# شرح كود (Chi-Square Selector)

## أولاً: استيراد المكتبات الأساسية :

**Pandas** تُستخدم لإدارة البيانات الجدولية (**DataFrames**).

استوردنا من مكتبة **sklearn** الأدوات الأساسية التالية:

**SelectKBest** و **chi2**: هذه الأدوات نستخدمها في عملية اختيار الميزات؛ حيث chi2 (اختبار مربع كاي) هي الدالة الإحصائية التي تقيس مدى ارتباط كل ميزة بالمتغير الهدف.

**MinMaxScaler**: أداة ضرورية لتطبيع البيانات قبل تطبيق اختبار \chi^2.

**LogisticRegression** و **accuracy_score**: نستخدمها لـ تقييم كفاءة الميزات المختارة.

**train_test_split**: لتقسيم البيانات لضمان تقييم عادل.

```python
import numpy as np
import pandas as pd
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

```

## ثانياً: تعريف الصنف:

نعرف الصنف **ChiSquareSelector** لتغليف المنطق وجعله قابلاً لإعادة الاستخدام.

ومن ثم يتم تهيئة **self.k_features** من خلال دالة البناء **init** ، وهو عدد الميزات المطلوبة (افتراضياً ١٠).

نقوم بتهيئة متغيرات أخرى (**self.selected_features_names_**, **self.selector_**) لتخزين أداة التحديد وأسماء الميزات بعد معالجة البيانات.

```python
class ChiSquareSelector:

    def __init__(self, k_features: int = 10):

        #تهيئة الأداة بعدد الميزات المطلوب اختيارها.

        self.k_features = k_features
        self.selector_ = None
        self.selected_features_names_ = None
```

## ثالثاً: تجهيز البيانات والتطبيع

( التطبيع هو أهم خطوة لأن اختبار **Chi-Square** لا يعمل بشكل جيد مع القيم السالبة أو الغير مطبعة)

وهنا قمنا باستخدام **MinMaxScaler** لتحويل قيم الميزات إلى مجال بين 0 و 1

ولضمان بقاء أسماء الأعمدة سليمة أعدنا البيانات المطبعة إلى **Pandas DataFrame**

```
def fit_transform(self, X: pd.DataFrame, y: pd.Series) -> tuple[list, float]:

    #X: (Pandas DataFrame)
    #y: (Pandas Series)


    # 1. إعداد البيانات Chi-square

    X_scaled = MinMaxScaler().fit_transform(X)
    X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
```

## رابعاً: تطبيق اختبار **Chi-Square**

قمنا بإنشاء **SelectKBest**، ومررنا له **score_func=chi2**. من أجل توجيه الأداة لاستخدام قيمة كاي مربع كمعيار لقياس أهمية الميزة.

قمنا باستخدم **k=min**(...) لضمان أننا لا نحاول اختيار عدد ميزات أكبر من العدد المتاح فعلياً في البيانات.

أيضاً استخدمنا **self.selector_.fit(X_scaled, y)**، لتدريب الأداة على حساب قيمة كاي مربع لكل ميزة وترتيبها.

```
# تطبيق Chi-square
self.selector_ = SelectKBest(score_func=chi2, k=min(self.k_features, X.shape[1]))

# التحديد تطبيق
self.selector_.fit(X_scaled, y)
```

## خامساً: تحديد الميزات المختارة

بعد التدريب، نحتاج إلى معرفة الميزات التي تم اختيارها لذلك استخدمنا

**get_support**:لأنها ترجع قناعاً منطقياً (**True/False**) يحدد الميزات الـ K التي حصلت على أعلى درجات كاي مربع.

نستطيع من خلال هذا القناع استخراج أسماء الأعمدة الفعلية من بيانات X الأصلية، وتخزينها في **self.selected_features_names_**.

```
# تحديد الميزات المختارة

selected_mask = self.selector_.get_support()
self.selected_features_names_ = list(X.columns[selected_mask])
```

## سادساً: التقيم على الميزات المختارة

في هذا الجزء نبرهن على كفاءة الاختيار

**X_final = X[self.selected_features_names_]:** هنا يتم عزل البيانات لـ الميزات المختارة فقط.

**train_test_split**: من خلالها يتم تقسيم البيانات (٣٠٪ للاختبار) لضمان أن التقييم يتم على بيانات لم يراها النموذج .

**LogisticRegression**: نستخدم نموذج الانحدار اللوجستي كـ نموذج أساسي إذا حقق النموذج دقة جيدة باستخدام الميزات المختارة.

**accuracy_score**: نحسب دقة النموذج على مجموعة الاختبار.

**return** : في النهاية، يتم إرجاع أسماء الميزات التي اخترناها ودرجة الدقة التي حققها النموذج باستخدام أسماء الميزات ، مما ينهي العملية بنتيجة قابلة للقياس.

```python
# التقييم على الميزات المختارة

if not self.selected_features_names_:
    return [], 0.0

# تقسيم البيانات للتقييم
X_final = X[self.selected_features_names_]
X_train, X_test, y_train, y_test = train_test_split(X_final, y, test_size=0.3, random_state=42)

# تدريب نموذج تصنيف (Logistic Regression)
final_model = LogisticRegression(max_iter=1000).fit(X_train, y_train)

# حساب الدقة
y_pred = final_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

return self.selected_features_names_, accuracy
```