Apply logistic regression for the given data set and analyse the performance of the algorithm

Data Set: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

Predicting Breast Cancer - Logistic Regression

Loading dataset:

```
# import dependencies
# data cleaning and manipulation
import pandas as pd
import numpy as np
# data visualization
import matplotlib.pyplot as plt
import seaborn as sns
# machine learning
from sklearn.preprocessing import StandardScaler
import sklearn.linear_model as skl_lm
from sklearn import preprocessing
from sklearn import neighbors
from sklearn.metrics import confusion_matrix, classification_report, precision_score
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
import statsmodels.formula.api as smf

# initialize some package settings
sns.set(style="whitegrid", color_codes=True, font_scale=1.3)
#%matplotlib inline
# read in the data and check the first 5 rows
df = pd.read_csv('./breastcancer.csv', index_col=0)
print(df.head())

# general summary of the dataframe
print(df.info())
```

```
id                                  ...
842302        M        17.99        ...            0.11890        NaN
842517        M        20.57        ...            0.08902        NaN
84300903      M        19.69        ...            0.08758        NaN
84348301      M        11.42        ...            0.17300        NaN
84358402      M        20.29        ...            0.07678        NaN
```

```
Data columns (total 32 columns):
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area_mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area_se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area_worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
Unnamed: 32                0 non-null float64
dtypes: float64(31), object(1)
memory usage: 146.7+ KB
None
```

Removing missing value column:

```
# remove the 'Unnamed: 32' column
df = df.drop('Unnamed: 32', axis=1)
# check the data type of each column
print(df.dtypes)
```

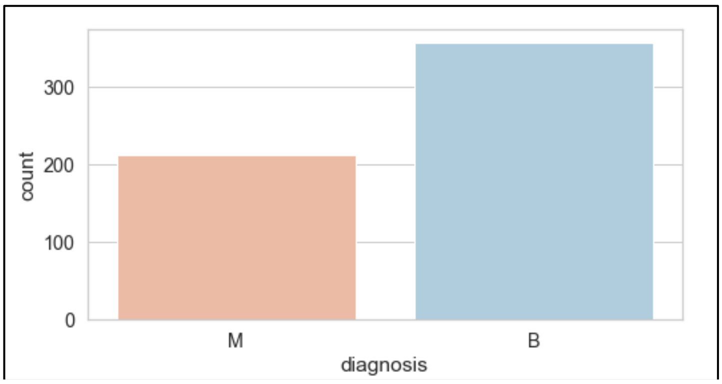
```
diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
```

Distribution of response variable classes:

```
# visualize distribution of classes
plt.figure(figsize=(8, 4))
sns.countplot(df['diagnosis'], palette='RdBu')
# count number of obvs in each class
benign, malignant = df['diagnosis'].value_counts()
print('Number of cells labeled Benign: ', benign)
print('Number of cells labeled Malignant : ', malignant)
print('')
print('% of cells labeled Benign', round(benign / len(df) * 100, 2), '%')
print('% of cells labeled Malignant', round(malignant / len(df) * 100, 2), '%')
```

```
Number of cells labeled Benign:  357
Number of cells labeled Malignant :  212

% of cells labeled Benign 62.74 %
% of cells labeled Malignant 37.26 %
```



Drop all unnecessary columns:

```
# first, drop all "worst" columns
cols = ['radius_worst',
        'texture_worst',
        'perimeter_worst',
        'area_worst',
        'smoothness_worst',
        'compactness_worst',
        'concavity_worst',
        'concave points_worst',
        'symmetry_worst',
        'fractal_dimension_worst']
df = df.drop(cols, axis=1)
# then, drop all columns related to the "perimeter" and "area" attributes
cols = ['perimeter_mean',
        'perimeter_se',
        'area_mean',
        'area_se']
df = df.drop(cols, axis=1)
# lastly, drop all columns related to the "concavity" and "concave points" attributes
cols = ['concavity_mean',
        'concavity_se',
        'concave points_mean',
        'concave points_se']
df = df.drop(cols, axis=1)
# verify remaining columns
print(df.columns)
```

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'smoothness_mean',
       'compactness_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'smoothness_se', 'compactness_se',
       'symmetry_se', 'fractal_dimension_se'],
      dtype='object')
```

Splitting data set to reduce overfitting:

```
# Split the data into training and testing sets
X = df
y = df['diagnosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=40)
```

Formula to be used for the logistic regression:

```
# Create a string for the formula
cols = df.columns.drop('diagnosis')
formula = 'diagnosis ~ ' + ' + '.join(cols)
print(formula, '\n')
```

```
diagnosis ~ radius_mean + texture_mean + smoothness_mean + compactness_mean + symmetry_mean +
fractal_dimension_mean + radius_se + texture_se + smoothness_se + compactness_se + symmetry_se +
fractal_dimension_se
```

```
# Run the model and report the results
model = smf.glm(formula=formula, data=X_train, family=sm.families.Binomial())
logistic_fit = model.fit()
print(logistic_fit.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================================
Dep. Variable:     ['diagnosis[B]', 'diagnosis[M]']   No. Observations:                398
Model:                                      GLM   Df Residuals:                    385
Model Family:                          Binomial   Df Model:                         12
Link Function:                            logit   Scale:                        1.0000
Method:                                    IRLS   Log-Likelihood:              -55.340
Date:                          Sun, 02 Feb 2020   Deviance:                     110.68
Time:                                  17:06:33   Pearson chi2:                   125.
No. Iterations:                               9
Covariance Type:                      nonrobust
============================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------
Intercept                 44.5427     11.787      3.779      0.000      21.441      67.644
radius_mean               -1.1610      0.301     -3.862      0.000      -1.750      -0.572
texture_mean              -0.4237      0.087     -4.866      0.000      -0.594      -0.253
smoothness_mean          -85.3981     40.976     -2.084      0.037    -165.709      -5.088
compactness_mean         -16.7104     22.510     -0.742      0.458     -60.829      27.408
symmetry_mean            -46.2721     17.767     -2.604      0.009     -81.095     -11.449
fractal_dimension_mean   -49.1536    121.888     -0.403      0.687    -288.050     189.742
radius_se                 -7.1916      2.806     -2.563      0.010     -12.691      -1.692
texture_se                 0.1849      0.784      0.236      0.814      -1.353       1.722
smoothness_se            163.6068    159.702      1.024      0.306    -149.403     476.616
compactness_se           -31.1808     42.772     -0.729      0.466    -115.012      52.650
symmetry_se               74.7366     51.458      1.452      0.146     -26.119     175.592
fractal_dimension_se     824.1245    412.040      2.000      0.045      16.541    1631.708
============================================================================================
```

Prediction:

```
# predict the test data and show the first 5 predictions
predictions = logistic_fit.predict(X_test)
print(predictions[1:6])
```

```
id
848406      0.324251
907915      0.996906
911201      0.964710
84799002    0.000544
8911164     0.838719
dtype: float64
```

```
# Note how the values are numerical.
# Convert these probabilities into nominal values and check the first 5 predictions
again.
predictions_nominal = [ "M" if x < 0.5 else "B" for x in predictions]
print(predictions_nominal[1:6])
```

```
['M', 'B', 'B', 'M', 'B']
```

We can confirm that probabilities closer to 0 have been labeled as "M", while the ones closer to 1 have been labelled as "B". Now we are able to evaluate the accuracy of our predictions by checking out the classification report and the confusion matrix.

```
print(classification_report(y_test, predictions_nominal, digits=3))
cfm = confusion_matrix(y_test, predictions_nominal)

true_negative = cfm[0][0]
false_positive = cfm[0][1]
false_negative = cfm[1][0]
true_positive = cfm[1][1]

print('Confusion Matrix: \n', cfm, '\n')

print('True Negative:', true_negative)
print('False Positive:', false_positive)
print('False Negative:', false_negative)
print('True Positive:', true_positive)
print('Correct Predictions',
      round((true_negative + true_positive) / len(predictions_nominal) * 100, 1), '%')
```

```
              precision    recall  f1-score   support

           B      0.982     0.965     0.974       115
           M      0.931     0.964     0.947        56

    accuracy                          0.965       171
   macro avg      0.957     0.965     0.961       171
weighted avg      0.966     0.965     0.965       171

Confusion Matrix:
 [[111   4]
 [  2  54]]

True Negative: 111
False Positive: 4
False Negative: 2
True Positive: 54
Correct Predictions 96.5 %
```

The model has accurately labelled 96.5% of the test data