Breast Cancer Prediction

This project has been made as a part of a learning process of IBM's Supervise Laerning Classification course. The dataset chosen is `Breast Cancer Prediction` dataset. The target column of the dataset is categorical. The supervised machine learning algorithms implemented here are Logistic Regression, KNN, XGBoost and SVC. Their metric scores are thoroughly analysed with L2 regularized logistic regression outperforming than the rest.

Data Preprocessing

In [3]:

```
# Importing the required libraries for visualization
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Visualization Prefrences.
%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
```

```
/var/folders/db/j89yx8ld557g0y6h36s2d06r0000gn/T/ipykernel_19197/232882520.py:2:
DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of
pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better
interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

In [225…

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Visualization Prefrences.
%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
```

In [4]:

```
# Data Retrieving
df = pd.read_csv("Breast_cancer_data.csv")
df.head()
```

Out[4]:

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnos
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	

In [5]:

```
# Extract Descriptive Data.
```

```
# Extract Descriptive Data:
pd.set_option("display.float", "{:.2f}".format)
df.describe()
```

Out[5]:

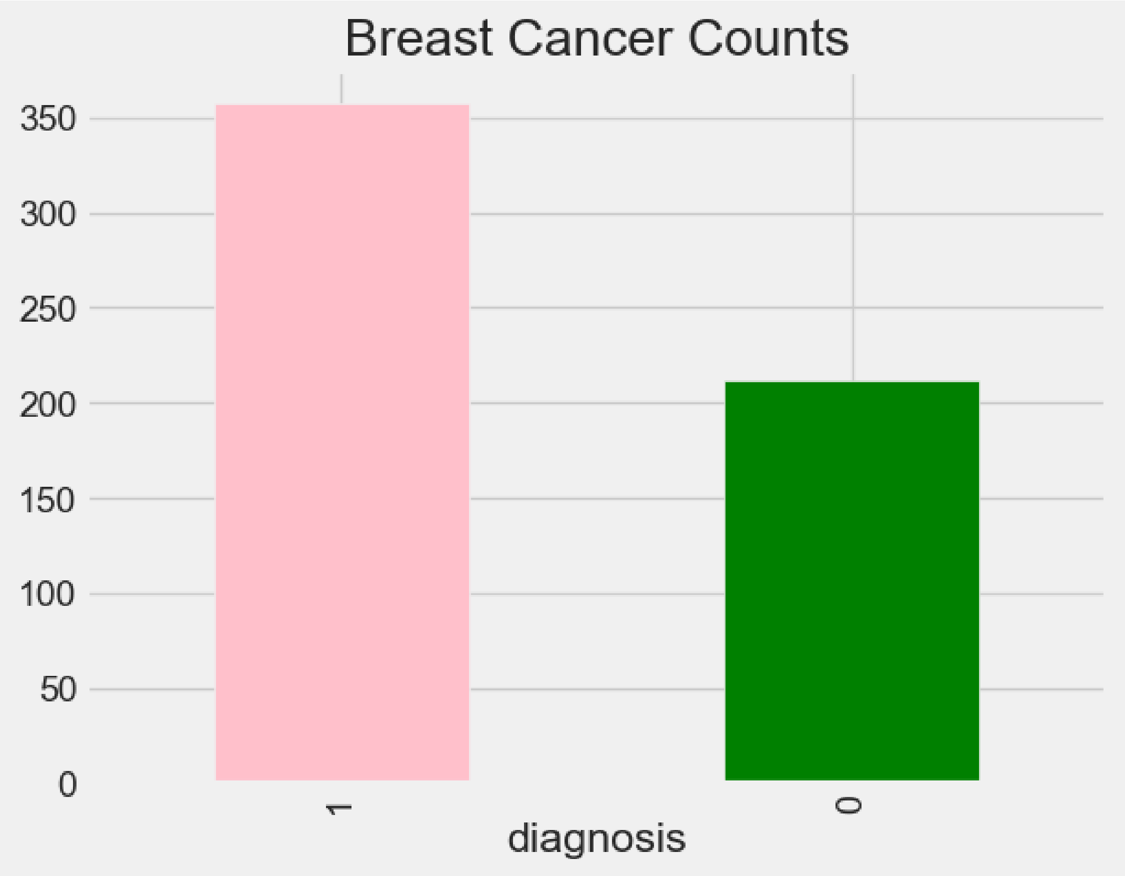
	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diag
count	569.00	569.00	569.00	569.00	569.00	5
mean	14.13	19.29	91.97	654.89	0.10	
std	3.52	4.30	24.30	351.91	0.01	
min	6.98	9.71	43.79	143.50	0.05	
25%	11.70	16.17	75.17	420.30	0.09	
50%	13.37	18.84	86.24	551.10	0.10	
75%	15.78	21.80	104.10	782.70	0.11	
max	28.11	39.28	188.50	2501.00	0.16	

In [6]:

```
#Viewing the status of women in the data set :
print(df.diagnosis.value_counts())
df.diagnosis.value_counts().plot(kind="bar", color=["pink", "green"], title =
```

```
diagnosis
1    357
0    212
Name: count, dtype: int64
```

Out[6]: `<Axes: title={'center': 'Breast Cancer Counts'}, xlabel='diagnosis'>`



In [7]:

```
# Check for Null Values
df.isna().sum()
```

Out[7]:

```
mean_radius        0
mean_texture       0
mean_perimeter     0
```

```
mean_area          0
mean_smoothness    0
diagnosis          0
dtype: int64
```

In [8]:

```python
# Categorical and Numerical Continious Features
categorical_val = []
continous_val = []
for column in df.columns:
    print('==============================')
    print(f"{column} : {df[column].unique()}")
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)

print('==============================')
print(f"Categorical Features : {categorical_val}")
print(f"Continous Features : {continous_val}")
```

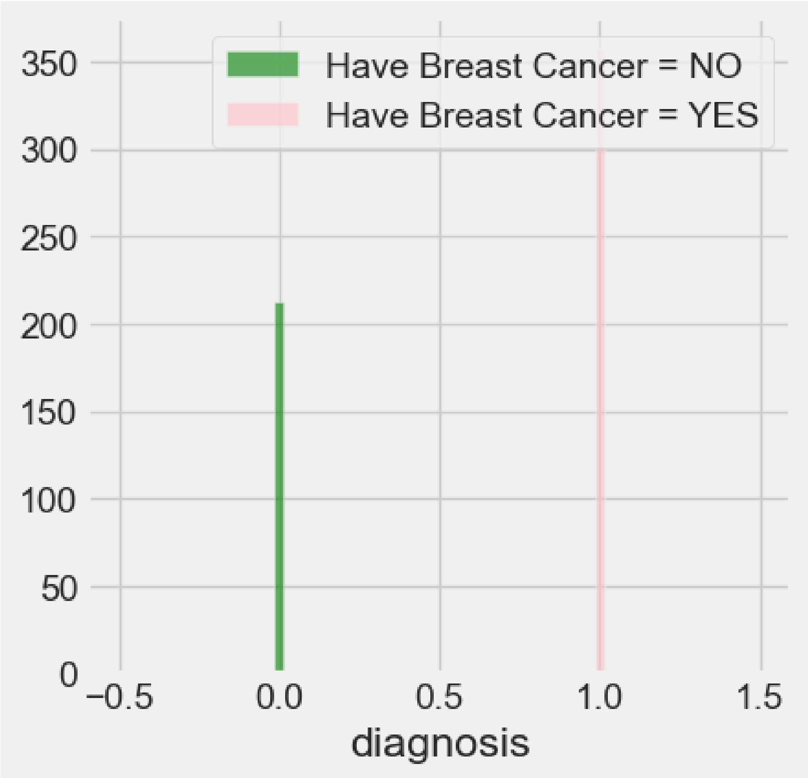
```
==============================
mean_radius : [17.99  20.57  19.69  11.42  20.29  12.45  18.25  13.71  13.    1
2.46
 16.02  15.78  19.17  15.85  13.73  14.54  14.68  16.13  19.81  13.54
 13.08   9.504 15.34  21.16  16.65  17.14  14.58  18.61  15.3   17.57
 18.63  11.84  17.02  19.27  16.74  14.25  13.03  14.99  13.48  13.44
 10.95  19.07  13.28  13.17  18.65   8.196 12.05  13.49  11.76  13.64
 11.94  18.22  15.1   11.52  19.21  14.71  13.05   8.618 10.17   8.598
  9.173 12.68  14.78   9.465 11.31   9.029 12.78  18.94   8.888 17.2
 13.8   12.31  16.07  13.53  18.05  20.18  12.86  11.45  13.34  25.22
 19.1   12.    18.46  14.48  19.02  12.36  14.64  14.62  15.37  13.27
 13.45  15.06  20.26  12.18   9.787 11.6   14.42  13.61   6.981  9.876
 10.49  13.11  11.64  22.27  11.34   9.777 12.63  14.26  10.51   8.726
 11.93   8.95  14.87  17.95  11.41  18.66  24.25  14.5   13.37  13.85
 19.    19.79  12.19  15.46  16.16  15.71  18.45  12.77  11.71  11.43
 14.95  11.28   9.738 16.11  12.9   10.75  11.9   11.8   14.44  13.74
  8.219  9.731 11.15  13.15  12.25  17.68  16.84  12.06  10.9   11.75
 19.19  19.59  12.34  23.27  14.97  10.8   16.78  17.47  12.32  13.43
 11.08  10.66   8.671  9.904 16.46  13.01  12.81  27.22  21.09  15.7
 15.28  10.08  18.31  11.81  12.3   14.22   9.72  14.86  12.91  13.77
 18.08  19.18  14.45  12.23  17.54  23.29  13.81  12.47  15.12  17.01
 15.27  20.58  28.11  17.42  14.19  13.86  11.89  10.2   19.8   19.53
 13.65  13.56  10.18  15.75  14.34  10.44  15.    12.62  12.83  17.05
 11.32  11.22  20.51   9.567 14.03  23.21  20.48  17.46  12.42  11.3
 13.75  19.4   10.48  13.2   12.89  10.65  20.94  11.5   19.73  17.3
 19.45  13.96  19.55  15.32  15.66  15.53  20.31  17.35  17.29  15.61
 17.19  20.73  10.6   13.59  12.87  10.71  14.29  11.29  21.75   9.742
 17.93  11.33  18.81  19.16  11.74  16.24  12.58  11.26  11.37  14.41
 14.96  12.95  11.85  12.72  10.91  20.09  11.46   9.    13.5   11.7
 14.61  12.76  11.54   8.597 12.49   9.042 12.43  10.25  20.16  20.34
 12.2   12.67  14.11  12.03  16.27  16.26  16.03  12.98  11.25  17.06
 12.99  18.77  10.05  23.51   9.606 11.06  19.68  10.26  14.76  11.47
 11.95  11.66  25.73  15.08  11.14  12.56  13.87   8.878  9.436 12.54
 13.3   16.5   13.4   20.44  20.2   12.21  21.71  22.01  16.35  15.19
 21.37  20.64  13.69  16.17  10.57  13.46  13.66  11.27  11.04  12.39
 14.6   13.88   8.734 15.49  21.61  12.1   14.06  13.51  12.8   17.91
 12.96  12.94  10.94  16.14  12.85  12.27  11.36   9.397 15.13   9.405
 15.5   12.7   11.16  11.57  14.69  11.61  10.03  11.13  14.9   12.4
 18.82  13.98  14.04  14.02  10.97  17.27  13.78  18.03  11.99  17.75
 14.8   14.53  21.1   11.87  13.38  11.63  13.21   9.755 17.08  27.42
 14.4   13.24  13.14   9.668 17.6   11.62   9.667 12.04  14.92  10.88
 14.2   13.9   11.49  16.25  12.16  13.47  13.7   15.73  19.44  11.68
 16.69  17.85  18.01  13.16  12.65  18.49  20.59  15.04  13.82  23.09
  9.268  9.676 12.22  16.3   14.81  15.05  19.89  12.88  12.75   9.295
 24.63   9.847  8.571 13.94  12.07  11.67  13.68  20.47  10.96  20.55
 14.27  11.69   7.729  7.691 14.47  14.74  13.62  10.32   9.683 10.82
 10.86   9.333 10.29  10.16   9.423 14.59  11.51  14.05  11.2   15.22
 20.92  21.56  20.13  16.6   20.6    7.76 ]
==============================
mean_texture : [10.38 17.77 21.25 20.38 14.34 15.7  19.98 20.83 21.82 24.04 23.2
```

```
 542.9  536.9  286.3  980.5  408.8  289.1  449.9  686.9  465.4  358.9
 506.9  618.4  599.4  404.9  815.8  455.3  602.9  546.3  571.1  747.2
 476.7  666.  1167.   420.5  857.6  466.5  992.1 1007.   538.7  680.9
 485.6  480.1 1068.  1320.   689.4  595.9  476.3 1682.   248.7  272.5
 453.1  366.5  819.8  731.3  426.   680.7  556.7  701.9  391.2 1052.
 493.1  493.8  257.8 1841.   388.1  571.   293.2  221.3  551.1  468.5
 594.2  445.2  422.9  416.2  575.5 1299.   365.6 1308.   629.8  406.4
 178.8  170.4  402.9  656.4  668.6  538.4  584.8  573.2  324.9  320.8
 285.7  360.5  378.4  507.9  264.   321.4  311.7  271.3  657.1  403.5
 600.4  386.   716.9 1347.  1479.  1261.   858.1 1265.   181. ]
==============================
mean_smoothness : [0.1184  0.08474 0.1096  0.1425  0.1003  0.1278  0.09463 0.118
9  0.1273
 0.1186  0.08206 0.0971  0.0974  0.08401 0.1131  0.1139  0.09867 0.117
 0.09831 0.09779 0.1075  0.1024  0.1073  0.09428 0.1121  0.1054  0.0944
 0.1082  0.09847 0.1064  0.1109  0.1197  0.09401 0.104   0.0961  0.09823
 0.08983 0.09387 0.1016  0.08162 0.1227  0.09081 0.1041  0.09714 0.1099
 0.086   0.1158  0.1031  0.08752 0.08637 0.07685 0.08261 0.1148  0.09056
 0.09524 0.1053  0.1137  0.0806  0.09752 0.1134  0.1243  0.1049  0.07721
 0.1122  0.1172  0.1044  0.08139 0.1066  0.09009 0.09783 0.1071  0.1007
 0.09172 0.09168 0.1291  0.1065  0.1286  0.09934 0.1102  0.1078  0.1063
 0.1215  0.09723 0.09874 0.09444 0.09029 0.08772 0.1132  0.08974 0.092
 0.07355 0.1022  0.1039  0.09078 0.1045  0.09488 0.08013 0.1005  0.09989
 0.1398  0.1142  0.08477 0.1326  0.08759 0.1037  0.09933 0.07837 0.115
 0.09768 0.09462 0.1162  0.1155  0.08402 0.09373 0.1447  0.1101  0.07115
 0.08785 0.09258 0.08217 0.1015  0.1092  0.1008  0.0943  0.09055 0.1051
 0.09639 0.1167  0.1164  0.0925  0.09721 0.08677 0.07793 0.1152  0.1091
 0.08138 0.0997  0.07944 0.1135  0.09405 0.1072  0.09754 0.09384 0.08654
 0.1115  0.07445 0.09311 0.07515 0.1089  0.08694 0.112   0.1012  0.08439
 0.08421 0.09594 0.08865 0.09855 0.1028  0.09048 0.1257  0.1006  0.08792
 0.09138 0.09699 0.06251 0.08739 0.1094  0.1141  0.09597 0.09059 0.09057
 0.09267 0.08588 0.09774 0.0808  0.08749 0.0695  0.1034  0.07941 0.12
 0.07371 0.08523 0.09872 0.09586 0.08968 0.1323  0.09965 0.08876 0.1002
 0.08182 0.0909  0.08871 0.1026  0.09363 0.08054 0.09383 0.0842  0.09646
 0.1061  0.1025  0.08445 0.09906 0.08371 0.07903 0.1088  0.06883 0.0778
 0.09159 0.08464 0.0907  0.09509 0.08355 0.08223 0.09812 0.09423 0.07926
 0.09592 0.08043 0.1027  0.107   0.07215 0.0876  0.09657 0.1013  0.09345
 0.1062  0.1035  0.0926  0.1335  0.1     0.08662 0.08999 0.0784  0.09726
 0.09469 0.09688 0.07956 0.09425 0.06429 0.09834 0.09037 0.08855 0.1225
 0.09379 0.08923 0.07948 0.09516 0.102   0.07813 0.07818 0.08393 0.08605
 0.06955 0.0802  0.08713 0.08757 0.08992 0.08372 0.09667 0.09198 0.08518
 0.09968 0.06576 0.08451 0.108   0.1068  0.08853 0.07474 0.08511 0.07005
 0.07376 0.08352 0.08814 0.07618 0.08794 0.08597 0.1074  0.07734 0.09746
 0.07557 0.08673 0.09309 0.07683 0.1169  0.1165  0.09491 0.09579 0.08306
 0.08313 0.1119  0.09116 0.1069  0.09751 0.08481 0.1033  0.09797 0.09882
 0.08386 0.08875 0.09076 0.07561 0.1149  0.07274 0.08743 0.08293 0.1009
 0.07436 0.08582 0.09676 0.09686 0.07937 0.0915  0.09905 0.09231 0.09742
 0.07963 0.1001  0.09446 0.08302 0.0988  0.09073 0.07517 0.08268 0.1216
 0.1237  0.07987 0.06935 0.1042  0.08363 0.08682 0.08108 0.07026 0.08365
 0.101   0.09996 0.116   0.1029  0.08045 0.1059  0.08044 0.07741 0.09087
 0.123   0.08872 0.07351 0.09879 0.1004  0.09495 0.07551 0.1036  0.08685
 0.08858 0.1077  0.07969 0.08515 0.0832  0.09773 0.1018  0.08546 0.08117
 0.09816 0.08801 0.08151 0.07896 0.09947 0.1133  0.08924 0.106   0.09136
 0.08458 0.08684 0.07966 0.08915 0.08331 0.08817 0.08142 0.08947 0.103
 0.09997 0.09179 0.08388 0.09684 0.06613 0.1032  0.08437 0.08583 0.09245
 0.09357 0.08791 0.08369 0.07984 0.09898 0.1084  0.06995 0.08508 0.07466
 0.08284 0.08675 0.08311 0.09289 0.1175  0.08946 0.08098 0.07699 0.0904
 0.08931 0.06828 0.1046  0.07991 0.0995  0.1043  0.09514 0.08641 0.1128
 0.07497 0.08192 0.07838 0.07372 0.07335 0.09587 0.1076  0.08928 0.1085
 0.09883 0.09342 0.1634  0.1255  0.1194  0.09427 0.1183  0.08099 0.08472
 0.1106  0.09832 0.09215 0.1218  0.1125  0.1371  0.09916 0.09492 0.09003
 0.1248  0.11    0.09277 0.09156 0.09687 0.1038  0.1236  0.08668 0.09984
 0.08837 0.08275 0.08671 0.09578 0.09246 0.09434 0.08877 0.08491 0.07431
 0.09566 0.08276 0.0924  0.08123 0.0903  0.08473 0.09261 0.09929 0.07449
 0.1048  0.111   0.0978  0.08455 0.1178  0.05263]
==============================
diagnosis : [0 1]
==============================
Categorical Features : ['diagnosis']
Continous Features : ['mean_radius', 'mean_texture', 'mean_perimeter', 'mean_are
```

```
a', 'mean_smoothness']
```

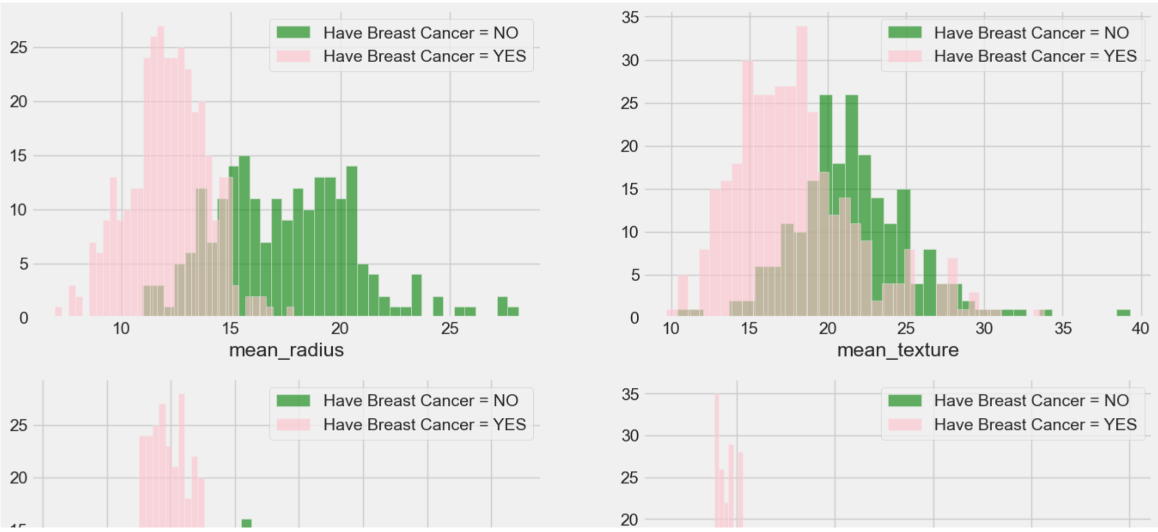
In [9]:

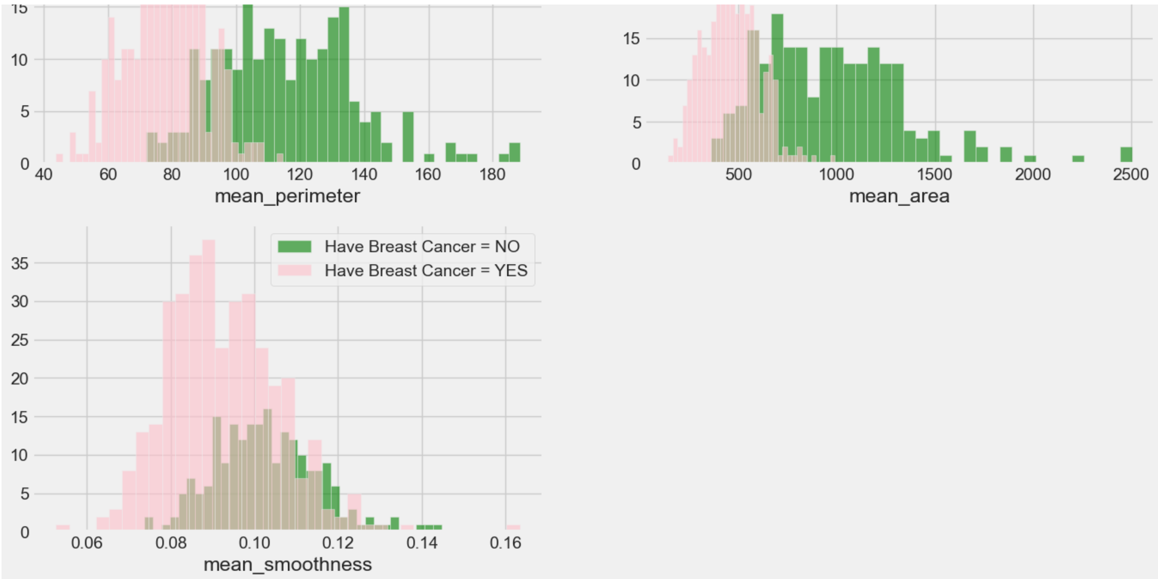
```
#Study of the relationship of categorical features and breast cancer:

plt.figure(figsize=(15, 15))

for i, column in enumerate(categorical_val, 1):
    plt.subplot(3, 3, i)
    df[df["diagnosis"] == 0][column].hist(bins=35, color='green', label='Have
    df[df["diagnosis"] == 1][column].hist(bins=35, color='pink', label='Have B
    plt.legend()
    plt.xlabel(column)
```



In [10]:

```
#Study of the relationship of continuous features and breast cancer:

plt.figure(figsize=(15, 15))

for i, column in enumerate(continous_val, 1):
    plt.subplot(3, 2, i)
    df[df["diagnosis"] == 0][column].hist(bins=35, color='green', label='Have
    df[df["diagnosis"] == 1][column].hist(bins=35, color='pink', label='Have B
    plt.legend()
    plt.xlabel(column)
```

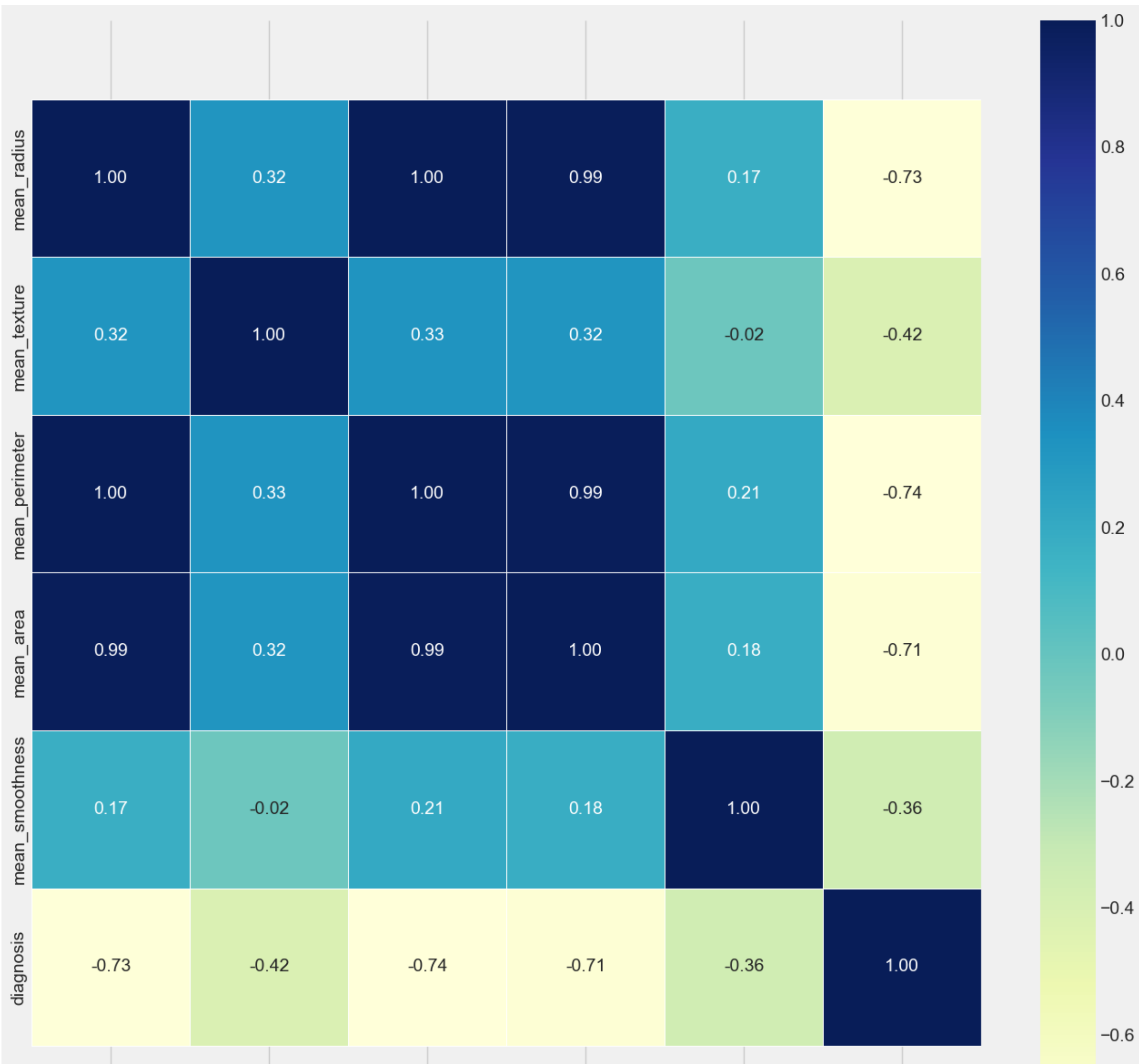


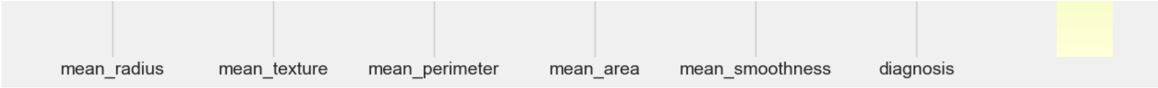


In [11]:

```python
# Studying the correlations between features using Heat Map!
corr_matrix = df.corr()
fig, ax = plt.subplots(figsize=(15, 15))
ax = sns.heatmap(corr_matrix,
                 annot=True,
                 linewidths=0.5,
                 fmt=".2f",
                 cmap="YlGnBu");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[11]: (6.5, -0.5)

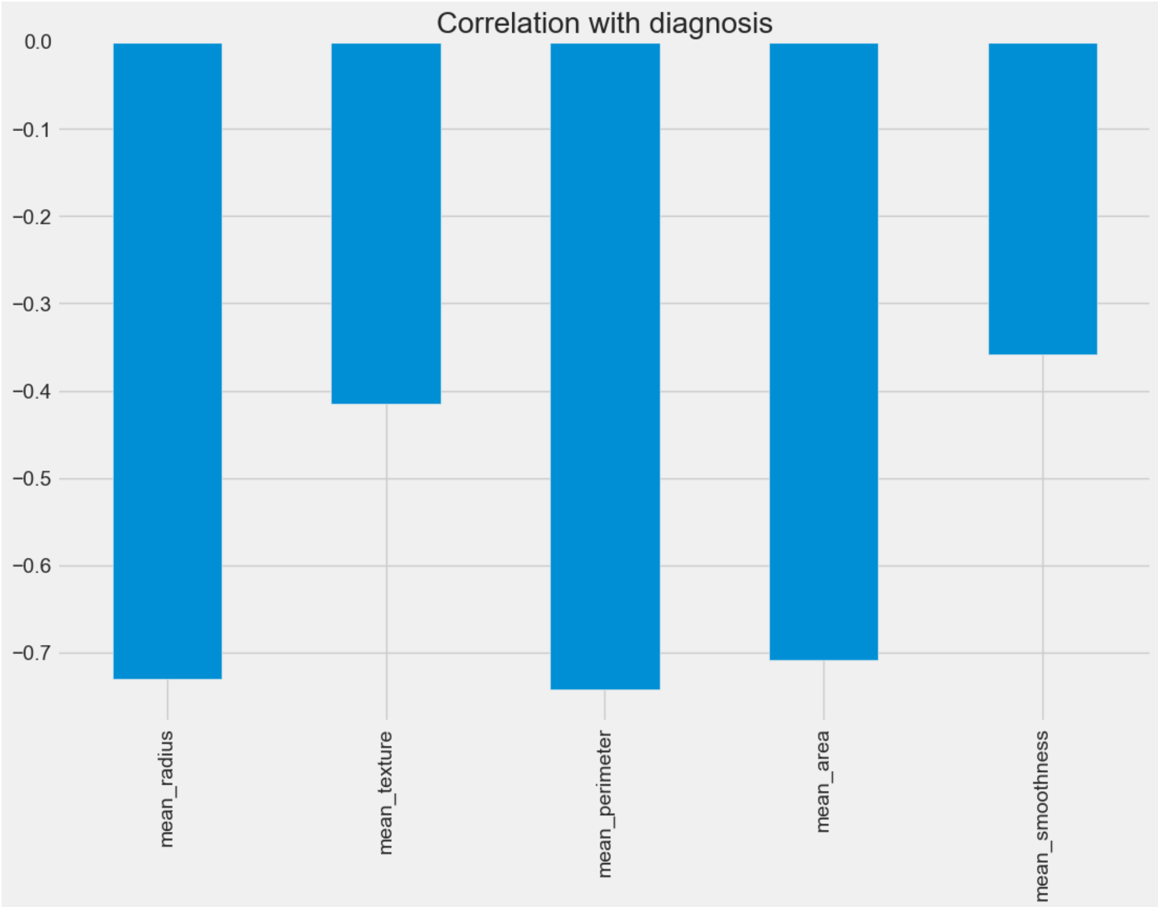




In [12]:

```
df.drop('diagnosis', axis=1).corrwith(df.diagnosis).plot(kind='bar', grid=True
                                                         title="Correlation with dia
```

Out[12]: `<Axes: title={'center': 'Correlation with diagnosis'}>`



In [13]:

```
dataset=df
```

Supervised Learning Algorithms Implementations

1: Logistic Regression Algorithm

In [15]:

```
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.metrics import classification_report, confusion_matrix, Confusion
feature_cols = [col_name for col_name in dataset.columns if col_name != 'diagn

# Get the split indexes
strat_shuf_split = StratifiedShuffleSplit(n_splits=1,
                                          test_size=0.3,
                                          random_state=42)

train_idx, test_idx = next(strat_shuf_split.split(dataset[feature_cols], datas
# Create the dataframes
X_train = dataset.loc[train_idx, feature_cols]
y_train = dataset.loc[train_idx, 'diagnosis']

X_test  = dataset.loc[test_idx, feature_cols]
y_test  = dataset.loc[test_idx, 'diagnosis']
```

In [16]:

```
### BEGIN SOLUTION
from sklearn.linear_model import LogisticRegression

# Standard logistic regression
lr = LogisticRegression(solver='liblinear').fit(X_train, y_train)
y_pred_0 = lr.predict(X_test)
clf_report = pd.DataFrame(classification_report(y_test, y_pred_0, output_dict=
clf_report
```

Out[16]:

	0	1	accuracy	macro avg	weighted avg
precision	0.92	0.91	0.91	0.91	0.91
recall	0.84	0.95	0.91	0.90	0.91
f1-score	0.88	0.93	0.91	0.90	0.91
support	64.00	107.00	0.91	171.00	171.00

In [21]:

```
from sklearn.linear_model import LogisticRegressionCV

# L1 regularized logistic regression
lr_l1 = LogisticRegressionCV(Cs=10, cv=4, penalty='l1', solver='liblinear', ma
y_pred_1 = lr_l1.predict(X_test)
clf_report = pd.DataFrame(classification_report(y_test, y_pred_1, output_dict=
clf_report
```

Out[21]:

	0	1	accuracy	macro avg	weighted avg
precision	0.83	0.96	0.91	0.90	0.91
recall	0.94	0.89	0.91	0.91	0.91
f1-score	0.88	0.92	0.91	0.90	0.91
support	64.00	107.00	0.91	171.00	171.00

In [22]:

```
# L2 regularized logistic regression
lr_l2 = LogisticRegressionCV(Cs=10, cv=4, penalty='l2', solver='liblinear').fi
y_pred_2 = lr_l2.predict(X_test)
clf_report = pd.DataFrame(classification_report(y_test, y_pred_2, output_dict=
clf_report
```

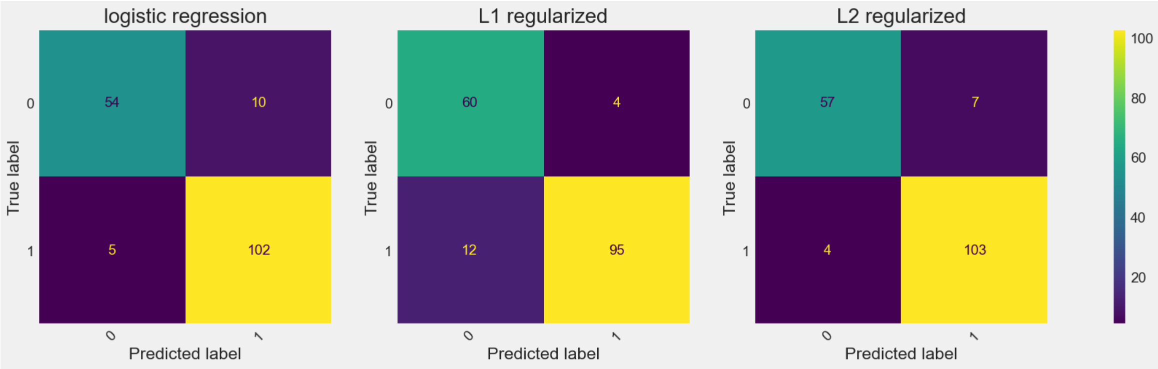
Out[22]:

	0	1	accuracy	macro avg	weighted avg
precision	0.93	0.94	0.94	0.94	0.94
recall	0.89	0.96	0.94	0.93	0.94
f1-score	0.91	0.95	0.94	0.93	0.94
support	64.00	107.00	0.94	171.00	171.00

In [25]:

```
classifiers = {
    "logistic regression": lr,
    "L1 regularized": lr_l1,
    "L2 regularized": lr_l2

}

f, axes = plt.subplots(1, 3, figsize=(20, 5))
for i, (key, classifier) in enumerate(classifiers.items()):
    y_pred = classifier.predict(X_test)
    cf_matrix = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(cf_matrix)
    disp.plot(ax=axes[i], xticks_rotation=45)
```

```
    disp.ax_.grid(False)
    disp.ax_.set_title(key)
    disp.im_.colorbar.remove()

f.colorbar(disp.im_, ax=axes)
plt.show()
```



2: KNN Algorithm

In [26]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_r
```

In [27]:

```
max_k = 40
f1_scores = list()
error_rates = list() # 1-accuracy

for k in range(1, max_k):

    knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
    knn = knn.fit(X_train, y_train)

    y_pred = knn.predict(X_test)
    f1 = f1_score(y_pred, y_test)
    f1_scores.append((k, round(f1_score(y_test, y_pred), 4)))
    error = 1-round(accuracy_score(y_test, y_pred), 4)
    error_rates.append((k, error))

f1_results = pd.DataFrame(f1_scores, columns=['K', 'F1 Score'])
error_results = pd.DataFrame(error_rates, columns=['K', 'Error Rate'])

# Get minimum error id
min_error_id = error_results['Error Rate'].idxmin()

# Get Best K
error_results.loc[min_error_id]
```

Out[27]:

```
K             5.00
Error Rate    0.11
Name: 4, dtype: float64
```

In [28]:

```
knn = KNeighborsClassifier(n_neighbors=25, weights='distance')
knn = knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

KNN_report = pd.DataFrame(classification_report(y_test, y_pred, output_dict=Tr
KNN_report
```

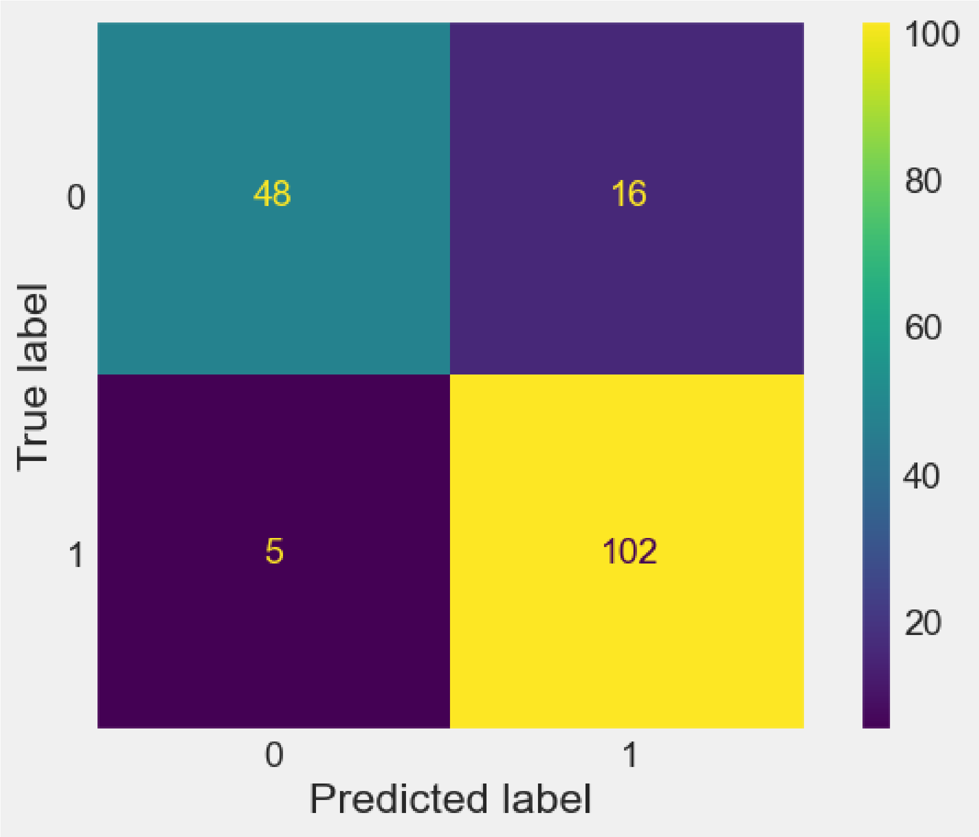
Out[28]:

	0	1	accuracy	macro avg	weighted avg
precision	0.91	0.86	0.88	0.89	0.88
recall	0.75	0.95	0.88	0.85	0.88

f1-score	0.82	0.91	0.88	0.86	0.87
support	64.00	107.00	0.88	171.00	171.00

In [29]:

```
cm = confusion_matrix(y_test, y_pred, labels=knn.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=knn.classes_
disp.plot()
plt.grid(False)
plt.show()
```



3: XGBoost Algorthim

In [30]:

```
%pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-2.0.3-py3-none-macosx_10_15_x86_64.macosx_11_0_x86_64.maco
sx_12_0_x86_64.whl (2.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 9.4 MB/s eta 0:00:000m
eta 0:00:01[36m0:00:01
Requirement already satisfied: numpy in ./classification/lib/python3.11/site-pac
kages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in ./classification/lib/python3.11/site-pac
kages (from xgboost) (1.12.0)
Installing collected packages: xgboost
Successfully installed xgboost-2.0.3

[notice] A new release of pip available: 22.3 -> 24.0
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [34]:

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

param_grid = {
    "max_depth": [7],
    "learning_rate": [0.05],
    "gamma": [0, 0.25, 1, 10],
    "reg_lambda": [0]
```

```
    "reg_lambda": [0],
    "scale_pos_weight": [1, 3, 5, 7, 10],
    "subsample": [0.1,0.2, 0.3, 0.4, 0.5, 0.8],
    "colsample_bytree": [0.5,0.7],
}


# Init classifier
xgb_cl = xgb.XGBClassifier(objective="binary:logistic")

# Init Grid Search
grid_cv = GridSearchCV(xgb_cl, param_grid, n_jobs=-1, cv=3, scoring="roc_auc")

# Fit
_ = grid_cv.fit(X_train, y_train)
```

In [35]:

```
grid_cv.best_params_
```

Out[35]:

```
{'colsample_bytree': 0.7,
 'gamma': 1,
 'learning_rate': 0.05,
 'max_depth': 7,
 'reg_lambda': 0,
 'scale_pos_weight': 3,
 'subsample': 0.4}
```

In [36]:

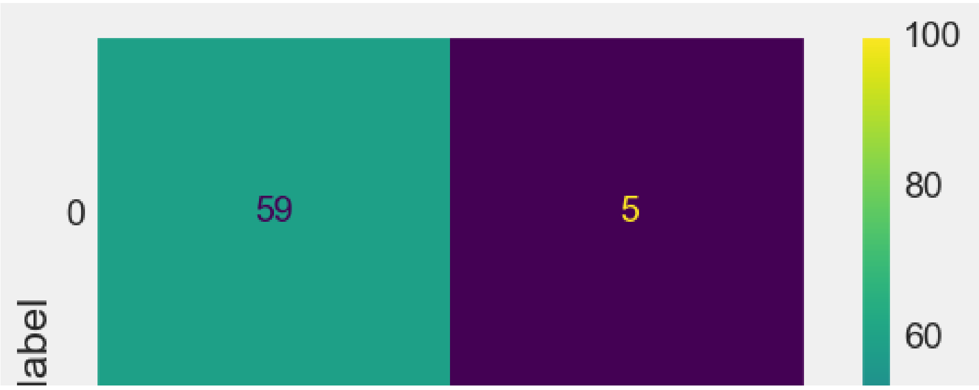
```
final_xgb_cl = xgb.XGBClassifier(
    **grid_cv.best_params_,
    objective="binary:logistic",
)

_ = final_xgb_cl.fit(X_train, y_train)

y_pred = final_xgb_cl.predict(X_test)

xgb_report = pd.DataFrame(classification_report(y_test, y_pred, output_dict=Tr
xgb_report
```

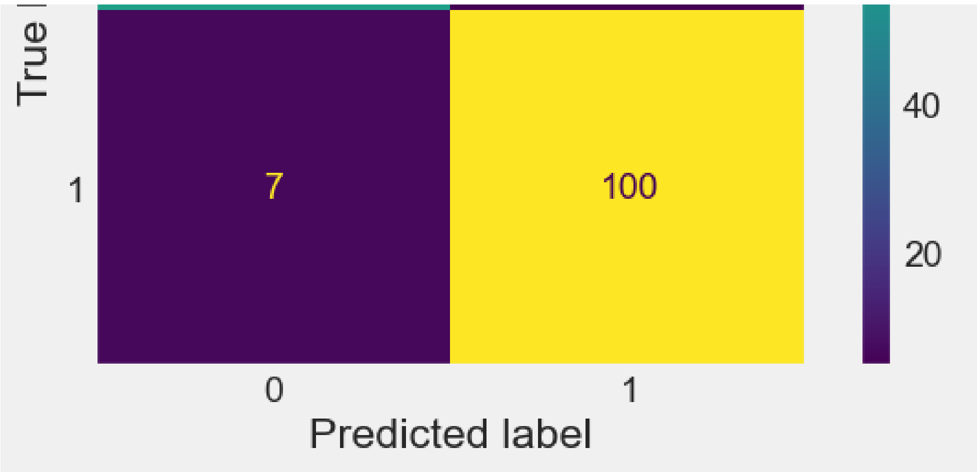
Out[36]:

	0	1	accuracy	macro avg	weighted avg
precision	0.89	0.95	0.93	0.92	0.93
recall	0.92	0.93	0.93	0.93	0.93
f1-score	0.91	0.94	0.93	0.93	0.93
support	64.00	107.00	0.93	171.00	171.00

In [37]:

```
cm = confusion_matrix(y_test, y_pred, labels=final_xgb_cl.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=final_xgb_cl
disp.plot()
plt.grid(False)
plt.show()
```





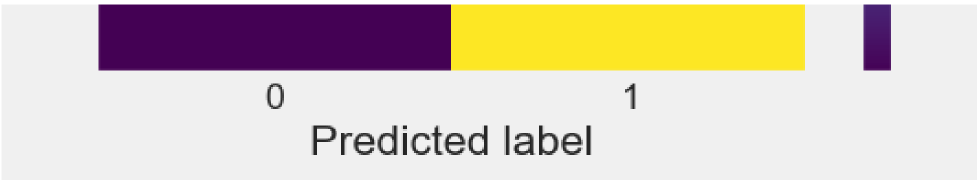
4: SVC Algorthim

In [38]:
```
from sklearn.svm import SVC

kwargs = {'kernel': 'rbf'}
svc = SVC(**kwargs)

SVC_cl = svc.fit(X_train, y_train)
y_pred = SVC_cl.predict(X_test)
SVC_cl_report = pd.DataFrame(classification_report(y_test, y_pred, output_dict
SVC_cl_report
```

Out[38]:

	0	1	accuracy	macro avg	weighted avg
precision	0.90	0.84	0.85	0.87	0.86
recall	0.69	0.95	0.85	0.82	0.85
f1-score	0.78	0.89	0.85	0.83	0.85
support	64.00	107.00	0.85	171.00	171.00

In [39]:
```
cm = confusion_matrix(y_test, y_pred, labels=SVC_cl.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=SVC_cl.class
disp.plot()
plt.grid(False)
plt.show()
```


0
1
Predicted label