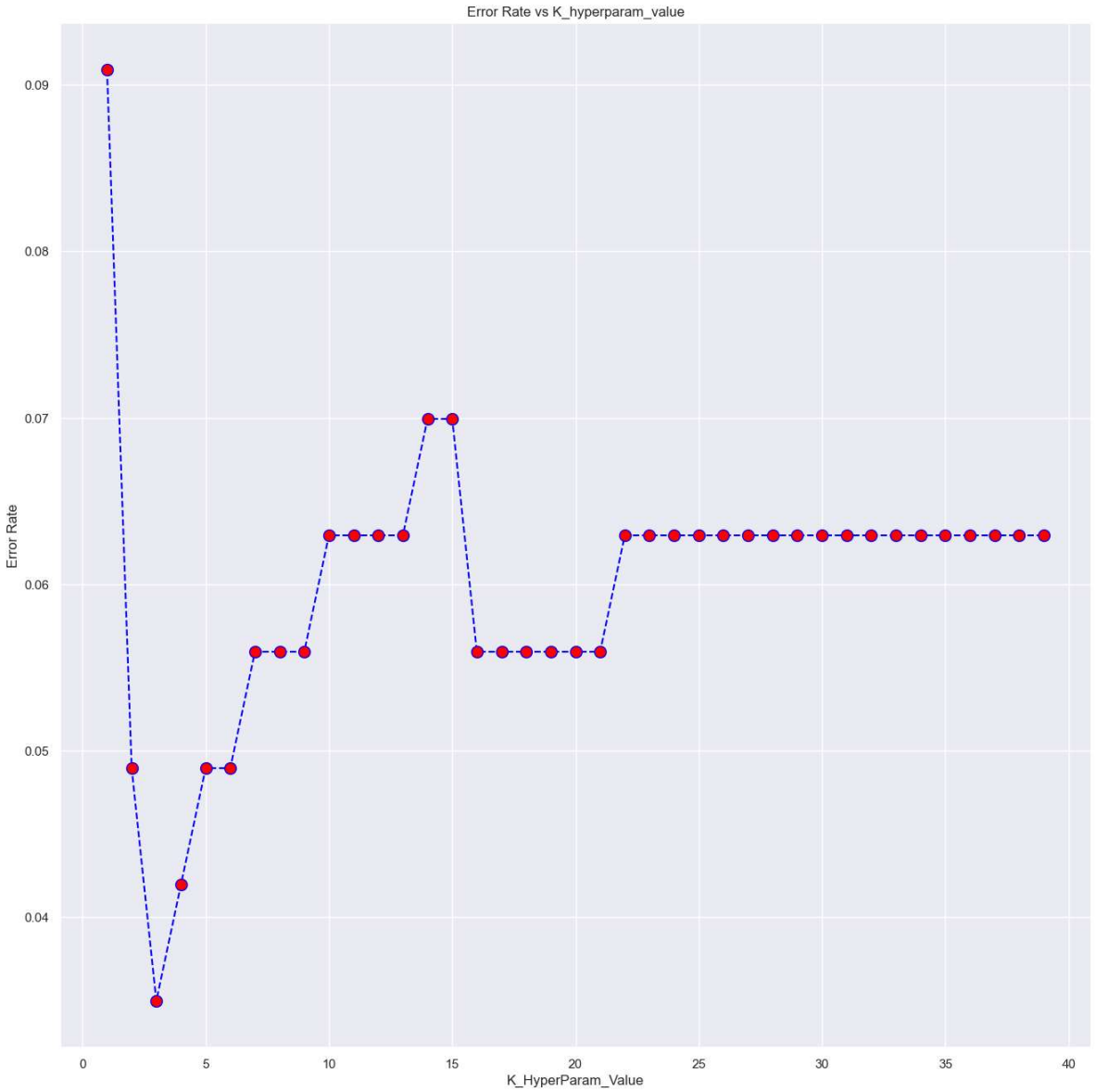
In [79]:
```
y_test = np.ravel(y_test)
```

In [80]:
```
error_rate  = []

for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train, y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [81]:
```
error_rate
```

Out[81]:
```
[0.09090909090909091,
 0.04895104895104895,
 0.03496503496503497,
 0.04195804195804196,
 0.04895104895104895,
 0.04895104895104895,
 0.055944055944055944,
 0.055944055944055944,
 0.055944055944055944,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06993006993006994,
 0.06993006993006994,
 0.055944055944055944,
 0.055944055944055944,
 0.055944055944055944,
 0.055944055944055944,
 0.055944055944055944,
 0.055944055944055944,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294,
 0.06293706293706294]
```

In [82]:
```
plt.figure(figsize=(16,16))
plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed', marker='o'
         markerfacecolor='red', markersize=10)
plt.title("Error Rate vs K_hyperparam_value")
plt.xlabel("K_HyperParam_Value")
plt.ylabel("Error Rate")
plt.show()
```



In [83]:
```
# Basis analysis, error_rate would be less when we choose k=3

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train)
```

Out[83]:
```
KNeighborsClassifier(n_neighbors=3)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [84]:
```
y_pred_train = knn.predict(x_train)
y_pred_test = knn.predict(x_test)
```

In [85]:
```
# Evaluate the model
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

In [86]:
```
print(confusion_matrix(y_train, y_pred_train))
print("****************"*5)
print(confusion_matrix(y_test, y_pred_test))
```

```
[[257   7]
 [ 15 147]]
**********************************************************************
***
[[90  3]
 [ 2 48]]
```

In [87]:
```
print(classification_report(y_train, y_pred_train))
print("****************"*5)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.96       264
           1       0.95      0.91      0.93       162

    accuracy                           0.95       426
   macro avg       0.95      0.94      0.94       426
weighted avg       0.95      0.95      0.95       426

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

           0       0.98      0.97      0.97        93
           1       0.94      0.96      0.95        50

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.97      0.97      0.97       143
```

In [88]:
```
print("Training Accuracy :", accuracy_score(y_train, y_pred_train))
print("****************"*5)
print("Test Accuracy :", accuracy_score(y_test, y_pred_test))
```

```
Training Accuracy : 0.9483568075117371
**********************************************************************
***
Test Accuracy : 0.965034965034965
```

In [89]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(knn, x_train, y_train, cv=10)
test_accuracy = cross_val_score(knn, x_test, y_test, cv=10)
print("Training Accuracy after CV :", training_accuracy.mean())
print("**********************"*5)
print("Test Accuracy after CV :", test_accuracy.mean())
```

```
Training Accuracy after CV : 0.9178848283499447
**********************************************************************
**********************************
Test Accuracy after CV : 0.9366666666666668
```

Support Vector Machine

In [90]:
```
from sklearn.svm import SVC

#kernel - linear
svm_linear = SVC(kernel='linear')
svm_linear.fit(x_train, y_train)
y_pred_train_linear = svm_linear.predict(x_train)
y_pred_test_linear = svm_linear.predict(x_test)

#kernel - sigmoid
svm_sigmoid = SVC(kernel='sigmoid')
svm_sigmoid.fit(x_train, y_train)
y_pred_train_sigmoid = svm_sigmoid.predict(x_train)
y_pred_test_sigmoid = svm_sigmoid.predict(x_test)

#kernel - poly
svm_poly = SVC(kernel='poly')
svm_poly.fit(x_train, y_train)
y_pred_train_poly = svm_poly.predict(x_train)
y_pred_test_poly = svm_poly.predict(x_test)

#kernel - rbf
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(x_train, y_train)
y_pred_train_rbf = svm_rbf.predict(x_train)
y_pred_test_rbf = svm_rbf.predict(x_test)
```

In [91]:
```
# Evulation

from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

In [92]:
```
print("Training Accuracy - Linear :", accuracy_score(y_train, y_pred_train_lin
print("Test Accuracy - Linear :", accuracy_score(y_test, y_pred_test_linear))
print("**************"*5)
print("Training Accuracy - sigmoid :", accuracy_score(y_train, y_pred_train_si
print("Test Accuracy - sigmoid :", accuracy_score(y_test, y_pred_test_sigmoid)
print("**************"*5)
print("Training Accuracy - poly :", accuracy_score(y_train, y_pred_train_poly)
print("Test Accuracy - poly :", accuracy_score(y_test, y_pred_test_poly))
print("**************"*5)
print("Training Accuracy - rbf :", accuracy_score(y_train, y_pred_train_rbf))
print("Test Accuracy - rbf :", accuracy_score(y_test, y_pred_test_rbf))
```

```
Training Accuracy - Linear : 0.9671361502347418
Test Accuracy - Linear : 0.972027972027972
************************************************************
Training Accuracy - sigmoid : 0.43896713615023475
Test Accuracy - sigmoid : 0.4825174825174825
************************************************************
Training Accuracy - poly : 0.9061032863849765
Test Accuracy - poly : 0.9300699300699301
************************************************************
Training Accuracy - rbf : 0.9131455399061033
Test Accuracy - rbf : 0.9370629370629371
```

In [93]:
```
print("Training Accuracy - Linear :", classification_report(y_train, y_pred_tr
print("**************"*5)
print("Test Accuracy - Linear :", classification_report(y_test, y_pred_test_li
```

```
Training Accuracy - Linear :               precision    recall  f1-score   su
pport

           0       0.97      0.98      0.97       264
           1       0.97      0.94      0.96       162

    accuracy                           0.97       426
   macro avg       0.97      0.96      0.96       426
weighted avg       0.97      0.97      0.97       426

************************************************************
Test Accuracy - Linear :               precision    recall  f1-score   suppor
t

           0       0.97      0.99      0.98        93
           1       0.98      0.94      0.96        50

    accuracy                           0.97       143
   macro avg       0.97      0.96      0.97       143
weighted avg       0.97      0.97      0.97       143
```

In [94]:
```
# cross validation method
from sklearn.model_selection import cross_val_score
train_accuracy = cross_val_score(svm_linear, x_train, y_train, cv=10)
test_accuracy = cross_val_score(svm_linear, x_test, y_test, cv=10)
print("Training accuracy :", train_accuracy)
print("Training Mean Accuracy :", train_accuracy.mean())
print("************"*5)
print("Test accuracy :", test_accuracy)
print("Test Mean Accuracy :", test_accuracy.mean())
```

```
Training accuracy : [0.88372093 0.90697674 0.93023256 0.95348837 0.90697674
0.95348837
 0.97619048 0.97619048 0.95238095 0.95238095]
Training Mean Accuracy : 0.9392026578073089
************************************************************
Test accuracy : [0.93333333 0.86666667 0.93333333 0.78571429 1.         0.857
14286
 0.92857143 1.         1.         0.85714286]
Test Mean Accuracy : 0.9161904761904761
```

Voting Classifier Model

In [95]:
```
from sklearn.ensemble  import VotingClassifier
```

In [100]:
```
estimators = [('Logistic Regression',logit),('Decision Tree',prunned_dtree),('
```

Hyperparameter: Hard Voting

In [106]:
```
hard_voting = VotingClassifier(estimators=estimators, voting='hard' )
hard_voting.fit(x_train,y_train)
```

Out[106]:
```
VotingClassifier(estimators=[('Logistic Regression',
                              LogisticRegression(multi_class='multinomial')),
                             ('Decision Tree',
                              DecisionTreeClassifier(max_depth=2)),
                             ('Bagging Algorith', BaggingClassifier()),
                             ('Random Forest',
                              RandomForestClassifier(criterion='entropy',
                                                     n_estimators=200)),
                             ('KNN', KNeighborsClassifier(n_neighbors=3)),
                             ('SVM', SVC(kernel='linear'))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [107]:
```
y_pred_train_hv= hard_voting.predict(x_train)
y_pred_test_hv= hard_voting.predict(x_test)
```

In [109]:
```
train_accuracy = cross_val_score(hard_voting, x_train, y_train, cv=10)
test_accuracy = cross_val_score(hard_voting, x_test, y_test, cv=10)
print("Training Accuracy :", train_accuracy.mean())
print("************"*5)
print("Test Accuracy :", test_accuracy.mean())
```

```
Training Accuracy : 0.9483942414174973
**********************************************************
Test Accuracy : 0.9514285714285714
```

Hyperparameter: Soft Voting

In [113]:
```
soft_voting = VotingClassifier(estimators=estimators, voting='soft', n_jobs=-1
soft_voting.fit(x_train,y_train)
```

Out[113]:
```
VotingClassifier(estimators=[('Logistic Regression',
                              LogisticRegression(multi_class='multinomial')),
                             ('Decision Tree',
                              DecisionTreeClassifier(max_depth=2)),
                             ('Bagging Algorith', BaggingClassifier()),
                             ('Random Forest',
                              RandomForestClassifier(criterion='entropy',
                                                     n_estimators=200)),
                             ('KNN', KNeighborsClassifier(n_neighbors=3)),
                             ('SVM', SVC(kernel='linear'))],
                 n_jobs=-1, voting='soft')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [114]:
```
y_pred_train_sv= soft_voting.predict(x_train)
y_pred_test_sv= soft_voting.predict(x_test)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[114], line 1
----> 1 y_pred_train_sv= soft_voting.predict(x_train)
      2 y_pred_test_sv= soft_voting.predict(x_test)

File ~\anaconda3\lib\site-packages\sklearn\ensemble\_voting.py:363, in Voting
Classifier.predict(self, X)
    361 check_is_fitted(self)
    362 if self.voting == "soft":
--> 363     maj = np.argmax(self.predict_proba(X), axis=1)
    365 else:  # 'hard' voting
    366     predictions = self._predict(X)

File ~\anaconda3\lib\site-packages\sklearn\ensemble\_voting.py:404, in Voting
Classifier.predict_proba(self, X)
    390 """Compute probabilities of possible outcomes for samples in X.
    391
    392 Parameters
   (...)
    400     Weighted average probability for each class per sample.
    401 """
    402 check_is_fitted(self)
    403 avg = np.average(
--> 404     self._collect_probas(X), axis=0, weights=self._weights_not_none
    405 )
    406 return avg

File ~\anaconda3\lib\site-packages\sklearn\ensemble\_voting.py:379, in Voting
Classifier._collect_probas(self, X)
    377 def _collect_probas(self, X):
    378     """Collect results from clf.predict calls."""
--> 379     return np.asarray([clf.predict_proba(X) for clf in self.estimator
s_])

File ~\anaconda3\lib\site-packages\sklearn\ensemble\_voting.py:379, in <listc
omp>(.0)
    377 def _collect_probas(self, X):
    378     """Collect results from clf.predict calls."""
--> 379     return np.asarray([clf.predict_proba(X) for clf in self.estimator
s_])

File ~\anaconda3\lib\site-packages\sklearn\utils\_available_if.py:32, in _Ava
ilableIfDescriptor.__get__(self, obj, owner)
     26 attr_err = AttributeError(
     27     f"This {repr(owner.__name__)} has no attribute {repr(self.attribu
te_name)}"
     28 )
     29 if obj is not None:
     30     # delegate only on instances, not the classes.
     31     # this is to allow access to the docstrings.
---> 32     if not self.check(obj):
     33         raise attr_err
     34     out = MethodType(self.fn, obj)

File ~\anaconda3\lib\site-packages\sklearn\svm\_base.py:829, in BaseSVC._chec
k_proba(self)
```

```
    827 def _check_proba(self):
    828     if not self.probability:
--> 829         raise AttributeError(
    830             "predict_proba is not available when  probability=False"
    831         )
    832     if self._impl not in ("c_svc", "nu_svc"):
    833         raise AttributeError("predict_proba only implemented for SVC
and NuSVC")

AttributeError: predict_proba is not available when  probability=False
```

In [ ]:

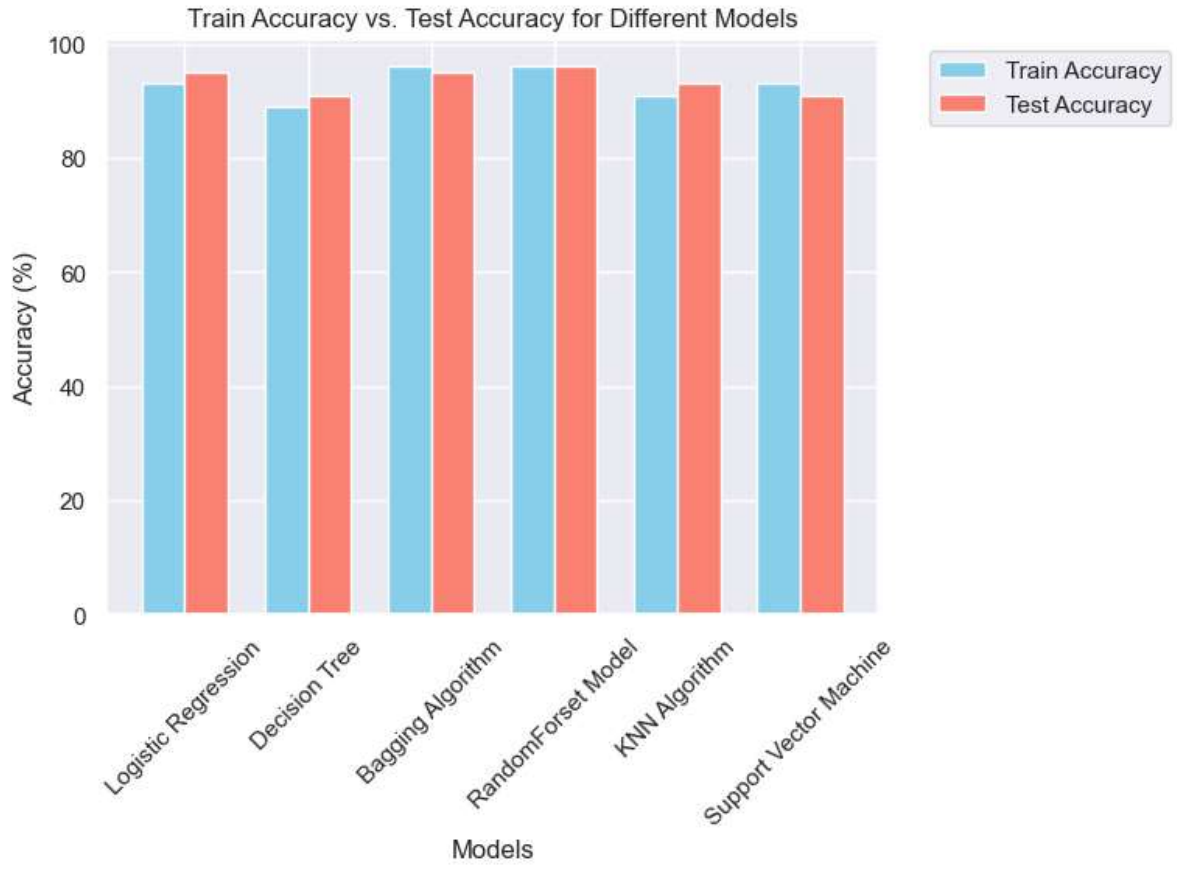
Visualize Every Model with their accuarcy values

In [97]:
```
data = [
    ['Model', 'Train Accuracy(%)', 'Test Accuracy(%)'],
    ['Logistic Regression', 93, '95'],
    ['Decision Tree', 89, '91'],
    ['Bagging Algorithm', 96, '95'],
    ['RandomForset Model', 96, '96'],
    ['KNN Algorithm', 91, '93'],
    ['Support Vector Machine', 93, '91'],

]
```

In [98]:
```
fig, ax = plt.subplots()
table = ax.table(cellText=data, loc='center')
table.auto_set_font_size(False)
table.set_fontsize(15)
table.scale(3, 3)

ax.axis('off')

plt.show()
```

Model	Train Accuracy(%)	Test Accuracy(%)
Logistic Regression	93	95
Decision Tree	89	91
Bagging Algorithm	96	95
RandomForset Model	96	96
KNN Algorithm	91	93
Support Vector Machine	93	91

In [99]:
```
# Extract model names and accuracy values
models = [item[0] for item in data[1:]]
train_accuracy = [float(item[1]) for item in data[1:]]
test_accuracy = [float(item[2]) for item in data[1:]]

# Generate positions for the bars
x = np.arange(len(models))

# Set width of the bars
bar_width = 0.35

# Plotting the bar plot
plt.bar(x - bar_width/2, train_accuracy, bar_width, label='Train Accuracy', cc
plt.bar(x + bar_width/2, test_accuracy, bar_width, label='Test Accuracy', colc

plt.xlabel('Models')
plt.ylabel('Accuracy (%)')
plt.title('Train Accuracy vs. Test Accuracy for Different Models')
plt.xticks(x, models, rotation=45)  # Set model names as x-axis labels
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
```



Conclusion: In this notebook, we implemented and compared various machine learning algorithms to classify breast cancer cells as benign or malignant. Each algorithm has its strengths and weaknesses, and the choice of the best model depends on the specific dataset

and problem at hand. By evaluating the performance of these models using appropriate metrics (e.g., accuracy, precision, recall, F1-score), we can select the most suitable model for