```
# Murilo Holtz Foltran, 133770

# implementaçao de SVG e MLP utilizando bibliotecas do sklearn
# partes do codigo foram baseados em
# conceitos encontrados em https://www.python-course.eu

# bibliotecas para criar modelo de machine learning:
from sklearn import preprocessing, model_selection, neighbors
from sklearn.preprocessing import StandardScaler

# plot para os graficos (Apenas visualizaçao)
from mlxtend.plotting import plot_decision_regions
```

```
df = pd.read_csv('Iris.csv')
```

```
df.head()
```

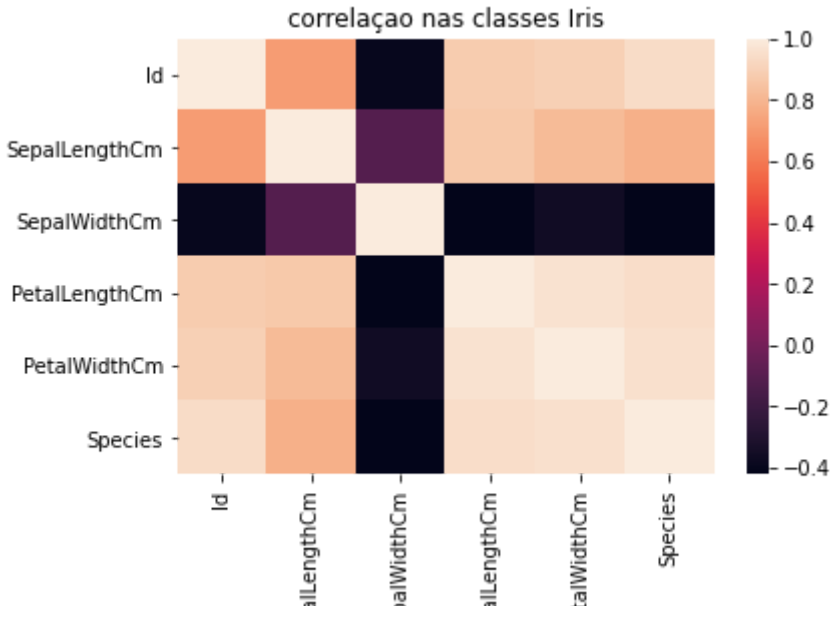
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
df = df.replace({"Species":  {"Iris-setosa":1,"Iris-versicolor":2, "Iris-virginica":3}})
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	1
1	2	4.9	3.0	1.4	0.2	1
2	3	4.7	3.2	1.3	0.2	1
3	4	4.6	3.1	1.5	0.2	1
4	5	5.0	3.6	1.4	0.2	1

```
plt.figure(1)
sns.heatmap(df.corr())
plt.title('correlaçao nas classes Iris')
```

```
Text(0.5, 1.0, 'correlaçao nas classes Iris')
```




```
X = df.iloc[:,:-1]
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
# separando 30% para teste e 70% para treino
```

SVM

```
#modelo SVM

from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)

#atributo fit: ajusta o modelo para os dados
classifier.fit(X_train, y_train)

#faz a previsao:
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
# 'matriz de confusao' para calcular a acurácia
cm = confusion_matrix(y_test, y_pred)
print(cm)

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("acurácia SVM: {:.2f} %".format(accuracies.mean()*100))
```

```
[[16  0  0]
 [ 0 18  0]
 [ 0  0 11]]
acurácia SVM: 100.00 %
```

MLP

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# ajustar pra dados de treino
scaler.fit(X_train)

# escalando dados de treino
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
print(X_train[:3])

# treino:
from sklearn.neural_network import MLPClassifier
# criando classificador do modelo:
mlp = MLPClassifier(hidden_layer_sizes=(10, 5), max_iter=1000)

# atributo fit: ajusta o modelo para os dados
mlp.fit(X_train, y_train)
```

```
[[-0.38183718 -1.02366372 -2.37846268 -0.18295039 -0.29145882]
 [ 0.87739032  0.69517462 -0.10190314  0.93066067  0.73721938]
 [ 1.50700407  0.92435306  0.58106472  1.04202177  1.6373128 ]]
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning:

Stochastic Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet.

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(10, 5), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=1000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

```
from sklearn.metrics import accuracy_score
# calculando acurácia
predictions_train = mlp.predict(X_train)
predictions_test = mlp.predict(X_test)
cm = confusion_matrix(predictions_train, y_train)
print(cm)
print("acurácia MLP: {:.2f} %".format(accuracy_score(y_train, predictions_train)*100))
```

```
[[34  0  0]
```

```
 [ 0 32  0]
 [ 0  0 39]]
acurácia MLP: 100.00 %
```

```
1
```

✓ 0s conclusão: 18:47