

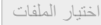
```
# 0) Basic setup
# =====
!pip install kaggle -q

from google.colab import files
import os

# Upload kaggle.json (from your Kaggle account)
print("Upload your kaggle.json file (from Kaggle > Account > Create New API Token)")
files.upload()

# Move kaggle.json to the correct place
os.makedirs("/root/.kaggle", exist_ok=True)
!mv kaggle.json /root/.kaggle/
!chmod 600 /root/.kaggle/kaggle.json
```

Upload your kaggle.json file (from Kaggle > Account > Create New API Token)

 archive (1).zip
archive (1).zip(application/x-zip-compressed) - 573312 bytes, last modified: 22100% - 2021/8/ done
 Saving archive (1).zip to archive (1) (2).zip
 mv: cannot stat 'kaggle.json': No such file or directory
 chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory

```
# 1) Download dataset from Kaggle
# We'll use: Pima Indians Diabetes Database
# https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database
# =====
!kaggle datasets download -d uciml/pima-indians-diabetes-database -p ./data

# Unzip
import zipfile

with zipfile.ZipFile("./data/pima-indians-diabetes-database.zip", "r") as z:
    z.extractall("./data")
```

Warning: Looks like you're using an outdated `kaggle` version (installed: 2.0.1), please consider upgrading to the latest version (2.0.2)
 Dataset URL: <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>
 License(s): CC0-1.0
 pima-indians-diabetes-database.zip: Skipping, found more recently modified local copy (use --force to force download)

```
# 2) Load dataset & handle missing values
# =====
import pandas as pd
import numpy as np

df = pd.read_csv("./data/diabetes.csv")
print("First 5 rows:")
```

```
display(df.head())

print("\nInfo:")
print(df.info())

print("\nMissing values per column:")
print(df.isna().sum())

# رسميًا، لكن بعض القيم = 0 تعتبر مفقودة منطقيًا NaN في هذا الداتا ما فيها
# (غير منطقي 0 = BMI مثل ضغط الدم = 0 أو)
# سنستبدل الأصفار في بعض الأعمدة بالقيمة الوسيطة (median)

cols_with_zero_as_missing = [
    "Glucose",
    "BloodPressure",
    "SkinThickness",
    "Insulin",
    "BMI"
]

for col in cols_with_zero_as_missing:
    df[col] = df[col].replace(0, np.nan)

print("\nMissing values after converting zeros to NaN:")
print(df.isna().sum())

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
df[cols_with_zero_as_missing] = imputer.fit_transform(df[cols_with_zero_as_missing])

print("\nMissing values after imputation:")
print(df.isna().sum())
```


First 5 rows:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# 3) Apply ML models + GridSearchCV on full features
# =====
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
X = df.drop("Outcome", axis=1)
y = df["Outcome"]
```

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

```
# Logistic Regression pipeline
log_reg_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=1000))
])
log_reg_param_grid = {
    "clf__C": [0.01, 0.1, 1, 10],
    "clf__penalty": ["l2"],
    "clf__solver": ["lbfgs"]
}
```

```
log_reg_grid = GridSearchCV(
    log_reg_pipeline,
    param_grid=log_reg_param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
```

```
log_reg_grid.fit(X_train, y_train)
```

```
print("=== Logistic Regression (All Features) ===")
print("Best params:", log_reg_grid.best_params_)
```

```

y_pred_lr = log_reg_grid.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Classification report:\n", classification_report(y_test, y_pred_lr))
# Random Forest pipeline
rf_pipeline = Pipeline([
    ("clf", RandomForestClassifier(random_state=42))
])

rf_param_grid = {
    "clf__n_estimators": [100, 200],
    "clf__max_depth": [None, 5, 10],
    "clf__min_samples_split": [2, 5]
}

rf_grid = GridSearchCV(
    rf_pipeline,
    param_grid=rf_param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
rf_grid.fit(X_train, y_train)

print("\n=== Random Forest (All Features) ===")
print("Best params:", rf_grid.best_params_)
y_pred_rf = rf_grid.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification report:\n", classification_report(y_test, y_pred_rf))

```

```

=== Logistic Regression (All Features) ===
Best params: {'clf__C': 1, 'clf__penalty': 'l2', 'clf__solver': 'lbfgs'}
Accuracy: 0.7077922077922078
Classification report:

```

	precision	recall	f1-score	support
0	0.75	0.82	0.78	100
1	0.60	0.50	0.55	54
accuracy			0.71	154
macro avg	0.68	0.66	0.67	154
weighted avg	0.70	0.71	0.70	154

```

=== Random Forest (All Features) ===
Best params: {'clf__max_depth': None, 'clf__min_samples_split': 2, 'clf__n_estimators': 100}
Accuracy: 0.7792207792207793
Classification report:

```

	precision	recall	f1-score	support
0	0.80	0.88	0.84	100
1	0.73	0.59	0.65	54
accuracy			0.78	154

macro avg	0.76	0.74	0.75	154
weighted avg	0.77	0.78	0.77	154

```
# 4) Two feature selection methods
# (a) SelectKBest
# (b) RFE
# =====
from sklearn.feature_selection import SelectKBest, f_classif, RFE

# (a) SelectKBest
k = 5 # number of top features
selector_kbest = SelectKBest(score_func=f_classif, k=k)
selector_kbest.fit(X_train, y_train)

kbest_mask = selector_kbest.get_support()
kbest_features = X.columns[kbest_mask]
print("\n=== SelectKBest top features ===")
print(kbest_features)

# (b) RFE with Logistic Regression
base_lr = LogisticRegression(max_iter=1000)
rfe = RFE(estimator=base_lr, n_features_to_select=5)
rfe.fit(X_train, y_train)

rfe_mask = rfe.get_support()
rfe_features = X.columns[rfe_mask]
print("\n=== RFE top features ===")
print(rfe_features)

=== SelectKBest top features ===
Index(['Glucose', 'SkinThickness', 'Insulin', 'BMI', 'Age'], dtype='object')

=== RFE top features ===
Index(['Pregnancies', 'Glucose', 'BMI', 'DiabetesPedigreeFunction', 'Age'], dtype='object')
```

```
# 5) Train ML models on selected features & evaluate
# =====

# Helper function to train & evaluate a model on a subset of features
def train_and_evaluate_on_features(feature_list, name_suffix=""):
    X_train_sub = X_train[feature_list]
    X_test_sub = X_test[feature_list]

    # Logistic Regression
    log_reg_grid.fit(X_train_sub, y_train)
    y_pred_lr_sub = log_reg_grid.predict(X_test_sub)
    print(f"\n=== Logistic Regression ({name_suffix}) ===")
```

```

print("Features:", list(feature_list))
print("Accuracy:", accuracy_score(y_test, y_pred_lr_sub))
print("Classification report:\n", classification_report(y_test, y_pred_lr_sub))

# Random Forest
rf_grid.fit(X_train_sub, y_train)
y_pred_rf_sub = rf_grid.predict(X_test_sub)
print(f"\n=== Random Forest ({name_suffix}) ===")
print("Features:", list(feature_list))
print("Accuracy:", accuracy_score(y_test, y_pred_rf_sub))
print("Classification report:\n", classification_report(y_test, y_pred_rf_sub))

# (a) Using SelectKBest features
train_and_evaluate_on_features(kbest_features, name_suffix="SelectKBest")

# (b) Using RFE features
train_and_evaluate_on_features(rfe_features, name_suffix="RFE")

```

```

=== Logistic Regression (SelectKBest) ===
Features: ['Glucose', 'SkinThickness', 'Insulin', 'BMI', 'Age']
Accuracy: 0.7207792207792207
Classification report:

```

	precision	recall	f1-score	support
0	0.77	0.82	0.79	100
1	0.62	0.54	0.57	54
accuracy			0.72	154
macro avg	0.69	0.68	0.68	154
weighted avg	0.71	0.72	0.72	154

```

=== Random Forest (SelectKBest) ===
Features: ['Glucose', 'SkinThickness', 'Insulin', 'BMI', 'Age']
Accuracy: 0.7272727272727273
Classification report:

```

	precision	recall	f1-score	support
0	0.77	0.82	0.80	100
1	0.62	0.56	0.59	54
accuracy			0.73	154
macro avg	0.70	0.69	0.69	154
weighted avg	0.72	0.73	0.72	154

```

=== Logistic Regression (RFE) ===
Features: ['Pregnancies', 'Glucose', 'BMI', 'DiabetesPedigreeFunction', 'Age']
Accuracy: 0.7012987012987013
Classification report:

```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.75	0.81	0.78	100
1	0.59	0.50	0.54	54
accuracy			0.70	154
macro avg	0.67	0.66	0.66	154
weighted avg	0.69	0.70	0.70	154

=== Random Forest (RFE) ===

Features: ['Pregnancies', 'Glucose', 'BMI', 'DiabetesPedigreeFunction', 'Age']

Accuracy: 0.7597402597402597

Classification report:

	precision	recall	f1-score	support
0	0.79	0.86	0.82	100
1	0.69	0.57	0.63	54
accuracy			0.76	154
macro avg	0.74	0.72	0.72	154
weighted avg	0.75	0.76	0.75	154

6) Notes for saving as PDF

=====

print("""

To submit:

1) Rename this notebook file to your full name, e.g.: Ghaida AL-suwaidan.ipynb

In Colab: File > Rename

""")

To submit:

1) Rename this notebook file to your full name, e.g.: Ghaida AL-suwaidan.ipynb

In Colab: File > Rename