Binomial logistic regression Diagnostic breast cancer Wisconsin

---

0.1 Introduction

This project aims to generate a binomial logistic regression model to classify, based on an examination, whether cancer is benign or malignant.

The work is carried out based on analysis, compression, data cleaning, metrics, testing and validation of the model, with the following work path:

- Development
- Understanding the data
- Data cleaning
  - Check null values
- Correlation analysis
- Data standardization:
- Exploratory data analysis
- Model creation
  - Split training and test data
  - Create and train the model
  - Evaluate the accuracy of the model with metrics
  - Evaluate the model with cross validation
- Analysis of results
  - Get the probabilities
  - Get the coefficients
  - Confusion Matrix
- Test
  - Test eliminating the characteristics with lower coefficients
  - Test balancing the number of class records with SMOTE
- Regularizers
  - L1 Lasso
  - L2 Ridge
- Conclusions

0.2 Data

The Data set "Breast Cancer Wisconsin (Diagnostic)" of kaggel contains Predict whether the cancer is benign or malignant. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

Attribute Information:

- ID number
- Diagnosis (M = malignant, B = benign)
- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

0.3 Development

0.3.1 Importing Libraries

[73]:
```
! pip install imblearn
```

```
Requirement already satisfied: imblearn in
/home/williamccs/miniconda3/envs/cookiecutter-personal/lib/python3.11/site-
packages (0.0)
Requirement already satisfied: imbalanced-learn in
/home/williamccs/miniconda3/envs/cookiecutter-personal/lib/python3.11/site-
packages (from imblearn) (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in
/home/williamccs/miniconda3/envs/cookiecutter-personal/lib/python3.11/site-
packages (from imbalanced-learn->imblearn) (1.25.2)
Requirement already satisfied: scipy>=1.5.0 in
/home/williamccs/miniconda3/envs/cookiecutter-personal/lib/python3.11/site-
packages (from imbalanced-learn->imblearn) (1.11.1)
Requirement already satisfied: scikit-learn>=1.0.2 in
/home/williamccs/miniconda3/envs/cookiecutter-personal/lib/python3.11/site-
packages (from imbalanced-learn->imblearn) (1.3.0)
Requirement already satisfied: joblib>=1.1.1 in
/home/williamccs/miniconda3/envs/cookiecutter-personal/lib/python3.11/site-
packages (from imbalanced-learn->imblearn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/williamccs/miniconda3/envs/cookiecutter-personal/lib/python3.11/site-
```

```
packages (from imbalanced-learn->imblearn) (3.2.0)
```

[74]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
import seaborn as sns
import sklearn.metrics as metrics
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTEENN
print('Imported libraries')
```

```
Imported libraries
```

0.3.2 Import the dataset

[75]:
```
df = pd.read_csv("/home/williamccs/cookiecutter-personal/projects_personal_ds/
↪logistic_regression_diagnostic_breast_cancer_wisconsin/data/data.csv")
df.head()
```

[75]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
0	0.11840	0.27760	0.3001	0.14710	
1	0.08474	0.07864	0.0869	0.07017	
2	0.10960	0.15990	0.1974	0.12790	
3	0.14250	0.28390	0.2414	0.10520	
4	0.10030	0.13280	0.1980	0.10430	

	…	texture_worst	perimeter_worst	area_worst	smoothness_worst	\
0	…	17.33	184.60	2019.0	0.1622	
1	…	23.41	158.80	1956.0	0.1238	
2	…	25.53	152.50	1709.0	0.1444	
3	…	26.50	98.87	567.7	0.2098	
4	…	16.67	152.20	1575.0	0.1374	

	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	\
0	0.6656	0.7119	0.2654	0.4601	

```
1            0.1866           0.2416                0.1860          0.2750
2            0.4245           0.4504                0.2430          0.3613
3            0.8663           0.6869                0.2575          0.6638
4            0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

There are 32 characteristics including the "diagnosis" label

[76]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
```

```
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

[77]:
```
df.describe()
```

[77]:

	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	…	texture_worst	perimeter_worst	area_worst
count	569.000000	…	569.000000	569.000000	569.000000
mean	0.181162	…	25.677223	107.261213	880.583128
std	0.027414	…	6.146258	33.602542	569.356993
min	0.106000	…	12.020000	50.410000	185.200000
25%	0.161900	…	21.080000	84.110000	515.300000
50%	0.179200	…	25.410000	97.660000	686.500000
75%	0.195700	…	29.720000	125.400000	1084.000000
max	0.304000	…	49.540000	251.200000	4254.000000

	smoothness_worst	compactness_worst	concavity_worst
count	569.000000	569.000000	569.000000
mean	0.132369	0.254265	0.272188
std	0.022832	0.157336	0.208624

```
min            0.071170          0.027290         0.000000
25%            0.116600          0.147200         0.114500
50%            0.131300          0.211900         0.226700
75%            0.146000          0.339100         0.382900
max            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

       Unnamed: 32
count          0.0
mean           NaN
std            NaN
min            NaN
25%            NaN
50%            NaN
75%            NaN
max            NaN

[8 rows x 32 columns]
```

[ ]:

Class distribution: - B : benign - M : Malignant

[78]:
```
print("Number of registers:", df["diagnosis"].value_counts(), "\n",␣
  ↪"_____________________")
print("Percentage of registers:", df["diagnosis"].value_counts()* 100 /len(df))
```

```
Number of registers: diagnosis
B    357
M    212
Name: count, dtype: int64

 _____________________
Percentage of registers: diagnosis
B    62.741652
M    37.258348
Name: count, dtype: float64
```

0.4 Data cleaning

- The column "Unnamed: 32" does not provide information.
- The id column is irrelevant for model training

[79]:
```
df_clean = df.drop(["id", "Unnamed: 32"], axis=1)
df_clean.head()
```

[79]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	…	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	…	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	…	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	…	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	…	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	…	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

[5 rows x 31 columns]

Check null values

[80]:
```
df_clean.isna().sum()
```

[80]:
```
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
```

Convert the target variable to numeric

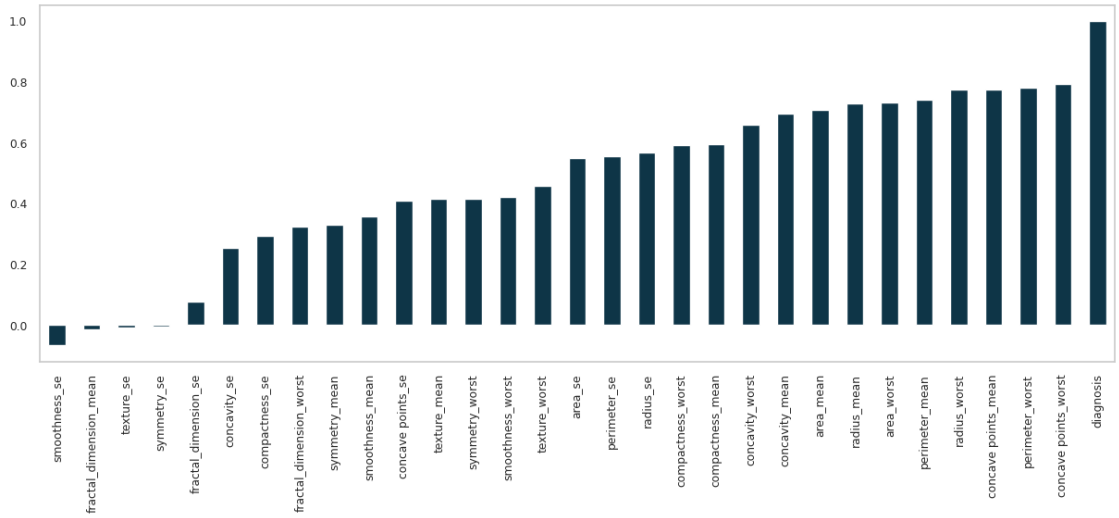
[81]:
```
df_clean["diagnosis"].replace(to_replace="M", value=1, inplace=True)
df_clean["diagnosis"].replace(to_replace="B", value=0, inplace=True)
df_clean["diagnosis"].value_counts()
```

[81]:
```
diagnosis
0    357
1    212
Name: count, dtype: int64
```

0.5 Correlation analysis

As we have many variables, we do the correlation analysis with the target variable "diagnosis"

[82]:
```
colors = ["#0E3547", "#92EAFF"]

plt.figure(figsize=(15,5))
df_clean.corr()["diagnosis"].sort_values(ascending=True).plot(kind="bar",␣
 ↪color=colors[0])
plt.grid()
plt.show()
```



Most attributes have high positive correlations with the target variable.

0.6 Data standardization

We use the MinMaxScaler method from sklearn

[83]:
```
scaler = StandardScaler()

df_processing_scaler = scaler.fit_transform(df_clean.drop("diagnosis", axis=1))
df_processing_scaler.shape
```

[83]:
```
(569, 30)
```

We convert the tensor into a pandas data frame

[84]:
```
columns = list(df_clean.columns)
columns.remove("diagnosis")
df_processing_scaler = pd.DataFrame(df_processing_scaler, columns=columns)
```

```
df_processing_scaler.head()
```

[84]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	\
0	1.097064	-2.073335	1.269934	0.984375	1.568466	
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	
2	1.579888	0.456187	1.566503	1.558884	0.942210	
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	
4	1.750297	-1.151816	1.776573	1.826229	0.280372	

	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	\
0	3.283515	2.652874	2.532475	2.217515	
1	-0.487072	-0.023846	0.548144	0.001392	
2	1.052926	1.363478	2.037231	0.939685	
3	3.402909	1.915897	1.451707	2.867383	
4	0.539340	1.371011	1.428493	-0.009560	

	fractal_dimension_mean	…	radius_worst	texture_worst	perimeter_worst	\
0	2.255747	…	1.886690	-1.359293	2.303601	
1	-0.868652	…	1.805927	-0.369203	1.535126	
2	-0.398008	…	1.511870	-0.023974	1.347475	
3	4.910919	…	-0.281464	0.133984	-0.249939	
4	-0.562450	…	1.298575	-1.466770	1.338539	

	area_worst	smoothness_worst	compactness_worst	concavity_worst	\
0	2.001237	1.307686	2.616665	2.109526	
1	1.890489	-0.375612	-0.430444	-0.146749	
2	1.456285	0.527407	1.082932	0.854974	
3	-0.550021	3.394275	3.893397	1.989588	
4	1.220724	0.220556	-0.313395	0.613179	

	concave points_worst	symmetry_worst	fractal_dimension_worst
0	2.296076	2.750622	1.937015
1	1.087084	-0.243890	0.281190
2	1.955000	1.152255	0.201391
3	2.175786	6.046041	4.935010
4	0.729259	-0.868353	-0.397100

[5 rows x 30 columns]

0.7 Exploratory data analysis

A correlation analysis is carried out with a scatter plot and for that we use the pairplot

[85]:
```
cmap = LinearSegmentedColormap.from_list('Custom', colors, N=256)

plt.figure(figsize=(10,10))
```

```
sns.set(style="whitegrid", context="notebook", font_scale=0.8)
sns.heatmap(df_clean.corr(), cmap=cmap, annot=True, annot_kws={"size": 5})
plt.title("Correlation matrix ", fontweight='bold')
plt.show()
```

There is a high positive correlation between the 4 main variables in their three typologies: “mean”, “standard error” and “worst”: - radio - texture - perimeter - area

---

0.8 Model creation

We separate the variables and the labels

[86]:
```
X = df_processing_scaler
y = df_clean["diagnosis"]
y
```

[86]:
```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

Split training and test data

[87]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪stratify= y, random_state=42)

# Print the proportions of the classes in both data sets
print("Proportion of classes in training data", y_train.value_counts() /␣
 ↪len(y_train), "\n", "____________________")
print("Proportion of classes in testing data", y_test.value_counts() /␣
 ↪len(y_test))
```

```
Proportion of classes in training data diagnosis
0    0.626761
1    0.373239
Name: count, dtype: float64
 ____________________
Proportion of classes in testing data diagnosis
0    0.629371
1    0.370629
Name: count, dtype: float64
```

Create and train the model

[88]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
```

[88]:
```
LogisticRegression()
```

Evaluate the accuracy of the model with metrics

- Precision: The proportion of correctly classified instances.
- Sensitivity: The proportion of positive instances correctly classified.
- Specificity: The proportion of negative instances correctly classified.

[89]:
```
# Predict the labels in the test set
y_pred = model.predict(X_test)

def metrics_model (y_test, y_pred):
    precision = metrics.precision_score(y_test, y_pred)
    recall = metrics.recall_score(y_test, y_pred)
    accuracy = metrics.accuracy_score(y_test, y_pred)

    print("Precision:", precision)
    print("Sensitivity:", recall)
    print("Specificity:", accuracy)

metrics_model(y_test, y_pred)
```

```
Precision: 0.98
Sensitivity: 0.9245283018867925
Specificity: 0.965034965034965
```

Evaluate the model with cross validation

[90]:
```
from sklearn.model_selection import cross_val_score

# Perform cross validation
precision = cross_val_score(model, X, y, cv=10, scoring="precision")

# Print the cross validation result
print("Mean precision:", precision.mean())
```

```
Mean precision: 0.986106719367589
```

0.9 Analysis of results

Get the probabilities:

- The probability that it is 0 in the first value, the probability that it is 1 in the second

[91]:
```
model.predict_proba(X_test)[:10]
```

[91]:
```
array([[5.30062727e-04, 9.99469937e-01],
       [9.96298095e-01, 3.70190495e-03],
       [2.74525194e-04, 9.99725475e-01],
       [9.99999979e-01, 2.13614367e-08],
       [9.68296969e-01, 3.17030315e-02],
       [9.96516486e-01, 3.48351422e-03],
```

```
       [9.99640898e-01, 3.59102219e-04],
       [9.98145777e-01, 1.85422341e-03],
       [9.93281821e-01, 6.71817880e-03],
       [9.00835015e-01, 9.91649855e-02]])
```

Get the coefficients

- The importance of each feature
- The positive results are the degrees of importance when it is equal to 1
- Negative results are the degrees of importance when it is equal to 0

[92]:
```
model.coef_
```

[92]:
```
array([[ 0.29267556,  0.57229482,  0.29484731,  0.41203614,  0.37267413,
        -0.48517893,  0.87742907,  0.95075314, -0.24967634, -0.09889822,
         1.23743861, -0.43639047,  0.78135018,  0.88518094,  0.26187264,
        -0.92978178, -0.07921269,  0.40655306, -0.28939838, -0.60083522,
         0.86563781,  1.37735369,  0.72083898,  0.88964341,  0.45481341,
        -0.20747016,  0.86479482,  0.71672175,  1.03556402, -0.01875716]])
```

[93]:
```
model.feature_names_in_
```

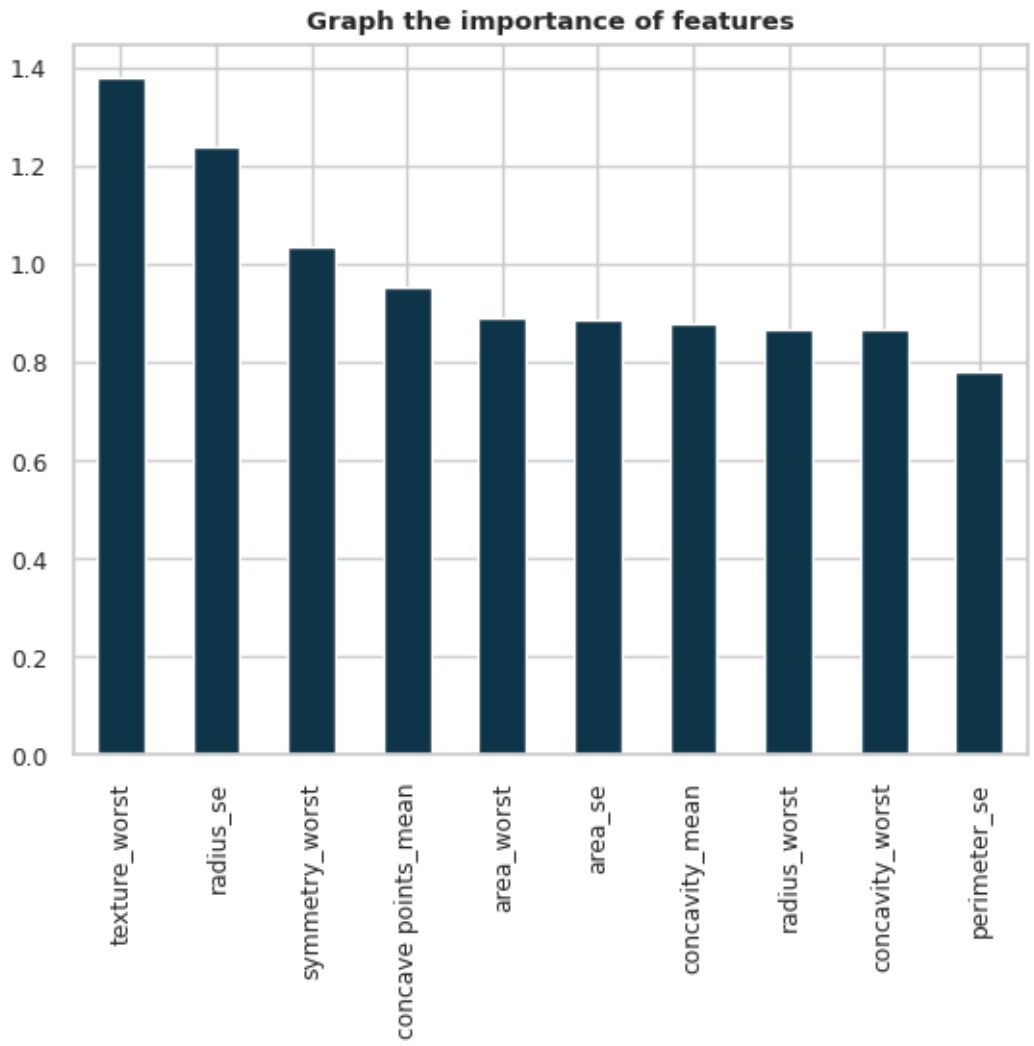
[93]:
```
array(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se',
       'smoothness_se', '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)
```

Graph the importance of features

- We make a ps.Series of the values, with the names of the columns
- We organize the Series from highest to lowest and select the first 10 positive and the first 10 negative

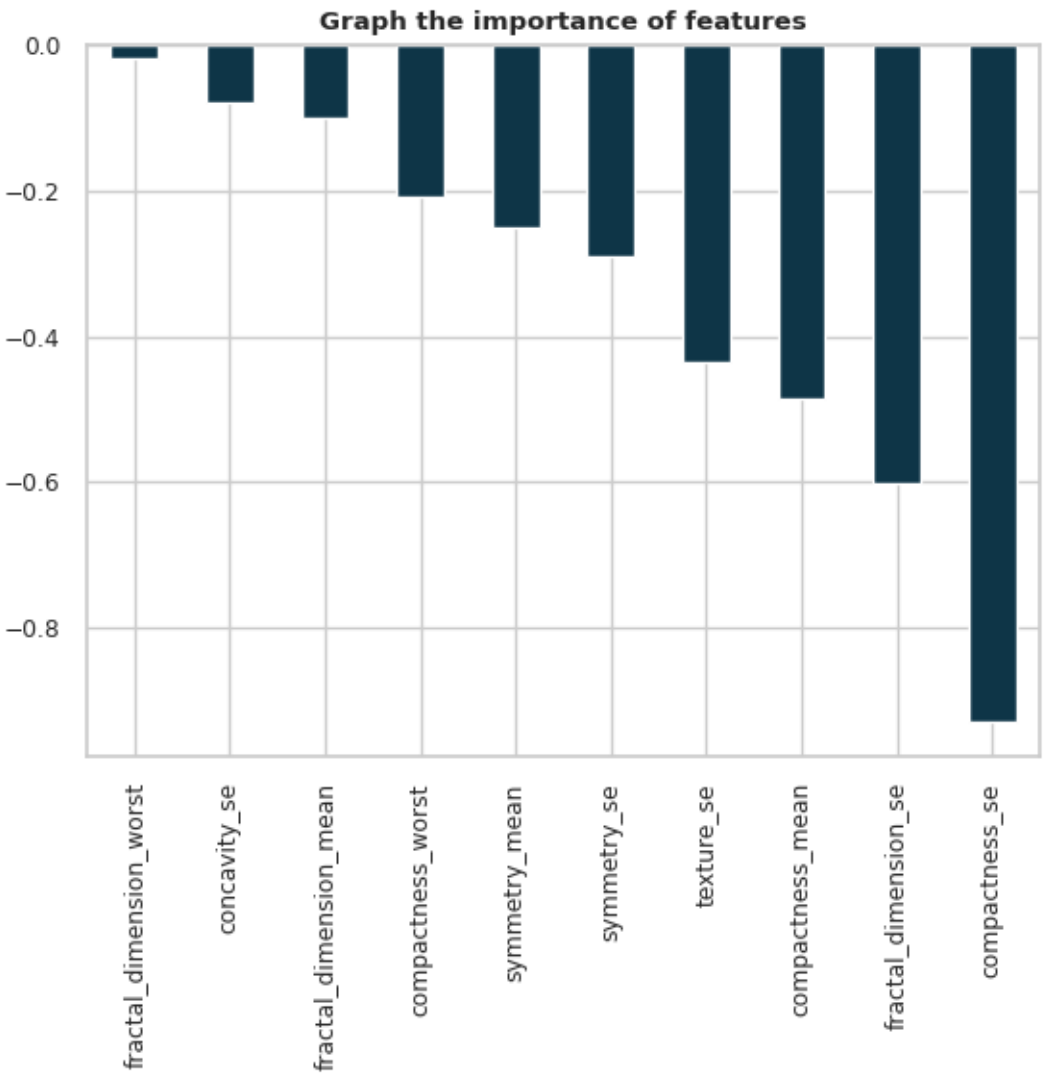
[94]:
```
weights = pd.Series(model.coef_[0], index= X.columns.values)
weights.sort_values(ascending=False)[:10].plot(kind="bar", color=colors[0])
plt.title("Graph the importance of features", fontweight='bold')
```

[94]:
```
Text(0.5, 1.0, 'Graph the importance of features')
```



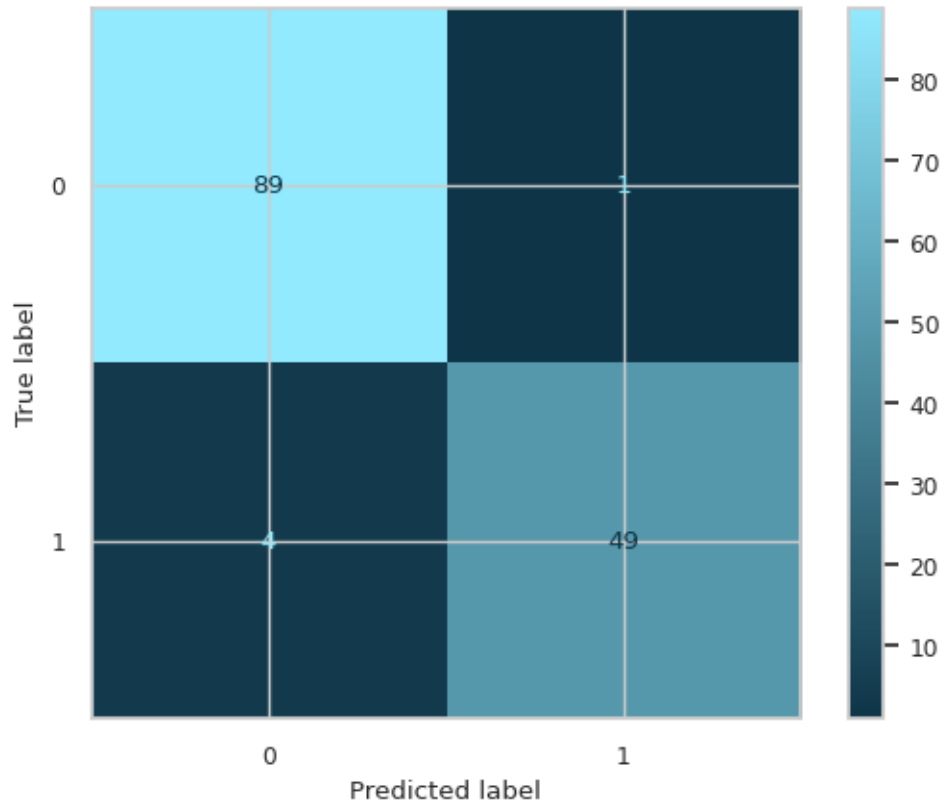
[95]:
```
weights.sort_values(ascending=False)[-10:].plot(kind="bar", color=colors[0])
plt.title("Graph the importance of features", fontweight='bold')
```

[95]: Text(0.5, 1.0, 'Graph the importance of features')



Confusion Matrix

[96]:
```
cm = metrics.confusion_matrix(y_test, y_pred, labels=model.classes_)

disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.
  ↪classes_)
disp.plot(cmap=cmap)
plt.show()
```



0.10 Test

Test eliminating the characteristics with lower coefficients

[97]:
```
columns_test = list(weights[(weights > 0.5) | (weights < -0.5)].index)
columns_test
```

[97]:
```
['texture_mean',
 'concavity_mean',
 'concave points_mean',
 'radius_se',
 'perimeter_se',
 'area_se',
 'compactness_se',
 'fractal_dimension_se',
 'radius_worst',
 'texture_worst',
 'perimeter_worst',
 'area_worst',
 'concavity_worst',
 'concave points_worst',
 'symmetry_worst']
```

[98]:
```
X_t = df_processing_scaler[columns_test]
y_t = df_clean["diagnosis"].values

X_train, X_test, y_train, y_test = train_test_split(X_t, y_t, test_size=0.25,␣
 ↪random_state=42)
model2 = LogisticRegression()
model2.fit(X_train, y_train)

predictions_test = model2.predict(X_test)
print(metrics.accuracy_score(y_test, predictions_test))
```

```
0.9790209790209791
```

Test balancing the number of class records with SMOTE

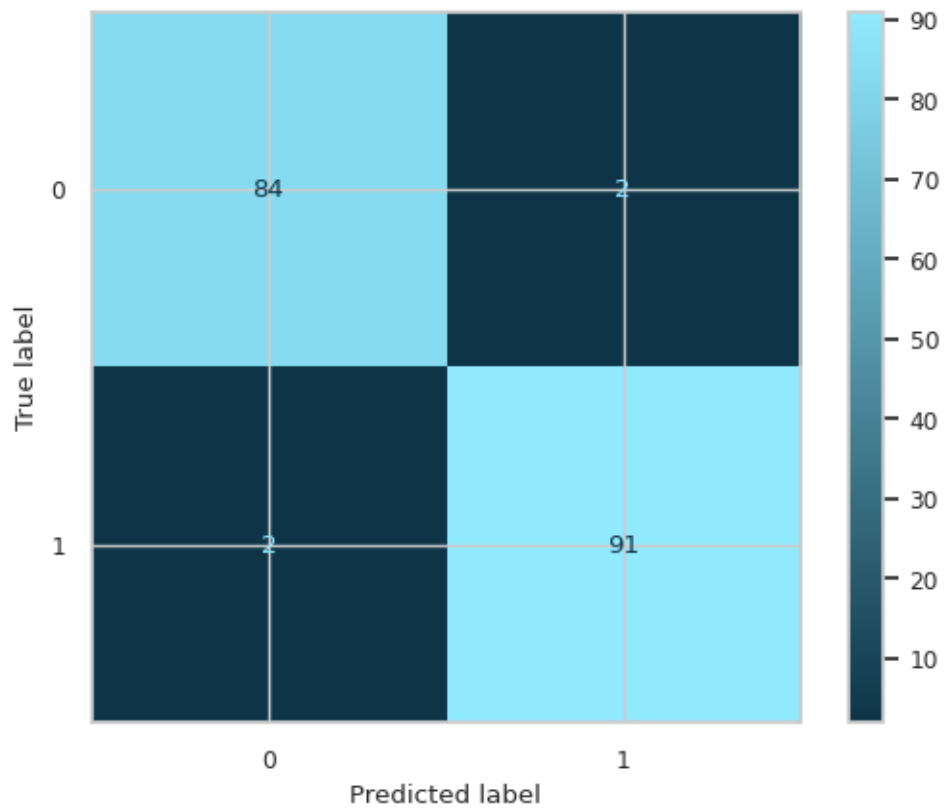
[99]:
```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

y_resampled.value_counts()
```

[99]:
```
diagnosis
1    357
0    357
Name: count, dtype: int64
```

[101]:
```
X_t_balanced = X_resampled
y_t_balanced = y_resampled

X_train, X_test, y_train, y_test = train_test_split(X_t_balanced, y_t_balanced,␣
 ↪test_size=0.25, random_state=42)
model3 = LogisticRegression()
model3.fit(X_train, y_train)

y_pred = model3.predict(X_test)
metrics_model(y_test, y_pred)
```

```
Precision: 0.978494623655914
Sensitivity: 0.978494623655914
Specificity: 0.9776536312849162
```

[102]:
```
cm = metrics.confusion_matrix(y_test, y_pred, labels=model.classes_)

disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.
 ↪classes_)
disp.plot(cmap=cmap)
plt.show()
```



There is no big difference between the model with all the variables, the model with the most relevant variables and the model with balanced classes.

0.11 Regularizers

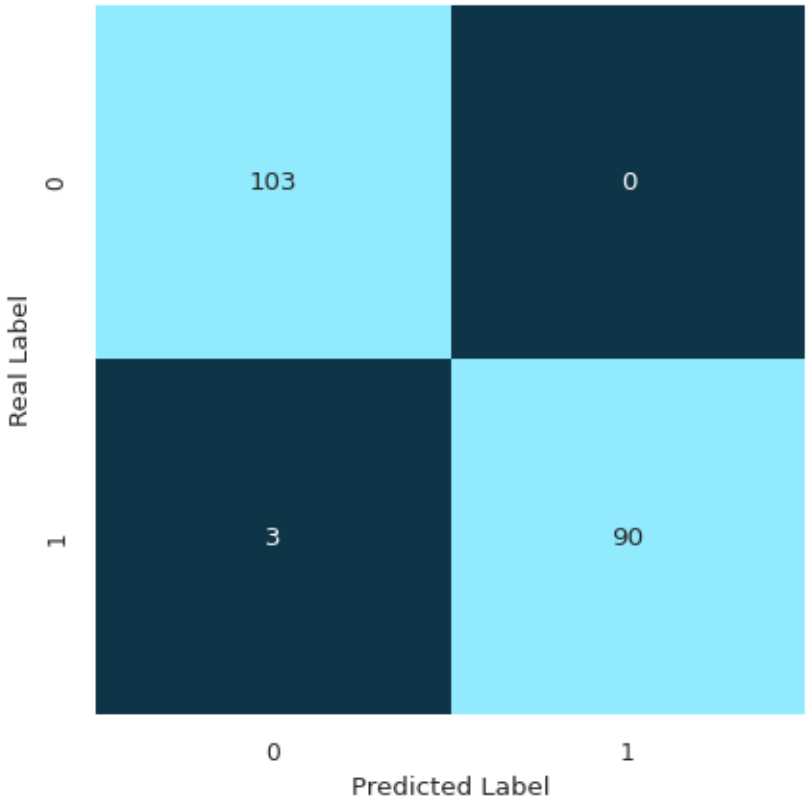
L1 Lasso Reduces complexity by eliminating features that do not contribute much to the model. It penalizes features that provide little information by making them zero, eliminating the noise they produce in the model.

[103]:
```
X = df_processing_scaler
y = df_clean["diagnosis"]

smote_enn = SMOTEENN(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote_enn.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
↪test_size=0.3, random_state=42)

lasso = LogisticRegression(max_iter=10000, penalty="l1", solver="saga", C = 0.5)
lasso.fit(X_train, y_train)
```

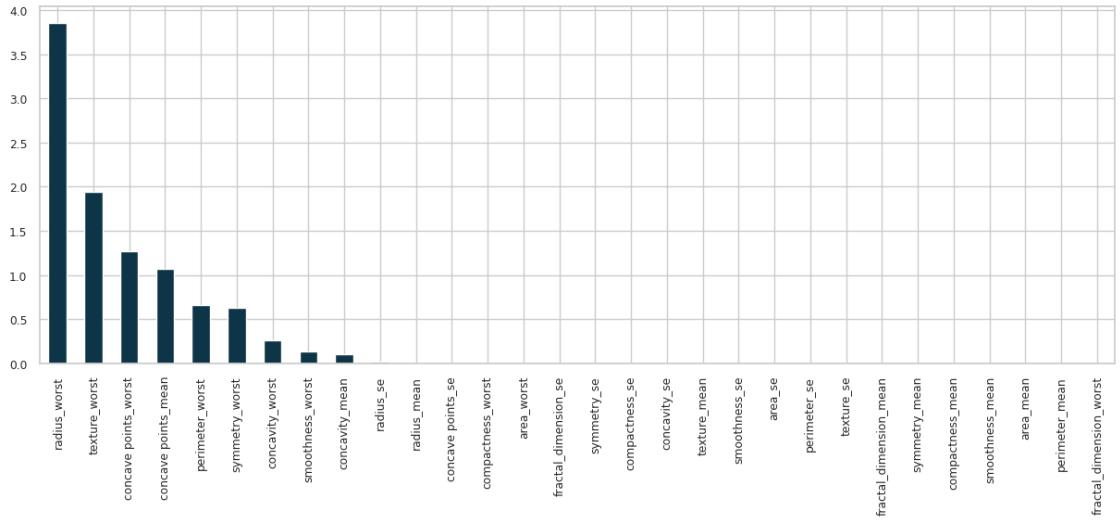
```
lasso.score(X_test, y_test)
```

[103]: 0.9846938775510204

[104]:
```
cm=metrics.confusion_matrix(lasso.predict(X_test),y_test)
sns.heatmap(
    cm,
    annot=True,
    cmap=colors,
    cbar=False,
    square=True,
    fmt="d"
    )
plt.ylabel('Real Label')
plt.xlabel('Predicted Label');
```



[105]:
```
weights = pd.Series(lasso.coef_[0], index=X.columns.values).
  ↪sort_values(ascending=False)
fig = plt.figure(figsize=(15,5))
weights.plot(kind='bar', color=colors[0]);
```

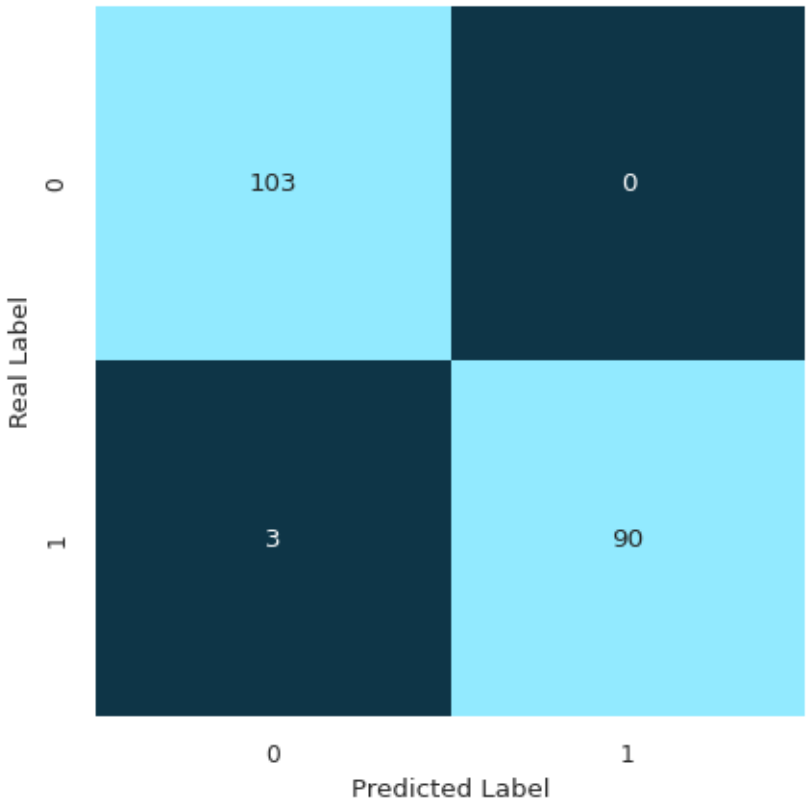


L2 Ridge Reduces complexity by decreasing the impact of certain features of our model. It penalizes irrelevant traits, but does not make them zero. It only limits the information they provide to our model.

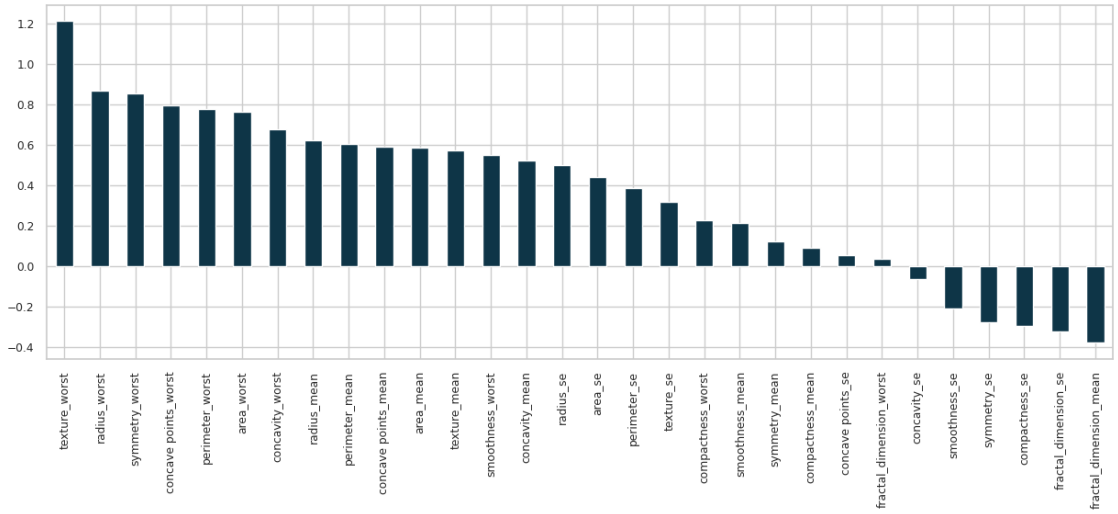
[106]:
```
ridge = LogisticRegression(max_iter=10000, penalty="l2", solver="saga", C = 0.5)
ridge.fit(X_train, y_train)
ridge.score(X_test, y_test)
```

[106]: 0.9846938775510204

[107]:
```
cm=metrics.confusion_matrix(ridge.predict(X_test),y_test)
sns.heatmap(
    cm,
    annot=True,
    cmap=colors,
    cbar=False,
    square=True,
    fmt="d"
    )
plt.ylabel('Real Label')
plt.xlabel('Predicted Label');
```



[108]:
```
weights = pd.Series(ridge.coef_[0], index=X.columns.values).
  ↪sort_values(ascending=False)
fig = plt.figure(figsize=(15,5))
weights.plot(kind='bar', color=colors[0]);
```



0.11.1 *Conclusions*

- The variables of the original data set have correlations necessary to generate a binary logistic regression model efficient enough not to use other optimization tools.
- When variable reduction tests or regularizers were used, no better results were obtained than the initial model.
- The results of the models were as follows:
  - The model with all the original variables without modification in their weights obtained an accuracy of: 0.9861
  - The model eliminating the characteristics with lower coefficients obtained an accuracy of: 0.9790
  - The model balancing the number of class records with SMOTE obtained an accuracy of: 0.9784
  - The model created using the L1 Lasso regularizer obtained an accuracy of: 0.9846
  - The model created using the L2 Ridge regularizer obtained an accuracy of: 0.9846