Paquetes a utilizar

```
import sklearn as skl
import pandas as pd
import numpy as np
import xgboost as xgb
from boruta import BorutaPy
```

Base de datos

```
df = pd.read_csv("data/wisconsin_breast_cancer_dataset.csv")
df = df.dropna()
df = df.rename(columns={'diagnosis':'Label'})
df['Label'].value_counts()
```

```
df.describe().T
```

```
df.dtypes
```

Variable dependiente que debe predecirse y codificación de datos categóricos

```
y = df["Label"].values
Y = skl.preprocessing.LabelEncoder().fit_transform(y)
```

Definir x, normalizar valores y definir variables independientes

```
X = df.drop(labels = ["Label", "id"], axis=1)
nombres_de_funciones = np.array(X.columns)
scaler = skl.preprocessing.StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
```

Train and test para verificar la precisión después de ajustar el modelo

```
X_train, X_test, y_train, y_test = skl.model_selection.train_test_split(X, Y, test_size=0.25
```

XGBOOST para ser utilizado por Boruta

```
modelo = xgb.XGBClassifier()
```

- Crear funciones de sombra: funciones aleatorias y valores aleatorios en columnas
- Entrenar Random Forest / XGBoost y calcular la importancia de la característica a través de la disminución media de la impureza
- Comprobar si las características reales tienen mayor importancia en comparación con las características de sombra
- Repetir esto para cada iteración
- Si la función original funcionó mejor, marcarla como importante

```
# definir el método de selección de características de Boruta
selector = BorutaPy(modelo, n_estimators='auto', verbose=2, random_state=1)
# encontrar todas las características relevantes
selector.fit(X_train, y_train)
# llamar a transform() en X para filtrarlo a las características seleccionadas
X_filtered = selector.transform(X_train)  # Aplicar selección de características y devolver
```

zip nombres de características, rangos y decisiones

```
feature_ranks = list(zip(nombres_de_funciones,
                         selector.ranking_,
                         selector.support_))
```

Resultados

```
for feat in feature_ranks:
    print('Feature: {:<30} Rank: {},  Keep: {}'.format(feat[0], feat[1], feat[2]))

# Ahora usar el subconjunto de funciones para ajustar el modelo XGBoost en los datos de ent
xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_filtered, y_train)
# Ahora predecir con datos de prueba usando el modelo entrenado
# Primero aplicar la transformación del selector de funciones para asegurarse de que se sel
X_test_filtered = selector.transform(X_test)
prediction_xgb = xgb_model.predict(X_test_filtered)
```

Precisión

```
skl.metrics.accuracy_score(y_test, prediction_xgb)
```