

```
from google.colab import files

uploaded = files.upload()
```

No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

1. لقراءة ملف الطلاب، ثم فحصنا حجم البيانات، أنواع الأعمدة، والقيم المفقودة للتأكد أن الداتا جاهزة للتحليل pandas قراءة البيانات وفحصها استخدمنا

```
import pandas as pd
```

```
df = pd.read_csv('Students_general_report_DB_ready 19 March.csv')
print(df.head())
```

	No. active courses	Exams	Exams Participation %	Homeworks	\
0	13	0	0.0	1	
1	13	0	0.0	1	
2	13	0	0.0	1	
3	13	0	0.0	1	
4	13	0	0.0	1	

	Homeworks Participation %	Interactive contents	Contents Participation %	\
0	100%	3	0%	
1	0%	3	0%	
2	100%	3	0%	
3	100%	3	33%	
4	0%	3	33%	

	Virtual Classrooms	Virtual Classrooms Participation %	AVG Participations %	\
0	0	0%	25%	
1	0	25%	0%	
2	0	25%	25%	
3	0	25%	33%	
4	0	25%	8%	

	Discussion Rooms	Discussion Rooms Participation %	Teachers messages	\
0	0	0%	0	
1	0	0%	0	
2	0	0%	0	
3	0	0%	0	
4	0	0%	0	

	Mailbox Participation %	Student login %	Interactive contents Phet	\
0	0%	100%	0	
1	0%	100%	0	
2	0%	100%	0	
3	0%	100%	0	
4	0%	100%	0	

	Final Mark	Abcense
0	50	70
1	70	60
2	90	50
3	100	20
4	100	100

```
import pandas as pd
```

```
df = pd.read_csv("Students_general_report_DB_ready 19 March.csv")
```

```
print("Shape:")
print(df.shape)
```

```
print("\nData Types:")
print(df.dtypes)
```

```
print("\nMissing Values:")
print(df.isnull().sum())
```

```
print("\nFirst 5 Rows:")
print(df.head())
```

Shape:
(1395, 18)

Data Types:
No. active courses int64

```

Exams                                int64
Exams Participation %                 float64
Homeworks                            int64
Homeworks Participation %             object
Interactive contents                  int64
Contents Participation %              object
Virtual Classrooms                    int64
Virtual Classrooms Participation %    object
AVG Participations %                 object
Discussion Rooms                      int64
Discussion Rooms Participation %       object
Teachers messages                     int64
Mailbox Participation %               object
Student login %                      object
Interactive contents Phet              int64
Final Mark                           int64
Absense                              int64
dtype: object

```

Missing Values:

```

No. active courses                    0
Exams                                0
Exams Participation %                 0
Homeworks                            0
Homeworks Participation %             0
Interactive contents                  0
Contents Participation %              0
Virtual Classrooms                    0
Virtual Classrooms Participation %    0
AVG Participations %                 0
Discussion Rooms                      0
Discussion Rooms Participation %       0
Teachers messages                     0
Mailbox Participation %               0
Student login %                      0
Interactive contents Phet              0
Final Mark                           0
Absense                              0
dtype: int64

```

First 5 Rows:

	No. active courses	Exams	Exams Participation %	Homeworks \
0	13	0	0.0	1
1	13	0	0.0	1
2	13	0	0.0	1
3	13	0	0.0	1
4	13	0	0.0	1

	Homeworks Participation %	Interactive contents	Contents Participation % \
0	100%	3	0%
1	0%	3	0%
2	100%	3	0%
3	100%	3	33%

تحويل النسب المئوية بعض الأعمدة كانت مكتوبة مثل 100% كنص، فحولناها إلى أرقام مثل 100.0 حتى نقدر النماذج تتعامل معها.

```

percent_cols = [
    'Homeworks Participation %',
    'Contents Participation %',
    'Virtual Classrooms Participation %',
    'AVG Participations %',
    'Discussion Rooms Participation %',
    'Mailbox Participation %',
    'Student login %'
]

for col in percent_cols:
    df[col] = (
        df[col]
        .astype(str)
        .str.replace('%', '', regex=False)
        .astype(float)
    )

```

```
print(df[percent_cols].dtypes)
```

```

Homeworks Participation %    float64
Contents Participation %     float64
Virtual Classrooms Participation % float64
AVG Participations %        float64
Discussion Rooms Participation % float64

```

```
Mailbox Participation %      float64
Student login %             float64
dtype: object
```

```
print(df.head())
```

```

  No. active courses  Exams  Exams Participation %  Homeworks  \
0                   13     0                   0.0         1
1                   13     0                   0.0         1
2                   13     0                   0.0         1
3                   13     0                   0.0         1
4                   13     0                   0.0         1

  Homeworks Participation %  Interactive contents  Contents Participation %  \
0                   100.0         3                   0.0
1                   0.0         3                   0.0
2                   100.0         3                   0.0
3                   100.0         3                  33.0
4                   0.0         3                  33.0

  Virtual Classrooms  Virtual Classrooms Participation %  \
0                   0                   0.0
1                   0                  25.0
2                   0                  25.0
3                   0                  25.0
4                   0                  25.0

  AVG Participations %  Discussion Rooms  Discussion Rooms Participation %  \
0                   25.0         0                   0.0
1                   0.0         0                   0.0
2                   25.0         0                   0.0
3                   33.0         0                   0.0
4                   8.0         0                   0.0

  Teachers messages  Mailbox Participation %  Student login %  \
0                   0                   0.0         100.0
1                   0                   0.0         100.0
2                   0                   0.0         100.0
3                   0                   0.0         100.0
4                   0                   0.0         100.0

  Interactive contents  Phet  Final Mark  Absense
0                   0       50         70
1                   0       70         60
2                   0       90         50
3                   0      100         20
4                   0      100        100
```

3. إنشاء الـ Target / Class قسماً الطلاب إلى 3 فئات حسب

```
def create_class(mark):
    if mark < 50:
        return 0
    elif mark < 75:
        return 1
    else:
        return 2

df['Class'] = df['Final Mark'].apply(create_class)

print(df['Class'].value_counts())
```

```
Class
1    828
2    326
0    241
Name: count, dtype: int64
```

2. تجهيز البيانات

```
X = df.drop(['Final Mark', 'Class'], axis=1)
y = df['Class']
```

3. تقسيم البيانات

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

4. لمعالجة عدم التوازن SMOTE النتيجة كانت غير متوازنة تقريبًا، لذلك استخدمنا Class لمعرفة عدد الطلاب في كل value_counts() فحص توازن البيانات استخدمنا.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)

X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

5. حتى يحافظ التقسيم على نفس توزيع الكلاسات stratify=y مع train_test_split باستخدام Training و Testing تقسيم البيانات قسمنا البيانات إلى.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train_smote)
X_test_scaled = scaler.transform(X_test)
```

6. Scaling باستخدام StandardScaler مثل Logistic Regression و SVM النماذج المستخدمة ولماذا . النماذج مقاييس الأرقام . تتأثر باختلاف مقاييس الأرقام.

✓ **Logistic Regression:** نموذج بسيط وأساسي للمقارنة كبدائية.

SVM: جيد مع التصنيف ويفصل بين الكلاسات باستخدام حدود قرار قوية.

Random Forest: ويقلل الخطأ Decision Tree قوي لأنه يستخدم أكثر من.

XGBoost: قوي جدًا وغالبًا يعطي نتائج عالية في التصنيف Boosting نموذج.

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

7. إنشاء النماذج ● Logistic Regression

```
lr = LogisticRegression(
    max_iter=3000,
    class_weight='balanced'
)
```

● SVM

```
svm = SVC(
    kernel='rbf',
    class_weight='balanced'
)
```

● Random Forest

```
rf = RandomForestClassifier(
    n_estimators=300,
    random_state=42,
    class_weight='balanced'
)
```

 XGBoost

```
from xgboost import XGBClassifier

xgb = XGBClassifier(
    objective='multi:softmax',
    num_class=3,
    eval_metric='mlogloss',
    learning_rate=0.1,
    max_depth=6,
    n_estimators=200,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
)
```

8. تدريب النماذج

```
lr.fit(X_train_scaled, y_train_smote)
svm.fit(X_train_scaled, y_train_smote)

rf.fit(X_train_smote, y_train_smote)
xgb.fit(X_train_smote, y_train_smote)
```

9. التنبؤ

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=0.8, device=None, early_stopping_rounds=None,
               enable_categorical=False, eval_metric='mlogloss',
               feature_types=None, feature_weights=None, gamma=None,
               grid_search=None, importance_type=None, init_score=None,
               is_distributed=False, keep_training_memory=None,
               label_encoder=None, max_bin=255, max_cat_to_onehot=4,
               max_delta_step=0.05, max_leaf_nodes=None, min_child_weight=1,
               missing=None, monotone_constraints=None, multi_strategy='separate',
               n_estimators=200, num_parallel_tree=1, nthread=None,
               objective='multi:softmax', num_class=3,
               print_eval_method=None, random_state=42,
               scale_pos_weight=None, seed=None, silent=False,
               subsample=0.8, tree_method='auto',
               use_label_encoder=False, verbose=False,
               watchlogid=None, watchmetric=None,
               watchout=None,
               xgb_model=None,
               xgb_train=None)
```

```
y_pred_lr = lr.predict(X_test_scaled)
y_pred_svm = svm.predict(X_test_scaled)

y_pred_rf = rf.predict(X_test)
y_pred_xgb = xgb.predict(X_test)
```

10. دالة التقييم

```
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import pandas as pd

def evaluate_model(name, y_test, y_pred):
    print("\n=====")
    print(name)
    print("=====")

    print("Accuracy:", accuracy_score(y_test, y_pred))

    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, zero_division=0))

    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
```

11. تقييم كل النماذج

```
evaluate_model("Logistic Regression", y_test, y_pred_lr)
evaluate_model("SVM", y_test, y_pred_svm)
evaluate_model("Random Forest", y_test, y_pred_rf)
evaluate_model("XGBoost", y_test, y_pred_xgb)
```

```
=====
Logistic Regression
=====
Accuracy: 0.6451612903225806

Classification Report:
precision    recall  f1-score   support
```

0	0.46	0.69	0.55	48
1	0.79	0.66	0.72	166
2	0.55	0.58	0.57	65
accuracy			0.65	279
macro avg	0.60	0.64	0.61	279
weighted avg	0.68	0.65	0.65	279

Confusion Matrix:

```
[[ 33  12   3]
 [ 29 109  28]
 [ 10  17  38]]
```

=====

SVM

=====

Accuracy: 0.7526881720430108

Classification Report:

	precision	recall	f1-score	support
0	0.56	0.62	0.59	48
1	0.80	0.87	0.83	166
2	0.81	0.54	0.65	65
accuracy			0.75	279
macro avg	0.72	0.68	0.69	279
weighted avg	0.76	0.75	0.75	279

Confusion Matrix:

```
[[ 30  14   4]
 [ 17 145   4]
 [   7  23  35]]
```

=====

Random Forest

=====

Accuracy: 0.8207885304659498

Classification Report:

	precision	recall	f1-score	support
0	0.78	0.67	0.72	48
1	0.84	0.92	0.87	166
2	0.80	0.69	0.74	65

12. مقارنة النماذج

```
results = pd.DataFrame({
    "Model": ["Logistic Regression", "SVM", "Random Forest", "XGBoost"],
    "Accuracy": [
        accuracy_score(y_test, y_pred_lr),
        accuracy_score(y_test, y_pred_svm),
        accuracy_score(y_test, y_pred_rf),
        accuracy_score(y_test, y_pred_xgb)
    ]
})

print(results.sort_values(by="Accuracy", ascending=False))
```

	Model	Accuracy
3	XGBoost	0.820789
2	Random Forest	0.820789
1	SVM	0.752688
0	Logistic Regression	0.645161

Confusion Matrix

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

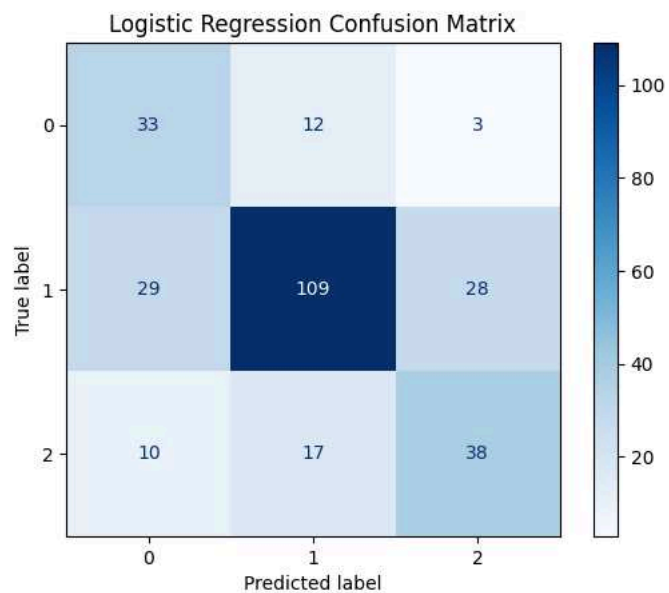
```
def plot_cm(model, X_test, y_test, title):
    y_pred = model.predict(X_test)
```

```
cm = confusion_matrix(y_test, y_pred)

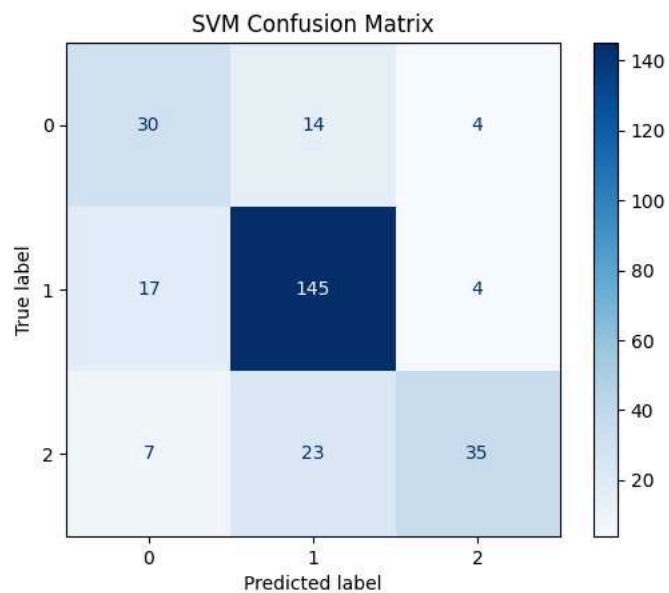
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1,2])
disp.plot(cmap='Blues', values_format='d')

plt.title(title)
plt.show()
```

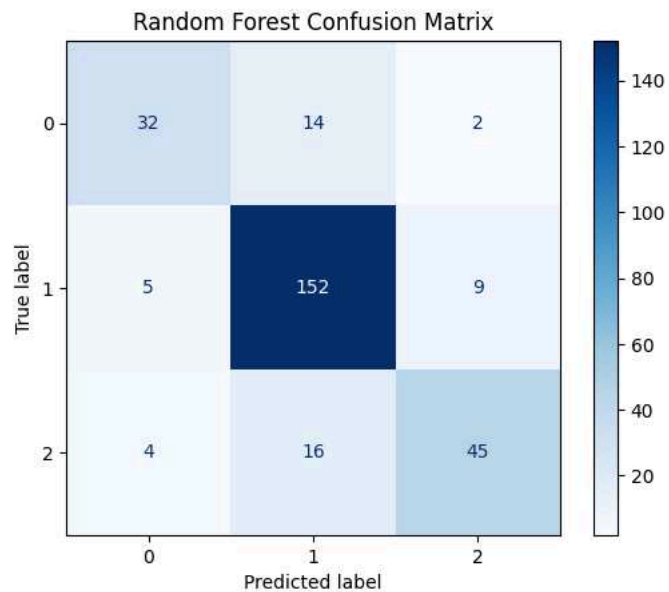
```
plot_cm(lr, X_test_scaled, y_test, "Logistic Regression Confusion Matrix")
```



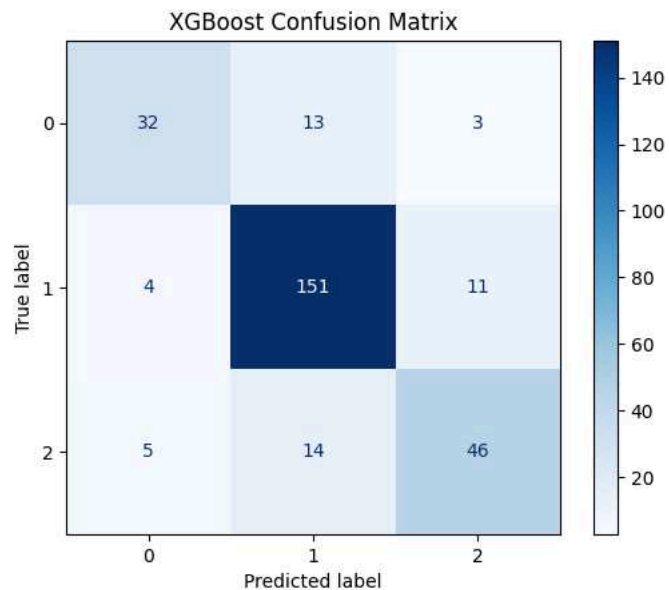
```
plot_cm(svm, X_test_scaled, y_test, "SVM Confusion Matrix")
```



```
plot_cm(rf, X_test, y_test, "Random Forest Confusion Matrix")
```



```
plot_cm(xgb, X_test, y_test, "XGBoost Confusion Matrix")
```



```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

fig, axes = plt.subplots(2, 2, figsize=(12, 10))

models = [
    (lr, X_test_scaled, "Logistic Regression"),
    (svm, X_test_scaled, "SVM"),
    (rf, X_test, "Random Forest"),
    (xgb, X_test, "XGBoost")
]

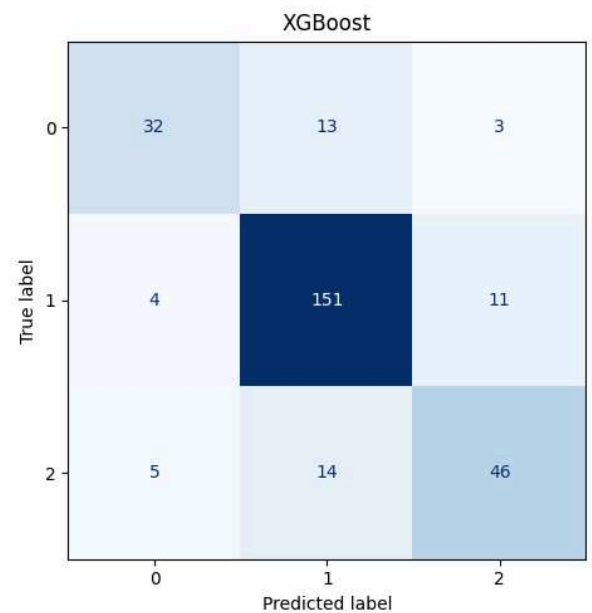
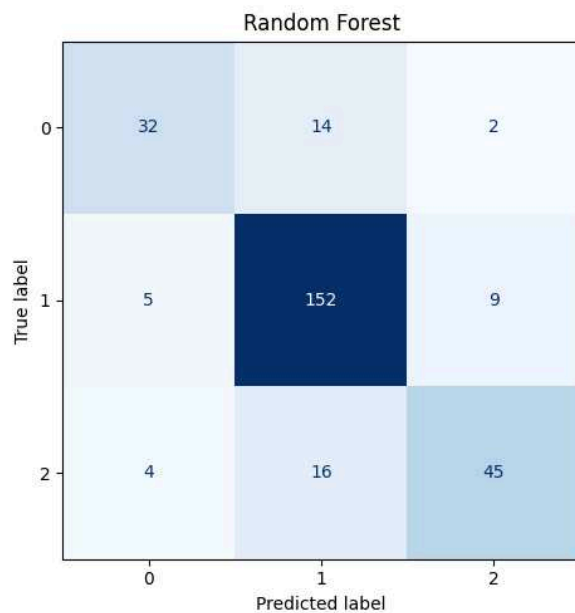
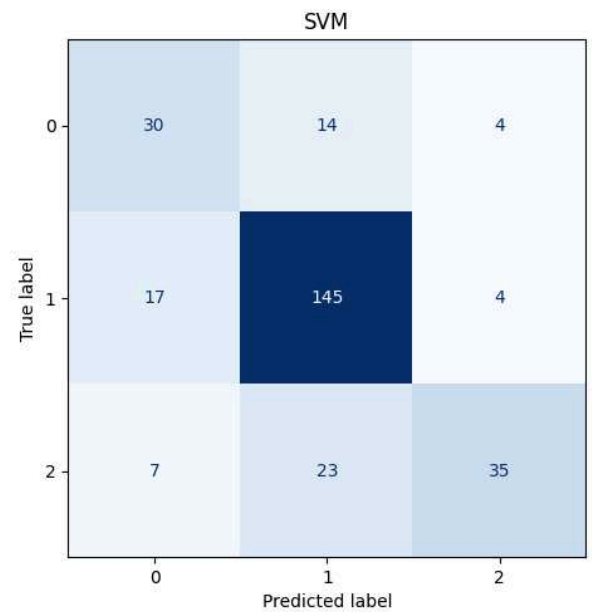
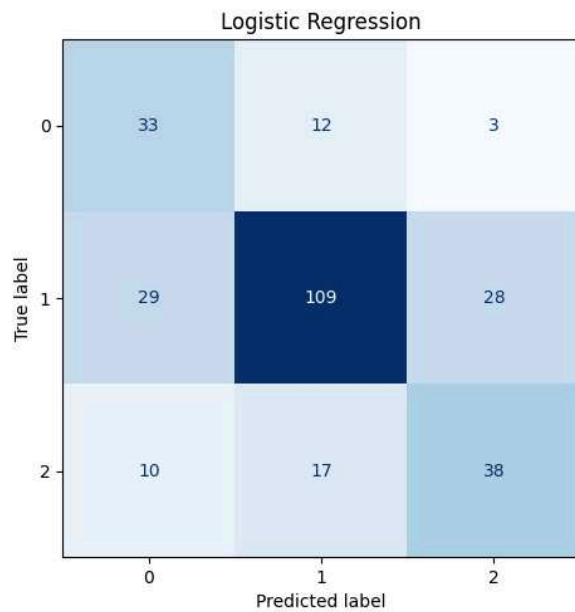
for ax, (model, X, title) in zip(axes.ravel(), models):

    y_pred = model.predict(X)
    cm = confusion_matrix(y_test, y_pred)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1,2])
    disp.plot(ax=ax, cmap='Blues', colorbar=False, values_format='d')

    ax.set_title(title)
```

```
plt.tight_layout()
plt.show()
```



Logistic Regression

أخطاء كثيرة بين الكلاسات. Accuracy = 64%.

SVM

تحسن واضح. Accuracy = 75%.

Random Forest

معظم الأرقام موجودة على القطر الرئيسي. أخطاء أقل. Accuracy = 82%.

XGBoost

أعلى قليلاً. جملة جاهزة للدكتور F1-score حقق Random Forest لكن Random Forest قريب جداً من

The Confusion Matrix shows that Random Forest and XGBoost achieved the best classification performance, as most observations were correctly classified along the diagonal. Random Forest produced fewer misclassifications across the three classes and achieved the highest Macro F1-score, making it the best-performing model in this study.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

def plot_cm_percentage(model, X_test, y_test, title, ax):

    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)

    # تحويلها لنسب مئوية
    cm_percent = cm / cm.sum(axis=1, keepdims=True) * 100

    sns.heatmap(
        cm_percent,
        annot=True,
        fmt=".1f",
        cmap="Blues",
        cbar=False,
        ax=ax,
        xticklabels=[0,1,2],
        yticklabels=[0,1,2]
    )

    accuracy = (y_pred == y_test).mean()

    ax.set_title(f"{title}\nAccuracy = {accuracy:.2f}")
    ax.set_xlabel("Predicted")
    ax.set_ylabel("Actual")
```

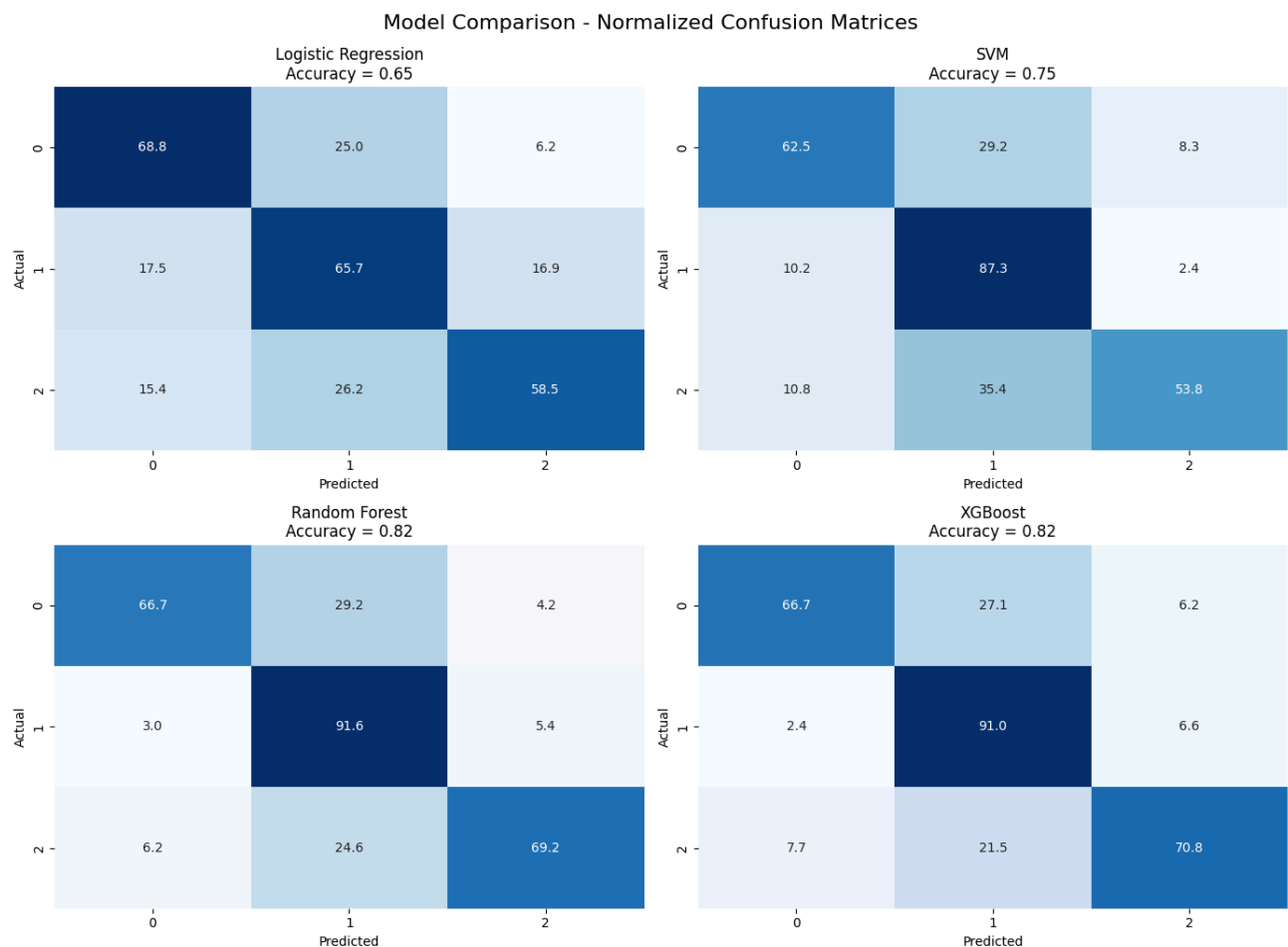
يعني حولت الأعداد إلى نسب مئوية ليسهل المقارنة بين الكلاسات. #يوضح الشكل نسبة التصنيف «Normalized Confusion Matrix العادية لأنها Confusion Matrix هذه أفضل من الد حقا أفضل أداء لأن نسبة XGBoost و Random Forest الصحيح والخطي لكل فئة. القيم الموجودة على القطر الرئيسي تمثل التصنيف الصحيح، بينما القيم خارج القطر تمثل الأخطاء. نلاحظ أن التصنيف الصحيح كانت الأعلى والأخطاء كانت الأقل مقارنة بباقي النماذج.

```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

models = [
    (lr, X_test_scaled, "Logistic Regression"),
    (svm, X_test_scaled, "SVM"),
    (rf, X_test, "Random Forest"),
    (xgb, X_test, "XGBoost")
]

for ax, (model, X, name) in zip(axes.ravel(), models):
    plot_cm_percentage(model, X, y_test, name, ax)

plt.tight_layout()
plt.suptitle("Model Comparison - Normalized Confusion Matrices", y=1.02, fontsize=16)
plt.show()
```



```
fig.savefig("confusion_matrices_comparison.png", dpi=300, bbox_inches='tight')
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import pandas as pd
```

```
def get_scores(model, X_test, y_test, name):
    y_pred = model.predict(X_test)

    return {
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision (Macro)": precision_score(y_test, y_pred, average='macro', zero_division=0),
        "Recall (Macro)": recall_score(y_test, y_pred, average='macro', zero_division=0),
        "F1-score (Macro)": f1_score(y_test, y_pred, average='macro', zero_division=0)
    }
```

```
results = []
```

```
results.append(get_scores(lr, X_test_scaled, y_test, "Logistic Regression"))
results.append(get_scores(svm, X_test_scaled, y_test, "SVM"))
results.append(get_scores(rf, X_test, y_test, "Random Forest"))
results.append(get_scores(xgb, X_test, y_test, "XGBoost"))
```

```
df_results = pd.DataFrame(results)

df_results = df_results.sort_values(by="F1-score (Macro)", ascending=False)

print(df_results)
```

	Model	Accuracy	Precision (Macro)	Recall (Macro)	\
2	Random Forest	0.820789	0.806408	0.758212	
3	XGBoost	0.820789	0.798490	0.761333	
1	SVM	0.752688	0.722071	0.678985	
0	Logistic Regression	0.645161	0.599638	0.642914	
	F1-score (Macro)				
2		0.778822			
3		0.777669			
1		0.689906			
0		0.611423			

```
df_results.style.format({
    "Accuracy": "{:.3f}",
    "Precision (Macro)": "{:.3f}",
    "Recall (Macro)": "{:.3f}",
    "F1-score (Macro)": "{:.3f}"
})
```