Data loading and Exploring | تحميل مجوعة البيانات واستكشافها

```
# تحميل المكتبات المطلوبة
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn import metrics
import pickle
from sklearn.tree import export_graphviz
from io import StringIO

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score


from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

import warnings
warnings.filterwarnings('ignore')
from IPython.display import Image
import pydotplus



# قراءة مجموعة البيانات
df =pd.read_csv('loan_prediction.csv')

#عرض أو خمسة أسطر من المجموعة
df.head(5)
```

	#	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0
1	1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0
2	2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0
3	3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0
4	4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# إعادة تسمية الأعمدة لسهولة القراءة
columns =['#','LoanID', 'Gender', 'Married', 'Dependents', 'Education',
       'SelfEmployed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'LoanAmountTerm', 'CreditHistory', 'PropertyArea', 'target']
df.columns= columns


# عرض حجم مجموعة البيانات
# حيث كما نلاحظ فهيا تتكزن من 614 سجل وكل سجيل يتكون من أربعة عشرة سطراً #
```

```
print(f'the shape of the data frame is {df.shape } , rows : {df.shape[0]} , columns : {df.shape[1]}\n\n')
```

```
the shape of the data frame is (614, 14) , rows : 614 , columns : 14
```

```
# عرض بعض المعلومات عن المجموعة
print(f'the information of the data frame is ')

df.info()
```

```
the information of the data frame is
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   #                  614 non-null    int64
 1   LoanID             614 non-null    object
 2   Gender             601 non-null    object
 3   Married            611 non-null    object
 4   Dependents         599 non-null    object
 5   Education          614 non-null    object
 6   SelfEmployed       582 non-null    object
 7   ApplicantIncome    614 non-null    int64
 8   CoapplicantIncome  614 non-null    float64
 9   LoanAmount         592 non-null    float64
 10  LoanAmountTerm     600 non-null    float64
 11  CreditHistory      564 non-null    float64
 12  PropertyArea       614 non-null    object
 13  target             614 non-null    object
dtypes: float64(4), int64(2), object(8)
memory usage: 67.3+ KB
```

```
# نلاحظ أن هنالك عمودين ها معرف القرض وقم السطر اليقدمان أي معلومة مفيدة
# لذلك سنقوم بإزالتهما من المجموعة
df = df.drop(['LoanID','#'],axis = 1)
```

```
# عرض مقدار القيم الفارغة أي غير الموجودة
# ونلاحظ أن عددها قليل نسبيا لذللك يمكننا حذفها
# كما يمكننا تبديلها بالأكثر تكراراً وهو ماسنفعله
df.isnull().sum()
```

	0
Gender	13
Married	3
Dependents	15
Education	0
SelfEmployed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
LoanAmountTerm	14
CreditHistory	50
PropertyArea	0
target	0

```
# استبدال القيم الفارغة بالقيمة الأكثر تكراراً من أجل الأعمدة الفئوية
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
df['SelfEmployed'].fillna(df['SelfEmployed'].mode()[0], inplace=True)
df['CreditHistory'].fillna(df['CreditHistory'].mode()[0], inplace=True)
df['LoanAmountTerm'].fillna(df['LoanAmountTerm'].mode()[0], inplace=True)
```

```
# استبدال القيمة الفارغة بالمتوسط من أجل الأعمدة العددية
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)


# نلاحظ أن جميع القيم الفارغة أزيلت
df.isnull().sum()
```

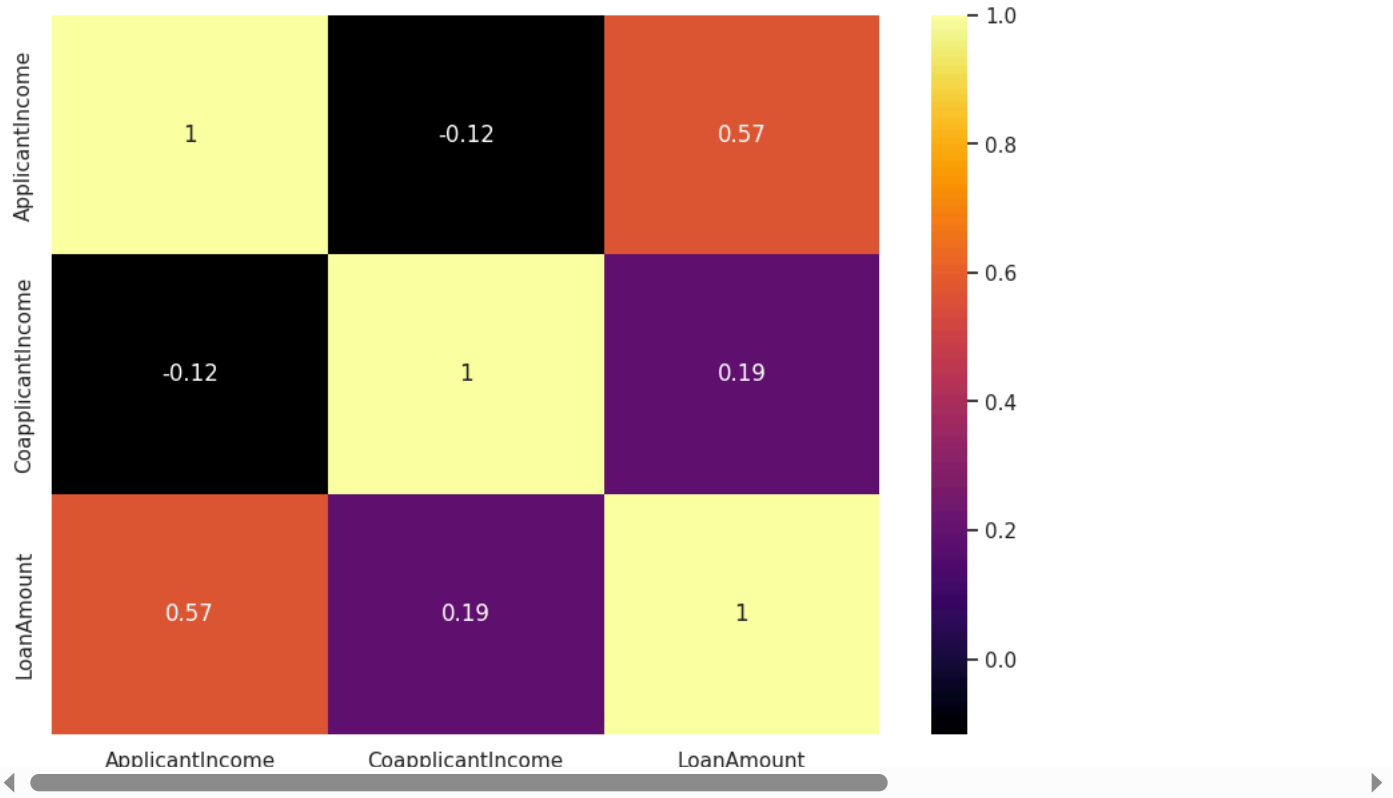
	0
Gender	0
Married	0
Dependents	0
Education	0
SelfEmployed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
LoanAmountTerm	0
CreditHistory	0
PropertyArea	0
target	0

```
# للحقول العددية وصف مجموعة البيانات
#
df.describe()
```

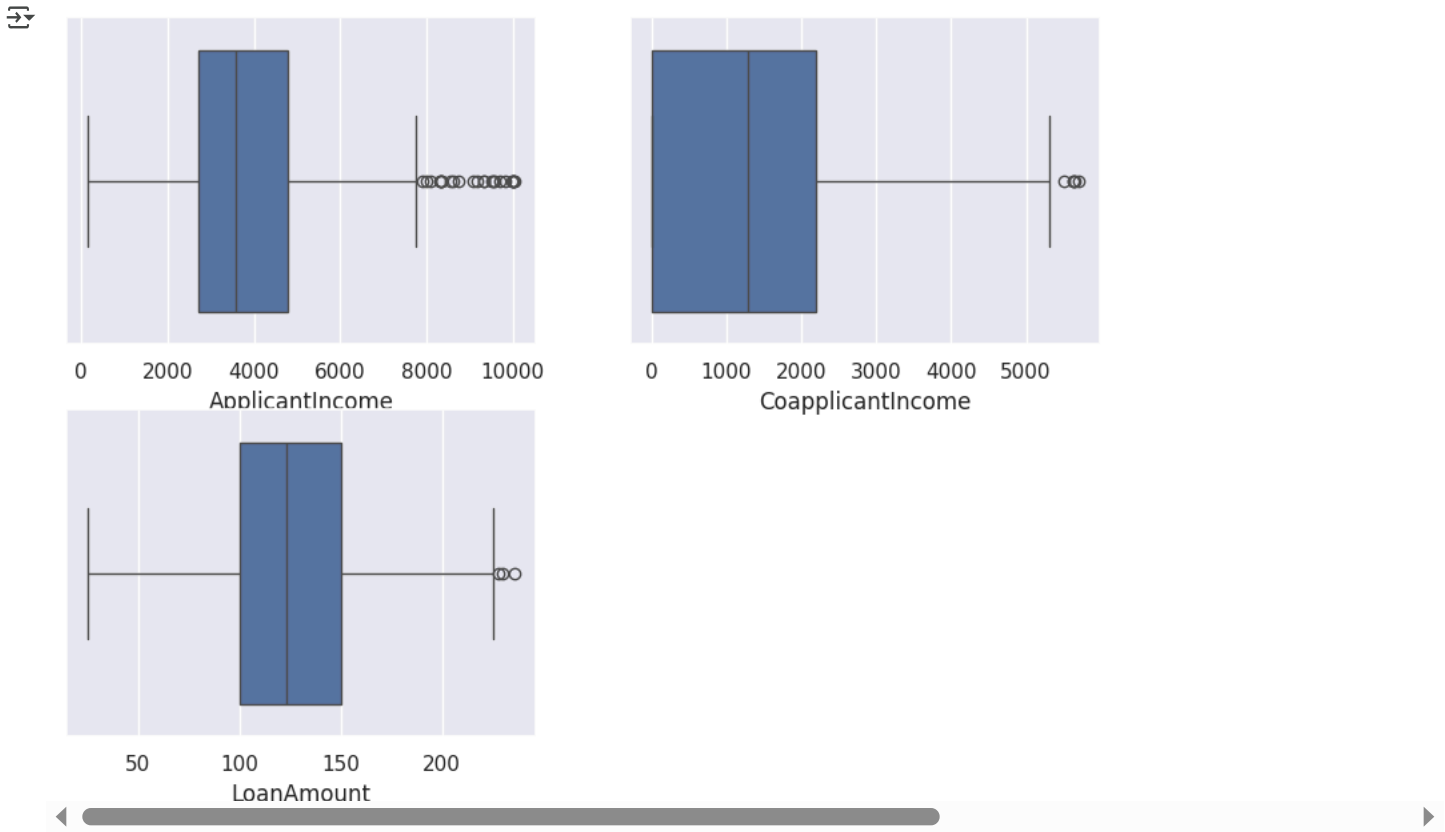
	ApplicantIncome	CoapplicantIncome	LoanAmount	LoanAmountTerm	CreditHistory
count	614.000000	614.000000	614.000000	614.000000	614.000000
mean	5403.459283	1621.245798	146.412162	342.410423	0.855049
std	6109.041673	2926.248369	84.037468	64.428629	0.352339
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.250000	360.000000	1.000000
50%	3812.500000	1188.500000	129.000000	360.000000	1.000000
75%	5795.000000	2297.250000	164.750000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

```
# تعريف الأعمدة العددية والفئوية
# لقد عرفنا العمود
#loan amount term على أنه فوي لأن له ثلاث قيم عددية فقط

numerical_columns = [  'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']

categorical_columns = [ 'Gender', 'Married', 'Dependents', 'Education', 'LoanAmountTerm',
                        'SelfEmployed', 'CreditHistory', 'PropertyArea' ]


# رسم مصفوفة ارتباط بين الحقول العددية
# نلاحظ أن هناك ارتباط بين مقدار القرض ودخل المتقدم
numeric_columns = df[numerical_columns]
plt.figure(figsize=(10,7))
sns.heatmap(numeric_columns.corr(), annot=True, cmap='inferno')
```

<Axes: >

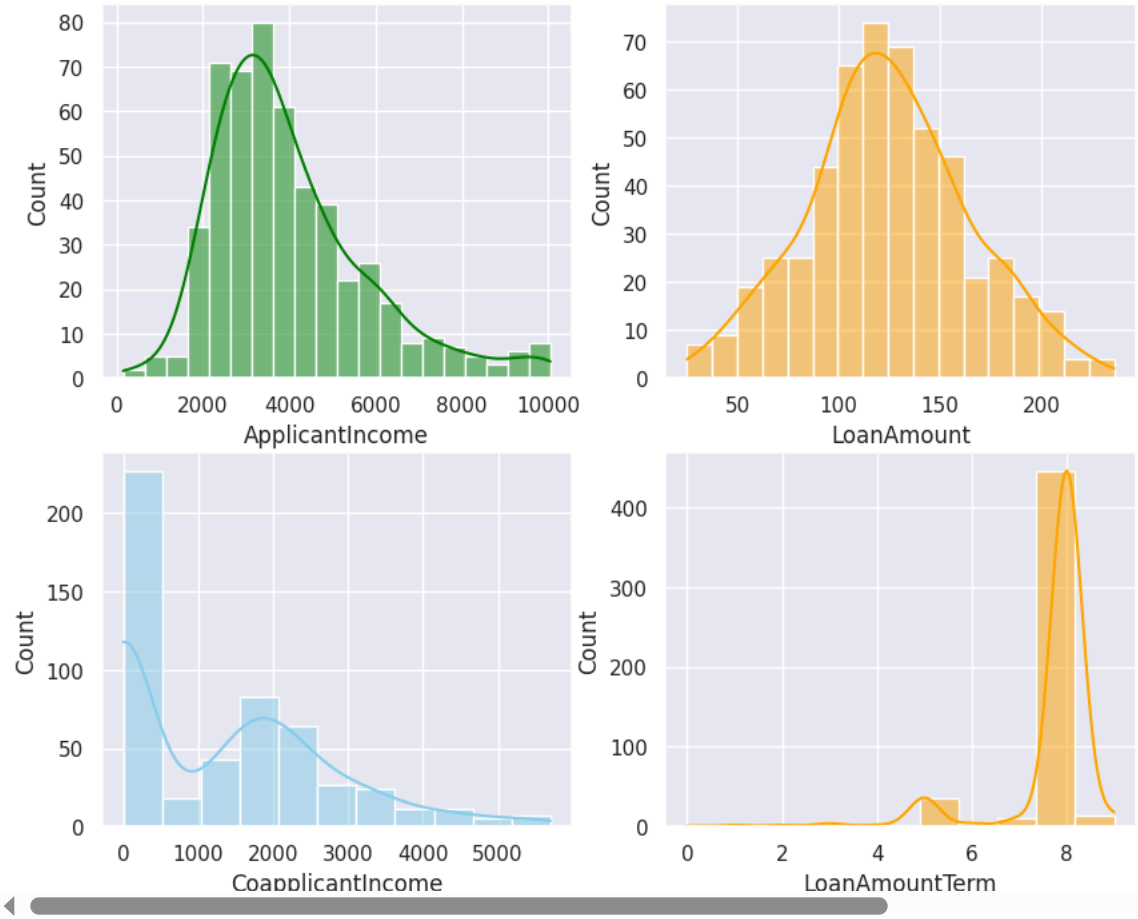



```
# ترميز الأعمدة الفئوية
# أي تويللها من نصوص إلى أرقام مثل
# (Y,N) => (1,0)
# تعريف المرمز الذي سوف نستخدمه للترميز
label_encoder = LabelEncoder()
for column in categorical_columns:
    # تطبيق الترميز
    df[column] = label_encoder.fit_transform(df[column])


# ترميز العمود الهدف من
# 1 و 0إلى y,n
df.target = df.target.map({'Y': 1 , 'N':0})


# إزالة القم الشائة
# وهي التي تقع خارج الربعين
for feature in numerical_columns:
    # ترعيف الربع الأدنى والربع الأعلى
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    # تعريف النطاق
    IQR = Q3 - Q1
    # تعريف الحدود
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # حذف القيم الخارجة
    df = df[(df[feature] >= lower_bound) & (df[feature] <= upper_bound)]


# رسم مخطط توزع القيم
plt.figure(figsize=(10, 7))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x=df[column])
```



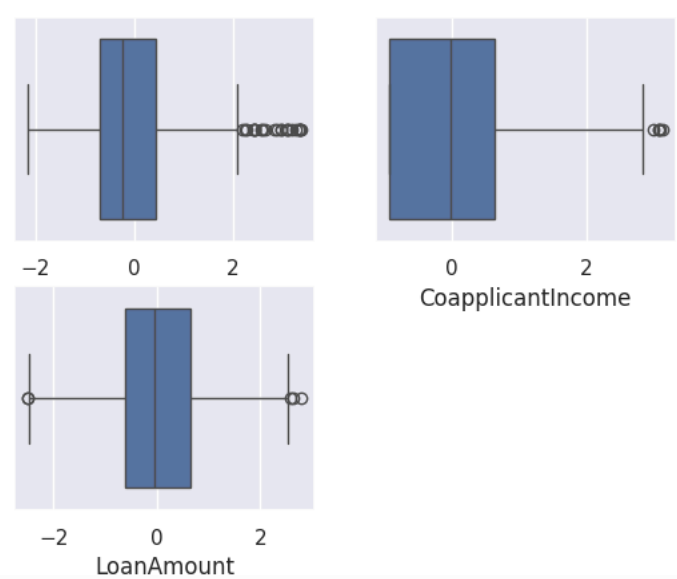

```
مخطط توزع القيم ولكن كمنحن #

sns.set(style="darkgrid")
fig, axs = plt.subplots(2, 2, figsize=(10, 8))

sns.histplot(data=df, x="CoapplicantIncome", kde=True, ax=axs[1, 0], color='skyblue')
sns.histplot(data=df, x="LoanAmount", kde=True, ax=axs[0, 1], color='orange')

sns.histplot(data=df, x="ApplicantIncome", kde=True, ax=axs[0, 0], color='green')
sns.histplot(data=df, x="LoanAmountTerm", kde=True, ax=axs[1, 1], color='orange')
```

```
<Axes: xlabel='LoanAmountTerm', ylabel='Count'>
```



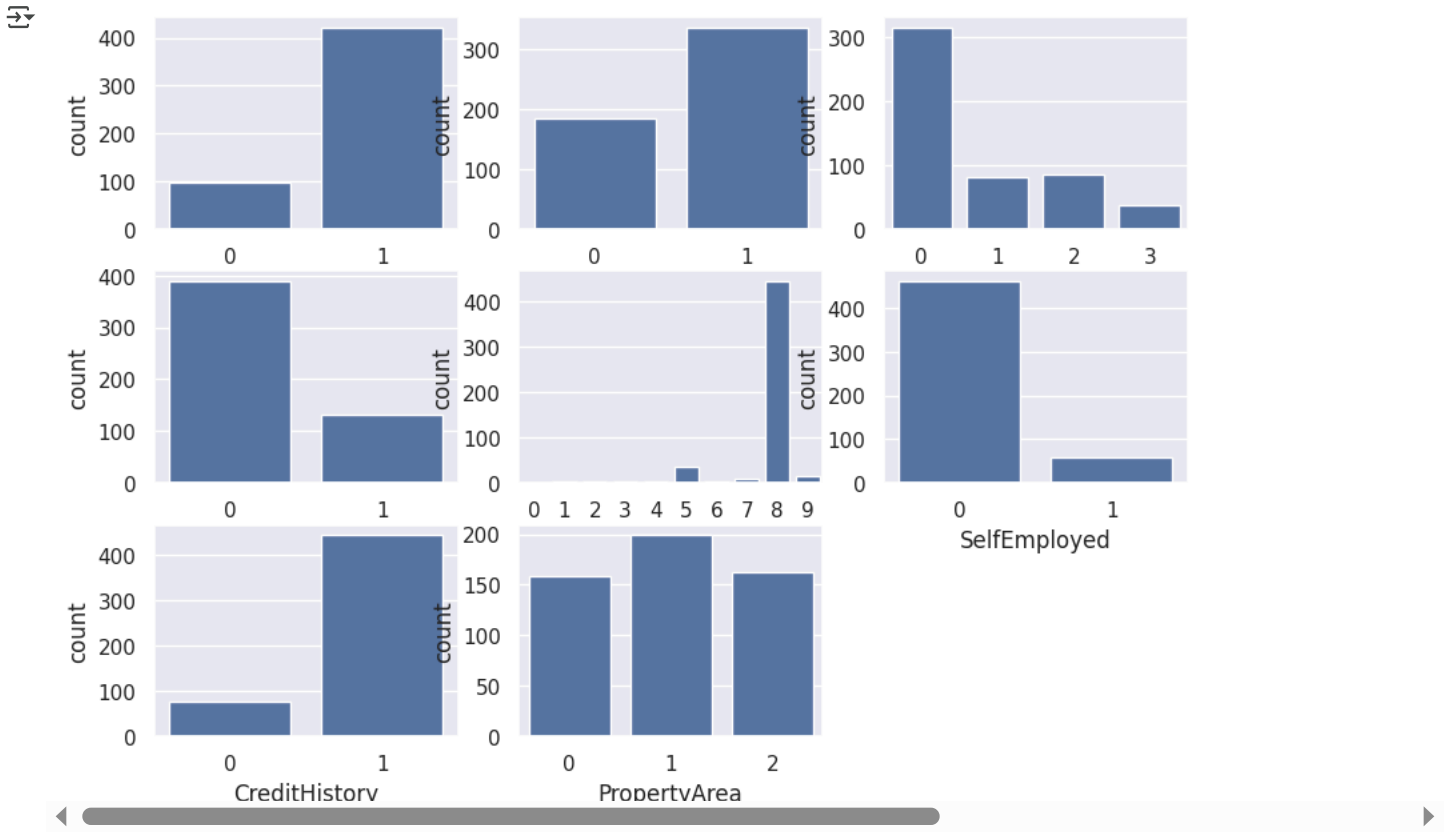

```
# تنظيم البيانات وإعادتها إلى المجال 0 1 من إجل القيم العددية لان بعض النماذج تتحسس لتوزع المجال
scaled_columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']
standard_scaler = StandardScaler()
df[scaled_columns] = standard_scaler.fit_transform(df[scaled_columns])
```

```
# مخططط التوزع بعد تنظيم البياناتplt.figure(figsize=(10, 7))
for i, column in enumerate(scaled_columns, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x=df[column])
```

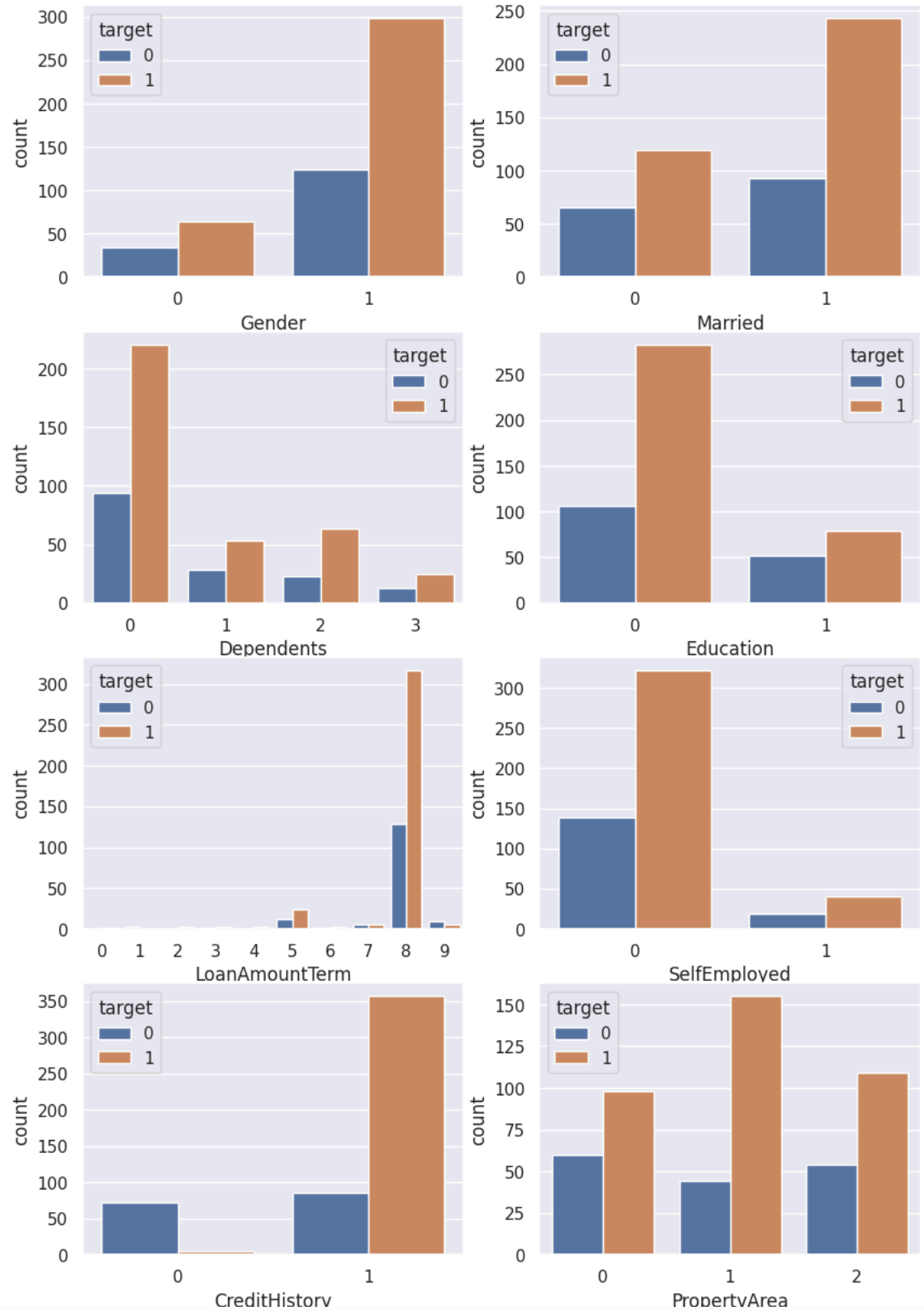



```
# مخطط توزع القيم الفئوية
plt.figure(figsize=(10, 7))
```

```
for i, column in enumerate(categorical_columns, 1):
    plt.subplot(3, 3, i)
    sns.countplot(x=df[column])
```




```
# مخطط توزع القيم الفئوية مع أخذ النتيجة بالاعتبار

figure ,axes = plt.subplots(4,2,figsize=(10,15))
for index,categorical_col in enumerate(categorical_columns):
    row,col = index//2,index%2
    sns.countplot(x=categorical_col,data=df,hue='target',ax=axes[row,col])
```




```
# عرض مجموعة البيانات بعد تلك المعالجات
df.head(5)
```

	Gender	Married	Dependents	Education	SelfEmployed	ApplicantIncome	CoapplicantIncome	LoanAmount	LoanAmountTerm	CreditHistory	P
0	1	0	0	0	0	1.033643	-0.943785	0.552042	8	1	
1	1	1	1	0	0	0.327548	0.136399	0.091120	8	1	
2	1	1	0	0	1	-0.555349	-0.943785	-1.460962	8	1	
3	1	1	0	1	0	-0.787925	0.745256	-0.109148	8	1	
4	1	0	0	0	0	1.117861	-0.943785	0.416557	8	1	

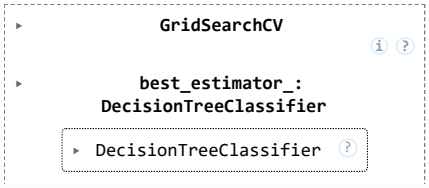
Next steps: Generate code with df | View recommended plots | New interactive sheet

تحضير المجموعة للتدريب

```
# تحضير البيانات للتدريب
X=df.drop('target',axis=1)
Y=df['target']


# KBinsDiscretizer تحويل القيم العددية إلى فئوية باستخدام التابع

numerical_colmns_cat =['ApplicantIncomeCat', 'CoapplicantIncomeCat', 'LoanAmountCat']
# تعريف التابع
discretizer = KBinsDiscretizer(n_bins=10, encode='ordinal',strategy='kmeans')

# تطبيق التابع
X[numerical_colmns_cat] = discretizer.fit_transform(X[numerical_columns])


# تقسيم المجموعة إلى تدريب واختبار
X_train ,X_test,y_train , y_test =train_test_split(X,Y,test_size=0.3,random_state=47)
```

استخدام نموذج أشجار القرار

```
# تعريف موسطات البحث
parameters = {
    # أكبر عمق للشجرة
    'max_depth': [8,10,15,13,15],
    # أقل عدد عناصر لتقسيم العقدة
    'min_samples_split': [35,45,50,60],
    # أقل عدد عناصر لاعتبار عقدة ورقة
    'min_samples_leaf': [20,30,25,35],
    # معيار تقويم الشجرة
    'criterion': [ 'entropy']
}

DT_grid_search = GridSearchCV(DecisionTreeClassifier(criterion='entropy'), parameters,scoring='f1', cv=5)


# تدريب الشجرة
DT_grid_search.fit(X_train[numerical_colmns_cat+ categorical_columns], y_train)
```




```
# عرض موسطات النموذج الأفضل
print(f'Best parameters: {DT_grid_search.best_params_}')
```

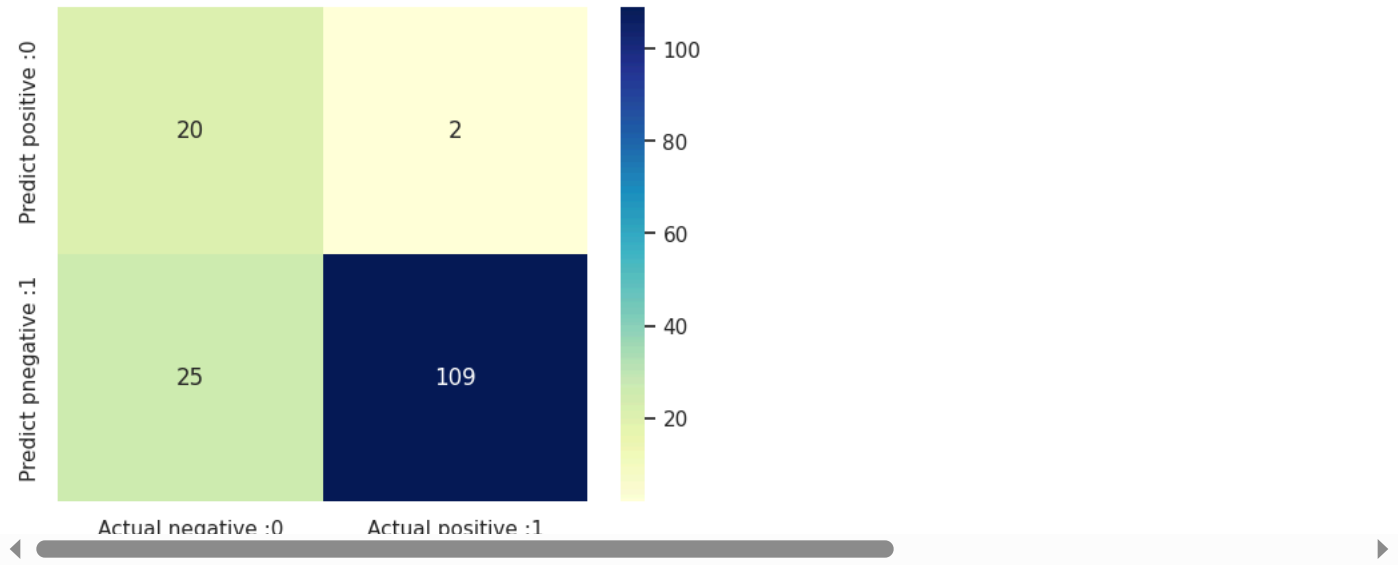
```
Best parameters: {'criterion': 'entropy', 'max_depth': 8, 'min_samples_leaf': 20, 'min_samples_split': 45}
```

```
DT_y_pred = DT_grid_search.best_estimator_.predict(X_test[numerical_colmns_cat+ categorical_columns])

# تقييم النموذج على مجموعة الاختبار
accuracy = accuracy_score(y_test, DT_y_pred)
print(f'Accuracy of DecisionTreeClassifier after scaling: {accuracy}')
```

```
Accuracy of DecisionTreeClassifier after scaling: 0.8269230769230769
```

```
# عرض مصفوفة التقييم

cm=  confusion_matrix(DT_y_pred,y_test)
cm_matrix = pd.DataFrame(data=cm,columns=['Actual negative :0','Actual positive :1']
                         ,index=['Predict positive :0','Predict pnegative :1'])
sns.heatmap(cm_matrix,annot=True,fmt='d',cmap='YlGnBu')
```

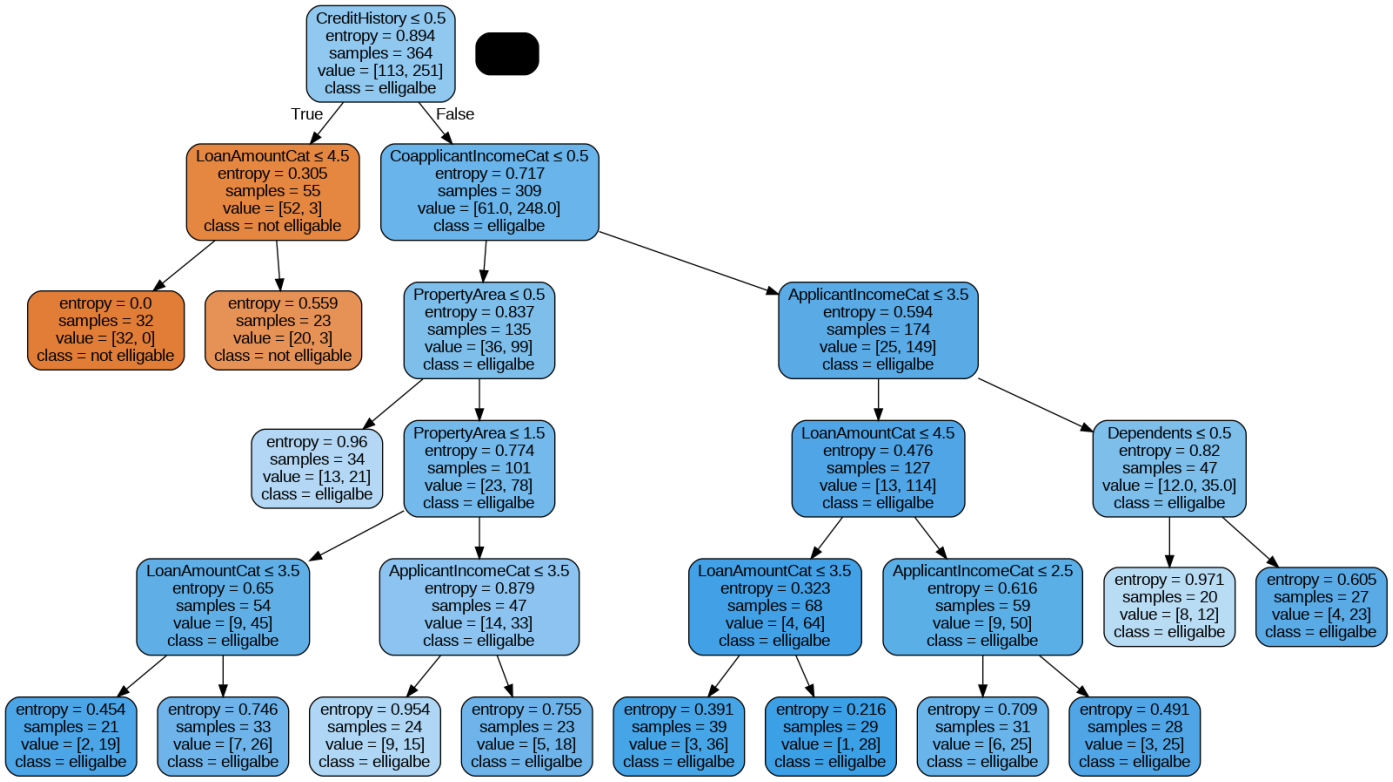
```
<Axes: >
```




```
DT_y_pred = DT_grid_search.best_estimator_.predict(X_test[numerical_colmns_cat+ categorical_columns])

# تقييم النموذج

# الصحة
accuracy = accuracy_score(y_test, DT_y_pred)
# الدقة
precision = precision_score(y_test, DT_y_pred)
# تقرير التصنيف
print(classification_report(y_test, DT_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.44      0.60        45
           1       0.81      0.98      0.89       111

    accuracy                           0.83       156
   macro avg       0.86      0.71      0.74       156
weighted avg       0.84      0.83      0.81       156
```

```
# رسم الشجرة
dot_data = StringIO()
export_graphviz(DT_grid_search.best_estimator_, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = numerical_colmns_cat+categorical_columns ,class_names=['not elligable','elligalbe'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('loan9.png')
Image(graph.create_png())
```



استخدام نموذج SVM

```
# تعريف المصنف
svc = SVC ()

# تعريف موسطات البحث
parameters = [
    {'C': range(1,100,10),'kernel':['linear']}
   ]
#
SVM_grid_search = GridSearchCV(estimator=svc,
                           param_grid=parameters,
                           scoring="f1",
                           cv=5,
                           verbose=1)


# تدريب النوذج
SVM_grid_search.fit(X_train[numerical_columns+ categorical_columns], y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

GridSearchCV

best_estimator_: SVC

SVC

```
svm_best=SVM_grid_search.best_estimator_

print(f'Best parameters: {svm_best}')
```

```
Best parameters: SVC(C=1, kernel='linear')
```

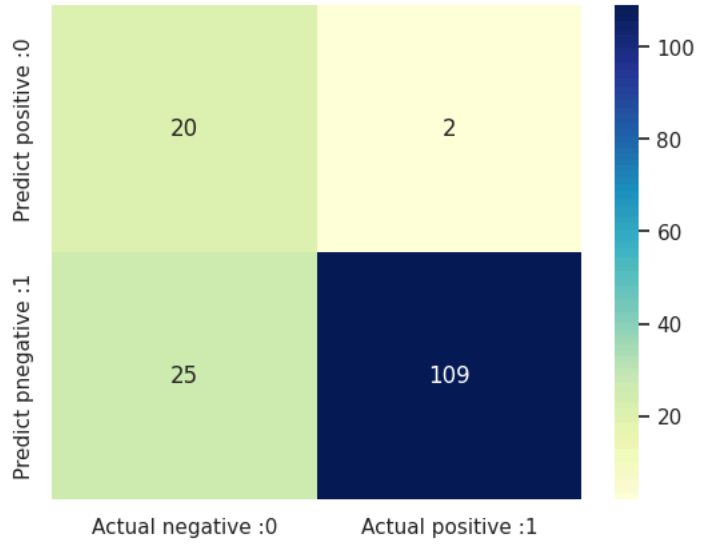
```
SVM_y_pred = svm_best.predict(X_test[numerical_columns+ categorical_columns])

# تقييم النموذج
accuracy = accuracy_score(y_test, SVM_y_pred)

# عرض حة النموذج
print(f'Accuracy of SVM after scaling: {accuracy}')
```

```
# تقرير التصنيف
print(classification_report(y_test, SVM_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.44      0.60        45
           1       0.81      0.98      0.89       111

    accuracy                           0.83       156
   macro avg       0.86      0.71      0.74       156
weighted avg       0.84      0.83      0.81       156
```

```
# عرض مصفوفة التقييم
cm=  confusion_matrix(SVM_y_pred,y_test)
cm_matrix = pd.DataFrame(data=cm,columns=['Actual negative :0','Actual positive :1']
                         ,index=['Predict positive :0','Predict pnegative :1'])
sns.heatmap(cm_matrix,annot=True,fmt='d',cmap='YlGnBu')
```

```
<Axes: >
```



استخدام نموذج أقرب جار KNN

```
# تعريف موسطات البحث
parameters={'n_neighbors':range(1,40),
            'metric':['euclidean','manhattan','cosine'],
            'weights':['uniform','distance']
           }
# تعريف النموذج
KNN_=KNeighborsClassifier()
# تعريق البحث
KNN_grid_search=GridSearchCV(estimator=KNN_,param_grid=parameters,cv=5,scoring="f1")
```

```
# تدريب النموذج
KNN_grid_search.fit(X_train[numerical_columns+ categorical_columns],y_train)
# عرض موسطات النوذج الأفضل
print(KNN_grid_search.best_params_)
```

```
{'metric': 'manhattan', 'n_neighbors': 8, 'weights': 'uniform'}
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.