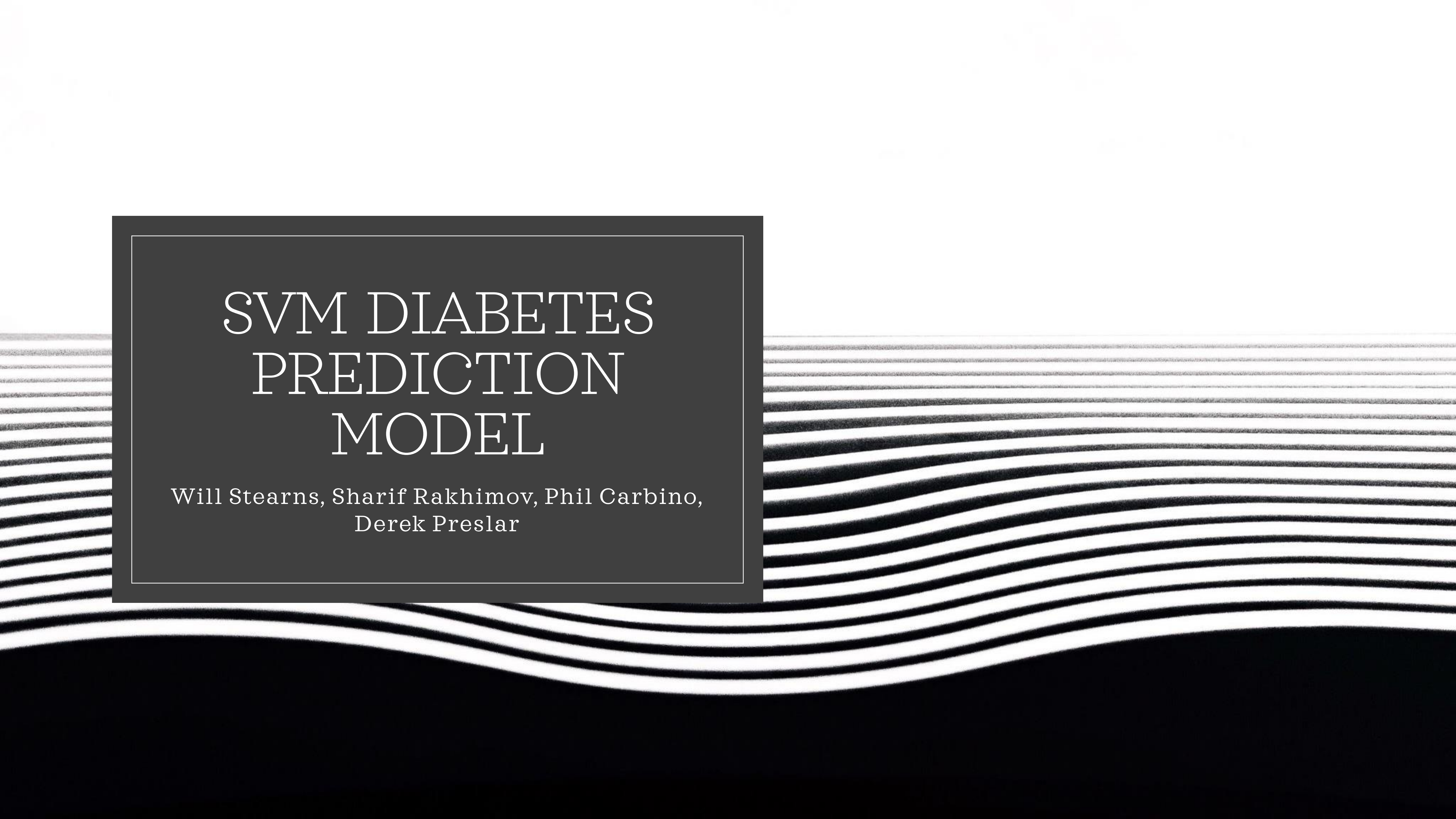


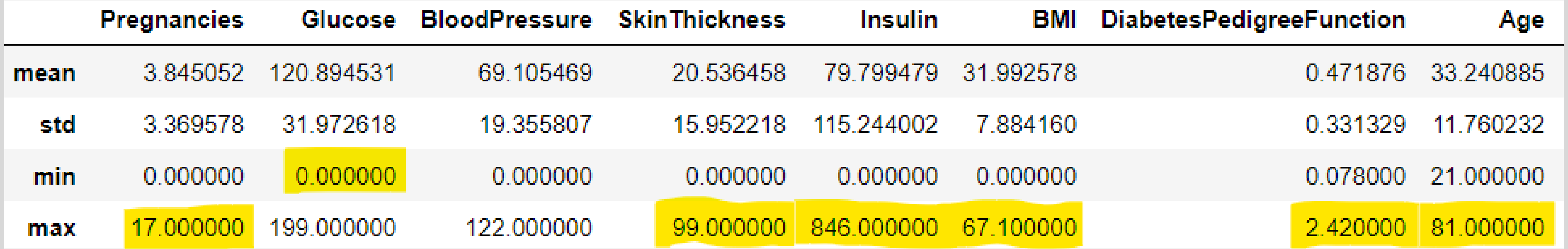
SVM DIABETES PREDICTION MODEL

Will Stearns, Sharif Rakhimov, Phil Carbino, Derek Preslar

Data Set: Diabetes Data

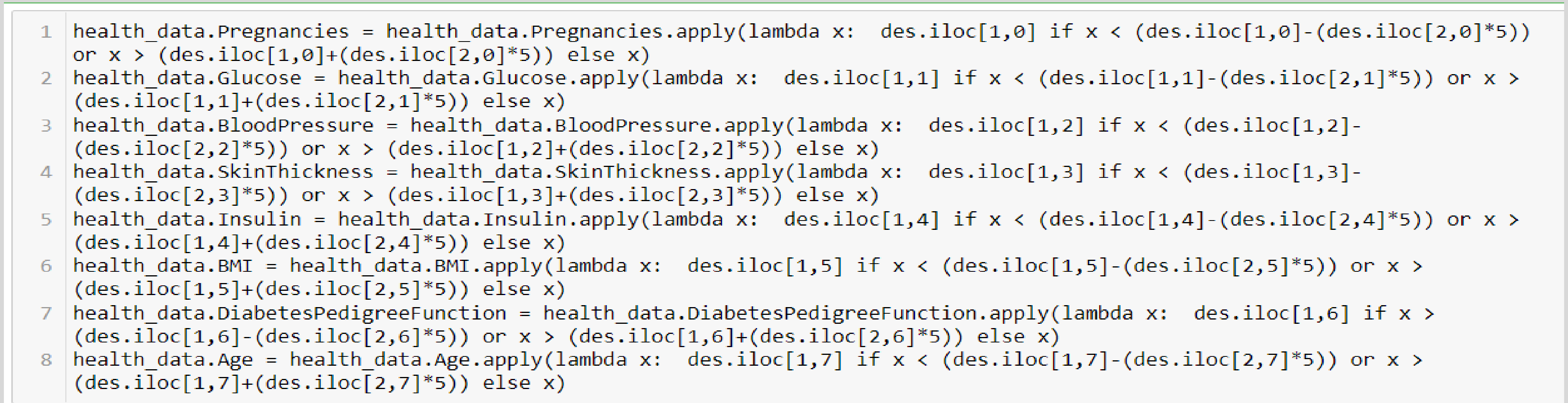
- The data gathered diagnostic measurements for diabetes by the National Institute of Diabetes and Digestive and Kidney Diseases
  - The data is kept in a csv format
  - All patients were female, at least 21 years old, and of Pima Indian heritage
- Goal is to create a Support Vector Machine model to predict whether a patient has diabetes based on diagnostic criteria

Descriptive statistics and potential outliers:

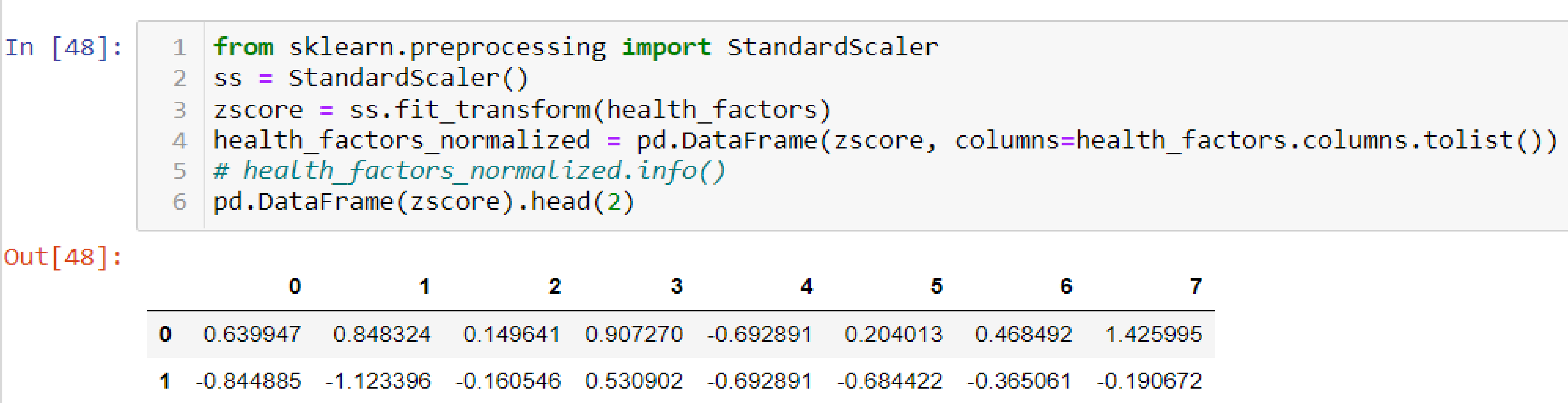


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

Data Normalization and Outlier mitigation codes



```python
health_data.Pregnancies = health_data.Pregnancies.apply(lambda x:  des.iloc[1,0] if x < (des.iloc[1,0]-(des.iloc[2,0]*5)) or x > (des.iloc[1,0]+(des.iloc[2,0]*5)) else x)
health_data.Glucose = health_data.Glucose.apply(lambda x:  des.iloc[1,1] if x < (des.iloc[1,1]-(des.iloc[2,1]*5)) or x > (des.iloc[1,1]+(des.iloc[2,1]*5)) else x)
health_data.BloodPressure = health_data.BloodPressure.apply(lambda x:  des.iloc[1,2] if x < (des.iloc[1,2]-(des.iloc[2,2]*5)) or x > (des.iloc[1,2]+(des.iloc[2,2]*5)) else x)
health_data.SkinThickness = health_data.SkinThickness.apply(lambda x:  des.iloc[1,3] if x < (des.iloc[1,3]-(des.iloc[2,3]*5)) or x > (des.iloc[1,3]+(des.iloc[2,3]*5)) else x)
health_data.Insulin = health_data.Insulin.apply(lambda x:  des.iloc[1,4] if x < (des.iloc[1,4]-(des.iloc[2,4]*5)) or x > (des.iloc[1,4]+(des.iloc[2,4]*5)) else x)
health_data.BMI = health_data.BMI.apply(lambda x:  des.iloc[1,5] if x < (des.iloc[1,5]-(des.iloc[2,5]*5)) or x > (des.iloc[1,5]+(des.iloc[2,5]*5)) else x)
health_data.DiabetesPedigreeFunction = health_data.DiabetesPedigreeFunction.apply(lambda x:  des.iloc[1,6] if x > (des.iloc[1,6]-(des.iloc[2,6]*5)) or x > (des.iloc[1,6]+(des.iloc[2,6]*5)) else x)
health_data.Age = health_data.Age.apply(lambda x:  des.iloc[1,7] if x < (des.iloc[1,7]-(des.iloc[2,7]*5)) or x > (des.iloc[1,7]+(des.iloc[2,7]*5)) else x)
```



In [48]:

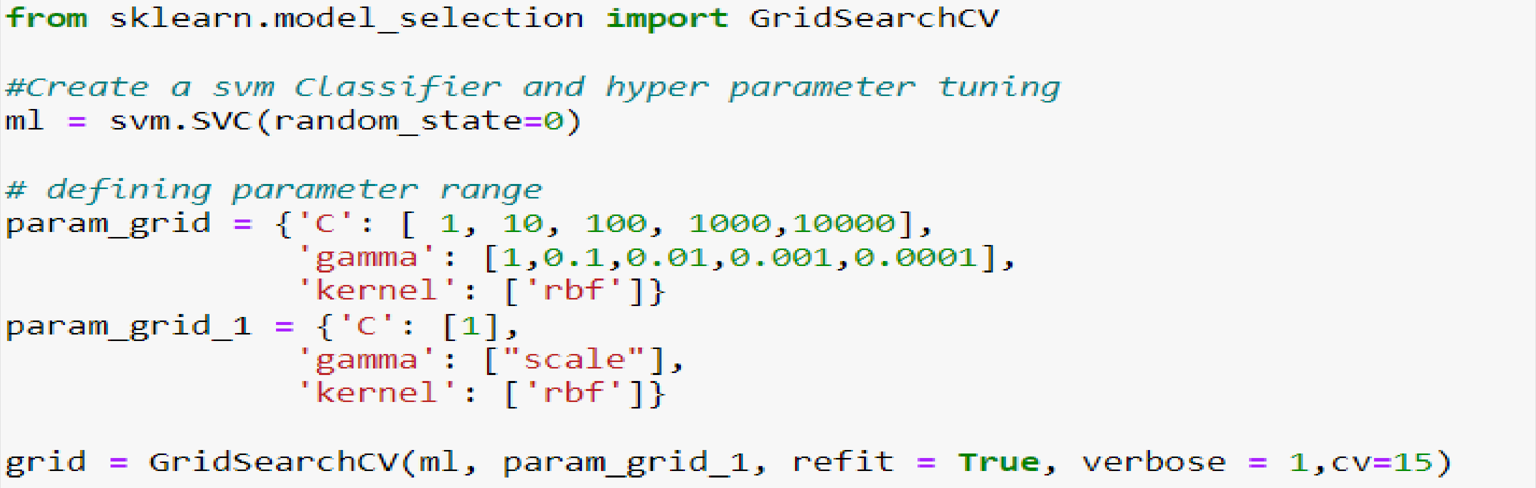
```python
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
zscore = ss.fit_transform(health_factors)
health_factors_normalized = pd.DataFrame(zscore, columns=health_factors.columns.tolist())
# health_factors_normalized.info()
pd.DataFrame(zscore).head(2)
```

Out[48]:

```
           0          1          2         3          4          5          6          7
0   0.639947   0.848324   0.149641  0.907270  -0.692891   0.204013   0.468492   1.425995
1  -0.844885  -1.123396  -0.160546  0.530902  -0.692891  -0.684422  -0.365061  -0.190672
```

Feature Engineering and Hyperparameters

- Preparing the dataset for model training:
  - All features were considered necessary based on descriptive statistics report analysis
  - Dataset was split into target (Outcome) and training (All the rest) features to feed the model
  - To ensure the best score, we selected 2 different sets of hyperparameter values using GridSearchCV along with 2 folds for cross validation on each set to train our model



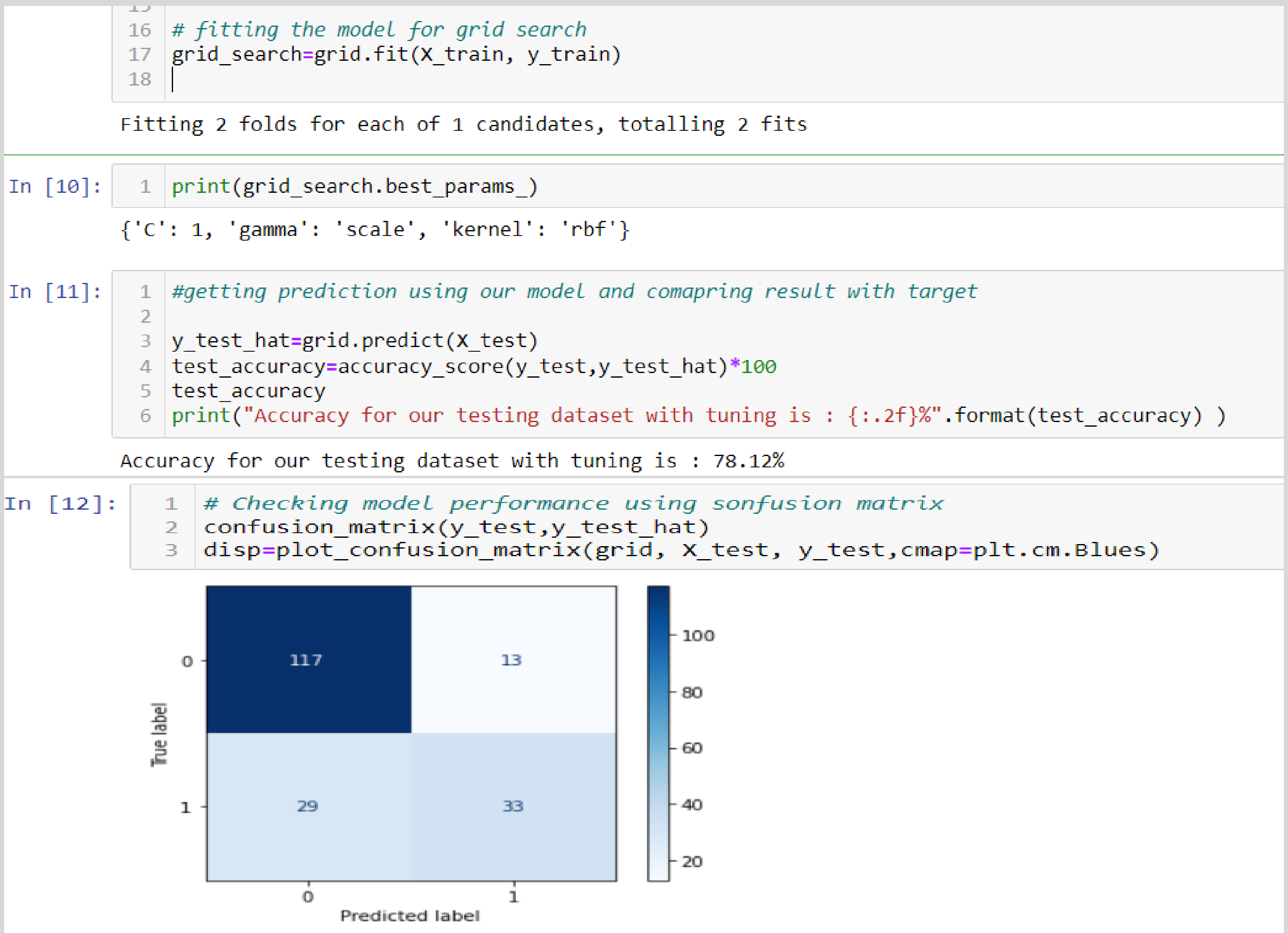
```
from sklearn.model_selection import GridSearchCV

#Create a svm Classifier and hyper parameter tuning
ml = svm.SVC(random_state=0)

# defining parameter range
param_grid = {'C': [ 1, 10, 100, 1000,10000],
              'gamma': [1,0.1,0.01,0.001,0.0001],
              'kernel': ['rbf']}
param_grid_1 = {'C': [1],
              'gamma': ["scale"],
              'kernel': ['rbf']}

grid = GridSearchCV(ml, param_grid_1, refit = True, verbose = 1,cv=15)
```

Default Hyperparameters



```
# fitting the model for grid search
grid_search=grid.fit(X_train, y_train)

```

```
Fitting 2 folds for each of 1 candidates, totalling 2 fits
```

```
In [10]: print(grid_search.best_params_)
```

```
{'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
```

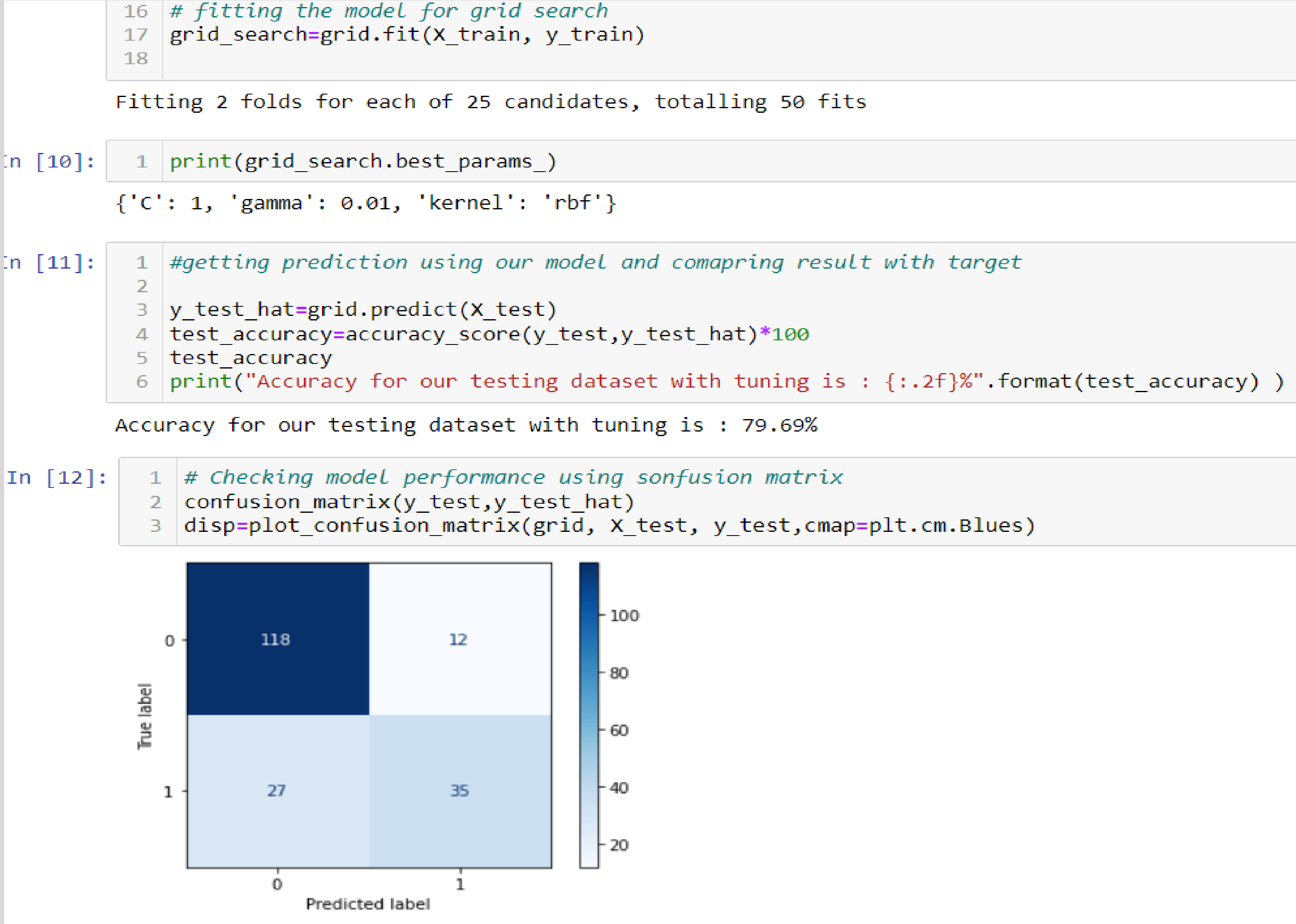
```
In [11]: #getting prediction using our model and comapring result with target

         y_test_hat=grid.predict(X_test)
         test_accuracy=accuracy_score(y_test,y_test_hat)*100
         test_accuracy
         print("Accuracy for our testing dataset with tuning is : {:.2f}%".format(test_accuracy) )
```

```
Accuracy for our testing dataset with tuning is : 78.12%
```

```
In [12]: # Checking model performance using sonfusion matrix
         confusion_matrix(y_test,y_test_hat)
         disp=plot_confusion_matrix(grid, X_test, y_test,cmap=plt.cm.Blues)
```

Tuning Hyperparameters



```
# fitting the model for grid search
grid_search=grid.fit(X_train, y_train)

```

```
Fitting 2 folds for each of 25 candidates, totalling 50 fits
```

```
In [10]: print(grid_search.best_params_)
```

```
{'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
```

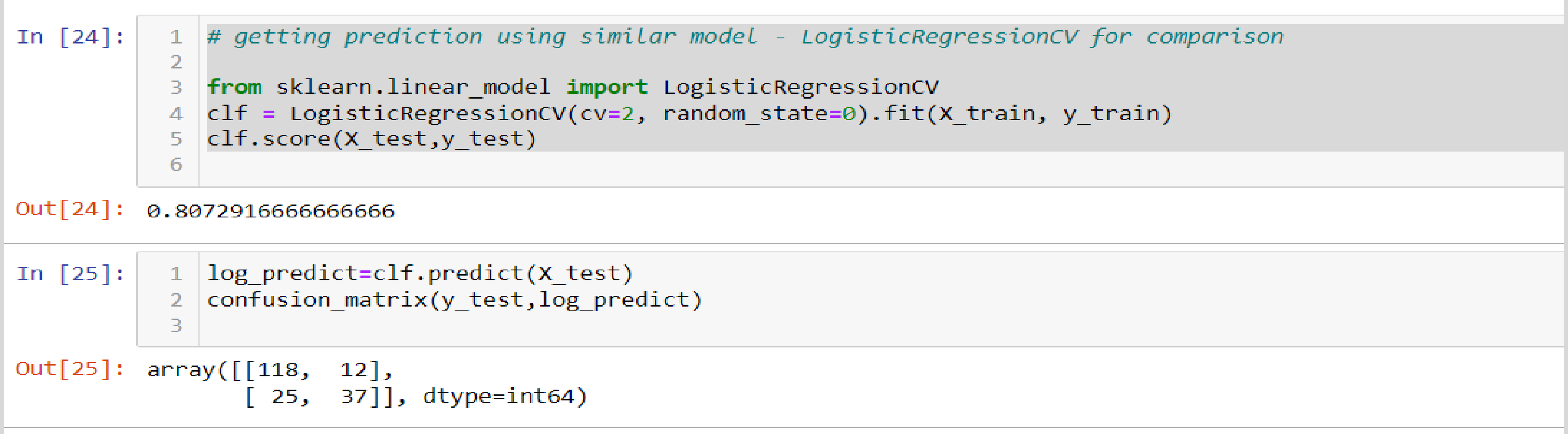
```
In [11]: #getting prediction using our model and comapring result with target

         y_test_hat=grid.predict(X_test)
         test_accuracy=accuracy_score(y_test,y_test_hat)*100
         test_accuracy
         print("Accuracy for our testing dataset with tuning is : {:.2f}%".format(test_accuracy) )
```

```
Accuracy for our testing dataset with tuning is : 79.69%
```

```
In [12]: # Checking model performance using sonfusion matrix
         confusion_matrix(y_test,y_test_hat)
         disp=plot_confusion_matrix(grid, X_test, y_test,cmap=plt.cm.Blues)
```

Comparing performance against a Logistic Regression Model



In [24]:
```python
# getting prediction using similar model - LogisticRegressionCV for comparison

from sklearn.linear_model import LogisticRegressionCV
clf = LogisticRegressionCV(cv=2, random_state=0).fit(X_train, y_train)
clf.score(X_test,y_test)

```

Out[24]:
```
0.8072916666666666
```

In [25]:
```python
log_predict=clf.predict(X_test)
confusion_matrix(y_test,log_predict)

```

Out[25]:
```
array([[118,  12],
       [ 25,  37]], dtype=int64)
```

- The SVM model achieved 78.12% accuracy on default hyperparameters and 79.69% after tuning hyperparameters
- The Logistic Regression model achieved 80.7% on default hyperparameters
- MCC for SVM = 0.51
- MCC for Logistic Regression = 0.54