In [1]:
```
import numpy as np # linear algebra
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import *

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
import warnings
warnings.filterwarnings("ignore")
df=pd.read_csv("breast_cancer_expanded_dataset.csv")
df.head()
```

Out[1]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_m
0	3.048642e+06	M	17.971931	10.422523	123.038998	1010.679004	0.118376	0.276847	0.30
1	3.248717e+06	M	20.504629	17.854892	133.010474	1329.620513	0.084951	0.078766	0.08
2	8.308423e+07	M	19.733084	21.226184	130.072458	1206.386366	0.109611	0.161120	0.19
3	8.440706e+07	M	11.423847	20.427818	77.653815	384.723555	0.142544	0.283623	0.24
4	8.465858e+07	M	20.295838	14.297520	134.974219	1300.068231	0.100374	0.132542	0.19

5 rows × 32 columns

In [2]:
```
df.isnull().sum()
```

Out[2]:
```
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 [3]:
```
import pandas as pd
# Step 1: Selecting a dataset
df = pd.read_csv('breast_cancer_expanded_dataset.csv')
# Display the first few rows of the dataset:")
print("First few rows of the dataset:")
print(df.head())
```

```
# Display basic information about the dataset
print("\nDataset information:")
print(df.info())

# Summary statistics
print("\nSummary statistics of numerical columns:")
print(df.describe())

# Check for missing values
print("\nMissing values:")
print(df.isnull().sum())
```

```
First few rows of the dataset:
             id diagnosis  radius_mean  texture_mean  perimeter_mean  \
0  3.048642e+06         M    17.971931     10.422523      123.038998
1  3.248717e+06         M    20.504629     17.854892      133.010474
2  8.308423e+07         M    19.733084     21.226184      130.072458
3  8.440706e+07         M    11.423847     20.427818       77.653815
4  8.465858e+07         M    20.295838     14.297520      134.974219

     area_mean  smoothness_mean  compactness_mean  concavity_mean  \
0  1010.679004         0.118376          0.276847        0.301246
1  1329.620513         0.084951          0.078766        0.085893
2  1206.386366         0.109611          0.161120        0.198109
3   384.723555         0.142544          0.283623        0.241326
4  1300.068231         0.100374          0.132542        0.196814

   concave points_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0             0.147314  ...     25.500017      17.217229       184.775788
1             0.070364  ...     25.012428      23.414819       158.779310
2             0.128497  ...     23.565253      25.634042       152.280293
3             0.105410  ...     14.876205      26.476193        99.041551
4             0.104399  ...     22.438373      16.638056       152.383430

    area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0  2023.940579          0.162339           0.666989         0.710214
1  1950.232129          0.124237           0.187702         0.241233
2  1702.704701          0.144531           0.427018         0.448880
3   564.829786          0.209767           0.863589         0.687396
4  1577.492830          0.137631           0.206669         0.401075

   concave points_worst  symmetry_worst  fractal_dimension_worst
0              0.265724        0.460163                 0.119134
1              0.186230        0.274851                 0.089053
2              0.243555        0.361381                 0.087787
3              0.257700        0.663555                 0.172784
4              0.162672        0.236337                 0.076846

[5 rows x 32 columns]

Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
```

```
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       5000 non-null   float64
 1   diagnosis                5000 non-null   object 
 2   radius_mean              5000 non-null   float64
 3   texture_mean             5000 non-null   float64
 4   perimeter_mean           5000 non-null   float64
 5   area_mean                5000 non-null   float64
 6   smoothness_mean          5000 non-null   float64
 7   compactness_mean         5000 non-null   float64
 8   concavity_mean           5000 non-null   float64
 9   concave points_mean      5000 non-null   float64
 10  symmetry_mean            5000 non-null   float64
 11  fractal_dimension_mean   5000 non-null   float64
 12  radius_se                5000 non-null   float64
 13  texture_se               5000 non-null   float64
 14  perimeter_se             5000 non-null   float64
 15  area_se                  5000 non-null   float64
 16  smoothness_se            5000 non-null   float64
 17  compactness_se           5000 non-null   float64
 18  concavity_se             5000 non-null   float64
 19  concave points_se        5000 non-null   float64
 20  symmetry_se              5000 non-null   float64
 21  fractal_dimension_se     5000 non-null   float64
 22  radius_worst             5000 non-null   float64
 23  texture_worst            5000 non-null   float64
 24  perimeter_worst          5000 non-null   float64
 25  area_worst               5000 non-null   float64
 26  smoothness_worst         5000 non-null   float64
 27  compactness_worst        5000 non-null   float64
 28  concavity_worst          5000 non-null   float64
 29  concave points_worst     5000 non-null   float64
 30  symmetry_worst           5000 non-null   float64
 31  fractal_dimension_worst  5000 non-null   float64
dtypes: float64(31), object(1)
memory usage: 1.2+ MB
None

Summary statistics of numerical columns:
                id  radius_mean  texture_mean  perimeter_mean    area_mean  \
```

```
count  5.000000e+03  5000.000000   5000.000000     5000.000000  5000.000000
mean   2.989844e+07    14.135533     19.261524       92.026807   655.583620
std    1.232414e+08     3.518936      4.278956       24.266795   351.286487
min   -4.121749e+06     6.927863      9.626705       43.569989   139.607998
25%    3.822221e+05    11.693910     16.160488       75.130449   419.537486
50%    1.546392e+06    13.362077     18.822353       86.258332   550.326850
75%    8.480591e+06    15.830531     21.754581      104.322449   788.919500
max    9.150227e+08    28.138334     39.343381      189.027651  2505.483736

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count      5000.000000       5000.000000     5000.000000          5000.000000
mean          0.096377          0.104419        0.089036             0.049047
std           0.014023          0.052902        0.079787             0.038808
min           0.052476          0.018415       -0.002386            -0.000848
25%           0.086344          0.064839        0.029247             0.020338
50%           0.095930          0.092950        0.061493             0.033679
75%           0.105262          0.130526        0.131597             0.074101
max           0.163653          0.345700        0.428172             0.201719

       symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
count    5000.000000  ...   5000.000000    5000.000000      5000.000000
mean        0.181302  ...     16.285191      25.648936       107.364586
std         0.027461  ...      4.831246       6.132811        33.566240
min         0.105473  ...      7.864242      11.984863        50.063108
25%         0.161981  ...     13.008701      21.065950        84.137958
50%         0.179427  ...     14.954424      25.369004        97.798772
75%         0.195724  ...     18.874108      29.673555       125.961081
max         0.304494  ...     36.116542      49.563874       251.791418

        area_worst  smoothness_worst  compactness_worst  concavity_worst  \
count  5000.000000       5000.000000        5000.000000      5000.000000
mean    882.095111          0.132389           0.254670         0.272709
std     568.549258          0.022837           0.157807         0.208886
min     178.654953          0.070779           0.024959        -0.006670
25%     514.527050          0.116578           0.146428         0.114010
50%     685.147018          0.131370           0.213824         0.227951
75%    1090.604440          0.146111           0.339797         0.384436
max    4260.299732          0.223145           1.059926         1.255012

       concave points_worst  symmetry_worst  fractal_dimension_worst
count           5000.000000     5000.000000              5000.000000
```

```
mean               0.114834        0.290495                0.083959
std                0.065792        0.062075                0.018086
min               -0.001301        0.155650                0.054791
25%                0.064416        0.250614                0.071371
50%                0.100302        0.282350                0.079984
75%                0.162539        0.318474                0.092066
max                0.291860        0.664396                0.207723

[8 rows x 31 columns]

Missing values:
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 [4]:
```
df.describe()
```

Out[4]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
count	5.000000e+03	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	2.989844e+07	14.135533	19.261524	92.026807	655.583620	0.096377	0.104419	0.089036
std	1.232414e+08	3.518936	4.278956	24.266795	351.286487	0.014023	0.052902	0.079787
min	-4.121749e+06	6.927863	9.626705	43.569989	139.607998	0.052476	0.018415	-0.002386
25%	3.822221e+05	11.693910	16.160488	75.130449	419.537486	0.086344	0.064839	0.029247
50%	1.546392e+06	13.362077	18.822353	86.258332	550.326850	0.095930	0.092950	0.061493
75%	8.480591e+06	15.830531	21.754581	104.322449	788.919500	0.105262	0.130526	0.131597
max	9.150227e+08	28.138334	39.343381	189.027651	2505.483736	0.163653	0.345700	0.428172

8 rows × 31 columns

In [5]:
```
# Define features and target
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

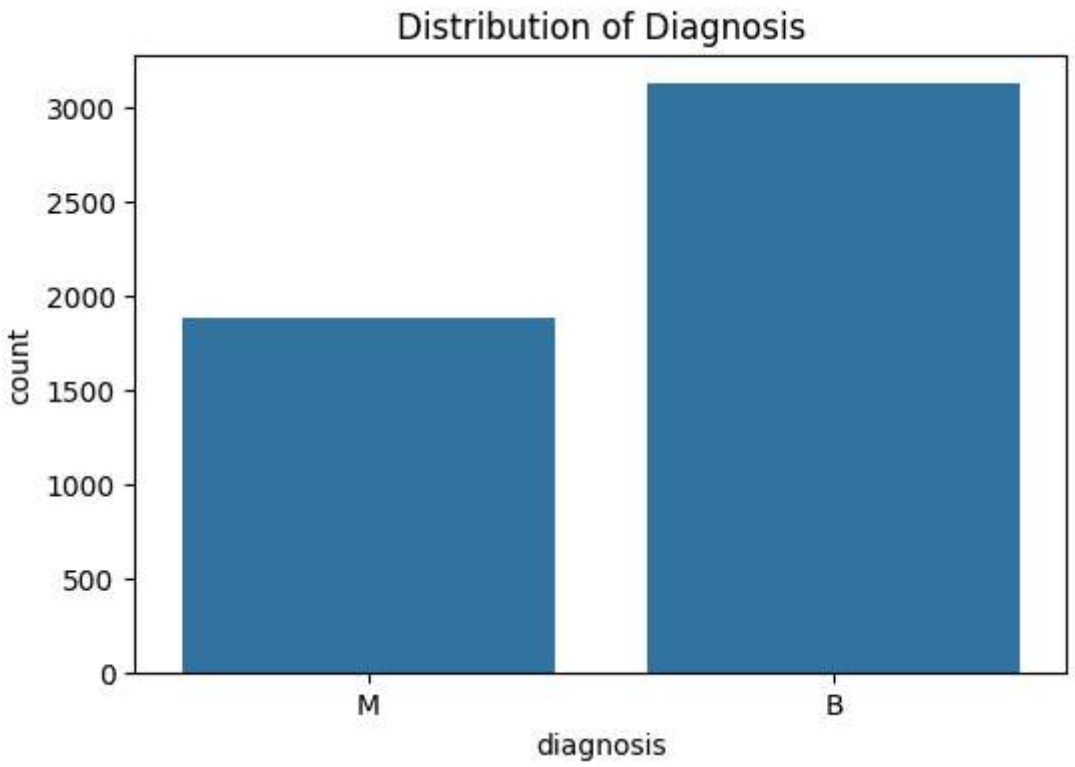
Out[5]:
```
array([[-0.21788517,  1.09032416, -2.06589728, ...,  2.29365626,
         2.73357064,  1.94507838],
       [-0.21626157,  1.81013029, -0.32876528, ...,  1.08527968,
        -0.25204188,  0.28169359],
       [ 0.43160086,  1.59085314,  0.45919067, ...,  1.95667569,
         1.14206513,  0.21164974],
       ...,
       [-0.15964576, -0.59215304,  1.32032701, ...,  0.0853743 ,
        -0.50309821, -0.07264692],
       [-0.18241475,  1.03305298,  2.05527866, ...,  1.24026865,
         0.11126857,  0.36769246],
       [-0.1656325 ,  0.20014292, -0.34792339, ..., -0.47784217,
         1.11704771, -0.60219062]])
```

In [6]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (3500, 31)
Shape of X_test: (1500, 31)
Shape of y_train: (3500,)
Shape of y_test: (1500,)
```

In [7]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='diagnosis', data=df)
plt.title('Distribution of Diagnosis')
plt.show()

# Descriptive Statistics
print("Descriptive Statistics:\n", df.describe())

# Assuming df is your DataFrame
numeric_df = df.select_dtypes(include='number')  # Selecting numeric columns
plt.figure(figsize=(12, 10))
sns.heatmap(numeric_df.corr(), annot=False, cmap='coolwarm')
```

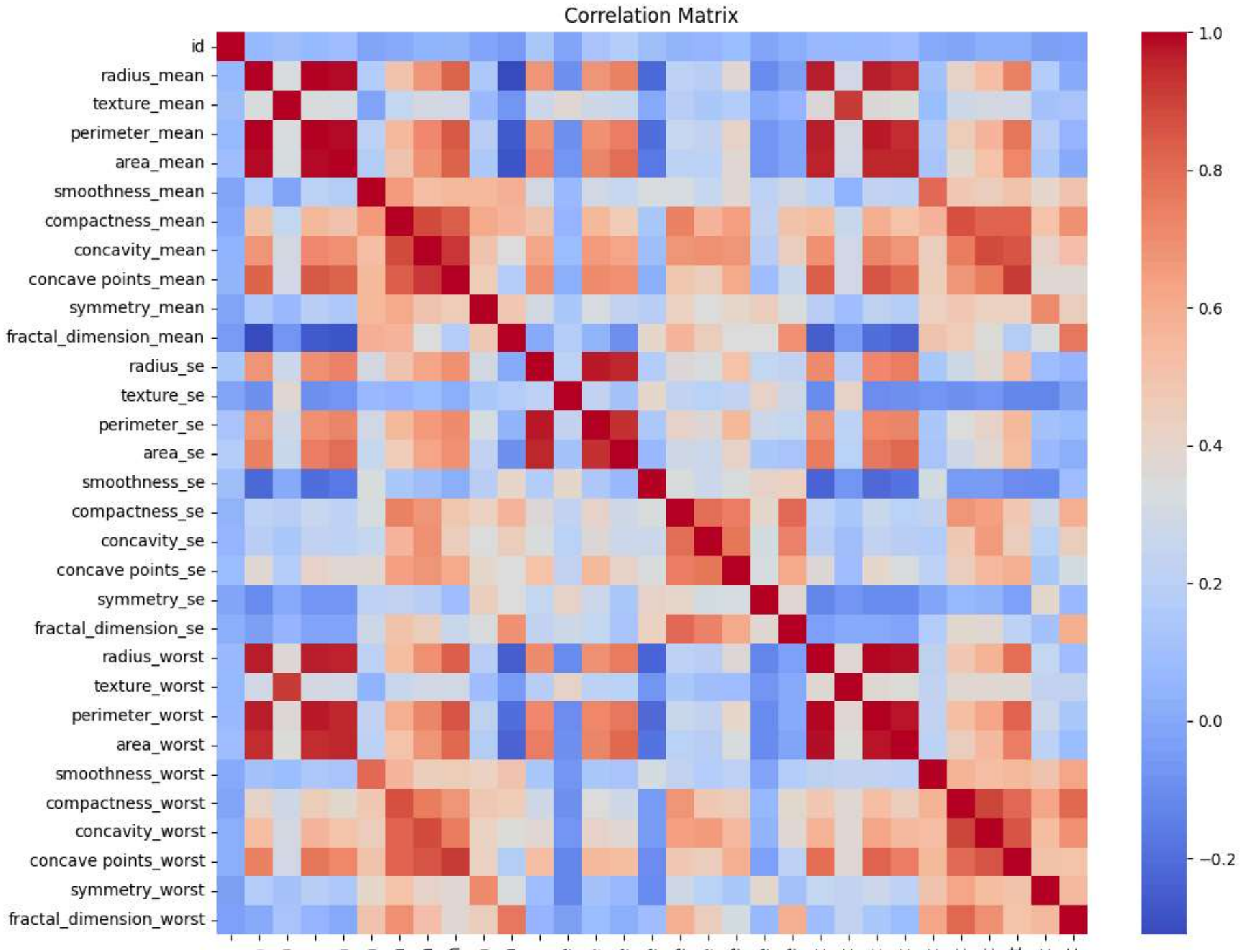
```
plt.title('Correlation Matrix')
plt.show()
```



```
Descriptive Statistics:
                 id  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  5.000000e+03  5000.000000   5000.000000     5000.000000  5000.000000
mean   2.989844e+07    14.135533     19.261524       92.026807   655.583620
std    1.232414e+08     3.518936      4.278956       24.266795   351.286487
min   -4.121749e+06     6.927863      9.626705       43.569989   139.607998
25%    3.822221e+05    11.693910     16.160488       75.130449   419.537486
50%    1.546392e+06    13.362077     18.822353       86.258332   550.326850
75%    8.480591e+06    15.830531     21.754581      104.322449   788.919500
max    9.150227e+08    28.138334     39.343381      189.027651  2505.483736

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count      5000.000000       5000.000000     5000.000000          5000.000000
mean          0.096377          0.104419        0.089036             0.049047
std           0.014023          0.052902        0.079787             0.038808
min           0.052476          0.018415       -0.002386            -0.000848
25%           0.086344          0.064839        0.029247             0.020338
50%           0.095930          0.092950        0.061493             0.033679
75%           0.105262          0.130526        0.131597             0.074101
max           0.163653          0.345700        0.428172             0.201719

       symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
count    5000.000000  ...   5000.000000    5000.000000      5000.000000
mean        0.181302  ...     16.285191      25.648936       107.364586
std         0.027461  ...      4.831246       6.132811        33.566240
min         0.105473  ...      7.864242      11.984863        50.063108
25%         0.161981  ...     13.008701      21.065950        84.137958
50%         0.179427  ...     14.954424      25.369004        97.798772
75%         0.195724  ...     18.874108      29.673555       125.961081
max         0.304494  ...     36.116542      49.563874       251.791418

        area_worst  smoothness_worst  compactness_worst  concavity_worst  \
count  5000.000000       5000.000000        5000.000000      5000.000000
mean    882.095111          0.132389           0.254670         0.272709
std     568.549258          0.022837           0.157807         0.208886
min     178.654953          0.070779           0.024959        -0.006670
25%     514.527050          0.116578           0.146428         0.114010
50%     685.147018          0.131370           0.213824         0.227951
75%    1090.604440          0.146111           0.339797         0.384436
max    4260.299732          0.223145           1.059926         1.255012
```

	concave points_worst	symmetry_worst	fractal_dimension_worst
count	5000.000000	5000.000000	5000.000000
mean	0.114834	0.290495	0.083959
std	0.065792	0.062075	0.018086
min	-0.001301	0.155650	0.054791
25%	0.064416	0.250614	0.071371
50%	0.100302	0.282350	0.079984
75%	0.162539	0.318474	0.092066
max	0.291860	0.664396	0.207723

[8 rows x 31 columns]

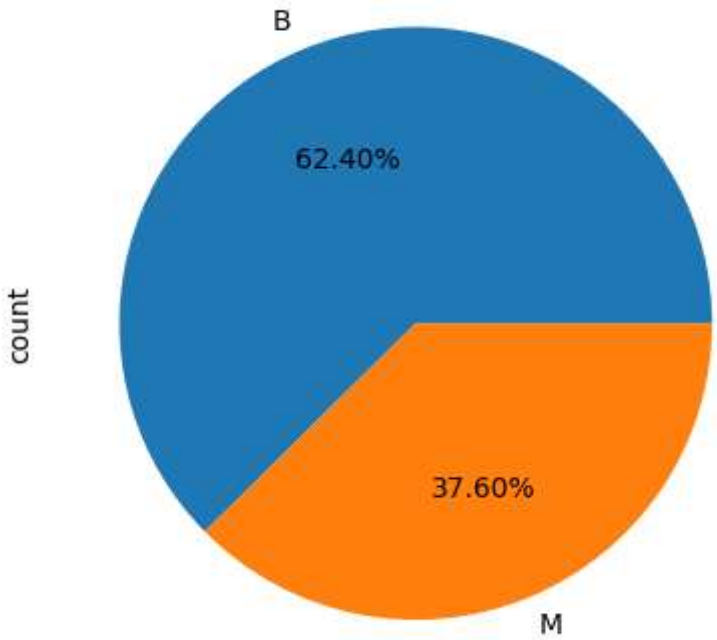

Correlation Matrix
id
radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
radius_se
texture_se
perimeter_se
area_se
smoothness_se
compactness_se
concavity_se
concave points_se
symmetry_se
fractal_dimension_se
radius_worst
texture_worst
perimeter_worst
area_worst
smoothness_worst
compactness_worst
concavity_worst
concave points_worst
symmetry_worst
fractal_dimension_worst
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

id
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

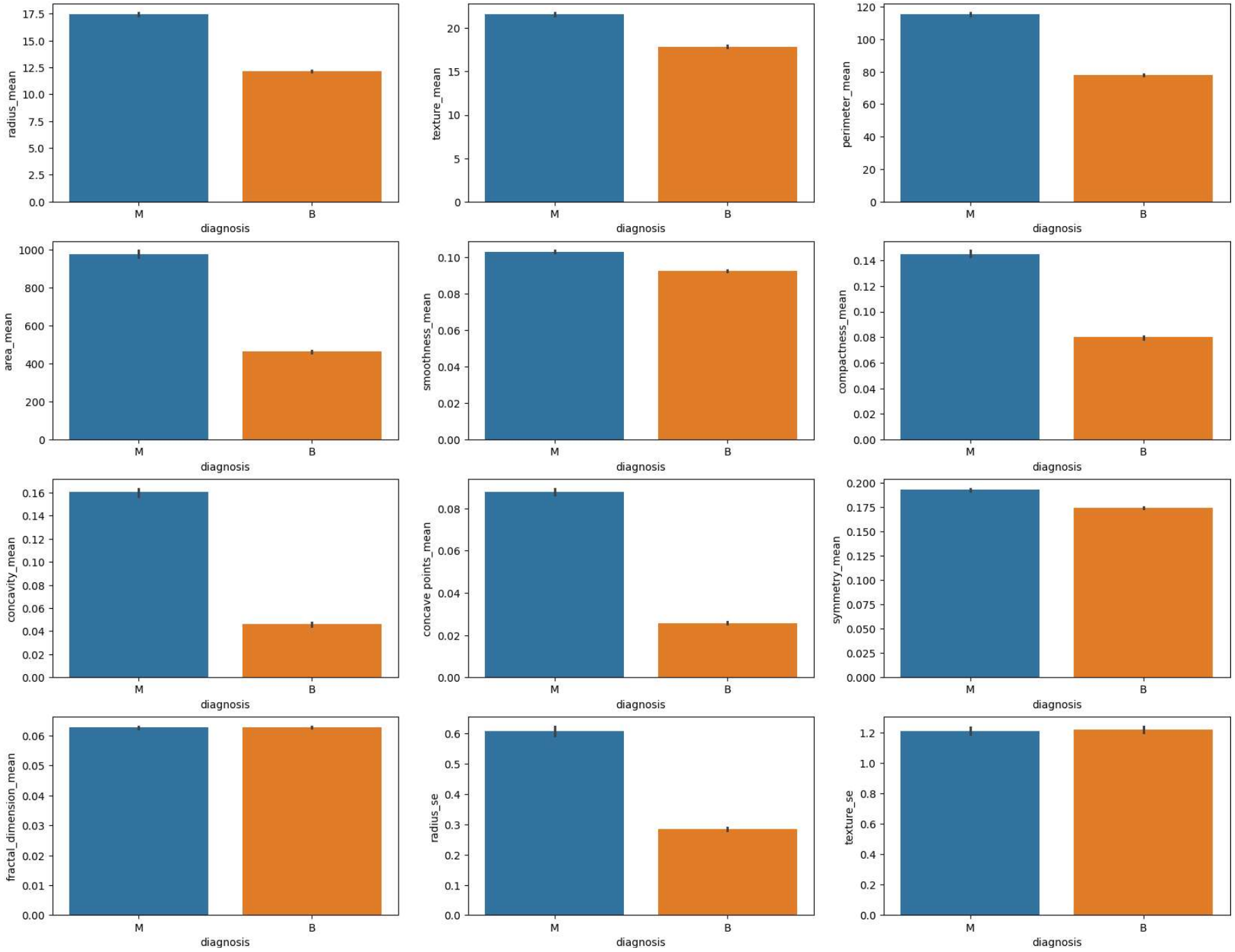
In [8]:
```
df.drop(columns='id',inplace=True)
```

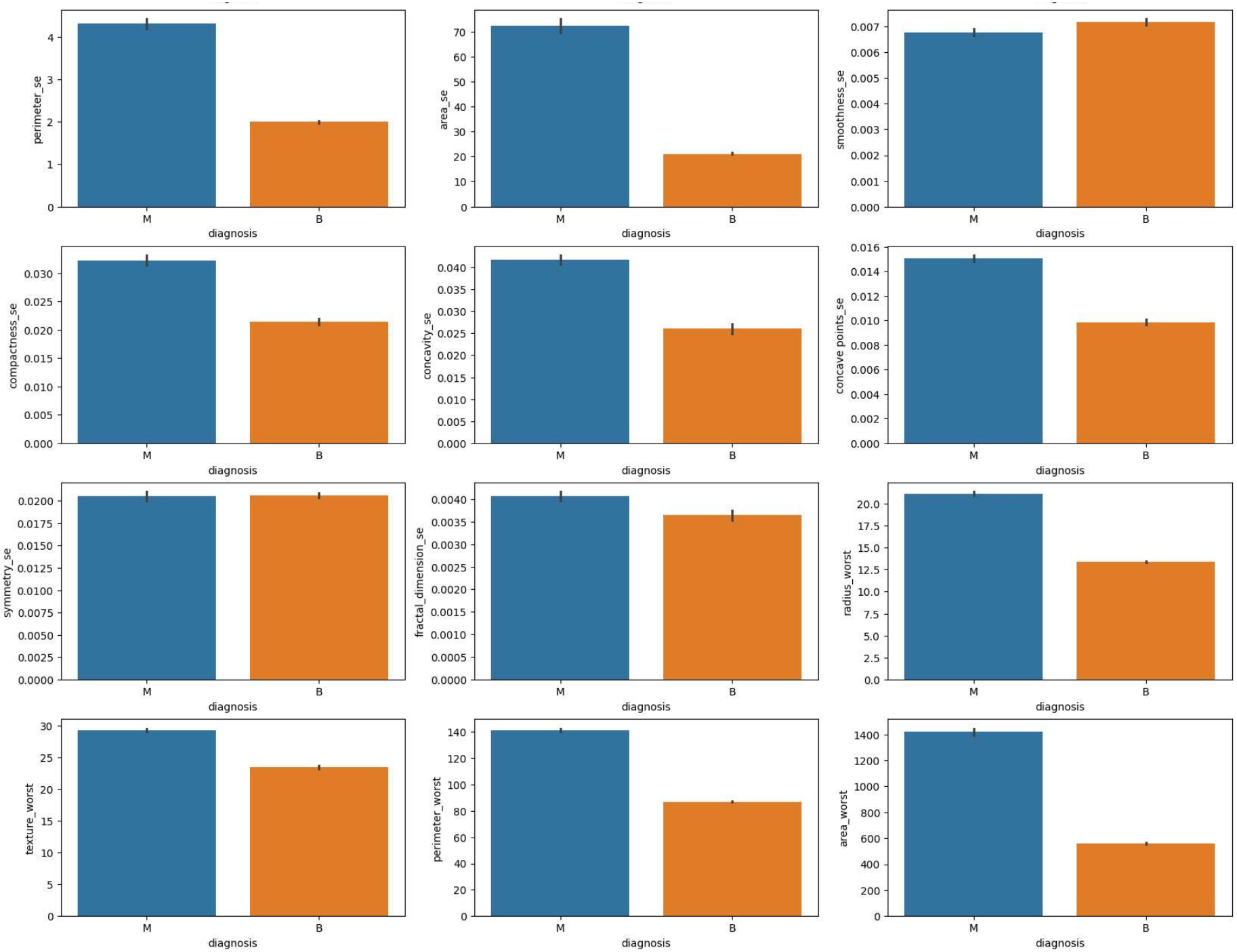
In [9]:
```
df['diagnosis'].value_counts().plot(kind='pie',autopct='%.2f%%')
```

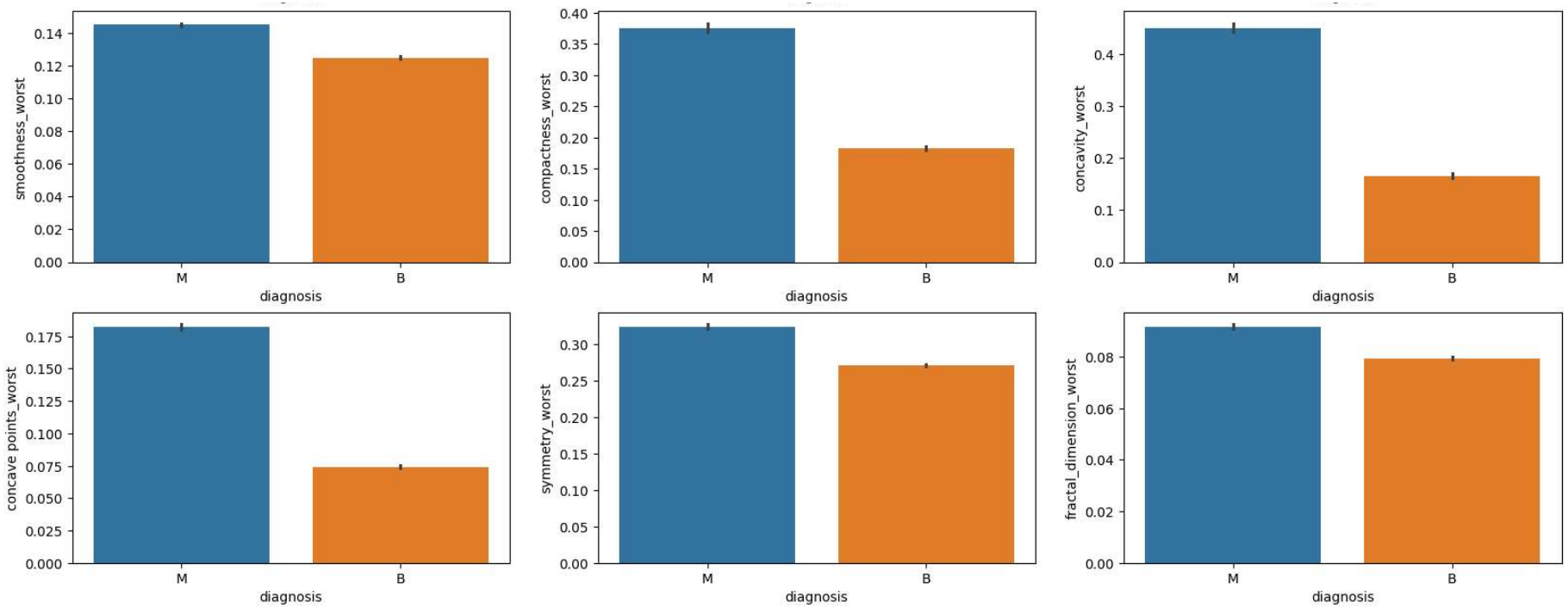
Out[9]: <Axes: ylabel='count'>



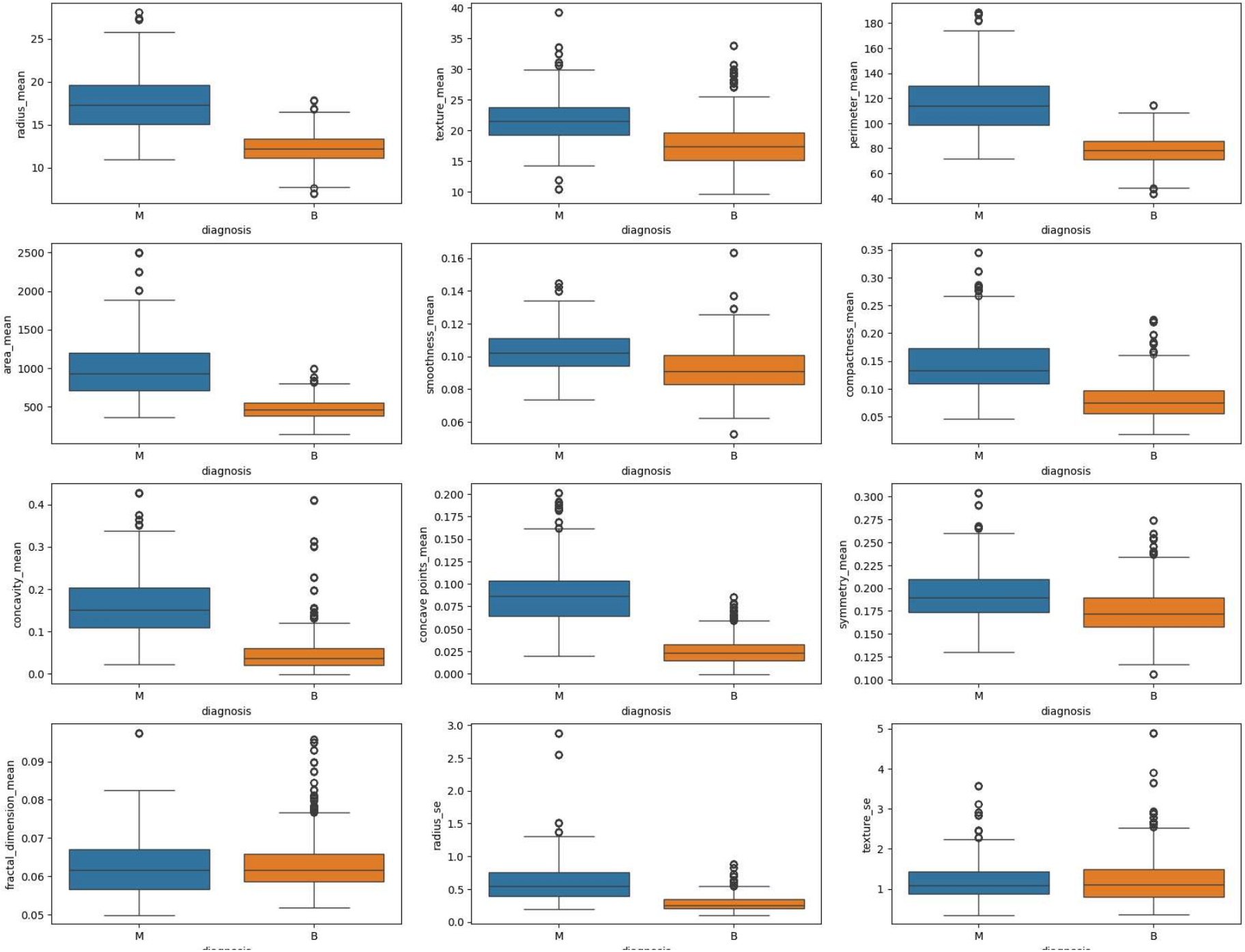
In [10]:
```
fig,ax=plt.subplots(figsize=(20,40),nrows=10,ncols=3)
ax=ax.flatten()
for i,col in enumerate(df.columns[1:]):
    # df[col]=np.log(df[col])
    sns.barplot(x='diagnosis',y=col,data=df,ax=ax[i],hue='diagnosis');
```

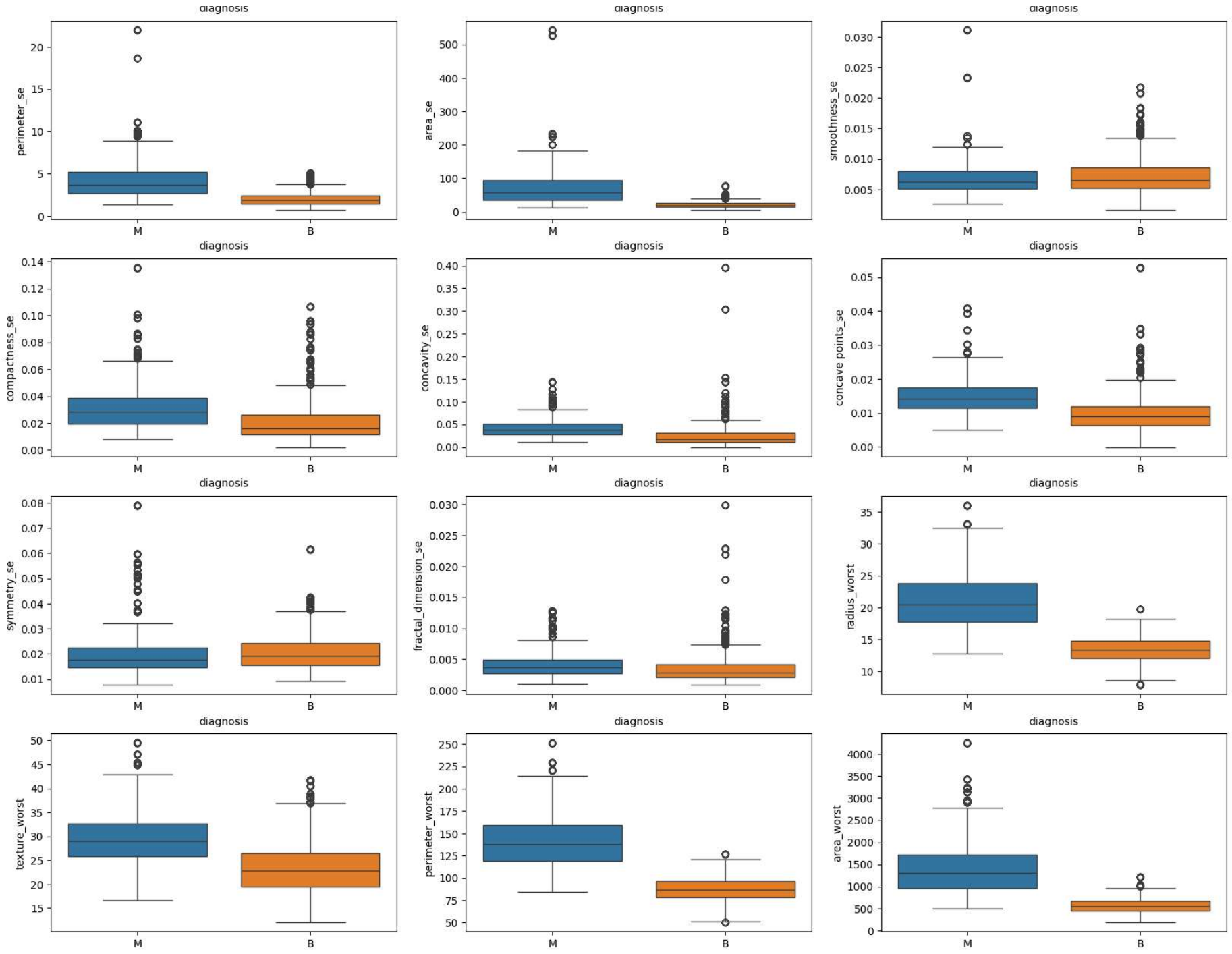

radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
radius_se
texture_se
M
B
diagnosis

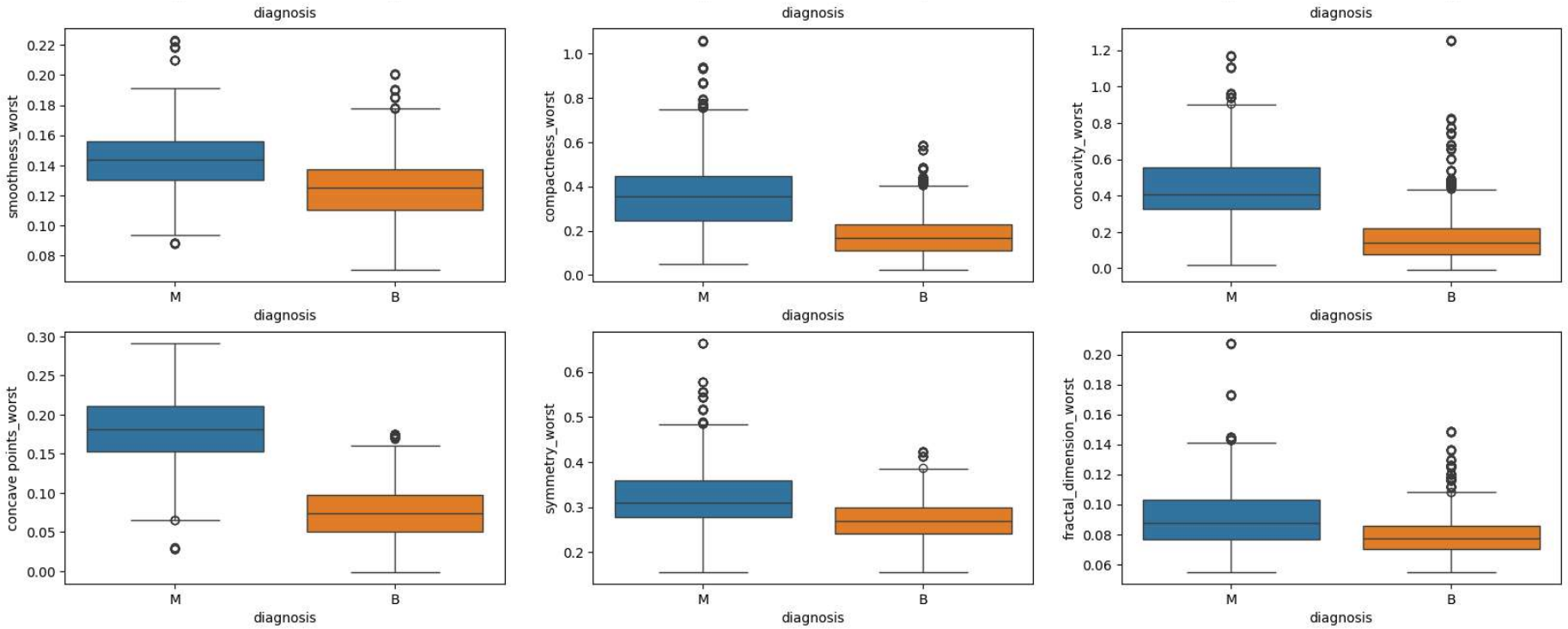

perimeter_se
area_se
smoothness_se
compactness_se
concavity_se
concave points_se
symmetry_se
fractal_dimension_se
radius_worst
texture_worst
perimeter_worst
area_worst
diagnosis
M
B



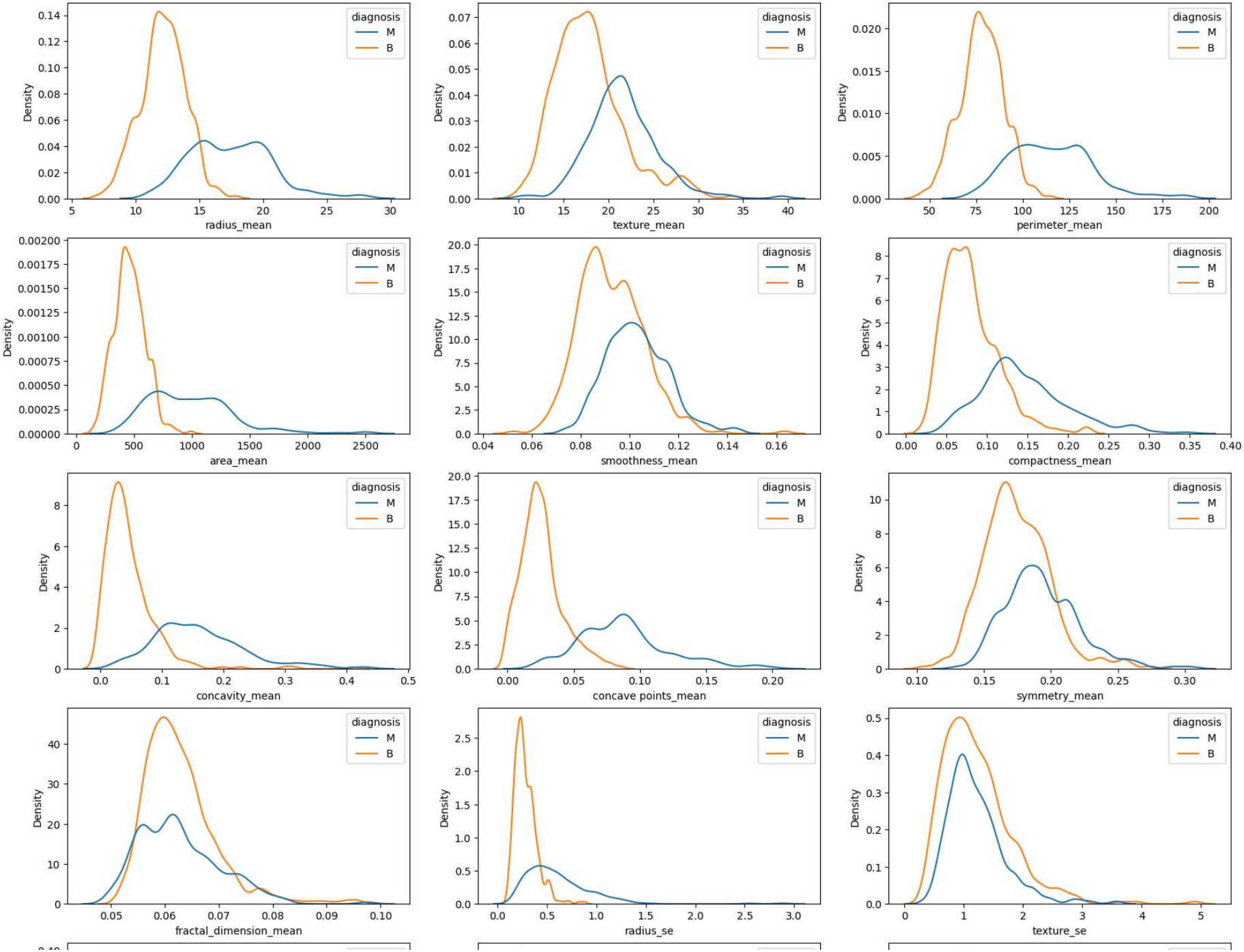
In [11]:
```
fig,ax=plt.subplots(figsize=(20,40),nrows=10,ncols=3)
ax=ax.flatten()
for i,col in enumerate(df.columns[1:]):
    # df[col]=np.log(df[col])
    sns.boxplot(x='diagnosis',y=col,data=df,ax=ax[i], hue='diagnosis');
```

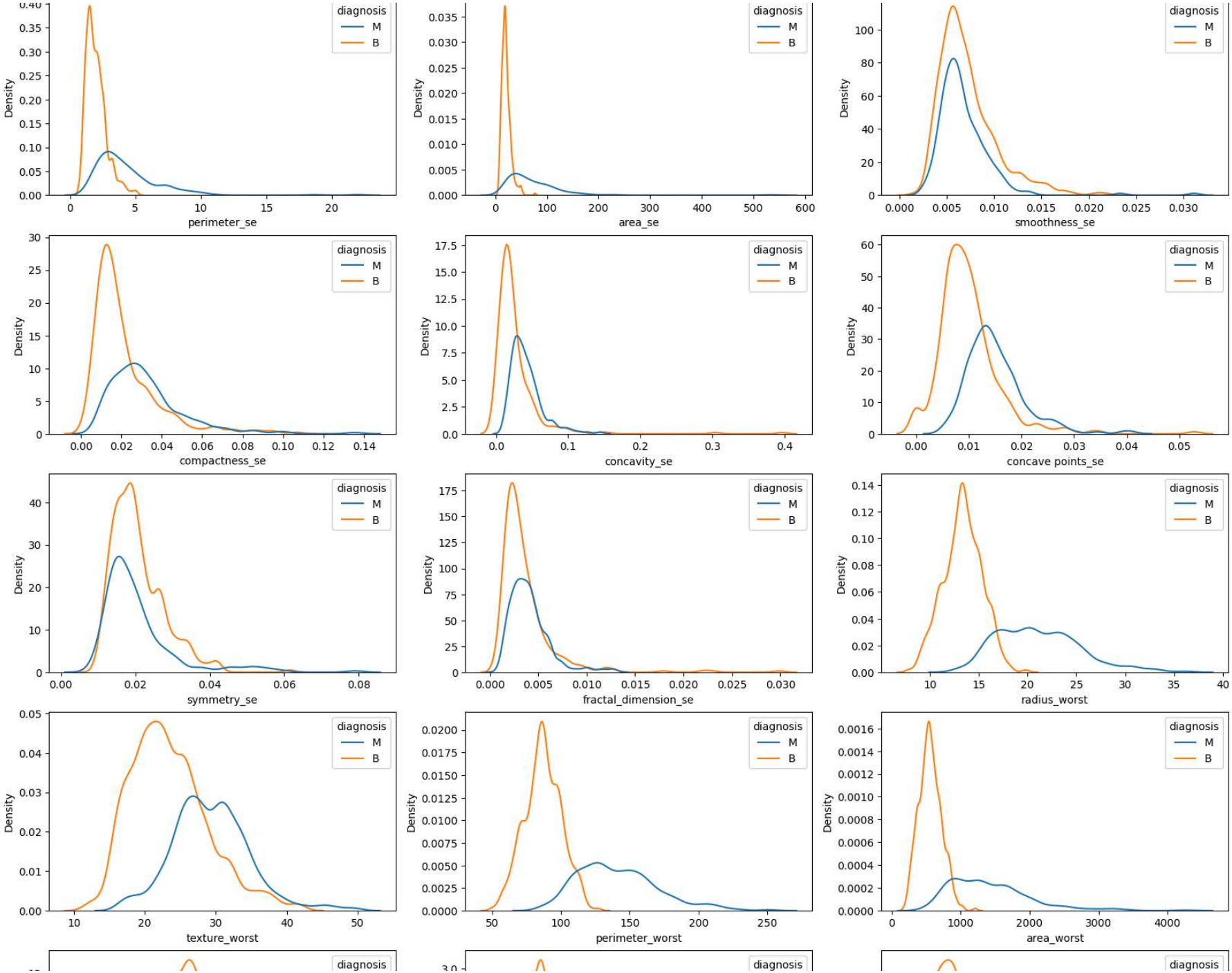

radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
radius_se
texture_se
M
B
diagnosis

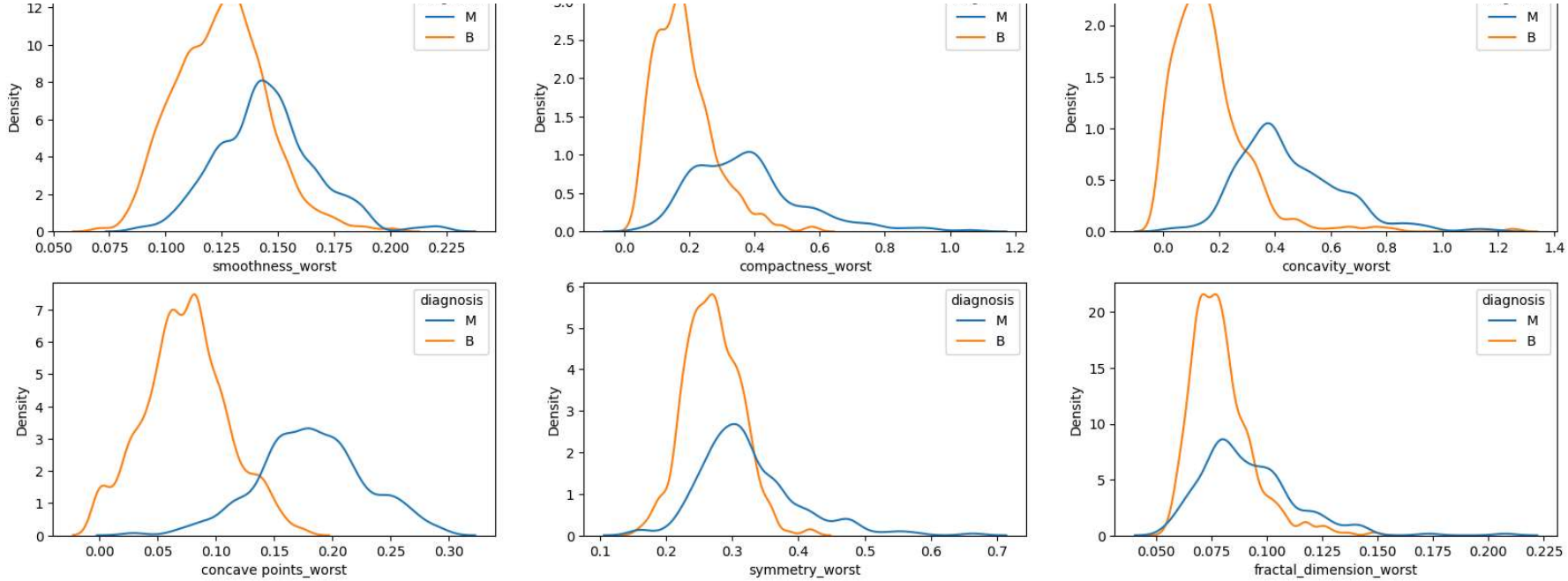

diagnosis
perimeter_se
area_se
smoothness_se
compactness_se
concavity_se
concave points_se
symmetry_se
fractal_dimension_se
radius_worst
texture_worst
perimeter_worst
area_worst
M
B



In [12]:
```
fig,ax=plt.subplots(figsize=(20,40),nrows=10,ncols=3)
ax=ax.flatten()
for i,col in enumerate(df.columns[1:]):
    # df[col]=np.log(df[col])
    sns.kdeplot(x=col,data=df,ax=ax[i],hue='diagnosis');
```


diagnosis
M
B
Density
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


diagnosis
M
B
Density
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

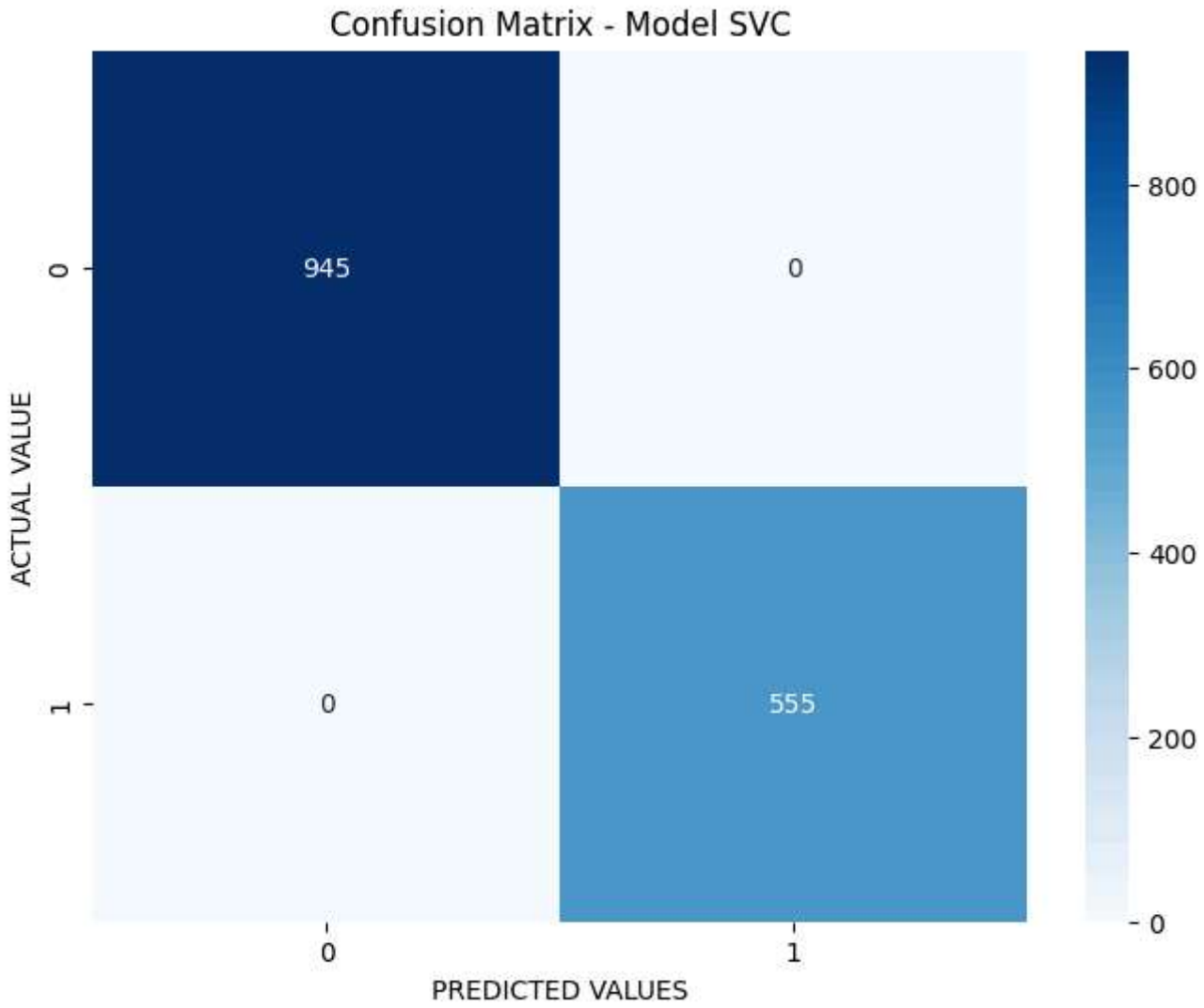


In [13]:
```
from sklearn.feature_selection import SelectKBest, f_classif

# Apply SelectKBest class to extract top 10 features
bestfeatures = SelectKBest(score_func=f_classif, k=10)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)

# Concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Feature','Score']  # naming the dataframe columns
print(featureScores.nlargest(10,'Score'))  # print 10 best features
```

	Feature	Score
28	concave points_worst	8540.218727
23	perimeter_worst	7914.084643
21	radius_worst	7583.737425
8	concave points_mean	7565.344127
3	perimeter_mean	6125.966416
24	area_worst	5830.067597
1	radius_mean	5686.059680
4	area_mean	5031.640067
7	concavity_mean	4661.311433
27	concavity_worst	3849.139920

In [14]:
```
from sklearn.neural_network import MLPClassifier

# Creating MLP classifier with adjusted parameters
mlp_classifier = MLPClassifier(max_iter=1000, solver='adam')  # Adjust max_iter and solver as needed

# Training the classifier
mlp_classifier.fit(X_train, y_train)

# Predicting on test set
mlp_predictions = mlp_classifier.predict(X_test)

# Evaluating accuracy
mlp_accuracy = accuracy_score(y_test, mlp_predictions)
print("Neural Network Accuracy:", mlp_accuracy)
mlp_conf_matrix = confusion_matrix(y_test, mlp_predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(mlp_conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=True)
plt.title("Confusion Matrix - Model SVC")
plt.xlabel("PREDICTED VALUES")
plt.ylabel("ACTUAL VALUE")
plt.show()
```

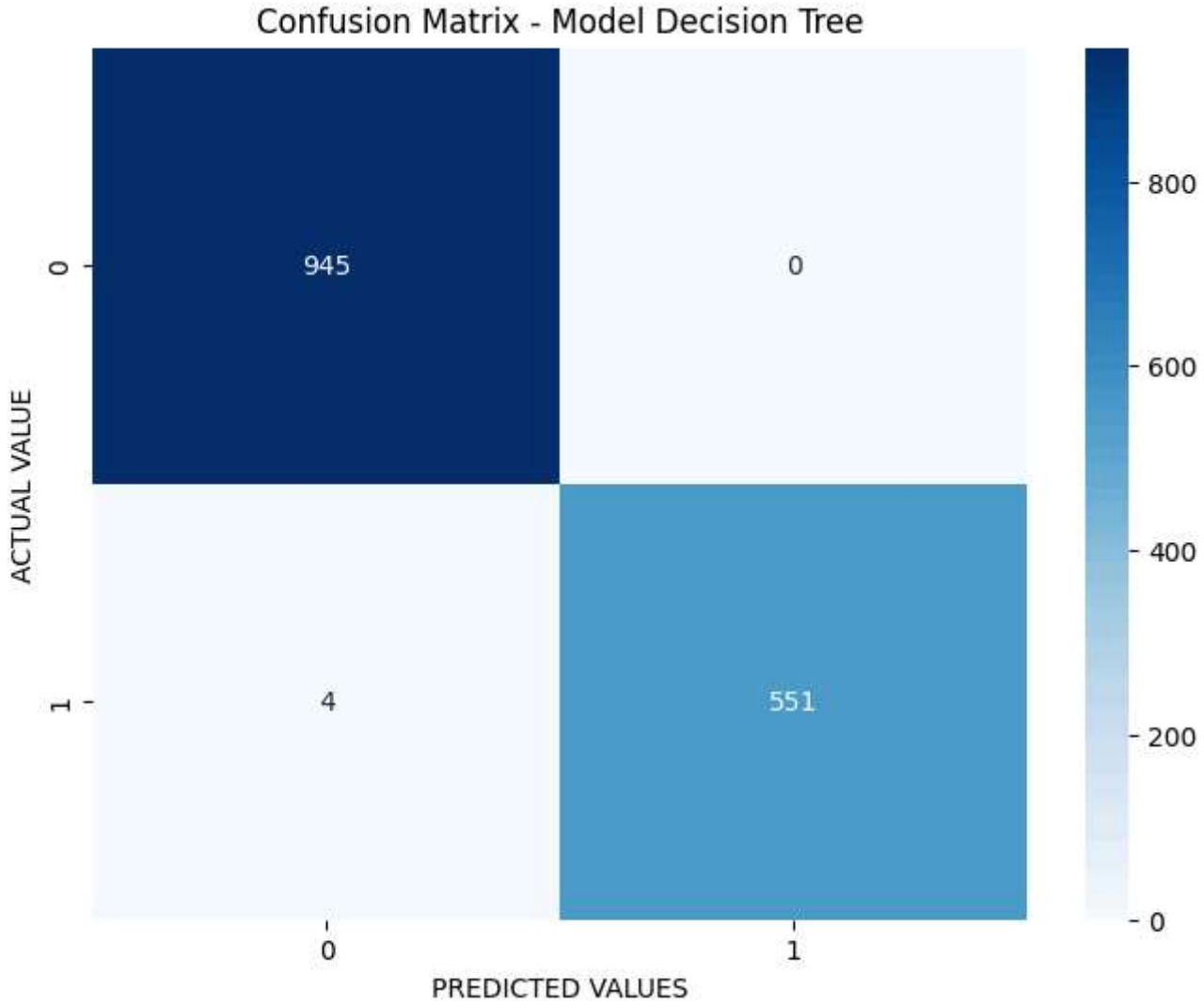
```
Neural Network Accuracy: 1.0
```



In [15]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# Assuming 'X_train', 'X_test', 'y_train', 'y_test' are already defined

# Creating Decision Tree classifier with default parameters
dt_classifier = DecisionTreeClassifier()  # Default settings; you can adjust parameters like max_depth

# Training the classifier
dt_classifier.fit(X_train, y_train)

# Predicting on test set
dt_predictions = dt_classifier.predict(X_test)

# Evaluating accuracy
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy)

# Generating confusion matrix
dt_conf_matrix = confusion_matrix(y_test, dt_predictions)

# Visualizing the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(dt_conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=True)
plt.title("Confusion Matrix - Model Decision Tree")
plt.xlabel("PREDICTED VALUES")
plt.ylabel("ACTUAL VALUE")
plt.show()
```

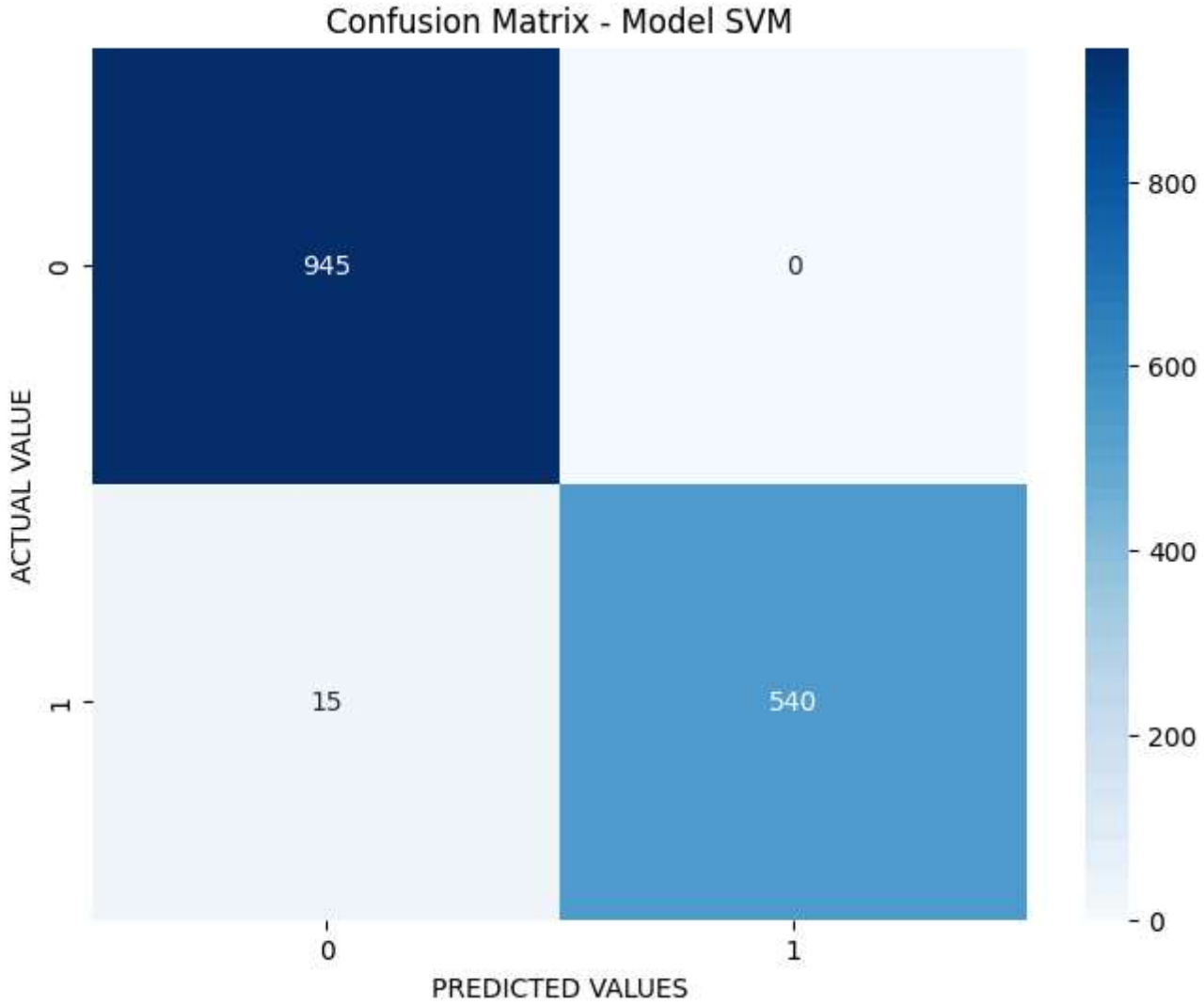
```
Decision Tree Accuracy: 0.9973333333333333
```



In [16]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# Assuming 'X_train', 'X_test', 'y_train', 'y_test' are already defined

# Creating SVM classifier with default parameters
svm_classifier = SVC(kernel='linear')  # Using a linear kernel; adjust as needed

# Training the classifier
svm_classifier.fit(X_train, y_train)

# Predicting on test set
svm_predictions = svm_classifier.predict(X_test)

# Evaluating accuracy
svm_accuracy = accuracy_score(y_test, svm_predictions)
print("SVM Accuracy:", svm_accuracy)

# Generating confusion matrix
svm_conf_matrix = confusion_matrix(y_test, svm_predictions)

# Visualizing the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(svm_conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=True)
plt.title("Confusion Matrix - Model SVM")
plt.xlabel("PREDICTED VALUES")
plt.ylabel("ACTUAL VALUE")
plt.show()
```

```
SVM Accuracy: 0.99
```



In [17]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```
# Assuming you have your data loaded into X_train and y_train

# Neural Network hyperparameters
param_grid_nn = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.001, 0.01]  # Regularization term
}

# Creating the GridSearchCV object
grid_search_nn = GridSearchCV(MLPClassifier(max_iter=1000, random_state=42), param_grid_nn, refit=True, verbose=2, cv=3)

# Fit the grid search to your data
grid_search_nn.fit(X_train, y_train)

# Get the best neural network model
best_nn_model = grid_search_nn.best_estimator_

# Use the best model to predict on your test set
y_pred = best_nn_model.predict(X_test)

# Calculate accuracy on the test set
test_accuracy = accuracy_score(y_test, y_pred)

# Displaying the best parameters and the score achieved with them
print("Best Neural Network Parameters:", grid_search_nn.best_params_)
print("Best Neural Network Score:", grid_search_nn.best_score_)
print("Test Set Accuracy:", test_accuracy)
```

```
Fitting 3 folds for each of 48 candidates, totalling 144 fits
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50,), solver=sgd; total time=   4.4s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50,), solver=sgd; total time=   4.6s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50,), solver=sgd; total time=   4.8s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50,), solver=adam; total time=   5.3s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50,), solver=adam; total time=   5.2s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50,), solver=adam; total time=   5.6s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100,), solver=sgd; total time=   5.3s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100,), solver=sgd; total time=   5.0s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100,), solver=sgd; total time=   5.5s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100,), solver=adam; total time=   5.7s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100,), solver=adam; total time=   6.0s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100,), solver=adam; total time=   5.9s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=sgd; total time=   8.8s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=sgd; total time=   7.9s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=sgd; total time=   8.9s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=adam; total time=   5.0s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=adam; total time=   5.5s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=adam; total time=   4.5s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=sgd; total time=   8.5s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=sgd; total time=   7.6s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=sgd; total time=   8.6s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=adam; total time=   4.6s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=adam; total time=   4.7s
[CV] END activation=tanh, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=adam; total time=   5.0s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50,), solver=sgd; total time=   4.5s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50,), solver=sgd; total time=   4.5s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50,), solver=sgd; total time=   4.9s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50,), solver=adam; total time=   5.5s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50,), solver=adam; total time=   5.3s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50,), solver=adam; total time=   5.8s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100,), solver=sgd; total time=   5.6s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100,), solver=sgd; total time=   5.2s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100,), solver=sgd; total time=   5.6s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100,), solver=adam; total time=   5.8s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100,), solver=adam; total time=   5.9s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100,), solver=adam; total time=   5.9s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50, 50), solver=sgd; total time=   8.8s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50, 50), solver=sgd; total time=   8.0s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50, 50), solver=sgd; total time=   9.0s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50, 50), solver=adam; total time=   4.9s
```

```
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50, 50), solver=adam; total time=   5.5s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(50, 50), solver=adam; total time=   4.6s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100, 100), solver=sgd; total time=   8.5s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100, 100), solver=sgd; total time=   7.5s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100, 100), solver=sgd; total time=   8.4s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100, 100), solver=adam; total time=   4.5s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100, 100), solver=adam; total time=   4.6s
[CV] END activation=tanh, alpha=0.001, hidden_layer_sizes=(100, 100), solver=adam; total time=   4.9s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50,), solver=sgd; total time=   4.5s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50,), solver=sgd; total time=   4.5s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50,), solver=sgd; total time=   4.9s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50,), solver=adam; total time=   5.4s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50,), solver=adam; total time=   5.2s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50,), solver=adam; total time=   5.6s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100,), solver=sgd; total time=   5.3s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100,), solver=sgd; total time=   5.0s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100,), solver=sgd; total time=   5.5s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100,), solver=adam; total time=   5.7s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100,), solver=adam; total time=   5.9s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100,), solver=adam; total time=   5.8s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50, 50), solver=sgd; total time=   8.8s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50, 50), solver=sgd; total time=   7.9s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50, 50), solver=sgd; total time=   8.9s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50, 50), solver=adam; total time=   5.0s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50, 50), solver=adam; total time=   5.5s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(50, 50), solver=adam; total time=   4.6s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100, 100), solver=sgd; total time=   8.5s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100, 100), solver=sgd; total time=   7.5s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100, 100), solver=sgd; total time=   8.4s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100, 100), solver=adam; total time=   4.5s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100, 100), solver=adam; total time=   4.6s
[CV] END activation=tanh, alpha=0.01, hidden_layer_sizes=(100, 100), solver=adam; total time=   5.3s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50,), solver=sgd; total time=   5.8s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50,), solver=sgd; total time=   5.5s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50,), solver=sgd; total time=   5.8s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50,), solver=adam; total time=   4.2s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50,), solver=adam; total time=   3.9s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50,), solver=adam; total time=   4.0s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100,), solver=sgd; total time=   6.9s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100,), solver=sgd; total time=   6.8s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100,), solver=sgd; total time=   7.0s
```

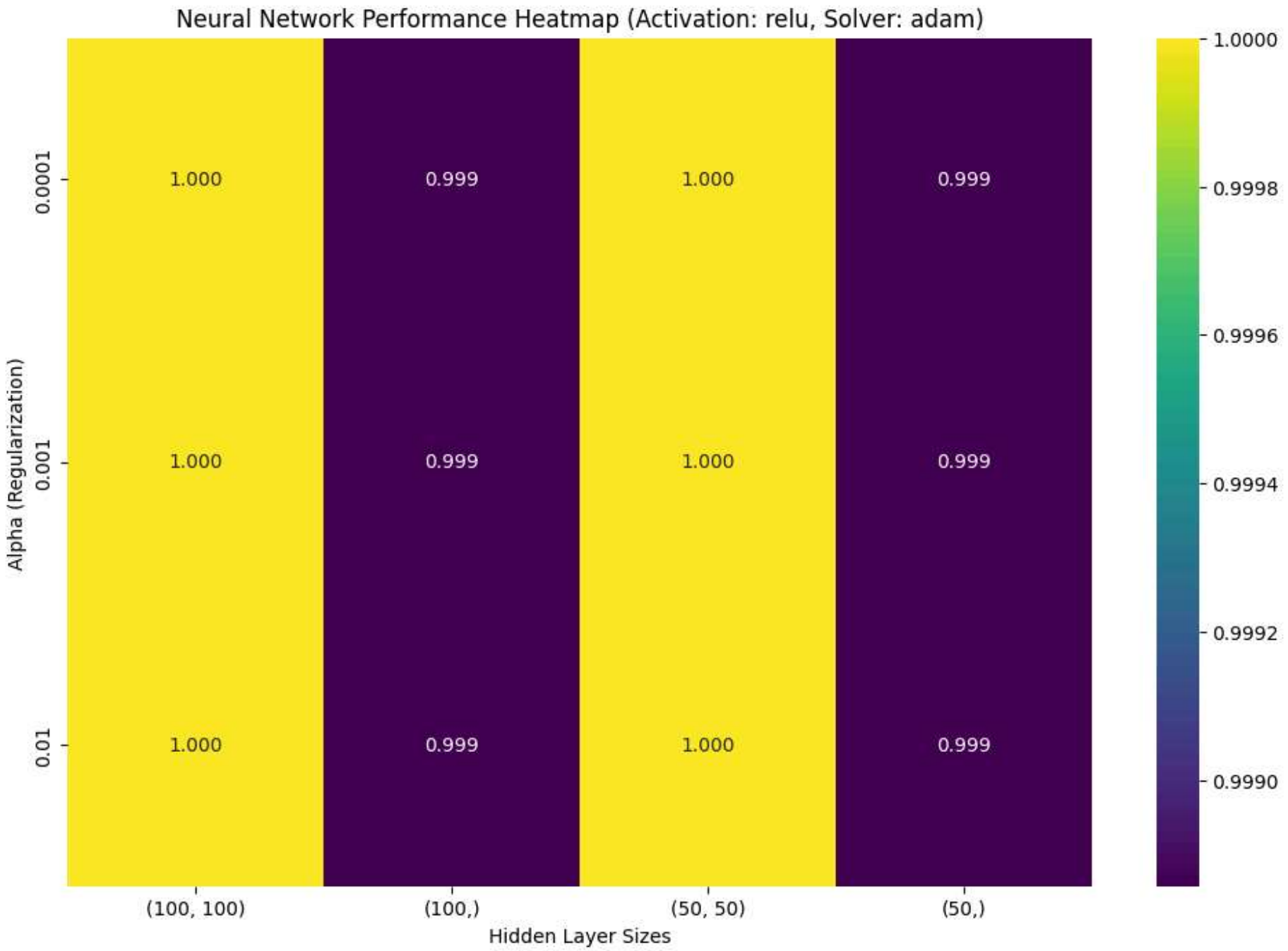
```
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100,), solver=adam; total time=   4.6s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100,), solver=adam; total time=   4.2s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100,), solver=adam; total time=   4.0s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=sgd; total time=  12.4s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=sgd; total time=  10.5s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=sgd; total time=  11.3s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=adam; total time=   3.4s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=adam; total time=   4.0s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(50, 50), solver=adam; total time=   3.2s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=sgd; total time=  12.8s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=sgd; total time=  11.8s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=sgd; total time=  12.9s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=adam; total time=   2.7s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=adam; total time=   2.7s
[CV] END activation=relu, alpha=0.0001, hidden_layer_sizes=(100, 100), solver=adam; total time=   2.7s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50,), solver=sgd; total time=   5.7s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50,), solver=sgd; total time=   5.5s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50,), solver=sgd; total time=   5.9s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50,), solver=adam; total time=   4.1s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50,), solver=adam; total time=   3.9s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50,), solver=adam; total time=   4.0s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100,), solver=sgd; total time=   6.1s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100,), solver=sgd; total time=   6.8s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100,), solver=sgd; total time=   6.8s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100,), solver=adam; total time=   4.3s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100,), solver=adam; total time=   4.1s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100,), solver=adam; total time=   4.0s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50, 50), solver=sgd; total time=  12.4s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50, 50), solver=sgd; total time=  10.5s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50, 50), solver=sgd; total time=  11.4s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50, 50), solver=adam; total time=   3.4s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50, 50), solver=adam; total time=   4.0s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(50, 50), solver=adam; total time=   3.3s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100, 100), solver=sgd; total time=  13.1s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100, 100), solver=sgd; total time=  12.4s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100, 100), solver=sgd; total time=  13.6s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100, 100), solver=adam; total time=   2.7s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100, 100), solver=adam; total time=   2.7s
[CV] END activation=relu, alpha=0.001, hidden_layer_sizes=(100, 100), solver=adam; total time=   2.7s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50,), solver=sgd; total time=   5.7s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50,), solver=sgd; total time=   5.6s
```

```
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50,), solver=sgd; total time=   5.9s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50,), solver=adam; total time=   3.8s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50,), solver=adam; total time=   3.9s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50,), solver=adam; total time=   4.0s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100,), solver=sgd; total time=   6.9s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100,), solver=sgd; total time=   6.9s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100,), solver=sgd; total time=   7.0s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100,), solver=adam; total time=   4.4s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100,), solver=adam; total time=   4.1s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100,), solver=adam; total time=   4.0s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50, 50), solver=sgd; total time=  12.7s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50, 50), solver=sgd; total time=  10.6s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50, 50), solver=sgd; total time=  11.3s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50, 50), solver=adam; total time=   3.2s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50, 50), solver=adam; total time=   4.0s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(50, 50), solver=adam; total time=   3.0s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100, 100), solver=sgd; total time=  13.1s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100, 100), solver=sgd; total time=  11.9s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100, 100), solver=sgd; total time=  13.2s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100, 100), solver=adam; total time=   2.8s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100, 100), solver=adam; total time=   3.3s
[CV] END activation=relu, alpha=0.01, hidden_layer_sizes=(100, 100), solver=adam; total time=   2.8s
Best Neural Network Parameters: {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50), 'solver': 'adam'}
Best Neural Network Score: 1.0
Test Set Accuracy: 1.0
```

In [18]:

```python
# Extracting results from the GridSearchCV object
results = pd.DataFrame(grid_search_nn.cv_results_)
params = results[['param_hidden_layer_sizes', 'param_alpha', 'mean_test_score']]
params['param_alpha'] = params['param_alpha'].astype(float)
params['mean_test_score'] = params['mean_test_score'].astype(float)

# Converting tuple representation for easier plotting
params['param_hidden_layer_sizes'] = params['param_hidden_layer_sizes'].apply(lambda x: str(x))

# Creating a pivot table for a chosen activation and solver, example: 'relu' and 'adam'
chosen_activation = 'relu'
chosen_solver = 'adam'
filtered_params = params[(results['param_activation'] == chosen_activation) & (results['param_solver'] == chosen_solver)]
pivot = filtered_params.pivot_table(index='param_alpha', columns='param_hidden_layer_sizes', values='mean_test_score')
```

```
# Plotting the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(pivot, annot=True, fmt=".3f", cmap='viridis', cbar=True)
plt.title(f'Neural Network Performance Heatmap (Activation: {chosen_activation}, Solver: {chosen_solver})')
plt.xlabel('Hidden Layer Sizes')
plt.ylabel('Alpha (Regularization)')
plt.show()
```


Neural Network Performance Heatmap (Activation: relu, Solver: adam)
Alpha (Regularization)
0.0001
0.001
0.01
1.000
0.999
1.000
0.999
1.000
0.999
1.000
0.999
(100, 100)
(100,)
(50, 50)
(50,)
Hidden Layer Sizes
1.0000
0.9998
0.9996
0.9994
0.9992
0.9990

In [19]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Decision Tree hyperparameters
param_grid_dt = {
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10]
}

# Creating the GridSearchCV object
grid_search_dt = GridSearchCV(DecisionTreeClassifier(), param_grid_dt, refit=True, verbose=2, cv=3)
grid_search_dt.fit(X_train, y_train)

# Displaying the best parameters and the score achieved with them
print("Best Decision Tree Parameters:", grid_search_dt.best_params_)
print("Best Decision Tree Score:", grid_search_dt.best_score_)
```

```
Fitting 3 folds for each of 45 candidates, totalling 135 fits
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=None, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
```

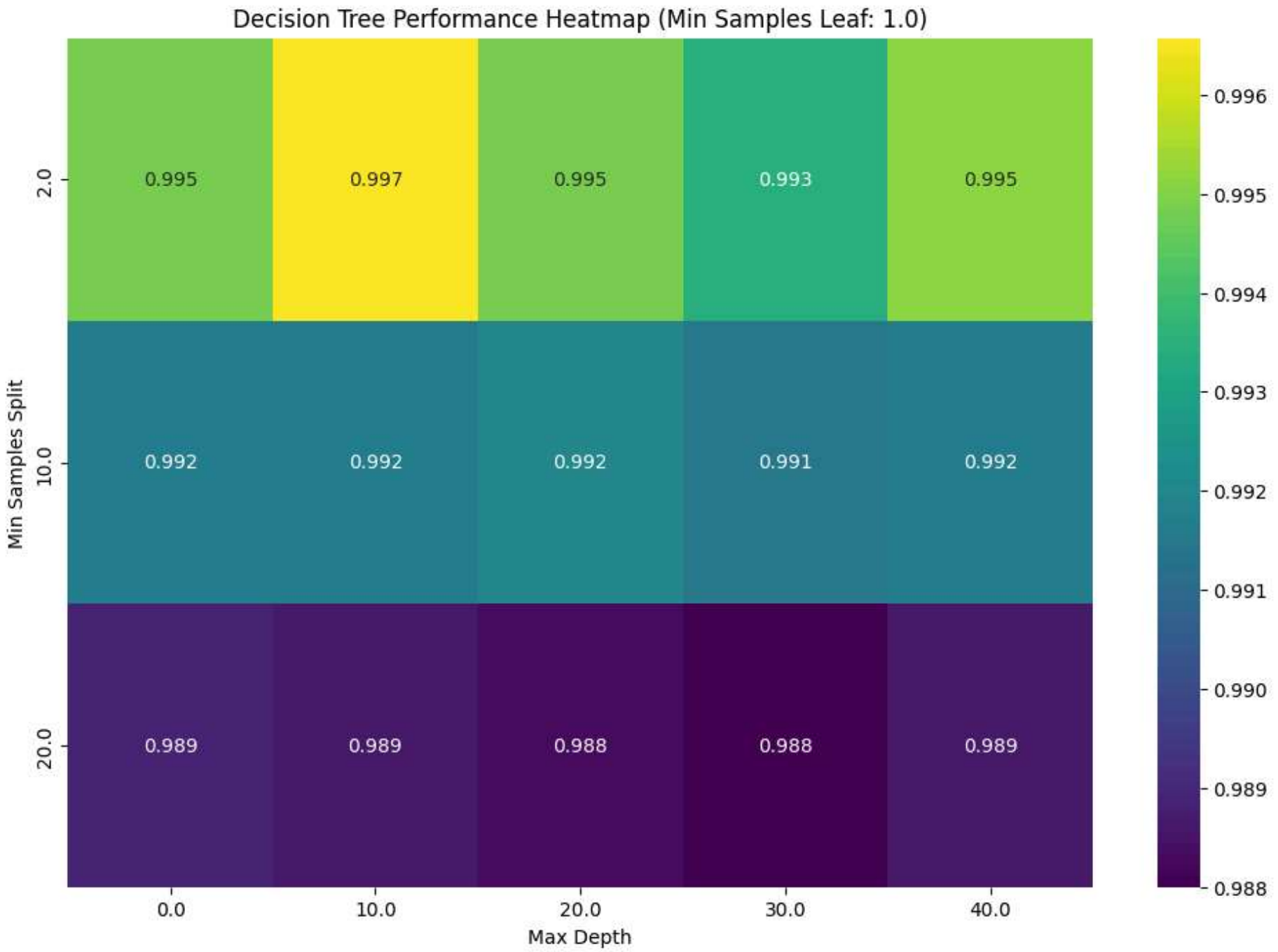
```
[CV] END max_depth=10, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=10; total time=   0.1s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=20; total time=   0.1s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
```

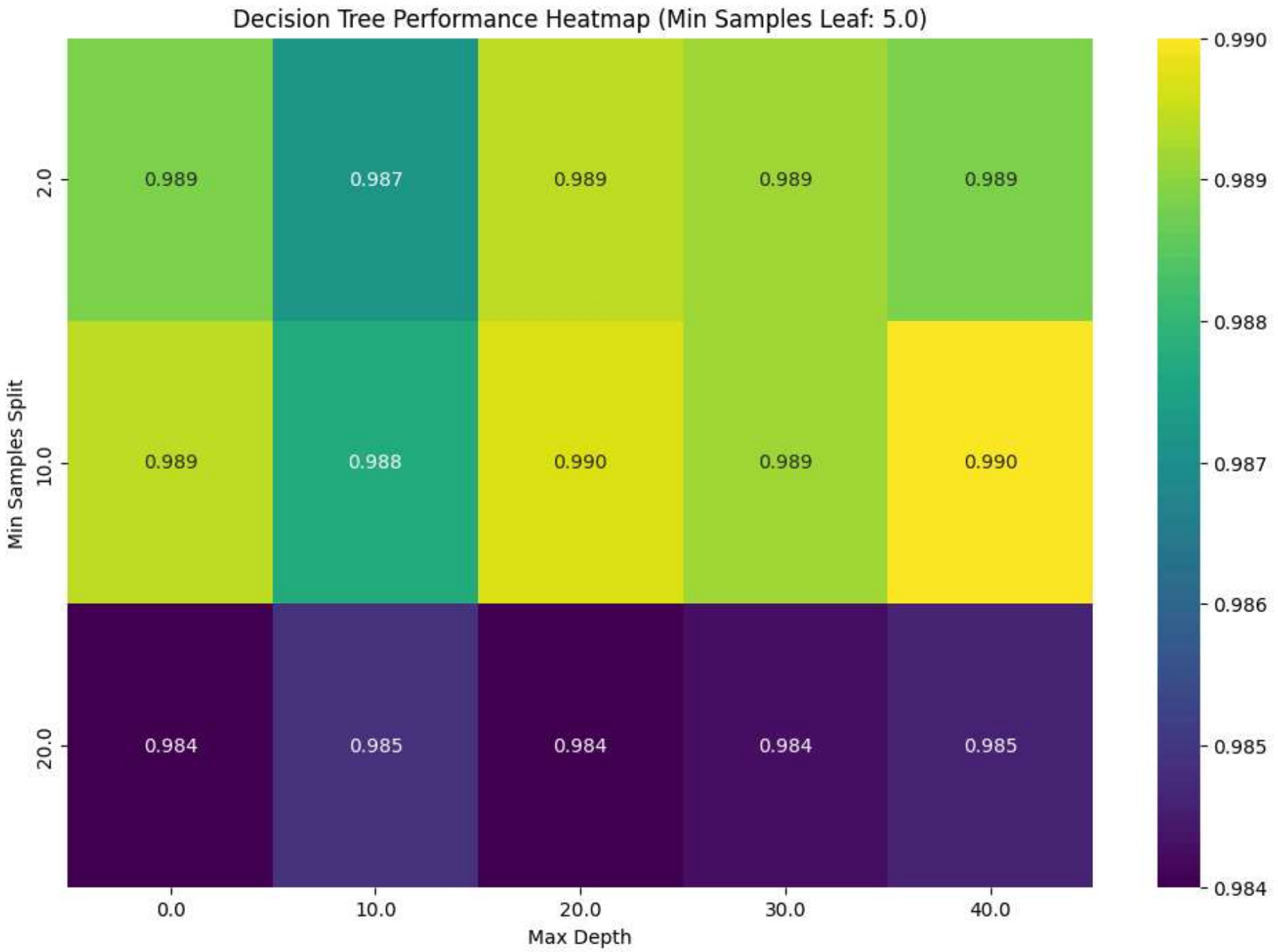
```
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=30, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=1, min_samples_split=20; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=5, min_samples_split=2; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
```

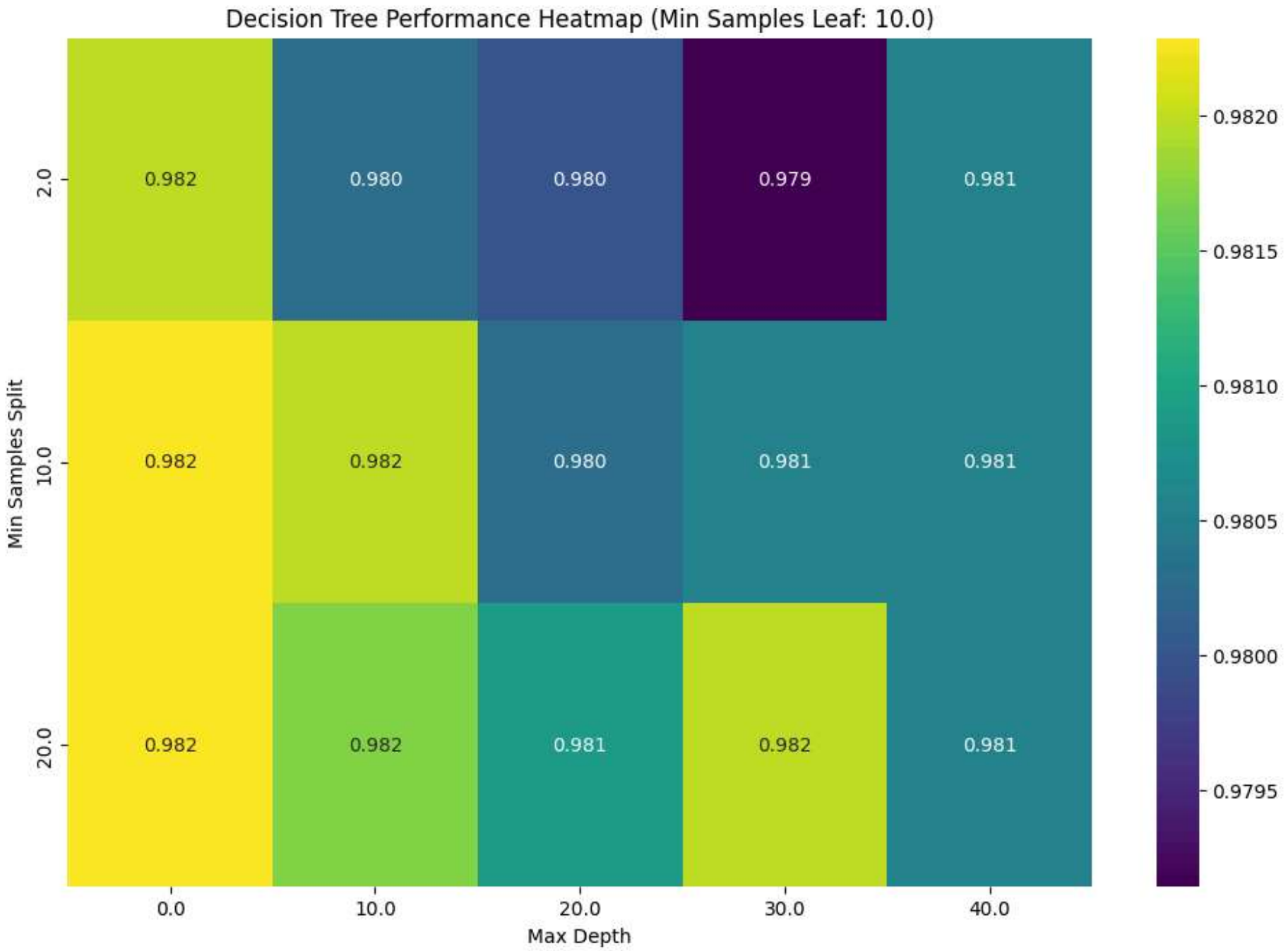
```
[CV] END max_depth=40, min_samples_leaf=5, min_samples_split=10; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=5, min_samples_split=20; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=10, min_samples_split=2; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=10, min_samples_split=10; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
[CV] END max_depth=40, min_samples_leaf=10, min_samples_split=20; total time=   0.0s
Best Decision Tree Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best Decision Tree Score: 0.9965716729794917
```

In [20]:

```python
# Extracting results from the GridSearchCV object
results = pd.DataFrame(grid_search_dt.cv_results_)
params = results[['param_max_depth', 'param_min_samples_split', 'param_min_samples_leaf', 'mean_test_score']]
params['param_max_depth'] = params['param_max_depth'].astype(float).fillna(0)  # Convert 'None' to a numeric value if needed
params['param_min_samples_split'] = params['param_min_samples_split'].astype(float)
params['param_min_samples_leaf'] = params['param_min_samples_leaf'].astype(float)
params['mean_test_score'] = params['mean_test_score'].astype(float)

# Plotting the heatmaps for each 'min_samples_leaf'
min_samples_leaves = params['param_min_samples_leaf'].unique()
for leaf in min_samples_leaves:
    pivot = params[params['param_min_samples_leaf'] == leaf].pivot_table(
        index='param_min_samples_split',
        columns='param_max_depth',
        values='mean_test_score'
    )
    plt.figure(figsize=(12, 8))
    sns.heatmap(pivot, annot=True, fmt=".3f", cmap='viridis', cbar=True)
    plt.title(f'Decision Tree Performance Heatmap (Min Samples Leaf: {leaf})')
    plt.xlabel('Max Depth')
    plt.ylabel('Min Samples Split')
    plt.show()
```


Decision Tree Performance Heatmap (Min Samples Leaf: 1.0)
Min Samples Split
2.0
10.0
20.0
0.995
0.997
0.995
0.993
0.995
0.992
0.992
0.992
0.991
0.992
0.989
0.989
0.988
0.988
0.989
0.0
10.0
20.0
30.0
40.0
Max Depth
0.996
0.995
0.994
0.993
0.992
0.991
0.990
0.989
0.988


Decision Tree Performance Heatmap (Min Samples Leaf: 5.0)
Min Samples Split
2.0
10.0
20.0
0.989
0.987
0.989
0.989
0.989
0.989
0.988
0.990
0.989
0.990
0.984
0.985
0.984
0.984
0.985
0.0
10.0
20.0
30.0
40.0
Max Depth
0.990
0.989
0.988
0.987
0.986
0.985
0.984

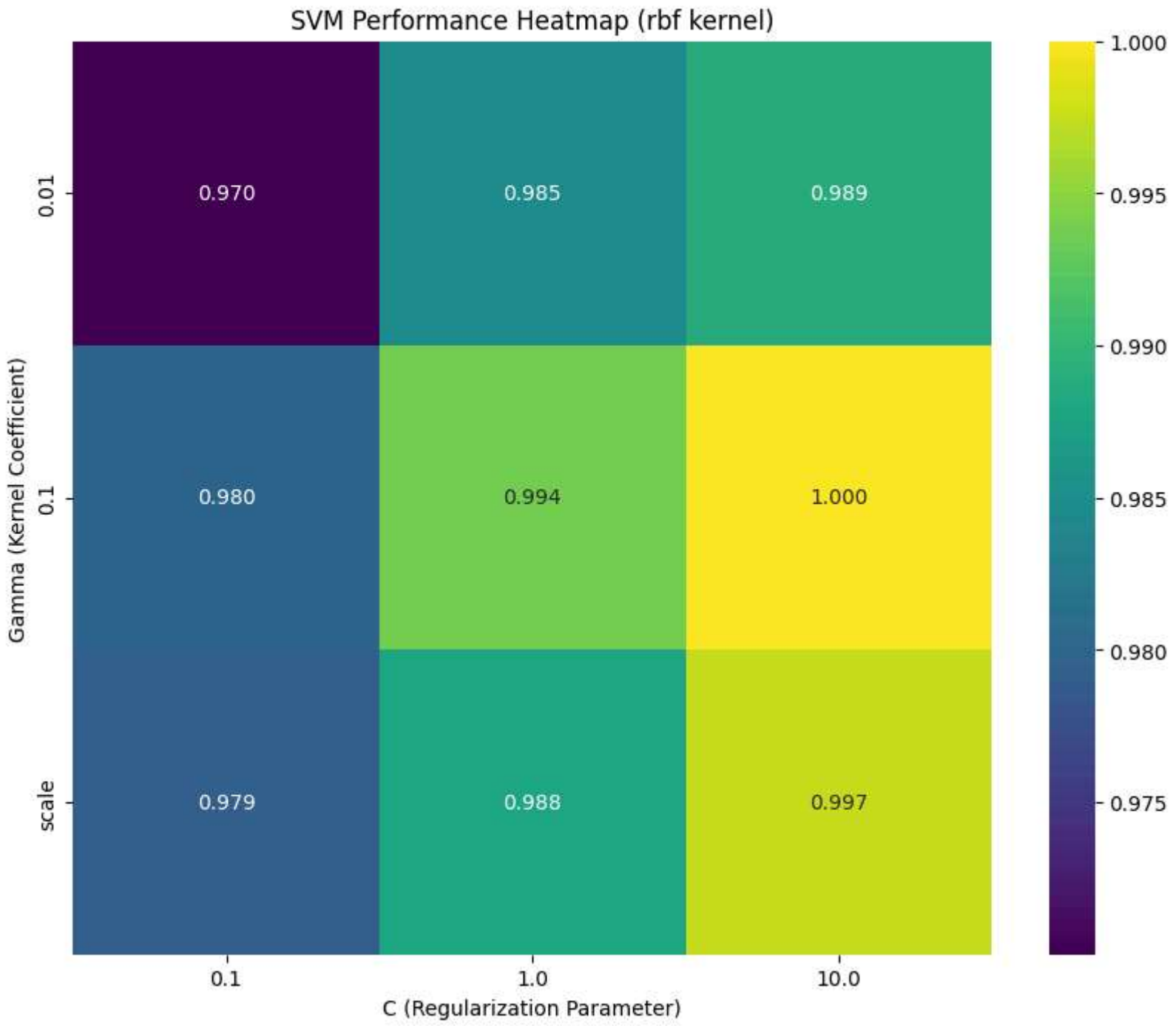

Decision Tree Performance Heatmap (Min Samples Leaf: 10.0)
Min Samples Split
2.0
10.0
20.0
0.982
0.980
0.980
0.979
0.981
0.982
0.982
0.980
0.981
0.981
0.982
0.982
0.981
0.982
0.981
0.0
10.0
20.0
30.0
40.0
Max Depth
0.9820
0.9815
0.9810
0.9805
0.9800
0.9795

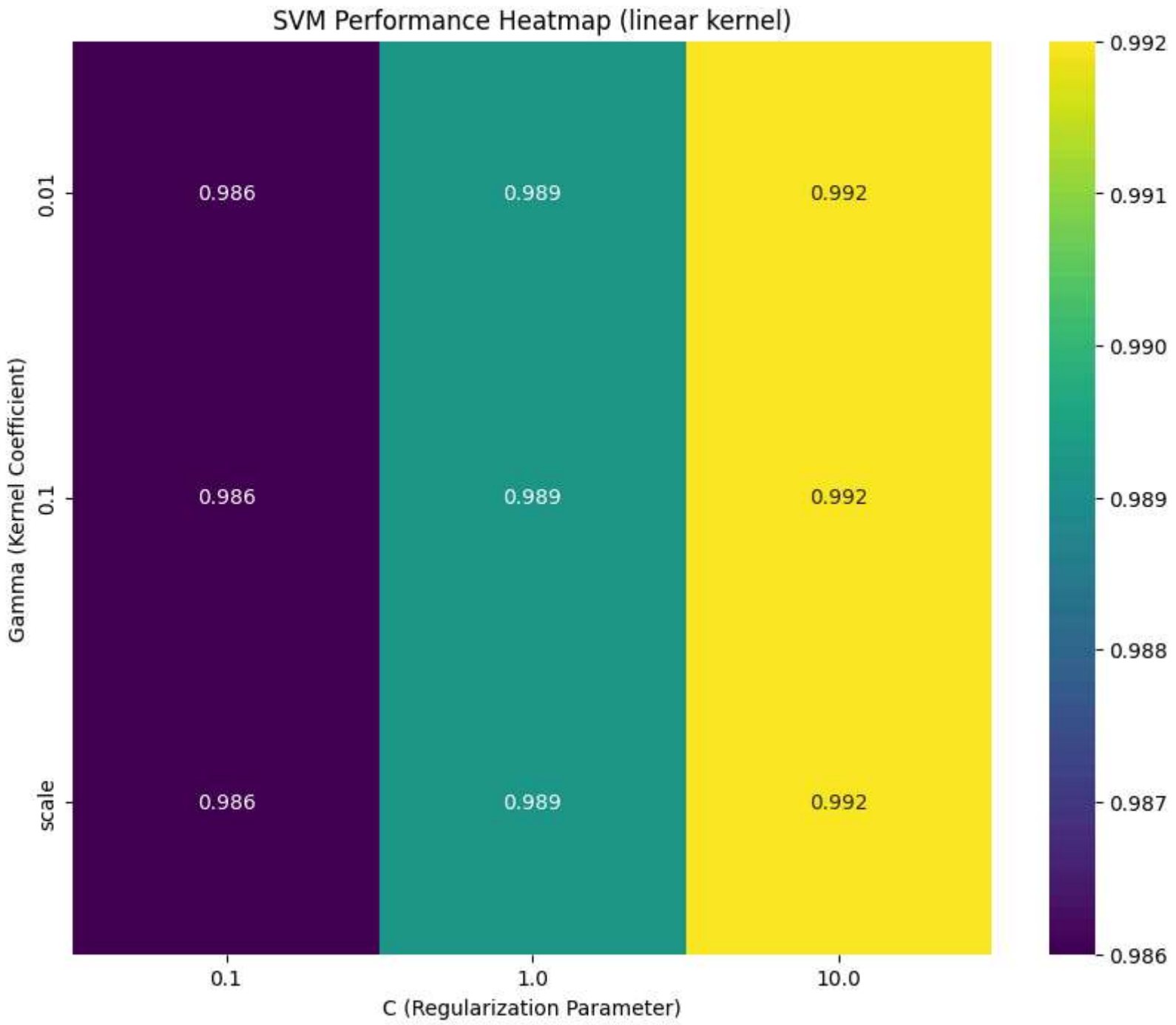
In [21]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# SVM hyperparameters
param_grid_svm = {
    'C': [0.1, 1, 10],
    'gamma': ['scale', 0.1, 0.01],
    'kernel': ['rbf', 'linear']
}

# Creating the GridSearchCV object
grid_search_svm = GridSearchCV(SVC(), param_grid_svm, refit=True, verbose=2, cv=3)
grid_search_svm.fit(X_train, y_train)

# Displaying the best parameters and the score achieved with them
print("Best SVM Parameters:", grid_search_svm.best_params_)
print("Best SVM Score:", grid_search_svm.best_score_)
```

```
Fitting 3 folds for each of 18 candidates, totalling 54 fits
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time=   0.2s
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time=   0.3s
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time=   0.2s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=   0.0s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=   0.0s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=   0.0s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   0.5s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   0.4s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   0.4s
[CV] END ..................C=0.1, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END ..................C=0.1, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END ..................C=0.1, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END ......................C=0.1, gamma=0.01, kernel=rbf; total time=   0.3s
[CV] END ......................C=0.1, gamma=0.01, kernel=rbf; total time=   0.3s
[CV] END ......................C=0.1, gamma=0.01, kernel=rbf; total time=   0.2s
[CV] END ..................C=0.1, gamma=0.01, kernel=linear; total time=   0.0s
[CV] END ..................C=0.1, gamma=0.01, kernel=linear; total time=   0.0s
[CV] END ..................C=0.1, gamma=0.01, kernel=linear; total time=   0.0s
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time=   0.1s
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time=   0.0s
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time=   0.1s
[CV] END ..................C=1, gamma=scale, kernel=linear; total time=   0.0s
[CV] END ..................C=1, gamma=scale, kernel=linear; total time=   0.0s
[CV] END ..................C=1, gamma=scale, kernel=linear; total time=   0.0s
[CV] END .........................C=1, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END .........................C=1, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END .........................C=1, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END ......................C=1, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END ......................C=1, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END ......................C=1, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END .........................C=1, gamma=0.01, kernel=rbf; total time=   0.1s
[CV] END .........................C=1, gamma=0.01, kernel=rbf; total time=   0.1s
[CV] END .........................C=1, gamma=0.01, kernel=rbf; total time=   0.1s
[CV] END ..................C=1, gamma=0.01, kernel=linear; total time=   0.0s
[CV] END ..................C=1, gamma=0.01, kernel=linear; total time=   0.0s
[CV] END ..................C=1, gamma=0.01, kernel=linear; total time=   0.0s
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=   0.0s
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=   0.0s
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=   0.0s
[CV] END ..................C=10, gamma=scale, kernel=linear; total time=   0.0s
```

```
[CV] END ..................C=10, gamma=scale, kernel=linear; total time=   0.0s
[CV] END ..................C=10, gamma=scale, kernel=linear; total time=   0.1s
[CV] END .......................C=10, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END .......................C=10, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END .......................C=10, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END .....................C=10, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END .....................C=10, gamma=0.1, kernel=linear; total time=   0.0s
[CV] END .....................C=10, gamma=0.1, kernel=linear; total time=   0.1s
[CV] END ......................C=10, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ......................C=10, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ......................C=10, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END ....................C=10, gamma=0.01, kernel=linear; total time=   0.0s
[CV] END ....................C=10, gamma=0.01, kernel=linear; total time=   0.0s
[CV] END ....................C=10, gamma=0.01, kernel=linear; total time=   0.1s
Best SVM Parameters: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Best SVM Score: 1.0
```

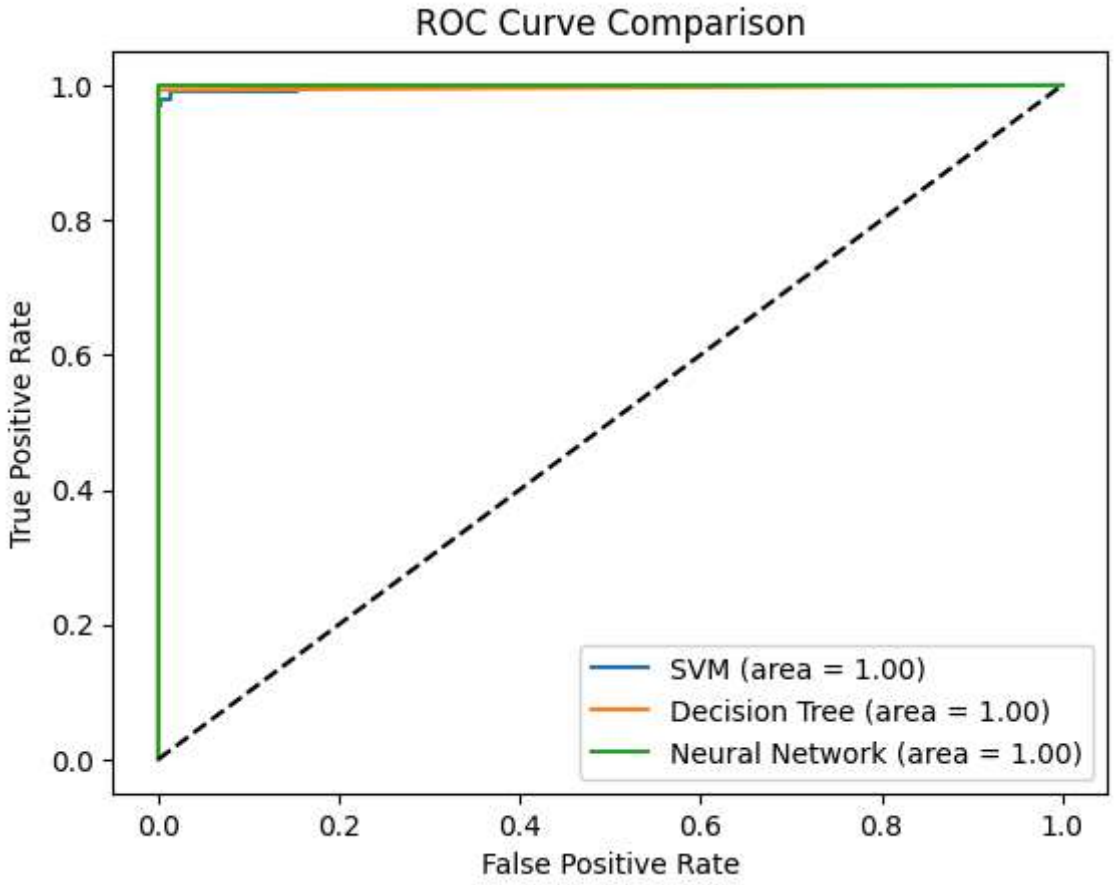
In [22]:
```python
# Extracting results from the GridSearchCV object
results = pd.DataFrame(grid_search_svm.cv_results_)
params = results[['param_C', 'param_gamma', 'param_kernel', 'mean_test_score']]
params['param_C'] = params['param_C'].astype(float)
params['param_gamma'] = params['param_gamma'].astype(str)
params['mean_test_score'] = params['mean_test_score'].astype(float)

# Create pivot tables for each kernel type and plot them
for kernel in params['param_kernel'].unique():
    pivot = params[params['param_kernel'] == kernel].pivot_table(
        index='param_gamma',
        columns='param_C',
        values='mean_test_score'
    )
    plt.figure(figsize=(10, 8))
    sns.heatmap(pivot, annot=True, fmt=".3f", cmap='viridis')
    plt.title(f'SVM Performance Heatmap ({kernel} kernel)')
    plt.xlabel('C (Regularization Parameter)')
    plt.ylabel('Gamma (Kernel Coefficient)')
    plt.show()
```


SVM Performance Heatmap (rbf kernel)
Gamma (Kernel Coefficient)
0.01
0.1
scale
0.970
0.985
0.989
0.980
0.994
1.000
0.979
0.988
0.997
0.1
1.0
10.0
C (Regularization Parameter)
1.000
0.995
0.990
0.985
0.980
0.975


SVM Performance Heatmap (linear kernel)
Gamma (Kernel Coefficient)
0.01
0.1
scale
0.986
0.989
0.992
0.986
0.989
0.992
0.986
0.989
0.992
0.1
1.0
10.0
C (Regularization Parameter)
0.992
0.991
0.990
0.989
0.988
0.987
0.986

In [23]:
```
from sklearn.metrics import roc_curve, auc, accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder

# Encode labels
encoder = LabelEncoder()
y_train = encoder.fit_transform(y_train)
y_test = encoder.transform(y_test)

models = {
    'SVM': SVC(probability=True),
    'Decision Tree': DecisionTreeClassifier(),
    'Neural Network': MLPClassifier(max_iter=1000)
}

results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    probability = model.predict_proba(X_test)[:, 1]  # Assuming positive class is 1
    fpr, tpr, _ = roc_curve(y_test, probability, pos_label=1)
    auc_score = auc(fpr, tpr)

    results[name] = {
        'Accuracy': accuracy_score(y_test, predictions),
        'ROC AUC': auc_score,
        'Classification Report': classification_report(y_test, predictions, output_dict=True)
    }

    # Plotting ROC Curve
    plt.plot(fpr, tpr, label=f'{name} (area = {auc_score:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc='best')
plt.show()

# Print the results
```

```
for model_name, metrics in results.items():
    print(f'\n{model_name} Model Metrics:')
    print('Accuracy:', metrics['Accuracy'])
    print('ROC AUC:', metrics['ROC AUC'])
    print('Classification Report:')
    print(pd.DataFrame(metrics['Classification Report']).transpose())
```



```
SVM Model Metrics:
Accuracy: 0.9886666666666667
ROC AUC: 0.9983545450212116
Classification Report:
              precision    recall  f1-score      support
0              0.982328  1.000000  0.991085   945.000000
1              1.000000  0.969369  0.984446   555.000000
accuracy       0.988667  0.988667  0.988667     0.988667
macro avg      0.991164  0.984685  0.987766  1500.000000
weighted avg   0.988867  0.988667  0.988629  1500.000000

Decision Tree Model Metrics:
Accuracy: 0.9973333333333333
ROC AUC: 0.9963963963963964
Classification Report:
              precision    recall  f1-score      support
0              0.995785  1.000000  0.997888   945.000000
1              1.000000  0.992793  0.996383   555.000000
accuracy       0.997333  0.997333  0.997333     0.997333
macro avg      0.997893  0.996396  0.997136  1500.000000
weighted avg   0.997345  0.997333  0.997331  1500.000000

Neural Network Model Metrics:
Accuracy: 1.0
ROC AUC: 1.0
Classification Report:
              precision  recall  f1-score  support
0                   1.0     1.0       1.0    945.0
1                   1.0     1.0       1.0    555.0
accuracy            1.0     1.0       1.0      1.0
macro avg           1.0     1.0       1.0   1500.0
weighted avg        1.0     1.0       1.0   1500.0
```

In [24]:

```python
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
import time

# Initialize classifiers
svm_classifier = SVC()
decision_tree_classifier = DecisionTreeClassifier()
```

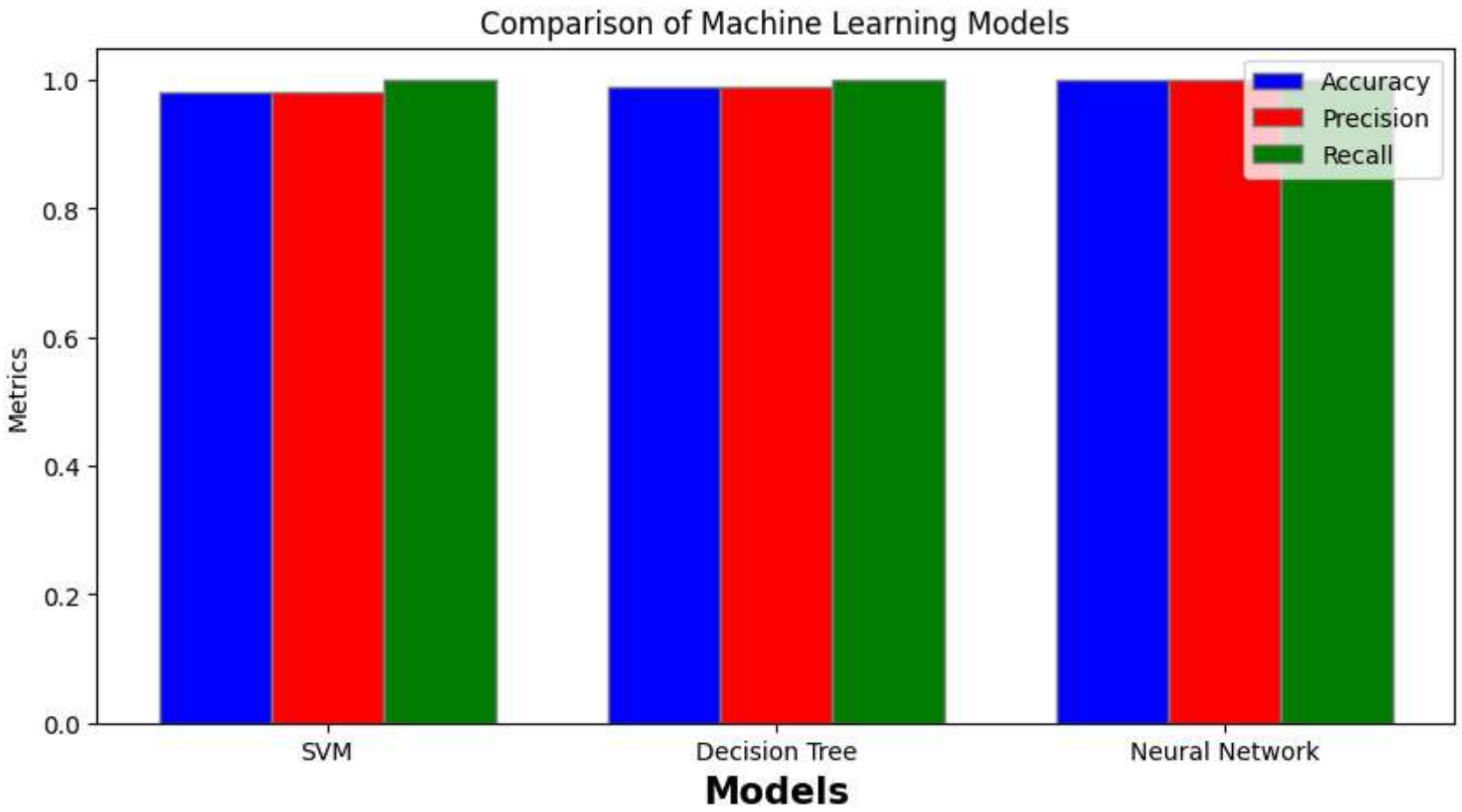
```
mlp_classifier = MLPClassifier()

classifiers = {
    "SVM": svm_classifier,
    "Decision Tree": decision_tree_classifier,
    "MLP": mlp_classifier
}

# Training time comparison
training_times = {}
for name, clf in classifiers.items():
    start_time = time.time()
    clf.fit(X_train, y_train)
    end_time = time.time()
    training_times[name] = end_time - start_time

# Prediction time comparison
prediction_times = {}
for name, clf in classifiers.items():
    start_time = time.time()
    clf.predict(X_test)
    end_time = time.time()
    prediction_times[name] = end_time - start_time

# Print out training and prediction times
print("Training Times:")
for name, time_taken in training_times.items():
    print(f"{name}: {time_taken:.4f} seconds")

print("\nPrediction Times:")
for name, time_taken in prediction_times.items():
    print(f"{name}: {time_taken:.4f} seconds")
```

```
Training Times:
SVM: 0.1137 seconds
Decision Tree: 0.1665 seconds
MLP: 5.4011 seconds

Prediction Times:
SVM: 0.1005 seconds
Decision Tree: 0.0000 seconds
MLP: 0.0121 seconds
```

In [25]:

```python
import matplotlib.pyplot as plt
import numpy as np

# Names of the models
models = ['SVM', 'Decision Tree', 'Neural Network']

# Metrics to compare
accuracy = [0.98, 0.99, 1.0]
precision = [0.98, 0.99, 1.0]
recall = [1.00, 1.0, 1.0]
f1_scores = [0.99, 0.99, 1.0]

# Creating index for each tick in bar plot
bar_width = 0.25
r1 = np.arange(len(accuracy))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]

# Creating the bar plot
plt.figure(figsize=(10, 5))
plt.bar(r1, accuracy, color='b', width=bar_width, edgecolor='grey', label='Accuracy')
plt.bar(r2, precision, color='r', width=bar_width, edgecolor='grey', label='Precision')
plt.bar(r3, recall, color='g', width=bar_width, edgecolor='grey', label='Recall')

# Adding labels
plt.xlabel('Models', fontweight='bold', fontsize=15)
plt.xticks([r + bar_width for r in range(len(accuracy))], models)
plt.ylabel('Metrics')
plt.title('Comparison of Machine Learning Models')

# Create legend & Show graphic
```

```
plt.legend()
plt.show()
```



In [ ]: