

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report,
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTETomek
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import warnings
from sklearn.model_selection import GridSearchCV
```

```
df = pd.read_csv('/content/drive/MyDrive/Project2/gender.csv')
```

```
df.head()
```

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long	gender
0	1	11.8	6.1	1	0	1	1	Male
1	0	14.0	5.4	0	0	1	0	Female
2	0	11.8	6.3	1	1	1	1	Male
3	0	14.4	6.1	0	1	1	1	Male
4	1	13.5	5.9	0	0	0	0	Female

Next steps: [Generate code with df](#)

[New interactive sheet](#)

```
df.isnull().sum()
```

	0
long_hair	0
forehead_width_cm	0
forehead_height_cm	0
nose_wide	0
nose_long	0
lips_thin	0
distance_nose_to_lip_long	0
gender	0

dtype: int64

```
df.shape
```

```
(5001, 8)
```

```
df.dtypes
```

```


```

	0
long_hair	int64
forehead_width_cm	float64
forehead_height_cm	float64
nose_wide	int64
nose_long	int64
lips_thin	int64
distance_nose_to_lip_long	int64
gender	int64

```
dtype: object
```

```
df.describe()
```

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long	gender
count	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000
mean	0.869626	13.181484	5.946311	0.493901	0.507898	0.493101	0.498900	0.493101
std	0.336748	1.107128	0.541268	0.500013	0.499988	0.500002	0.500049	0.500002
min	0.000000	11.400000	5.100000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	12.200000	5.500000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	13.100000	5.900000	0.000000	1.000000	0.000000	0.000000	0.000000
75%	1.000000	14.000000	6.400000	1.000000	1.000000	1.000000	1.000000	1.000000
max	1.000000	15.500000	7.100000	1.000000	1.000000	1.000000	1.000000	1.000000

```
df['gender']=df['gender'].map({ "Male":1, "Female":0}) # تحويل البيانات النصية لبيانات رقمية
```

```
df.columns
```

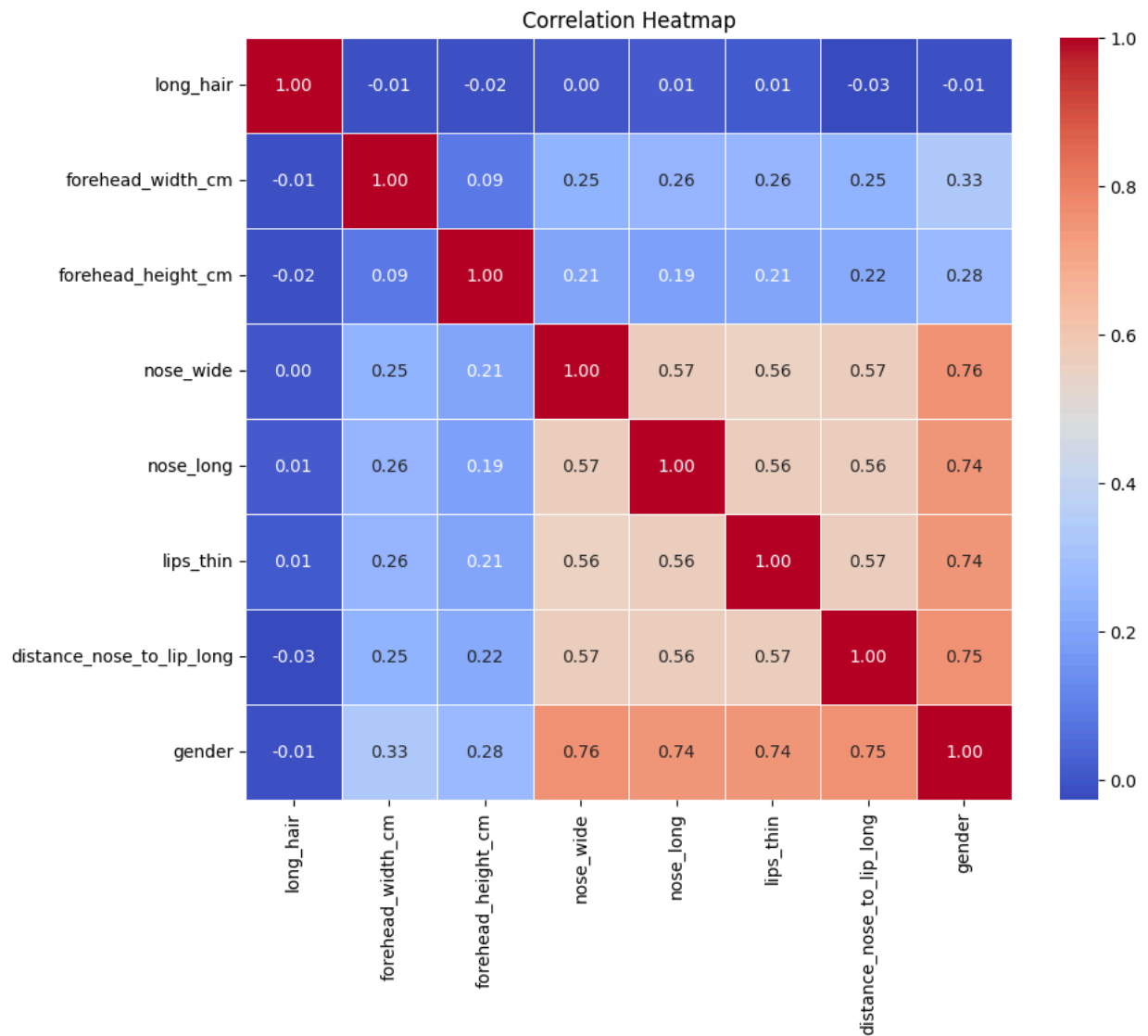
```
Index(['long_hair', 'forehead_width_cm', 'forehead_height_cm', 'nose_wide',  
      'nose_long', 'lips_thin', 'distance_nose_to_lip_long', 'gender'],  
      dtype='object')
```

```
plt.figure(figsize=(10,8))
```

```
sns.heatmap(df.corr(),annot=True , fmt=".2f" ,cmap='coolwarm', linewidths=0.5,cbar=True )
```

```
plt.title('Correlation Heatmap')
```

```
plt.show()
```

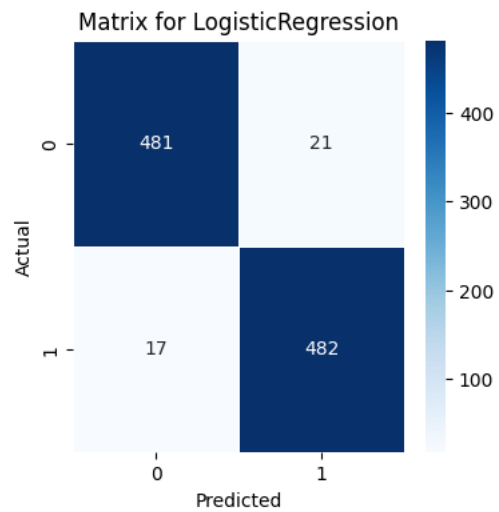


```
X=df.drop(['gender'],axis=1)
y=df['gender']
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=45)
```

```
model=LogisticRegression()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(4,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', linewidths=0.5)
plt.title('Matrix for LogisticRegression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```



Accuracy: 0.962037962037962

Precision: 0.9582504970178927

Recall: 0.9659318637274549

F1 Score: 0.9620758483033932

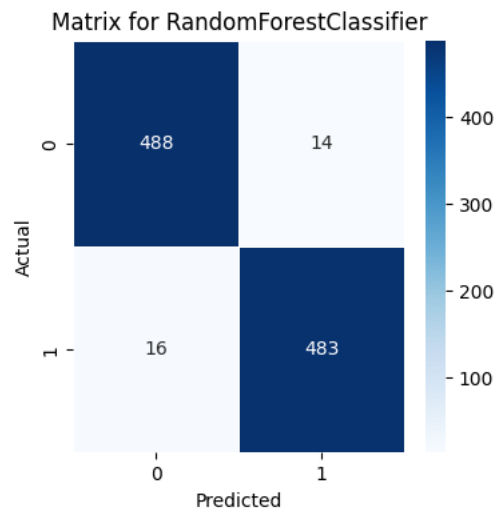
	precision	recall	f1-score	support
0	0.97	0.96	0.96	502
1	0.96	0.97	0.96	499
accuracy			0.96	1001
macro avg	0.96	0.96	0.96	1001
weighted avg	0.96	0.96	0.96	1001

```

model=RandomForestClassifier()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(4,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', linewidths=0.5)
plt.title('Matrix for RandomForestClassifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

```

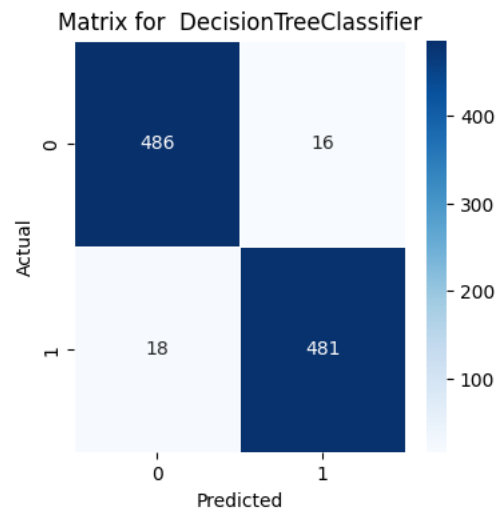


Accuracy: 0.97002997002997
Precision: 0.971830985915493
Recall: 0.9679358717434869
F1 Score: 0.9698795180722891

	precision	recall	f1-score	support
0	0.97	0.97	0.97	502
1	0.97	0.97	0.97	499
accuracy			0.97	1001
macro avg	0.97	0.97	0.97	1001
weighted avg	0.97	0.97	0.97	1001

```
model= DecisionTreeClassifier()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(4,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', linewidths=0.5)
plt.title('Matrix for DecisionTreeClassifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```



Accuracy: 0.9660339660339661

Precision: 0.9678068410462777

Recall: 0.9639278557114228

F1 Score: 0.9658634538152611

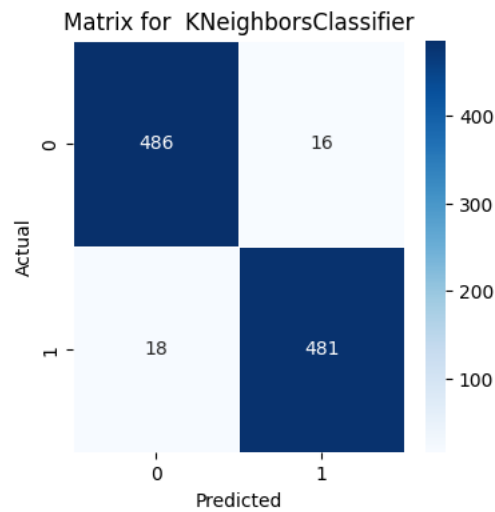
	precision	recall	f1-score	support
0	0.96	0.97	0.97	502
1	0.97	0.96	0.97	499
accuracy			0.97	1001
macro avg	0.97	0.97	0.97	1001
weighted avg	0.97	0.97	0.97	1001

```

model= KNeighborsClassifier()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(4,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', linewidths=0.5)
plt.title('Matrix for KNeighborsClassifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

```



Accuracy: 0.9660339660339661

Precision: 0.9678068410462777

Recall: 0.9639278557114228

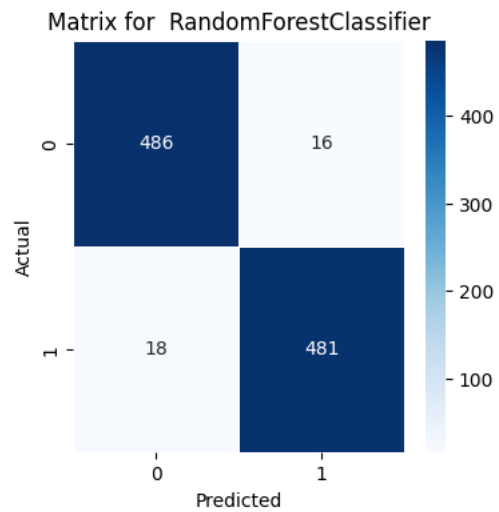
F1 Score: 0.9658634538152611

	precision	recall	f1-score	support
0	0.96	0.97	0.97	502
1	0.97	0.96	0.97	499
accuracy			0.97	1001
macro avg	0.97	0.97	0.97	1001
weighted avg	0.97	0.97	0.97	1001

```

rf_classifier = RandomForestClassifier(random_state=45)
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(estimator=rf_classifier,
    param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_rf_classifier = grid_search.best_estimator_
y_pred = best_rf_classifier.predict(X_test)
plt.figure(figsize=(4,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', linewidths=0.5)
plt.title('Matrix for RandomForestClassifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

```



Accuracy: 0.97002997002997

Precision: 0.971830985915493

Recall: 0.9679358717434869

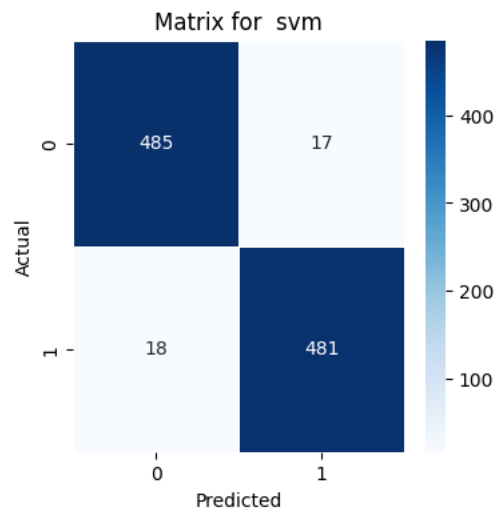
F1 Score: 0.9698795180722891

	precision	recall	f1-score	support
0	0.97	0.97	0.97	502
1	0.97	0.97	0.97	499
accuracy			0.97	1001
macro avg	0.97	0.97	0.97	1001
weighted avg	0.97	0.97	0.97	1001

```

svm_model = SVC(random_state=45)
param_grid = {
    'C': [0.1, 1],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale'],
    'degree': [3, 4, 5],
}
grid_search = GridSearchCV(estimator=svm_model,
    param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_svm = grid_search.best_estimator_
y_pred = best_svm.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(4,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', linewidths=0.5)
plt.title('Matrix for svm')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

```



Accuracy: 0.965034965034965
Precision: 0.9658634538152611
Recall: 0.9639278557114228
F1 Score: 0.9648946840521565

	precision	recall	f1-score	support
0	0.96	0.97	0.97	502
1	0.97	0.96	0.96	499
accuracy			0.97	1001
macro avg	0.97	0.97	0.97	1001
weighted avg	0.97	0.97	0.97	1001

```
ros = RandomOverSampler()
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)
print("Before:")
pd.Series(y_train).value_counts()
```

Before:

	count
gender	
1	2001
0	1999

dtype: int64

```
print("After:")
pd.Series(y_resampled).value_counts()
```

After:

	count
gender	
0	2001
1	2001

dtype: int64

```
rus = RandomUnderSampler()
X_rus, y_rus = rus.fit_resample(X_train, y_train)
print("After Undersampling:", pd.Series(y_rus).value_counts())
```

After Undersampling: gender
0 1999
1 1999
Name: count, dtype: int64

```
sm = SMOTE()
X_sm, y_sm = sm.fit_resample(X_train, y_train)
print("After SMOTE:",pd.Series(y_sm).value_counts())
```

```
After SMOTE: gender
0      2001
1      2001
Name: count, dtype: int64
```

```
smt = SMOTETomek()
X_smt, y_smt = smt.fit_resample(X_train, y_train)
print("\nAfter SMOTE + Tomek Links:")
print(pd.Series(y_smt).value_counts())
```

```
After SMOTE + Tomek Links:
gender
0      1962
1      1962
Name: count, dtype: int64
```

```
def evaluate_model(Xt, yt, method_name):
    print(f"\n===== {method_name} =====")
    rf_classifier = RandomForestClassifier(random_state=45)
    rf_classifier.fit(Xt, yt)
    rf_classifier.preds = rf_classifier.predict(X_test)
    probs = rf_classifier.predict_proba(X_test)[: , 1]
    print(classification_report(y_test, rf_classifier.preds))
    print("ROC-AUC:", roc_auc_score(y_test, probs))
```

```
evaluate_model(X_train, y_train, "Original Imbalanced Data")
```

```
ros = RandomOverSampler(random_state=45)
X_ros, y_ros = ros.fit_resample(X_train, y_train)
evaluate_model(X_ros, y_ros, "Random Oversampling")
```

```
rus = RandomUnderSampler(random_state=45)
X_rus, y_rus = rus.fit_resample(X_train, y_train)
evaluate_model(X_rus, y_rus, "Random Undersampling")
```

```
smote = SMOTE(random_state=45)
X_sm, y_sm = smote.fit_resample(X_train, y_train)
evaluate_model(X_sm, y_sm, "SMOTE Oversampling")
```

```
smt = SMOTETomek()
X_smt, y_smt = smt.fit_resample(X_train, y_train)
evaluate_model(X_smt, y_smt, "Tomek Links")
```

	precision	recall	f1-score	support
0	0.96	0.96	0.96	502
1	0.96	0.96	0.96	499
accuracy			0.96	1001
macro avg	0.96	0.96	0.96	1001
weighted avg	0.96	0.96	0.96	1001

```
ROC-AUC: 0.9935129222588603
```

```
===== Random Oversampling =====
```

	precision	recall	f1-score	support
0	0.95	0.97	0.96	502
1	0.97	0.95	0.96	499
accuracy			0.96	1001
macro avg	0.96	0.96	0.96	1001
weighted avg	0.96	0.96	0.96	1001

```
ROC-AUC: 0.9936127234548779
```

```
===== Random Undersampling =====
```

	precision	recall	f1-score	support
0	0.96	0.96	0.96	502
1	0.96	0.96	0.96	499
accuracy			0.96	1001

ROC-AUC: 0.9933013437233031

===== SMOTE Oversampling =====