Diabetes_Logistic_Regression

May 17, 2023

[ ]:
```
#Let's start with importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
#read the data file
data = pd.read_csv("/config/workspace/Dataset/diabetes.csv")
data.head()
```

[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[ ]:
```
data.describe()
```

[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479
std	3.369578	31.972618	19.355807	15.952218	115.244002
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000
75%	6.000000	140.250000	80.000000	32.000000	127.250000

```
max      17.000000  199.000000     122.000000      99.000000  846.000000

               BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[ ]:
```
data.isnull().sum()
```

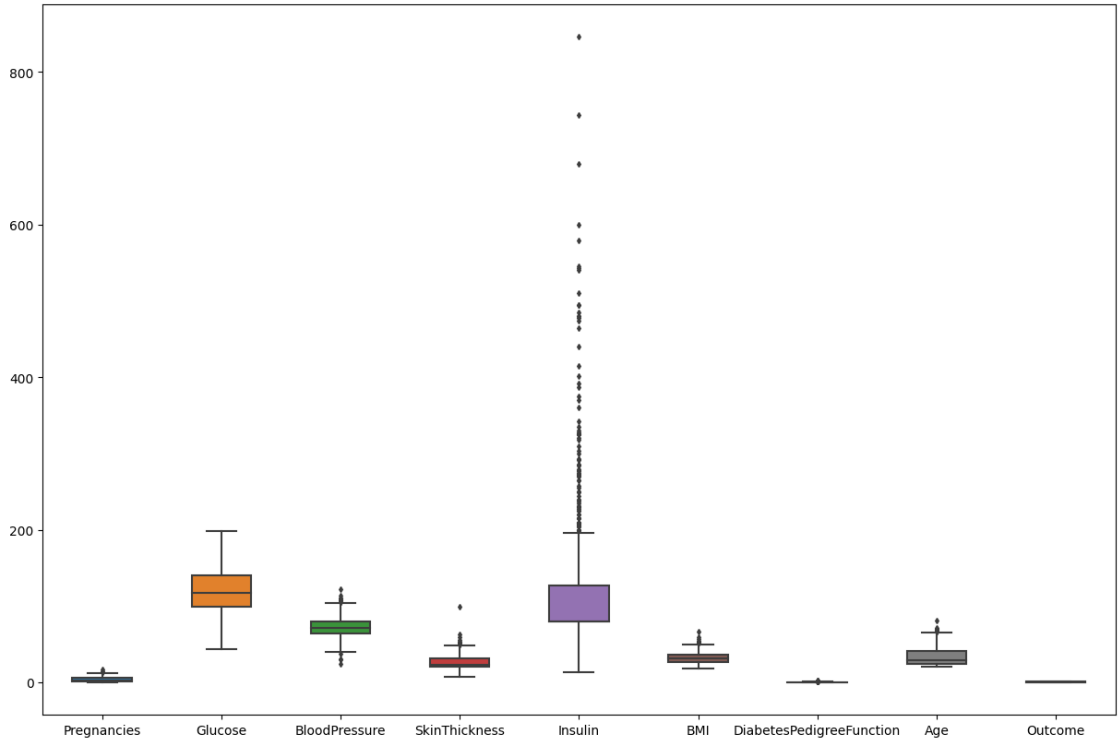
[ ]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

We can see there few data for columns Glucose , Insulin, skin thickenss, BMI and Blood Pressure which have value as 0. That's not possible,right? you can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

[ ]:
```
#here few misconception is there like BMI can not be zero, BP can't be zero,␣
 ↪glucose, insuline can't be zero so lets try to fix it
# now replacing zero values with the mean of the column
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].
 ↪mean())
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].
 ↪mean())
```

[ ]:
```
#now we have dealt with the 0 values and data looks better. But, there still␣
 ↪are outliers present in some columns.lets visualize it
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=data, width= 0.5,ax=ax,  fliersize=3)
```

[ ]:
```
<Axes: >
```



[ ]:
```
data.head()
```

[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	85.0	66.0	29.000000	79.799479	26.6	
2	8	183.0	64.0	20.536458	79.799479	23.3	
3	1	89.0	66.0	23.000000	94.000000	28.1	
4	0	137.0	40.0	35.000000	168.000000	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

[ ]:
```
#segregate the dependent and independent variable
X = data.drop(columns = ['Outcome'])
y = data['Outcome']
```

[ ]:
```
# separate dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
↪25,random_state=0)
```

```
X_train.shape, X_test.shape
```

[ ]:
```
((576, 8), (192, 8))
```

[ ]:
```
import pickle
##standard Scaling- Standardization
def scaler_standard(X_train, X_test):
    #scaling the data
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    #saving the model
    file = open('/config/workspace/Model/standardScalar.pkl','wb')
    pickle.dump(scaler,file)
    file.close()

    return X_train_scaled, X_test_scaled
```

[ ]:
```
X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
```

[ ]:
```
X_train_scaled
```

[ ]:
```
array([[ 1.50755225, -1.09947934, -0.89942504, …, -1.45561965,
        -0.98325882, -0.04863985],
       [-0.82986389, -0.1331471 , -1.23618124, …,  0.09272955,
        -0.62493647, -0.88246592],
       [-1.12204091, -1.03283573,  0.61597784, …, -0.03629955,
         0.39884168, -0.5489355 ],
       …,
       [ 0.04666716, -0.93287033, -0.64685789, …, -1.14021518,
        -0.96519215, -1.04923114],
       [ 2.09190629, -1.23276654,  0.11084355, …, -0.36604058,
        -0.5075031 ,  0.11812536],
       [ 0.33884418,  0.46664532,  0.78435594, …, -0.09470985,
         0.51627505,  2.953134  ]])
```

[ ]:
```
log_reg = LogisticRegression()

log_reg.fit(X_train_scaled,y_train)
```

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

[ ]:
```
## Hyperparameter Tuning
## GridSearch CV
from sklearn.model_selection import GridSearchCV
import numpy as np
import warnings
```

```
warnings.filterwarnings('ignore')
# parameter grid
parameters = {
    'penalty' : ['l1','l2'],
    'C'       : np.logspace(-3,3,7),
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
}
```

[ ]:
```
logreg = LogisticRegression()
clf = GridSearchCV(logreg,                    # model
                   param_grid = parameters,   # hyperparameters
                   scoring='accuracy',        # metric for scoring
                   cv=10)                     # number of folds

clf.fit(X_train_scaled,y_train)
```

[ ]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01,
1.e+02, 1.e+03]),
                         'penalty': ['l1', 'l2'],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear']},
             scoring='accuracy')
```

[ ]:
```
clf.best_params_
```

[ ]:
```
{'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
```

[ ]:
```
clf.best_score_
```

[ ]:
```
0.763793103448276
```

let's see how well our model performs on the test data set.

[ ]:
```
y_pred = clf.predict(X_test_scaled)
```

accuracy = accuracy_score(y_test,y_pred) accuracy

[ ]:
```
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

[ ]:
```
array([[117,  13],
       [ 26,  36]])
```

[ ]:
```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

[ ]:
```
Accuracy = (true_positive + true_negative) / (true_positive +false_positive +␣
↪false_negative + true_negative)
Accuracy
```

[ ]:
```
0.796875
```

[ ]:
```
Precision = true_positive/(true_positive+false_positive)
Precision
```

[ ]:
```
0.9
```

[ ]:
```
Recall = true_positive/(true_positive+false_negative)
Recall
```

[ ]:
```
0.8181818181818182
```

[ ]:
```
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

[ ]:
```
0.8571428571428572
```

[ ]:
```
import pickle
file = open('/config/workspace/Model/modelForPrediction.pkl','wb')
pickle.dump(log_reg,file)
file.close()
```

[ ]: