ASumbaraju_Project_Milestone2

July 22, 2021

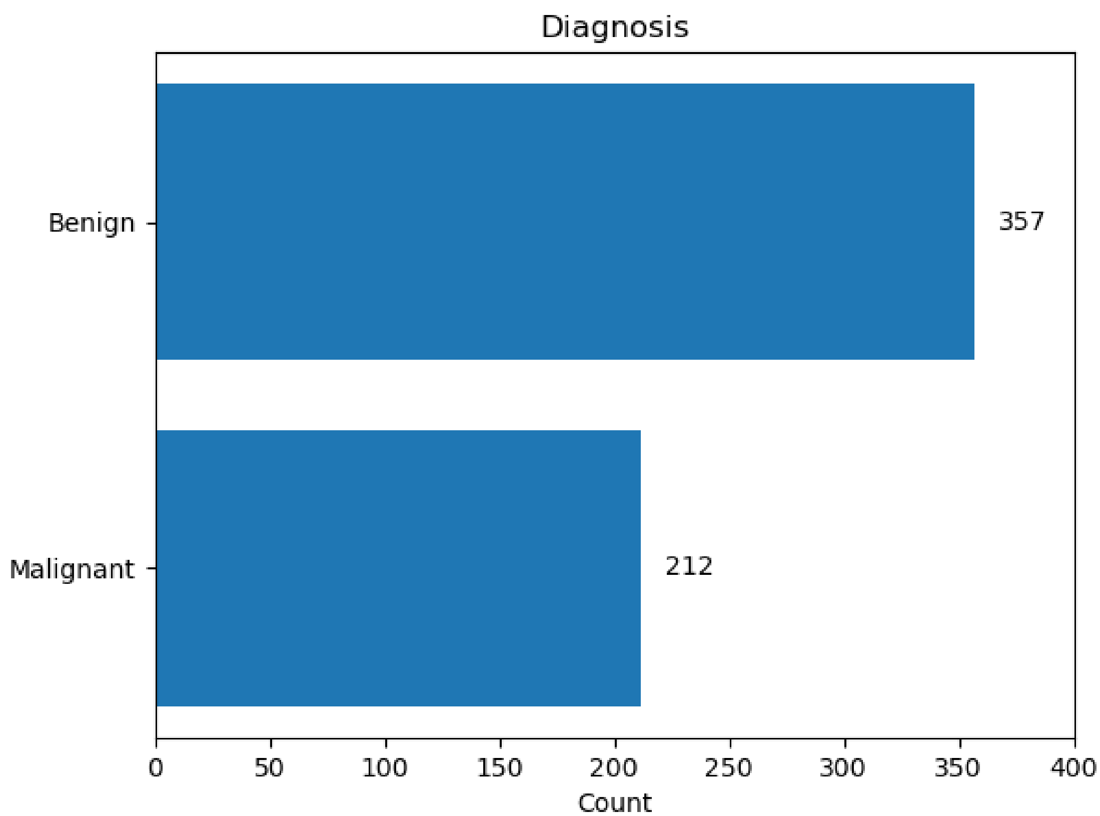
1 DSC 550 Project Milestone2

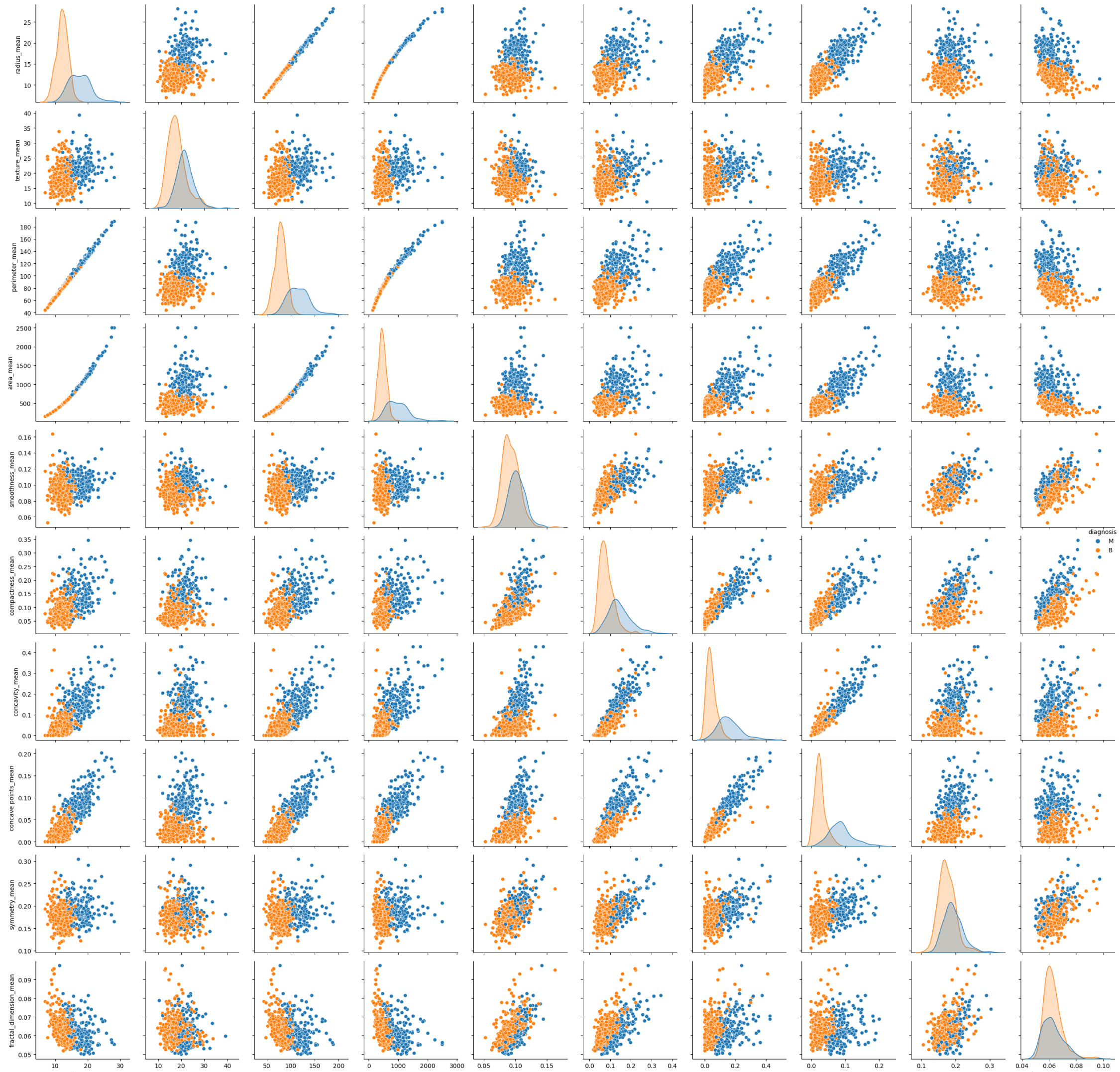
2 Aditya Sumbaraju

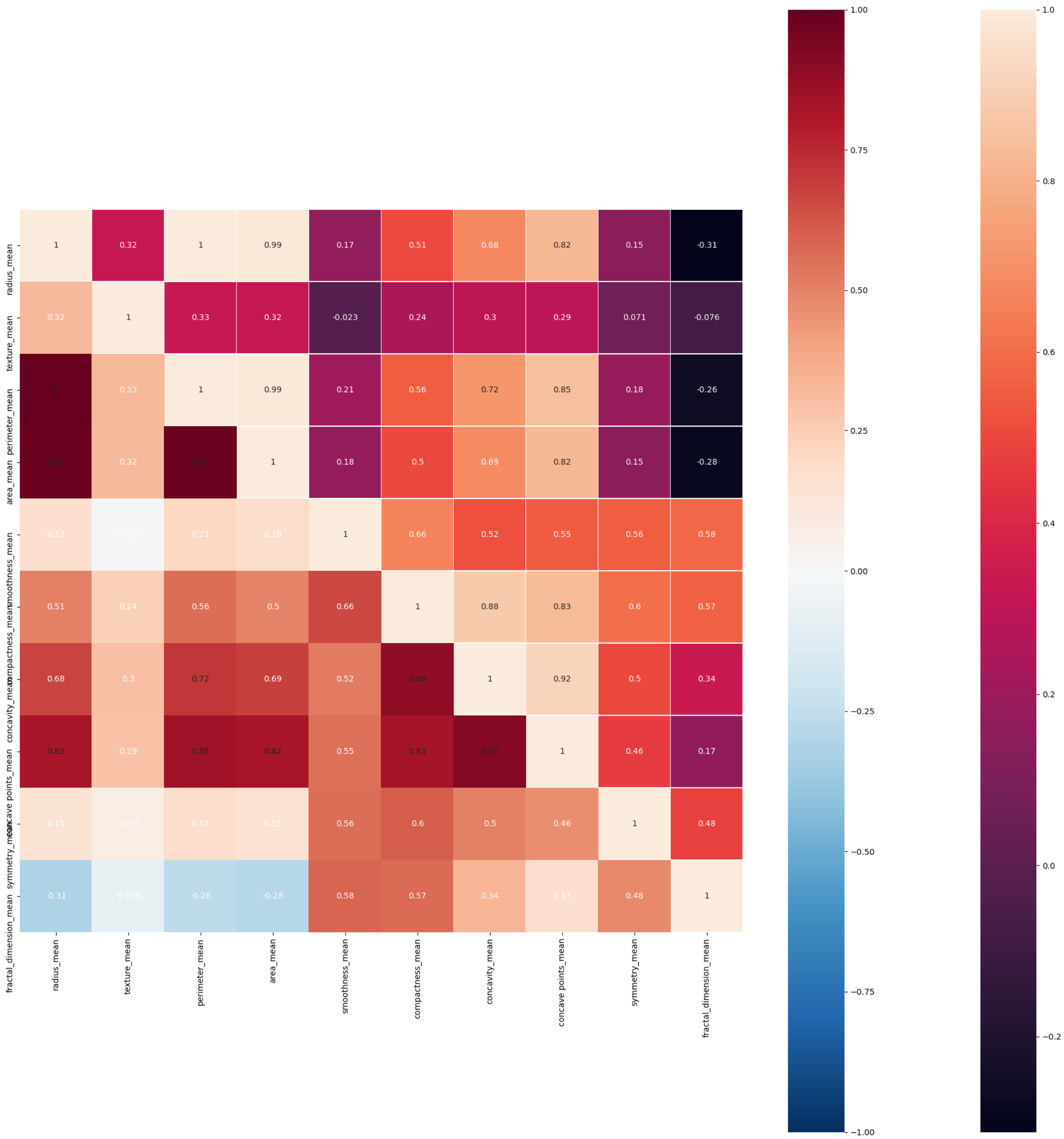
3 07/22/2021

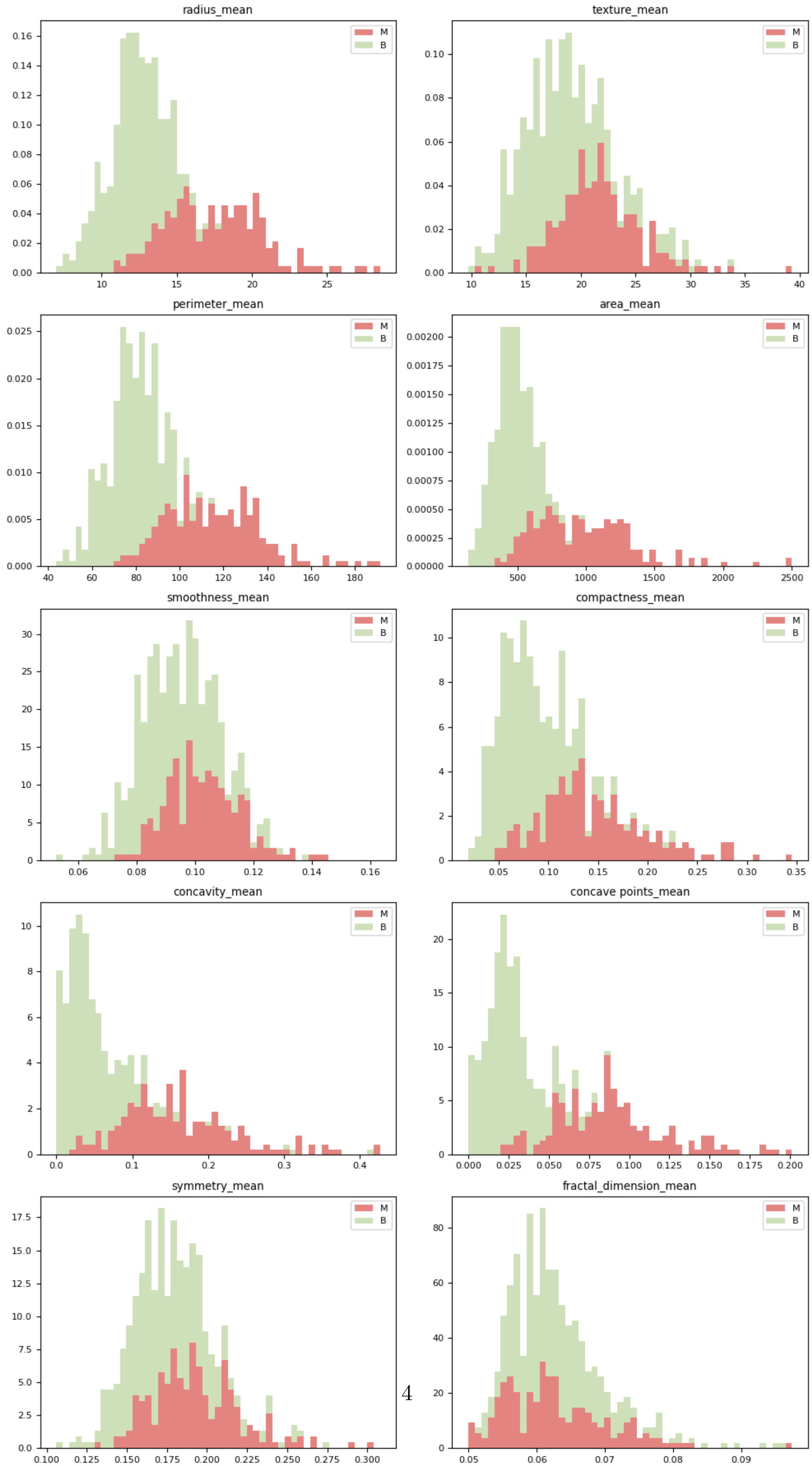
[1]:
```
#Recap of Milestone1. Import the Charts from Milestone1
from ipynb.fs.full.ASumbaraju_Project_Milestone1 import *
```

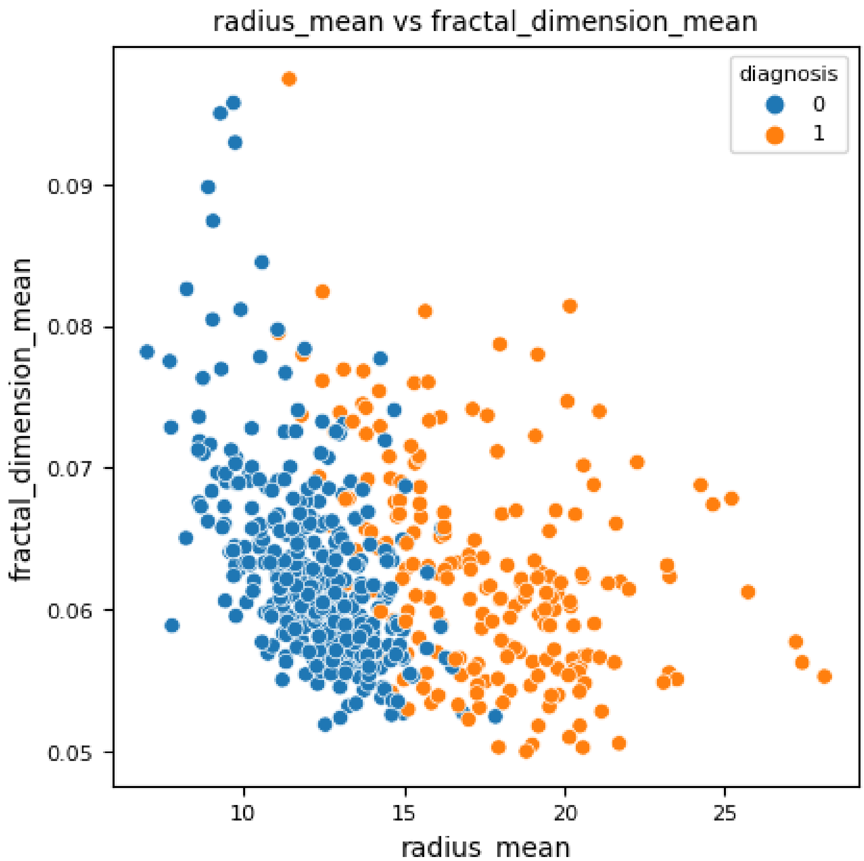
```
<Figure size 640x480 with 0 Axes>
```

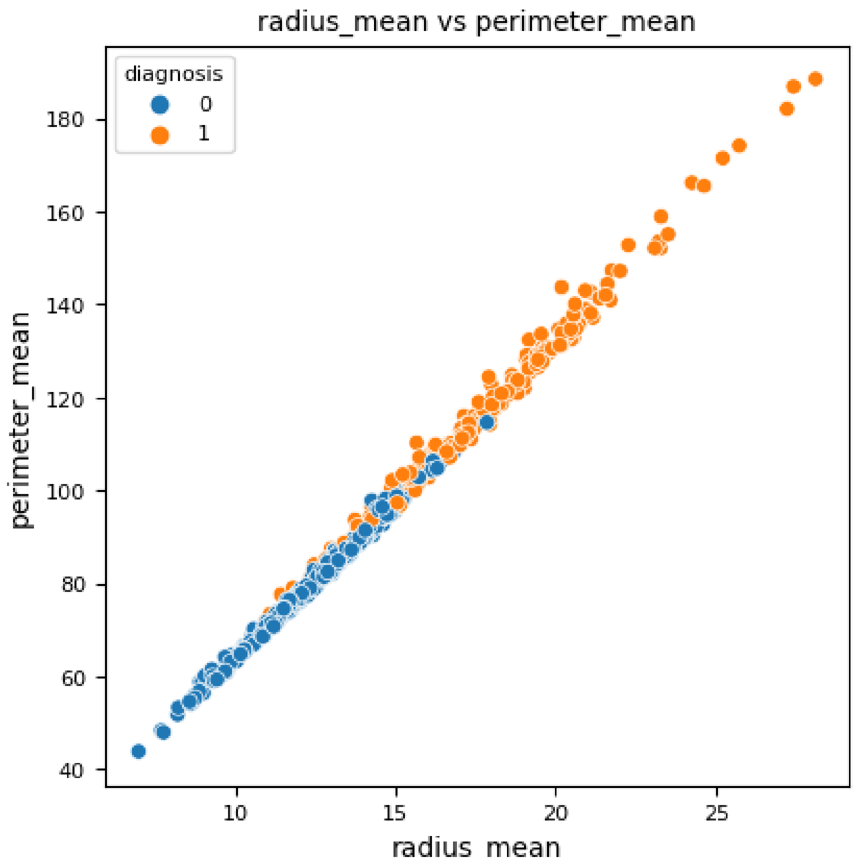


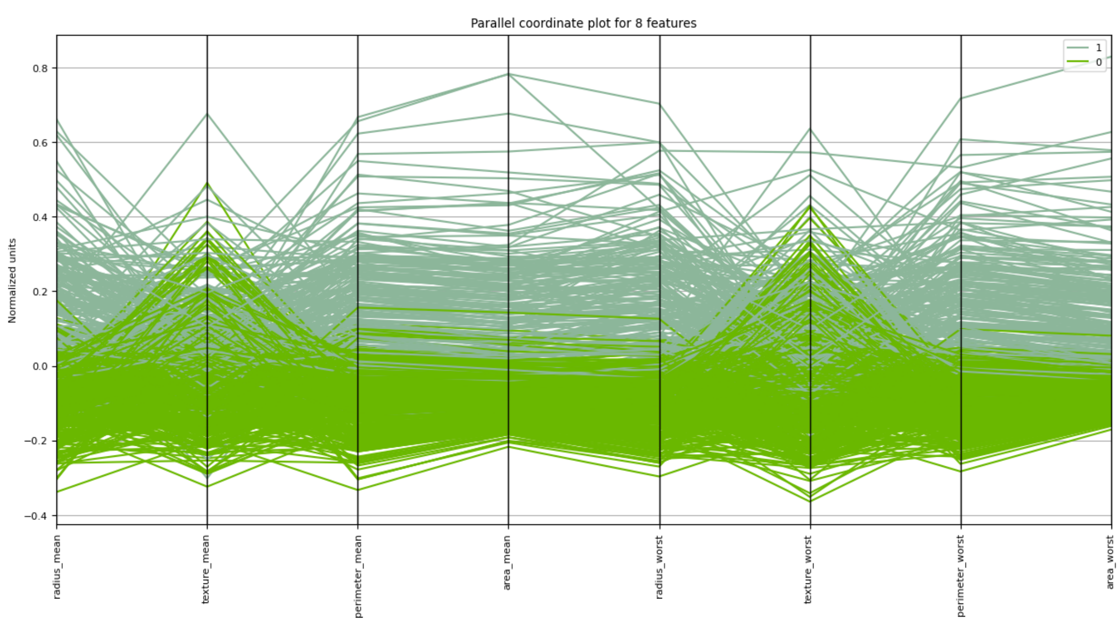

radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
diagnosis
M
B


radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean


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


radius_mean vs fractal_dimension_mean
diagnosis
0
1
0.09
0.08
0.07
0.06
0.05
fractal_dimension_mean
10
15
20
25
radius_mean


radius_mean vs perimeter_mean
diagnosis
0
1
180
160
140
120
100
80
60
40
perimeter_mean
10
15
20
25
radius_mean


Parallel coordinate plot for 8 features
1
0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
Normalized units
radius_mean
texture_mean
perimeter_mean
area_mean
radius_worst
texture_worst
perimeter_worst
area_worst
Numerical variables

4 Project Milestone 2

5 In Milestone 2, you should drop any features that are not useful for your model building. You should explain and justify why the feature dropped is not useful. You should address any missing data issues. Build any new features that you need for your model, e.g., create dummy variables for categorical features if necessary. Explain your process at each step. You can use any methods/tools you think are most appropriate

[176]:
```
#Importing the necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings

from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from yellowbrick.classifier import ConfusionMatrix
from yellowbrick.classifier import ClassificationReport
from yellowbrick.classifier import ROCAUC
from sklearn.model_selection import cross_val_score
from scipy import interp
from sklearn.metrics import roc_auc_score
from sklearn.feature_selection import SelectKBest, f_classif
warnings.filterwarnings(action="ignore")
```

[146]:
```
#Step 1: Load data into a dataframe
bc_df = "C:\BU\DSC550\project\data/data.csv"
data = pd.read_csv(bc_df)
```

[147]:
```
print('********************************************************************************␣
 ↪')
print('Numerical variable summary: \n')
print(data.describe())
print('********************************************************************************␣
 ↪')
print('Categorical variable summary: \n')
print(data.describe(include = ['O']))
print('********************************************************************************␣
 ↪')
```

```
********************************************************************************
Numerical variable summary:
```

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

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       symmetry_mean  …  radius_worst  texture_worst  perimeter_worst  \
count     569.000000  …    569.000000     569.000000       569.000000
mean        0.181162  …     16.269190      25.677223       107.261213
std         0.027414  …      4.833242       6.146258        33.602542
min         0.106000  …      7.930000      12.020000        50.410000
25%         0.161900  …     13.010000      21.080000        84.110000
50%         0.179200  …     14.970000      25.410000        97.660000
75%         0.195700  …     18.790000      29.720000       125.400000
max         0.304000  …     36.040000      49.540000       251.200000

        area_worst  smoothness_worst  compactness_worst  concavity_worst  \
count   569.000000        569.000000         569.000000       569.000000
mean    880.583128          0.132369           0.254265         0.272188
std     569.356993          0.022832           0.157336         0.208624
min     185.200000          0.071170           0.027290         0.000000
25%     515.300000          0.116600           0.147200         0.114500
50%     686.500000          0.131300           0.211900         0.226700
75%    1084.000000          0.146000           0.339100         0.382900
max    4254.000000          0.222600           1.058000         1.252000

       concave points_worst  symmetry_worst  fractal_dimension_worst
count            569.000000      569.000000               569.000000
mean               0.114606        0.290076                 0.083946
std                0.065732        0.061867                 0.018061
min                0.000000        0.156500                 0.055040
25%                0.064930        0.250400                 0.071460
50%                0.099930        0.282200                 0.080040
75%                0.161400        0.317900                 0.092080
```

```
max                 0.291000        0.663800                  0.207500

[8 rows x 31 columns]
********************************************************************************
Categorical variable summary:

       diagnosis
count        569
unique         2
top            B
freq         357
********************************************************************************
```

[148]:
```
# removing id and unnamed: 32 column which is not necessary for our model
bc_df = "C:\BU\DSC550\project\data/data.csv"
data = pd.read_csv(bc_df)
data = data.drop(['id'],axis = 1)
```

[169]:
```
# Mapping our target variable to 1 and 0
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['diagnosis'] = le.fit_transform(data['diagnosis'])
data.diagnosis.value_counts(normalize = True)
# dataset seem to be pretty balanced - Categorical variable Benign'B' stands␣
↪63% and Malignant 'M' stands 37%
```

[169]:
```
0    0.627417
1    0.372583
Name: diagnosis, dtype: float64
```

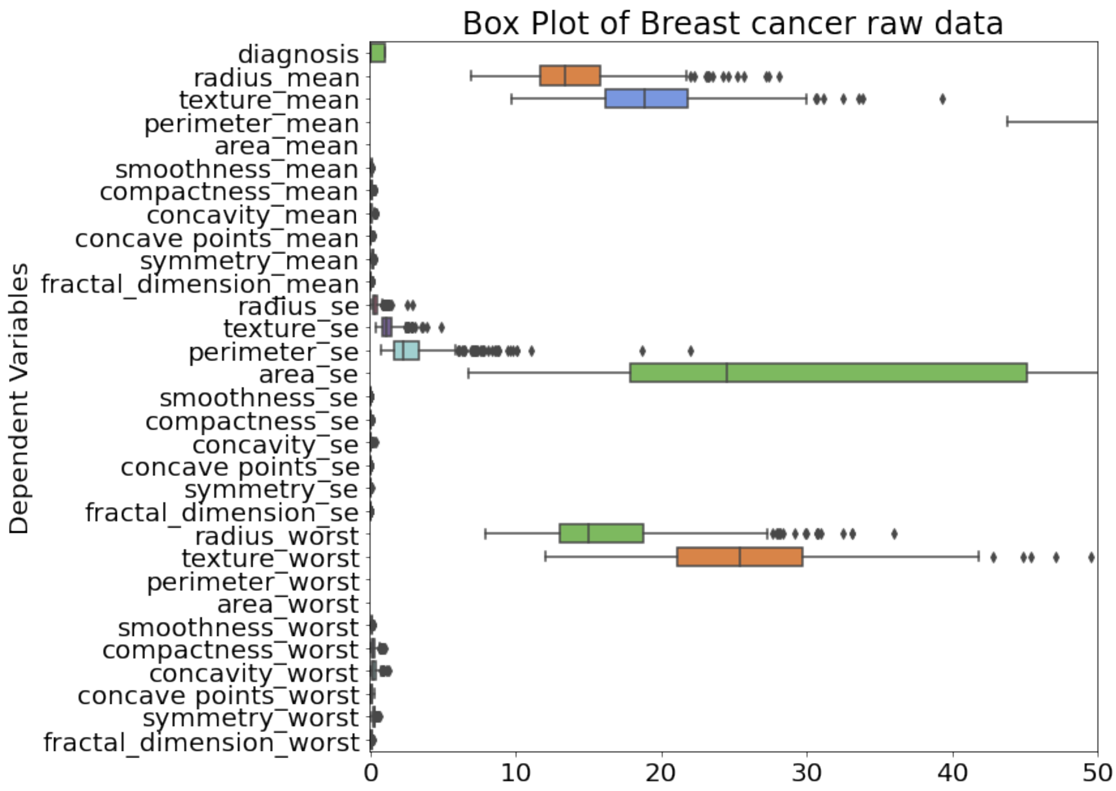
[175]:
```
# Finding correlation for all features using sns' heatmap
plt.figure(figsize=(20,20))
sns.heatmap(data.corr(),annot=False,cmap='coolwarm')
```

[175]:
```
<AxesSubplot:>
```

[152]:
```
def fig_box_plot(data_frame, data_set_name, xlim=None):

    fig, ax = plt.subplots(figsize=(10, 10))
    colors = ['#78C850', '#F08030', '#6890F0','#F8D030', '#F85888', '#705898',␣
 ↪'#98D8D8']
    #ax.set_axis_bgcolor('#fafafa')
    if xlim is not None:
        plt.xlim(*xlim)
    plt.ylabel('Dependent Variables')
    plt.title("Box Plot of {0}"\
        .format(data_set_name))
    #including all dependent variables
```

```
    ax = sns.boxplot(data = data_frame,
                     orient = 'h',
                     palette = colors)

    plt.show()
    plt.close()
```

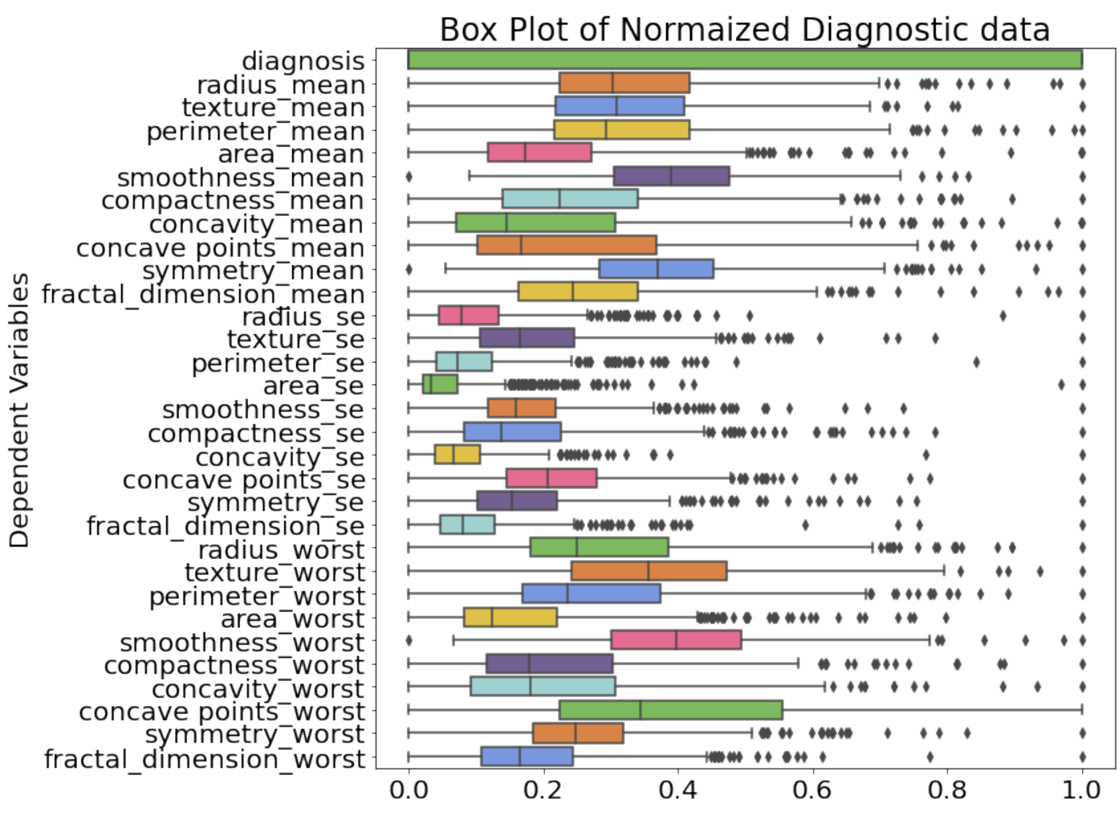
[154]:
```
fig_box_plot(data,
             'Breast cancer raw data',
             (-.05, 50))
```



[155]:
```
# fix the box plot by normalizing the data

def norm_data_frame(data_frame):
    #Intializing an empty data frame
    df_norm = pd.DataFrame()
    for col in data_frame:
        #normalize all columns that  have at > 20 unique values
        if ((len(np.unique(data_frame[col])) > 20) & (data_frame[col].dtype !=␣
↪'object')):
            df_norm[col]=((data_frame[col] - data_frame[col].min()) /
```

```
                                    (data_frame[col].max() - data_frame[col].
 ↪min()))
        else:
            df_norm[col] = data_frame[col]
    return df_norm
#Normalized dataframe values ranging (0, 1)
```

[157]:
```
fig_box_plot(norm_data_frame(data),
              'Normaized Diagnostic data',
              (-.05, 1.05))
# If we observe the median of area_mean,concave points_se,area_worst, concave␣
 ↪points_worst they are slightly towards first quartile q1
# We can consider these features for dropping
```



[158]:
```
#Converting all categorical features to numerical features using Pandas␣
 ↪get_dummies().
#This will allow RFECV to be used for feature selection.
df_dummies = pd.get_dummies(data)
```

[159]:
```
feature_space = data.iloc[:, data.columns != 'diagnosis']
feature_class = data.iloc[:, data.columns == 'diagnosis']
```

[160]:
```
#Shape of the dummy data frame
print('Shape of the dummy variable data frame: {}'.format(df_dummies.shape))

print('Features:', feature_space)
print('Target:', feature_class)
```

```
Shape of the dummy variable data frame: (569, 31)
Features:      radius_mean  texture_mean  perimeter_mean  area_mean
smoothness_mean  \
0          17.99         10.38          122.80     1001.0          0.11840
1          20.57         17.77          132.90     1326.0          0.08474
2          19.69         21.25          130.00     1203.0          0.10960
3          11.42         20.38           77.58      386.1          0.14250
4          20.29         14.34          135.10     1297.0          0.10030
..           …            …              …         …               …
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.28390         0.24140              0.10520         0.2597
4             0.13280         0.19800              0.10430         0.1809
..                …              …                    …              …
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     fractal_dimension_mean  …  radius_worst  texture_worst  \
0                   0.07871  …        25.380          17.33
1                   0.05667  …        24.990          23.41
2                   0.05999  …        23.570          25.53
3                   0.09744  …        14.910          26.50
4                   0.05883  …        22.540          16.67
..                      …  …           …             …
564                 0.05623  …        25.450          26.40
565                 0.05533  …        23.690          38.25
566                 0.05648  …        18.980          34.12
567                 0.07016  …        25.740          39.42
568                 0.05884  …         9.456          30.37

     perimeter_worst  area_worst  smoothness_worst  compactness_worst  \
```

```
0              184.60      2019.0           0.16220            0.66560
1              158.80      1956.0           0.12380            0.18660
2              152.50      1709.0           0.14440            0.42450
3               98.87       567.7           0.20980            0.86630
4              152.20      1575.0           0.13740            0.20500
..                  …           …                 …                 …
564            166.10      2027.0           0.14100            0.21130
565            155.00      1731.0           0.11660            0.19220
566            126.70      1124.0           0.11390            0.30940
567            184.60      1821.0           0.16500            0.86810
568             59.16       268.6           0.08996            0.06444

     concavity_worst  concave points_worst  symmetry_worst  \
0             0.7119                0.2654          0.4601
1             0.2416                0.1860          0.2750
2             0.4504                0.2430          0.3613
3             0.6869                0.2575          0.6638
4             0.4000                0.1625          0.2364
..                 …                     …               …
564           0.4107                0.2216          0.2060
565           0.3215                0.1628          0.2572
566           0.3403                0.1418          0.2218
567           0.9387                0.2650          0.4087
568           0.0000                0.0000          0.2871

     fractal_dimension_worst
0                    0.11890
1                    0.08902
2                    0.08758
3                    0.17300
4                    0.07678
..                         …
564                  0.07115
565                  0.06637
566                  0.07820
567                  0.12400
568                  0.07039

[569 rows x 30 columns]
Target:      diagnosis
0            1
1            1
2            1
3            1
4            1
..          …
564          1
565          1
```

```
566          1
567          1
568          0

[569 rows x 1 columns]
```

[161]:
```
# fetch the required features using RFECV - Recursive Feature Elimination and␣
 ↪Cross-Validation Selection
# witht he help of RFECV we can eliminate the irrelevent features based on␣
 ↪scoring.
# I am using f1_weighted - It results in an F-score that is not between␣
 ↪precision and recall

logreg = LogisticRegression()
rfecv = RFECV(estimator = logreg, step = 1, scoring = "f1_weighted")
rfecv.fit(feature_space, np.ravel(feature_class))
required_features = pd.DataFrame(rfecv.transform(feature_space))
```

[177]:
```
# fetch the results of required_features
print('Count of required features: {} out of {}'.format(rfecv.n_features_,␣
 ↪len(df_dummies.columns)))
print(rfecv.support_)
print (df_dummies.columns)

#We can keep 22 of the 26 features of the dummy data frame.
# Map "False" with df_dummies.columns to identify the columns that is not␣
 ↪required for modeling.
# eliminated Columns are area_mean,concave points_se,area_worst
```

```
Count of required features: 24 out of 31
[ True  True  True False  True  True  True  True  True False  True  True
  True  True False  True  True False False False  True  True  True  True
  True  True  True  True  True  True]
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

[163]:
```
#Let try to select top 5 features based on F_scores
selector = SelectKBest(f_classif, k = 5)
X = data.drop(['diagnosis'], axis = 1)
y = data['diagnosis']
```

```
X_new = selector.fit_transform(X, y)
col_names = X.columns.values[selector.get_support()]
F_scores = selector.scores_[selector.get_support()]
names_scores = list(zip(col_names, F_scores))
scores_df = pd.DataFrame(data = names_scores, columns=['Feature_names',␣
↪'F_Scores'])
#Sort the dataframe for better visualization
scores_df_sorted = scores_df.sort_values(['F_Scores', 'Feature_names'],␣
↪ascending = [False, True])
print(scores_df_sorted)
```

```
          Feature_names    F_Scores
4   concave points_worst  964.385393
3        perimeter_worst  897.944219
1    concave points_mean  861.676020
2           radius_worst  860.781707
0         perimeter_mean  697.235272
```

[164]:
```
# SelectKBest proves that the top 5 features are not contributing to␣
↪eliminiated features using RFECV and box plots
# hence it is safe to eliminate the features area_mean,concave␣
↪points_se,area_worst
```

6 Logistic Regression

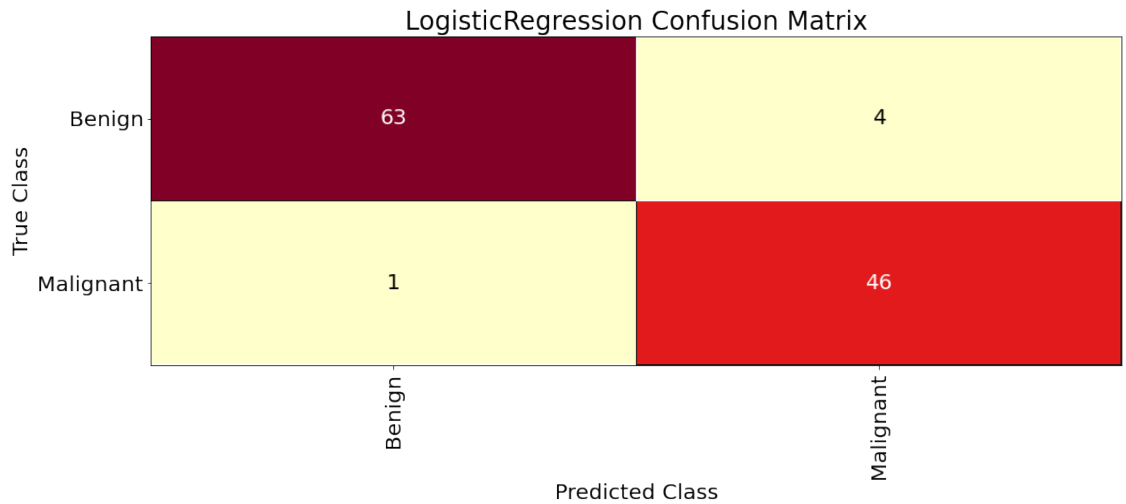
[165]:
```
# Train Test Split
# remove the features area_mean,concave points_se,area_worst from training and␣
↪test subsets
from sklearn.model_selection import train_test_split

X = data.drop(['diagnosis','area_mean','concave points_se','area_worst'], axis␣
↪= 1)
y = data['diagnosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,␣
↪random_state = 0)
# number of samples in each set
print("No. of samples in training set: ", X_train.shape[0])
print("No. of samples in validation set:", X_test.shape[0])

# Benign and Malignant
print('\n')
print('No. of Benign and Malignant diagnosis in the training set:')
print(y_train.value_counts())

print('\n')
print('No. of Benign and Malignant diagnosis in the validation set:')
```

```
print(y_test.value_counts())
```

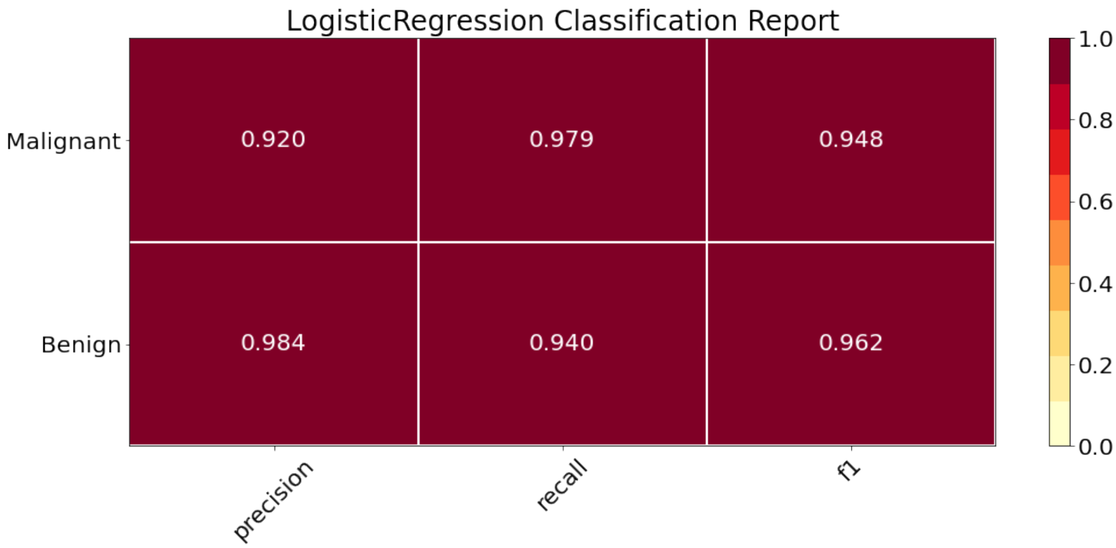
```
No. of samples in training set:  455
No. of samples in validation set: 114


No. of Benign and Malignant diagnosis in the training set:
0    290
1    165
Name: diagnosis, dtype: int64


No. of Benign and Malignant diagnosis in the validation set:
0    67
1    47
Name: diagnosis, dtype: int64
```

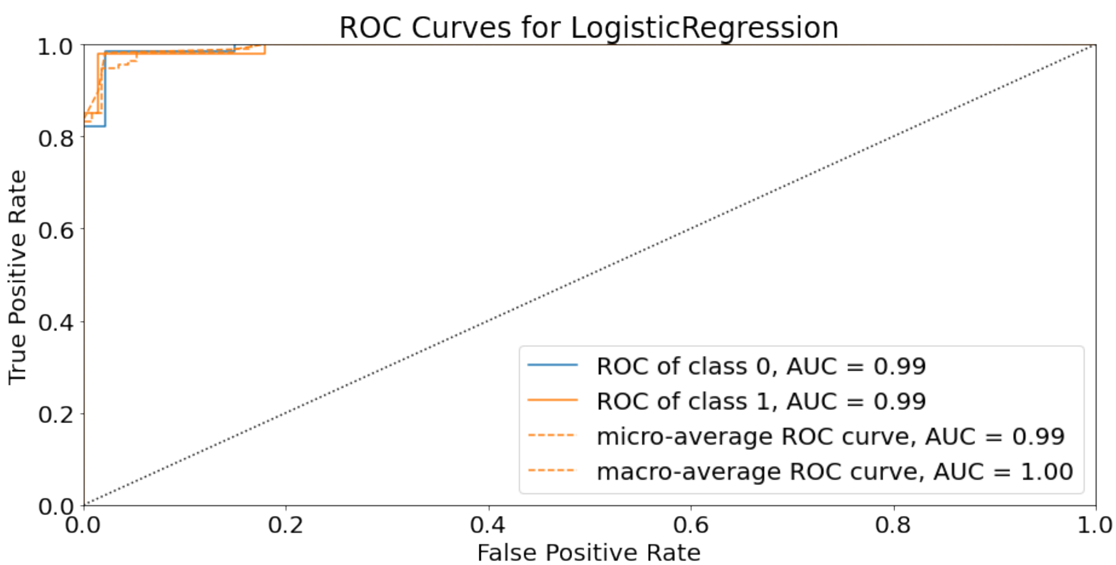
[166]:
```
# Instantiate the classification model
model = LogisticRegression()

#The ConfusionMatrix visualizer taxes a model
classes = ['Benign','Malignant']
cm = ConfusionMatrix(model, classes=classes, percent=False)

#Fit fits the passed model. This is unnecessary if you pass the visualizer a␣
↪pre-fitted model
cm.fit(X_train, y_train)

#To create the ConfusionMatrix, we need some test data. Score runs predict() on␣
↪the data
#and then creates the confusion_matrix from scikit learn.
cm.score(X_test, y_test)

# change fontsize of the labels in the figure
for label in cm.ax.texts:
    label.set_size(20)

#How did we do?
# the true positive value for Benign is 63, it means 63 positive class data␣
↪points were correctly classified by the model
# true negitive value for malignant is 46, it means 46 negative class data␣
↪points were correctly classified by the model
# this proves the predicted values match the actual values
# the false positive is 1 and false negitive is 4 - these values are␣
↪significantly low in number; hence we can say we are good with this model
# if calculate the accuracy based on confusion matrix
```

```
# formulae (TP+TN)/(TP+TN+FP+FN)= (63+46)/(63+46+1+4)= 109/114= 0.95 i.e 95 %␣
↪of accuracy.
cm.poof()
plt.show()

# reference: https://www.analyticsvidhya.com/blog/2020/04/
↪confusion-matrix-machine-learning/
```



[167]:
```
# Precision, Recall, and F1 Score
# set the size of the figure and the font size
#%matplotlib inline
plt.rcParams['figure.figsize'] = (15, 7)
plt.rcParams['font.size'] = 20

# Instantiate the visualizer
visualizer = ClassificationReport(model, classes=classes)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
g = visualizer.poof()

# ROC and AUC
#Instantiate the visualizer
visualizer = ROCAUC(model)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
g = visualizer.poof()
```





[168]:
```
#Area under the ROC Curve
#AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has␣
↪an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.
# AUC is an effective way to summarize the overall diagnostic accuracy of the␣
↪test. It takes values from 0 to 1, where a value of 0 indicates a perfectly␣
↪inaccurate test and a value of 1 reflects a perfectly accurate test.
# in our case study:
# ROC- Receiver Operating Characteristic Class 0 - Benign AUC=0.99 and
# ROC- Receiver Operating Characteristic Class 1 - Malignant AUC=0.99
# this indicates that test considered as excellent
```

```
# ROC curves above the diagonal line are considered to have reasonable
↪discriminating ability to diagnose patients with and without the breast
↪cancer.

# reference: https://www.sciencedirect.com/science/article/pii/S1556086415306043
```

7 End Of Project Milestone2

[ ]:
```

```