data_601_bc_final_project

May 28, 2024

[1]:
```
# for google colab to install the right modules
!pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Requirement already satisfied: pandas>=1.0.0 in
c:\users\hainz\anaconda3\lib\site-packages (from ucimlrepo) (2.2.1)
Requirement already satisfied: certifi>=2020.12.5 in
c:\users\hainz\anaconda3\lib\site-packages (from ucimlrepo) (2024.2.2)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\hainz\anaconda3\lib\site-packages (from pandas>=1.0.0->ucimlrepo)
(2024.1)
Requirement already satisfied: pytz>=2020.1 in
c:\users\hainz\anaconda3\lib\site-packages (from pandas>=1.0.0->ucimlrepo)
(2021.3)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\hainz\anaconda3\lib\site-packages (from pandas>=1.0.0->ucimlrepo)
(2.8.2)
Requirement already satisfied: numpy<2,>=1.22.4 in
c:\users\hainz\anaconda3\lib\site-packages (from pandas>=1.0.0->ucimlrepo)
(1.26.4)
Requirement already satisfied: six>=1.5 in c:\users\hainz\anaconda3\lib\site-
packages (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.16.0)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
```

1 Import Necessary Modules

[2]:
```
# fetch the dataset
from ucimlrepo import fetch_ucirepo

# import data analysis modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# import classifiers
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier,␣
 ↪VotingClassifier

# metrics
from sklearn.metrics import f1_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# stat model implementation module
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

2 Objective

- View the features of importance to better gauge the impact of each variable.
- Create visualizations to interpret which features may contribute to a benign or malignant tumor.
- Determine which predictor variables are significant in classifying a cancer as malignant or benign by using a machine learning model to predict the target variable.
- Analyze the performances of several different binary classification algorithms to determine which one is most reliable for our study.
- Compute and plot F1 scores of the algorithms and communicate findings of the best machine learning model.

[3]:
```
# fetch dataset from web
breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)
```

3 Data Cleaning and Preprocessing

[4]:
```
# data (as pandas dataframes)
bc_features_df = breast_cancer_wisconsin_diagnostic.data.features
bc_features_df.head(3) # view first three rows as sample
```

[4]:
```
   radius1  texture1  perimeter1   area1  smoothness1  compactness1  \
0    17.99     10.38       122.8  1001.0      0.11840       0.27760
1    20.57     17.77       132.9  1326.0      0.08474       0.07864
2    19.69     21.25       130.0  1203.0      0.10960       0.15990

   concavity1  concave_points1  symmetry1  fractal_dimension1  …  radius3  \
0      0.3001          0.14710     0.2419             0.07871  …    25.38
1      0.0869          0.07017     0.1812             0.05667  …    24.99
2      0.1974          0.12790     0.2069             0.05999  …    23.57
```

```
   texture3  perimeter3   area3  smoothness3  compactness3  concavity3  \
0     17.33       184.6  2019.0       0.1622        0.6656      0.7119
1     23.41       158.8  1956.0       0.1238        0.1866      0.2416
2     25.53       152.5  1709.0       0.1444        0.4245      0.4504

   concave_points3  symmetry3  fractal_dimension3
0           0.2654     0.4601             0.11890
1           0.1860     0.2750             0.08902
2           0.2430     0.3613             0.08758

[3 rows x 30 columns]
```

[5]:
```
# store target variable
bc_targets_df = breast_cancer_wisconsin_diagnostic.data.targets
bc_targets_df.head(3) # view first three rows as reference
```

[5]:
```
  Diagnosis
0         M
1         M
2         M
```

[6]:
```
# Checking for missing values
bc_features_df.isnull().sum()
```

[6]:
```
radius1               0
texture1              0
perimeter1            0
area1                 0
smoothness1           0
compactness1          0
concavity1            0
concave_points1       0
symmetry1             0
fractal_dimension1    0
radius2               0
texture2              0
perimeter2            0
area2                 0
smoothness2           0
compactness2          0
concavity2            0
concave_points2       0
symmetry2             0
fractal_dimension2    0
radius3               0
texture3              0
```

```
perimeter3            0
area3                 0
smoothness3           0
compactness3          0
concavity3            0
concave_points3       0
symmetry3             0
fractal_dimension3    0
dtype: int64
```

[7]:
```
#  limit the first 10 columns for simplicity
bc_features_df = bc_features_df.iloc[:, 0:10]
bc_features_df.head(3) # view the first three for reference
```

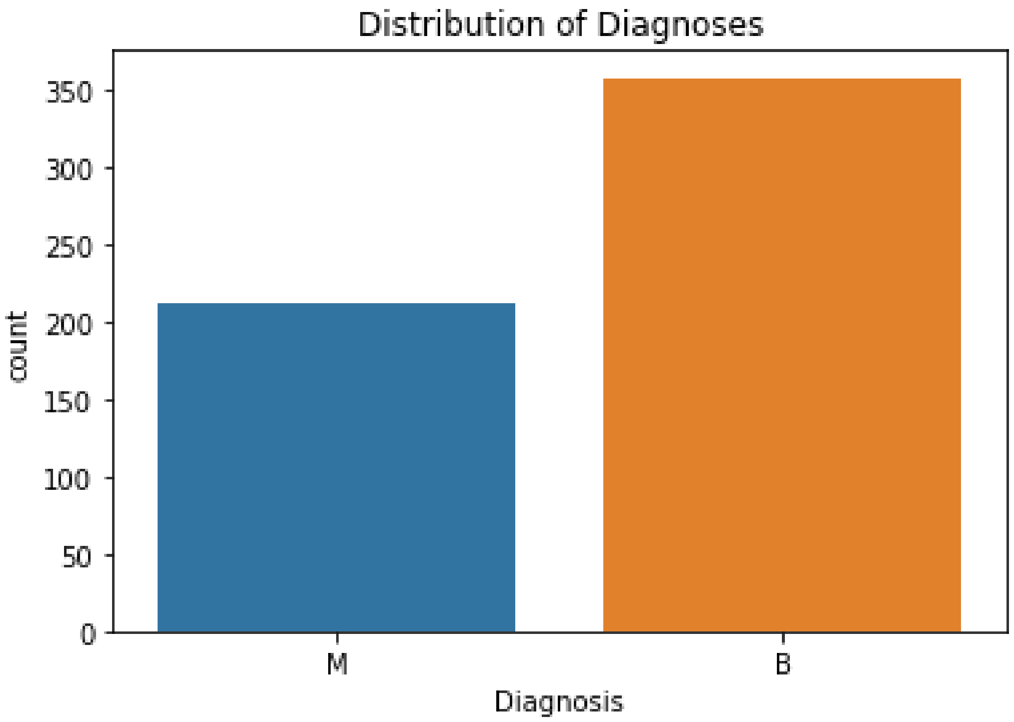
[7]:
```
   radius1  texture1  perimeter1   area1  smoothness1  compactness1  \
0    17.99     10.38       122.8  1001.0      0.11840       0.27760
1    20.57     17.77       132.9  1326.0      0.08474       0.07864
2    19.69     21.25       130.0  1203.0      0.10960       0.15990

   concavity1  concave_points1  symmetry1  fractal_dimension1
0      0.3001          0.14710     0.2419             0.07871
1      0.0869          0.07017     0.1812             0.05667
2      0.1974          0.12790     0.2069             0.05999
```

[8]:
```
# assess value_counts
target_value_counts = pd.DataFrame(bc_targets_df.value_counts()).reset_index().
 ↪rename(columns={0: 'Frequency'})
target_value_counts
```

[8]:
```
  Diagnosis  Frequency
0         B        357
1         M        212
```

4 Exploratory Data Analysis (EDA)

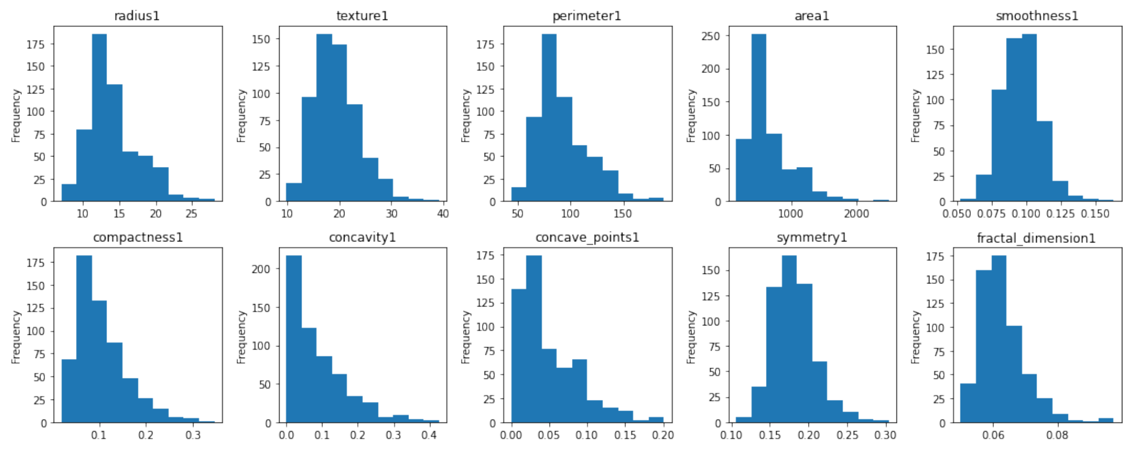
[9]:
```
# Plotting the Distribution of Diagnoses From Dataset
# Checking distribution of target variable
sns.countplot(x=bc_targets_df['Diagnosis'])
plt.title('Distribution of Diagnoses')
plt.show()
```



Since the labels in our dataset are not evenly distributed, this means we will have to work with the F1 score instead of accuracy.

4.0.1 Histogram of Each Feature Of Importance

[10]:
```
# Number of rows and columns for subplots
n_rows = 2
n_cols = 5

# Create a figure and a set of subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 6))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Iterate through the columns and plot histograms
for i, col in enumerate(bc_features_df.columns):
    bc_features_df[col].plot(kind="hist", ax=axes[i], title=col)

# Adjust layout to prevent overlap
plt.tight_layout()

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

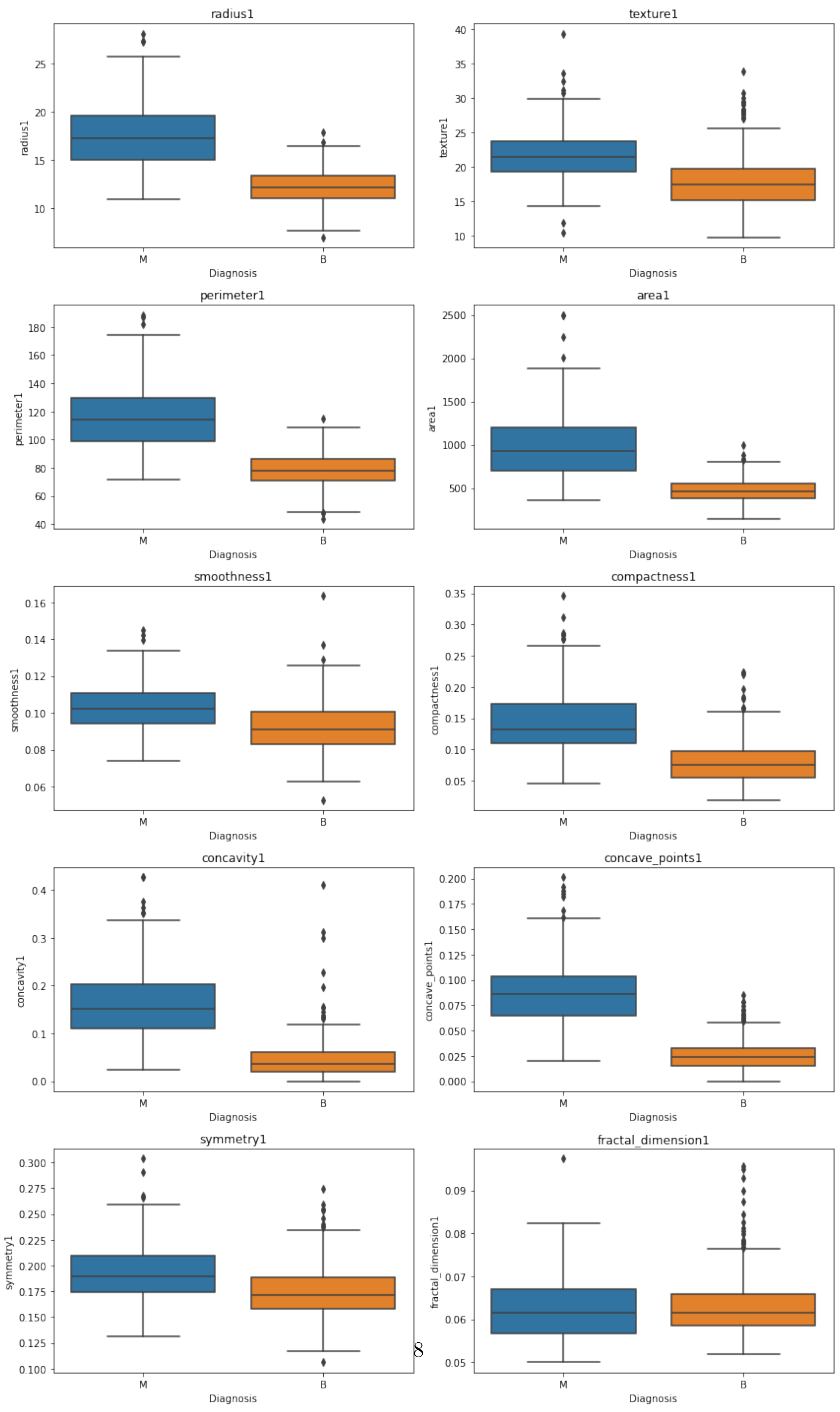


This series of distributions for the features followed a either general right-skewed or normal distribution, which is common in most medical datasets that showcase characteristics in a cancer study.

4.0.2 Boxplots of Feature Distribution For Respective Target Diagnosis

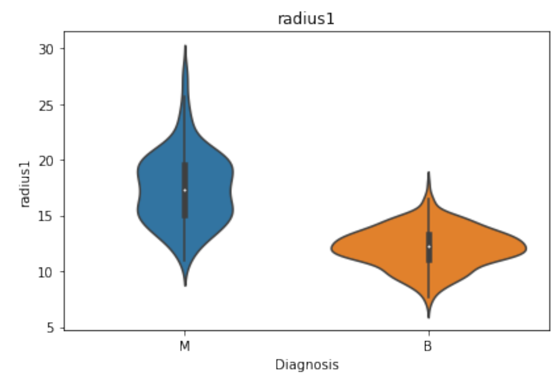
[11]:
```
# Assessing feature distributions
fig, axes = plt.subplots(5, 2, figsize=(12, 20))
axes = axes.flatten()
for i, col in enumerate(bc_features_df.columns):
    sns.boxplot(x=bc_targets_df['Diagnosis'], y=bc_features_df[col], ax=axes[i])
    axes[i].set_title(col)
```

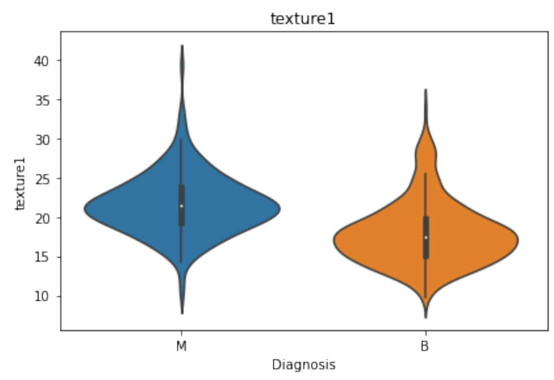
```
plt.tight_layout()
plt.show()
```

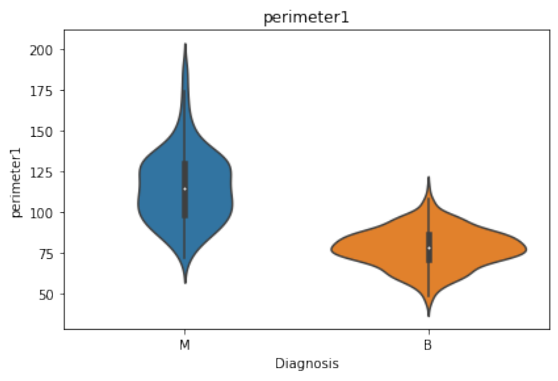

radius1
texture1
perimeter1
area1
smoothness1
compactness1
concavity1
concave_points1
symmetry1
fractal_dimension1
Diagnosis
M
B

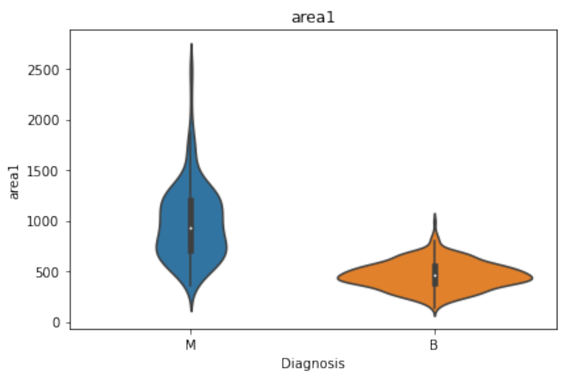
4.0.3 Violin Plots of Feature Distribution For Respective Target Diagnosis

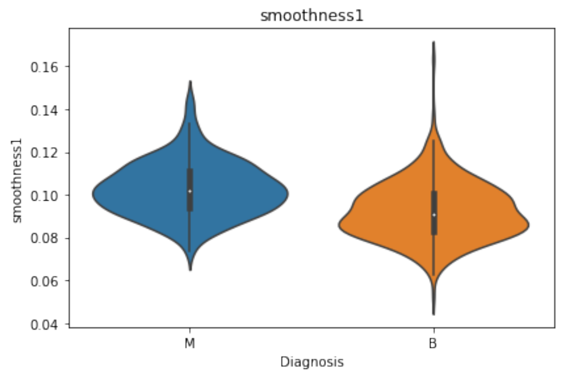
```
[12]: # Assuming bc_features_df is your DataFrame with features and bc_targets_df␣
       ↪contains the target variable 'Diagnosis'
      fig, axes = plt.subplots(5, 2, figsize=(12, 20))  # Adjust the grid size and␣
       ↪figure size as needed
      axes = axes.flatten()

      for i, col in enumerate(bc_features_df.columns):
          sns.violinplot(x=bc_targets_df['Diagnosis'], y=bc_features_df[col],␣
       ↪ax=axes[i])
          axes[i].set_title(col)

      plt.tight_layout()
      plt.show()
```

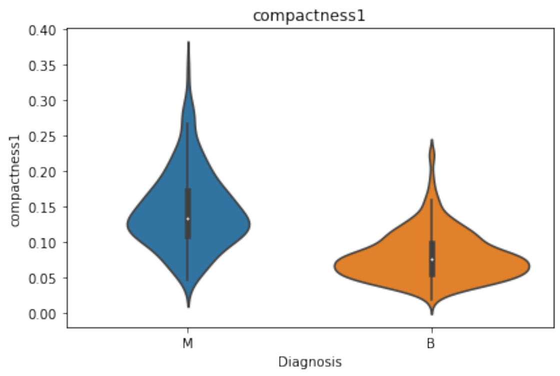

radius1
radius1
30
25
20
15
10
5
M
B
Diagnosis

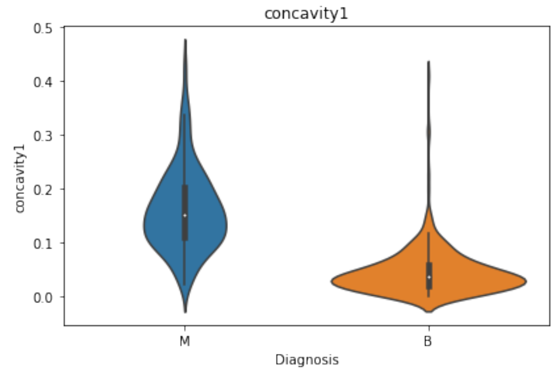

texture1
texture1
40
35
30
25
20
15
10
M
B
Diagnosis

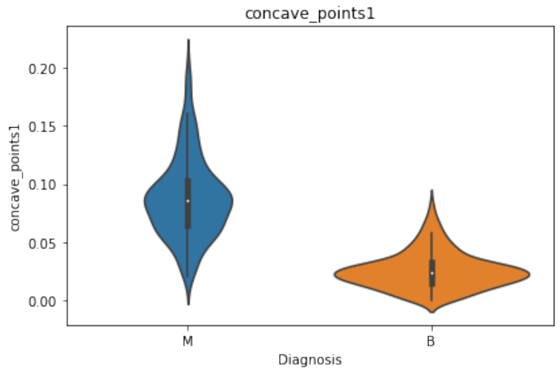

perimeter1
perimeter1
200
175
150
125
100
75
50
M
B
Diagnosis

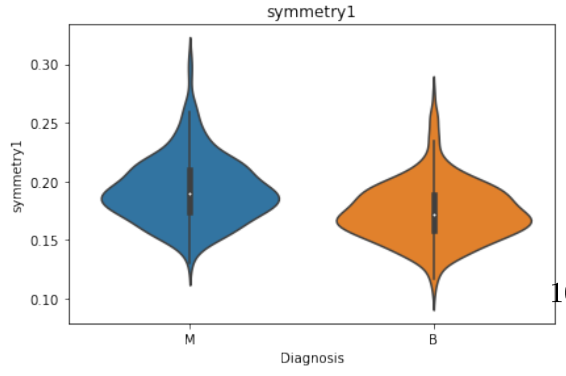

area1
area1
2500
2000
1500
1000
500
0
M
B
Diagnosis

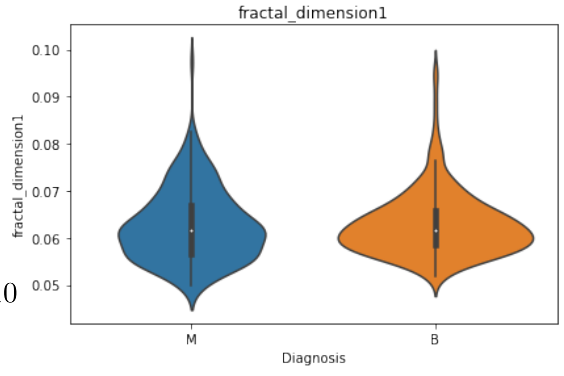

smoothness1
smoothness1
0.16
0.14
0.12
0.10
0.08
0.06
0.04
M
B
Diagnosis


compactness1
compactness1
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
M
B
Diagnosis


concavity1
concavity1
0.5
0.4
0.3
0.2
0.1
0.0
M
B
Diagnosis

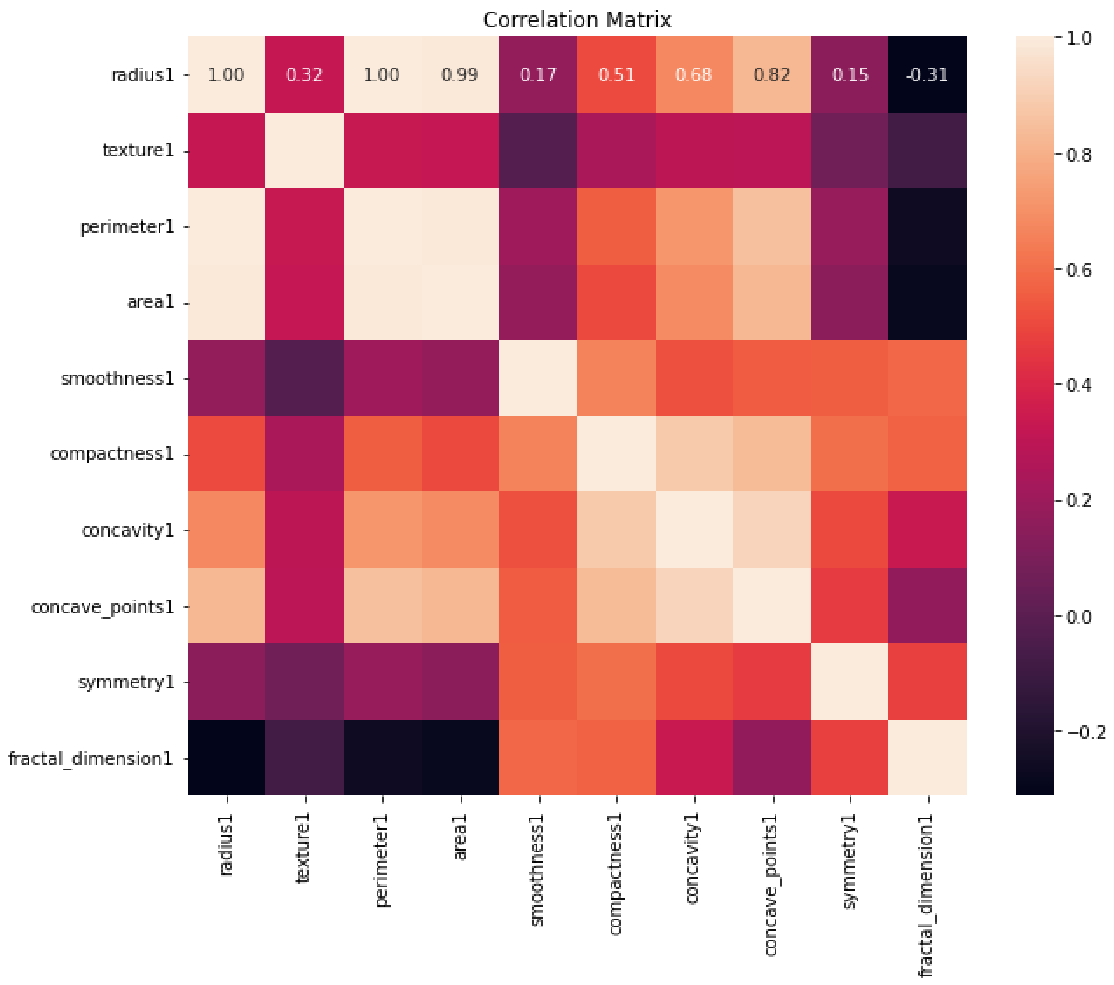

concave_points1
concave_points1
0.20
0.15
0.10
0.05
0.00
M
B
Diagnosis


symmetry1
symmetry1
0.30
0.25
0.20
0.15
0.10
M
B
Diagnosis


fractal_dimension1
fractal_dimension1
0.10
0.09
0.08
0.07
0.06
0.05
M
B
Diagnosis

4.0.4 Correlation Matrix

[13]:
```
plt.figure(figsize=(10,8))
sns.heatmap(bc_features_df.corr(), annot=True, fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



4.0.5 Logistic Regression Model

[14]:
```
# Convert target to numeric
bc_targets_df = bc_targets_df.replace(['M', 'B'], [1, 0])

# Logistic Regression Model
```

```python
cancer_data = pd.concat([bc_features_df.iloc[:, [0, 2, 3]], bc_targets_df],␣
  ↪axis=1)
log_reg_formula = "Diagnosis ~ radius1 + area1 + perimeter1"
cancer_log_reg = smf.glm(formula=log_reg_formula, data=cancer_data, family=sm.
  ↪families.Binomial()).fit()
print(cancer_log_reg.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:              Diagnosis   No. Observations:                  569
Model:                            GLM   Df Residuals:                      565
Model Family:                Binomial   Df Model:                            3
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -120.47
Date:                Tue, 28 May 2024   Deviance:                       240.94
Time:                        09:29:26   Pearson chi2:                     441.
No. Iterations:                     8   Pseudo R-squ. (CS):             0.5923
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      7.1041      6.030      1.178      0.239      -4.715      18.923
radius1       -9.0784      1.428     -6.359      0.000     -11.877      -6.280
area1          0.0338      0.011      3.170      0.002       0.013       0.055
perimeter1     1.0827      0.146      7.396      0.000       0.796       1.370
==============================================================================
```

[15]:
```python
# make our predictions
cancer_log_reg_predictions = cancer_log_reg.predict() # predict function
y_pred_log_reg = [1 if prediction >= 0.5 else 0 for prediction in␣
  ↪cancer_log_reg_predictions]
print(y_pred_log_reg)
```

```
[1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,
0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1,
1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0,
0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,
0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
1, 1, 0]
```

[16]:
```
# Get the confusion matrix for our logistic regression
print(confusion_matrix(cancer_data['Diagnosis'], y_pred_log_reg))
```

```
[[339  18]
 [ 36 176]]
```

[17]:
```
# Get the F1-score for our logistic regression
print("Logistic Regression F1 Score:", f1_score(cancer_data['Diagnosis'],␣
 ↪y_pred_log_reg))
```

```
Logistic Regression F1 Score: 0.8669950738916257
```

4.0.6 Training Our Machine Learning Model

Split data

[18]:
```
x = cancer_data[['radius1', 'perimeter1', 'area1']]
y = cancer_data['Diagnosis']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

[19]:
```
X_train.shape
```

[19]:
```
(398, 3)
```

[20]:
```
X_test.shape
```

[20]:
```
(171, 3)
```

4.0.7 Classifier Implementation

Preprocessing For Training Model

[21]:
```
# Scaling the features
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()  # SVM perfoms better with Scaler
X_scaled = scaler.fit_transform(x)

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,␣
 ↪random_state=42)
```

Support Vector Machine (SVM)

```
[22]: # SVM Model
svm_model = SVC(kernel='linear')  # Using linear kernel for simplicity
svm_model.fit(X_train, y_train.values.ravel())

# Predictions
y_pred_svm = svm_model.predict(X_test)
y_pred_svm
```

```
[22]: array([0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

```
[23]: f1_svm = f1_score(y_test, y_pred_svm)
print("Support Vector Machine F1 score:", f1_svm)
```

```
Support Vector Machine F1 score: 0.8672566371681417
```

Decision Tree

```
[24]: # Decision Tree Model
decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(X_train, y_train)

# Predictions
y_pred_dt = decision_tree_model.predict(X_test)
y_pred_dt
```

```
[24]: array([1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0], dtype=int64)
```

```
[25]: print("Decision Tree Classifier Classification Report:\n",␣
  ↪classification_report(y_test, y_pred_dt))
```

```
Decision Tree Classifier Classification Report:
               precision    recall  f1-score   support
```

```
           0       0.90      0.88      0.89       108
           1       0.80      0.84      0.82        63

    accuracy                           0.87       171
   macro avg       0.85      0.86      0.86       171
weighted avg       0.87      0.87      0.87       171
```

[26]:
```
f1_dt = f1_score(y_test, y_pred_dt)
print("Decision Tree F1 score:", f1_dt)
```

```
Decision Tree F1 score: 0.8217054263565892
```

KNN

[27]:
```
# K-NN Model
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
y_pred_knn
```

[27]:
```
array([0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0], dtype=int64)
```

[28]:
```
print("Voting Classifier Classification Report:\n",␣
 ↪classification_report(y_test, y_pred_knn))
```

```
Voting Classifier Classification Report:
               precision    recall  f1-score   support

           0       0.90      0.94      0.92       108
           1       0.90      0.83      0.86        63

    accuracy                           0.90       171
   macro avg       0.90      0.88      0.89       171
weighted avg       0.90      0.90      0.90       171
```

[29]:
```
f1_knn = f1_score(y_test, y_pred_knn)
print("KNN: F1 Score", f1_knn)
```

```
KNN: F1 Score 0.8595041322314049
```

Random Forest

[30]:
```
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
y_pred_rfc = rfc.predict(X_test)
```

[31]:
```
print("Random Forest Classifier Classification Report:\n",␣
 ↪classification_report(y_test, y_pred_rfc))
```

```
Random Forest Classifier Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.98      0.95       108
           1       0.96      0.86      0.91        63

    accuracy                           0.94       171
   macro avg       0.94      0.92      0.93       171
weighted avg       0.94      0.94      0.93       171
```

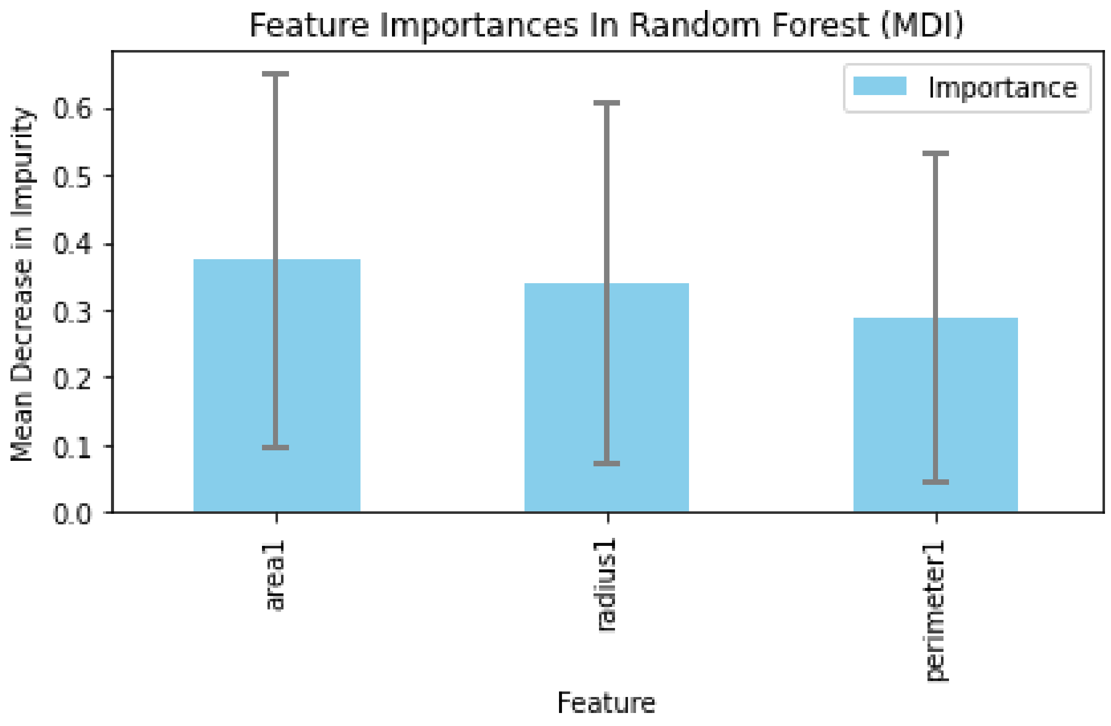
[32]:
```
f1_rfc = f1_score(y_test, y_pred_rfc)
print("Random Forest F1 score:", f1_rfc)
```

```
Random Forest F1 score: 0.9075630252100839
```

4.0.8 Feature Importances In Random Forest (MDI)

[33]:
```
'''
Determine the most significant predictors in random forest;
decided to use MDI instead of feature permutation so we do not have to
refit our model.
'''

# Webpage consulted for implementation: https://scikit-learn.org/stable/
 ↪auto_examples/ensemble/plot_forest_importances.html

# Extract feature importances and standard deviations
feature_importances = rfc.feature_importances_
importances_std = np.std([tree.feature_importances_ for tree in rfc.
 ↪estimators_], axis=0)

# Get the actual feature names from the dataset
feature_names = x.columns
# Create a DataFrame for easier handling
importances_df = pd.DataFrame({'Feature': feature_names, 'Importance':␣
 ↪feature_importances, 'Std': importances_std})

# Sort the features by importance
importances_df = importances_df.sort_values(by='Importance', ascending=False)
```

```
# Plotting
fig, ax = plt.subplots()
importances_df.plot(kind='bar', x='Feature', y='Importance', yerr='Std', ax=ax,␣
 ↪color='skyblue', error_kw=dict(ecolor='gray', lw=2, capsize=5, capthick=2))
ax.set_title("Feature Importances In Random Forest (MDI)")
ax.set_ylabel("Mean Decrease in Impurity")
plt.tight_layout()
plt.show()
```



4.0.9 Ensemble of Machine Learning Models

Voting

[34]:
```
voting_model = VotingClassifier(estimators=[
    ('svm', SVC(kernel='linear')),
    ('dt', DecisionTreeClassifier()),
    ('knn', KNeighborsClassifier())], voting='hard')
voting_model.fit(X_train, y_train)
y_pred_voting = voting_model.predict(X_test)
```

[35]:
```
f1_voting = f1_score(y_test, y_pred_voting)
print("Voting F1 score:", f1_voting)
```

```
Voting F1 score: 0.8695652173913043
```

```
[36]: print("Voting Classifier Classification Report:\n",␣
        ↪classification_report(y_test, y_pred_voting))
```

```
Voting Classifier Classification Report:
               precision    recall  f1-score   support

           0       0.89      0.98      0.93       108
           1       0.96      0.79      0.87        63

    accuracy                           0.91       171
   macro avg       0.93      0.89      0.90       171
weighted avg       0.92      0.91      0.91       171
```

Bagging

```
[37]: bagging_model = BaggingClassifier(DecisionTreeClassifier(), n_estimators=10,␣
        ↪random_state=42)
      bagging_model.fit(X_train, y_train)
      y_pred_bagging = bagging_model.predict(X_test)
```

```
[38]: f1_bag = f1_score(y_test, y_pred_bagging)
      print("Bagging F1 score:", f1_bag)
```

```
Bagging F1 score: 0.8688524590163935
```

```
[39]: print("Bagging Classifier Classification Report:\n",␣
        ↪classification_report(y_test, y_pred_bagging))
```

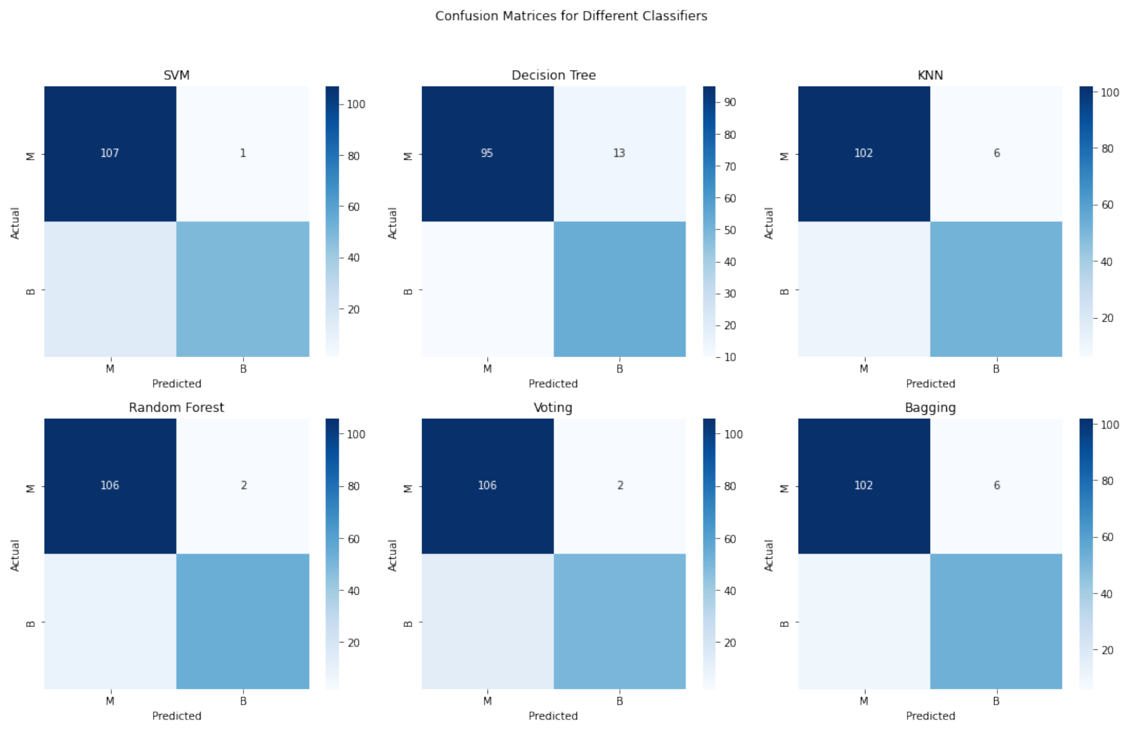
```
Bagging Classifier Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.94      0.93       108
           1       0.90      0.84      0.87        63

    accuracy                           0.91       171
   macro avg       0.90      0.89      0.90       171
weighted avg       0.91      0.91      0.91       171
```

4.1 Confusion Matrix Of Predictions Across All Models

```
[40]: # our classifier predictions grouped together
      y_pred_models = [y_pred_svm, y_pred_dt, y_pred_knn, y_pred_rfc, y_pred_voting,␣
        ↪y_pred_bagging]
      print(f"We have {len(y_pred_models)} models for review!")
```

```
We have 6 models for review!
```

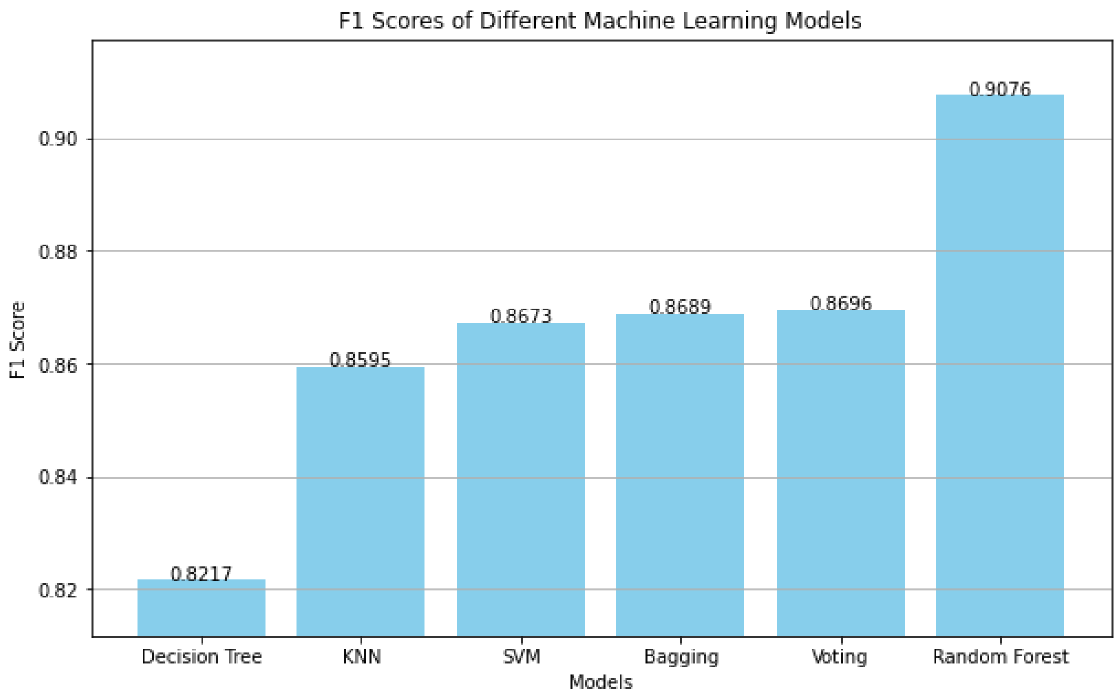
```
[41]: classifier_names = ["SVM", "Decision Tree", "KNN", "Random Forest", "Voting",␣
        ↪"Bagging"]

      # Compute confusion matrices for each model
      conf_matrices = [confusion_matrix(y_test, y_pred) for y_pred in y_pred_models]

      # Setting up the matplotlib figure
      fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))
      fig.suptitle('Confusion Matrices for Different Classifiers')

      # Labels for the axes
      labels = ['M', 'B']

      # Enumerate through the confusion matrices and plot each one
      for i, ax in enumerate(axes.flatten()):
          sns.heatmap(conf_matrices[i], annot=True, fmt="d", ax=ax, cmap='Blues',␣
        ↪xticklabels=labels, yticklabels=labels)
          ax.set_title(classifier_names[i])
          ax.set_xlabel('Predicted')
          ax.set_ylabel('Actual')

      # Adjust layout to avoid overlap
      plt.tight_layout(rect=[0, 0.03, 1, 0.95])

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



4.1.1 F1 Scores Of The Different Machine Learning Models

[42]:
```
f1_scores = [f1_svm, f1_dt, f1_knn, f1_rfc, f1_voting, f1_bag]
```

[43]:
```
# F1 scores for different models (example values)
models = ['SVM', 'Decision Tree', 'KNN', 'Random Forest', 'Voting', 'Bagging']

# Sorting the models based on their F1 scores
sorted_indices = sorted(range(len(f1_scores)), key=lambda k: f1_scores[k])
sorted_models = [models[i] for i in sorted_indices]
sorted_f1_scores = [f1_scores[i] for i in sorted_indices]

# Creating the plot
plt.figure(figsize=(10, 6))
plt.bar(sorted_models, sorted_f1_scores, color='skyblue')
plt.xlabel('Models')
plt.ylabel('F1 Score')
plt.title('F1 Scores of Different Machine Learning Models')
plt.ylim(min(sorted_f1_scores)-0.01, max(sorted_f1_scores)+0.01)
plt.grid(axis='y')

# Adding the score on top of each bar
for i in range(len(sorted_f1_scores)):
    plt.text(i, sorted_f1_scores[i], round(sorted_f1_scores[i], 4), ha='center')

# Show plot
plt.show()
```



This bar graph shows the F1 scores of different machine learning models. We will take the highest one which is the random forest.

5 Conclusion

- The logistic regression model considered radius, perimeter, and area to all be significant.
- The SVM model, after hyperparameter tuning, showed improved performance.
- Random Forest's feature importance indicates radius and perimeter as the two most significant predictors.
- The ensemble voting classifier combines predictions from SVM, Random Forest, and KNN, resulting in a robust model.
- Out of all the algorithms that were considered during this study, data scientists should consider random forest the most for classifying future cancers as malignant or benign.
- In descending order, the three most important cancer attributes for data scientists to look at are radius, perimeter, and area.
- Finally, the logistic regression and decision tree models are among the least accurate models that data scientist should not consider to make cancer classifications.

5.1 References

Benign & Malignant Tumors: Orthopedics & Sports Medicine. Available online: https://health.uconn.edu/orthopedics-sports-medicine/conditions-and-treatments/a-z-index/benign-malignant-tumors/ (accessed on 12 Dec 2023).