

Dataset Link : Name : Rahaf Barakat - ID : 253002370 - Pima Indians Diabetes Database

```
In [92]: from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
In [93]: 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
```

Dataset Link : <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

```
In [94]: df = pd.read_csv('/content/drive/MyDrive/My Project 1/Project 2/diabetes.csv')
```

```
In [95]: # Load Dataset
df.head()
```

```
Out[95]:
```

	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

```
In [96]: df.dtypes
```

```
Out[96]:
```

	0
Pregnancies	int64
Glucose	int64
BloodPressure	int64
SkinThickness	int64
Insulin	int64
BMI	float64
DiabetesPedigreeFunction	float64
Age	int64
Outcome	int64

dtype: object

```
In [97]: df.columns
```

```
Out[97]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
               'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
              dtype='object')
```

```
In [98]: df.isnull().sum()
```

Out[98]:

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

```
In [99]: # Replace zero values with NaN in some columns

columns_with_zero = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

# Keep only columns that actually exist in the dataframe
existing_columns = [col for col in columns_with_zero if col in df.columns]

# Replace 0 with NaN only for existing columns
df[existing_columns] = df[existing_columns].replace(0, np.nan)

# Check missing values
df.isnull().sum()
```

Out[99]:

	0
Pregnancies	0
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

```
In [100]: # Drop columns with >30% missing values

missing_percentage = df.isnull().mean() * 100
threshold = 30

columns_to_drop = missing_percentage[missing_percentage > threshold].index

df = df.drop(columns=columns_to_drop)

print(f"Columns dropped: {list(columns_to_drop)}")
print(f"Remaining columns: {df.columns}")

Columns dropped: ['Insulin']
Remaining columns: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'BMI',
                          'DiabetesPedigreeFunction', 'Age', 'Outcome'],
                          dtype='object')
```

```
In [101]: # Encode categorical columns

label_encoder = LabelEncoder()
```

```
for column in df.select_dtypes(include=['object']).columns:
    df[column] = label_encoder.fit_transform(df[column])

df.head()
```

Out[101...

	Pregnancies	Glucose	BloodPressure	SkinThickness	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	26.6	0.351	31	0
2	8	183.0	64.0	NaN	23.3	0.672	32	1
3	1	89.0	66.0	23.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	43.1	2.288	33	1

In [102...

```
# Fill remaining missing values with mean

for column in df.select_dtypes(include=['float64', 'int64']).columns:
    mean_value = df[column].mean()
    df[column] = df[column].fillna(mean_value)
```

In [103...

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

Out[103...

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

In [104...

```
df.head()
```

Out[104...

	Pregnancies	Glucose	BloodPressure	SkinThickness	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.00000	33.6	0.627	50	1
1	1	85.0	66.0	29.00000	26.6	0.351	31	0
2	8	183.0	64.0	29.15342	23.3	0.672	32	1
3	1	89.0	66.0	23.00000	28.1	0.167	21	0
4	0	137.0	40.0	35.00000	43.1	2.288	33	1

In [105...

```
df['Outcome'].value_counts()
```

Out[105...

	count
Outcome	
0	500
1	268

dtype: int64

In [106...

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV
```

```

from sklearn.preprocessing import StandardScaler
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.preprocessing import StandardScaler

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report

import matplotlib.pyplot as plt
import seaborn as sns

```

```

In [107.. label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

```

```

In [108.. X = df.drop('Outcome', axis=1).values
y = df['Outcome'].values

```

```

In [109.. # Scale features
scaler = StandardScaler()
X = scaler.fit_transform(X)

```

```

In [110.. X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.30, random_state=42, stratify=y)

```

```

In [111.. def evaluate_model(model_name, y_true, y_pred):

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    # Create a report
    report = classification_report(y_true, y_pred)

    # Output results
    metrics = {
        'Model Name': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
        'Classification Report': report
    }

    # Plot Confusion Matrix
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=np.unique(y_true),
                yticklabels=np.unique(y_true))
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    return metrics

```

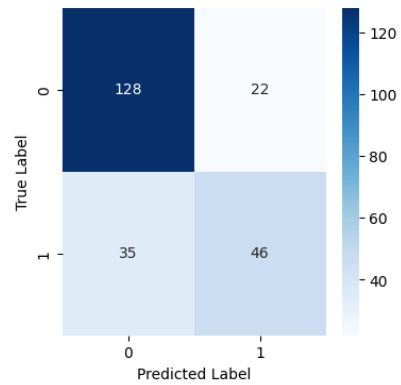
```

In [112.. rf_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)
# Train the model
rf_classifier.fit(X_train, y_train)
# Make predictions
y_pred = rf_classifier.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)

```

```
# Print report separately for better readability
else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
          f"{key}: \n{value}")
```

Confusion Matrix for RandomForestClassifier

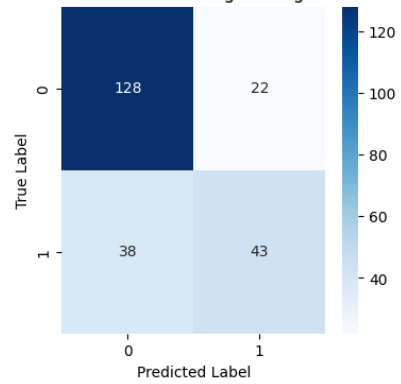


Model Name:
RandomForestClassifier
Accuracy: 0.7532
Precision: 0.7471
Recall: 0.7532
F1 Score: 0.7476

	precision	recall	f1-score	support
0	0.79	0.85	0.82	150
1	0.68	0.57	0.62	81
accuracy			0.75	231
macro avg	0.73	0.71	0.72	231
weighted avg	0.75	0.75	0.75	231

```
In [113]: LR = LogisticRegression()
# Train the model
LR.fit(X_train, y_train)
# Make predictions
y_pred = LR.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
              f"{key}: \n{value}")
```

Confusion Matrix for LogisticRegression



Model Name:

LogisticRegression

Accuracy: 0.7403

Precision: 0.7327

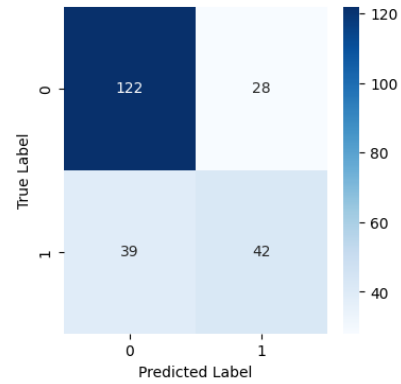
Recall: 0.7403

F1 Score: 0.7326

	precision	recall	f1-score	support
0	0.77	0.85	0.81	150
1	0.66	0.53	0.59	81
accuracy			0.74	231
macro avg	0.72	0.69	0.70	231
weighted avg	0.73	0.74	0.73	231

```
In [114... DT = DecisionTreeClassifier()
# Train the model
DT.fit(X_train, y_train)
# Make predictions
y_pred = DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

Confusion Matrix for DecisionTreeClassifier



Model Name:

DecisionTreeClassifier

Accuracy: 0.7100

Precision: 0.7024

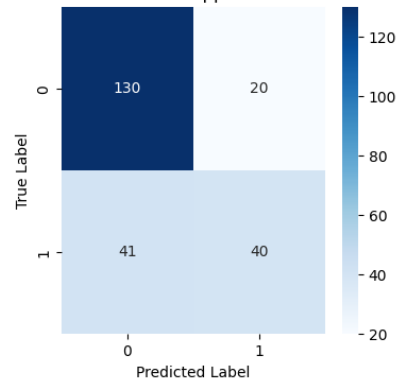
Recall: 0.7100

F1 Score: 0.7045

	precision	recall	f1-score	support
0	0.76	0.81	0.78	150
1	0.60	0.52	0.56	81
accuracy			0.71	231
macro avg	0.68	0.67	0.67	231
weighted avg	0.70	0.71	0.70	231

```
In [115]: SVM = SVC()
# Train the model
SVM.fit(X_train, y_train)
# Make predictions
y_pred = SVM.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

Confusion Matrix for Support Vector Machine



Model Name:

Support Vector Machine

Accuracy: 0.7359

Precision: 0.7274

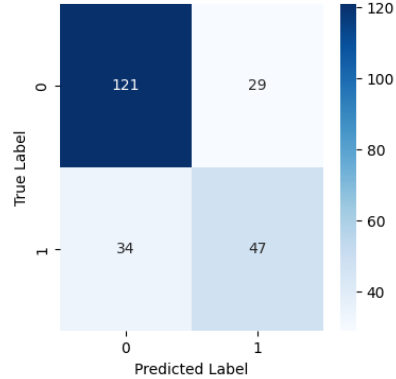
Recall: 0.7359

F1 Score: 0.7249

	precision	recall	f1-score	support
0	0.76	0.87	0.81	150
1	0.67	0.49	0.57	81
accuracy			0.74	231
macro avg	0.71	0.68	0.69	231
weighted avg	0.73	0.74	0.72	231

```
In [116... gnb = GaussianNB()
# Train the model
gnb.fit(X_train, y_train)
# Make predictions
y_pred = gnb.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Gaussian Naive Bayes', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

Confusion Matrix for Gaussian Naive Bayes

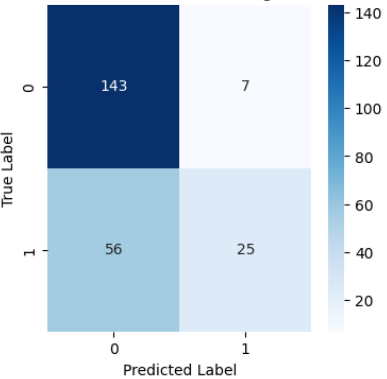


Model Name:
Gaussian Naive Bayes
Accuracy: 0.7273
Precision: 0.7238
Recall: 0.7273
F1 Score: 0.7252

	precision	recall	f1-score	support
0	0.78	0.81	0.79	150
1	0.62	0.58	0.60	81
accuracy			0.73	231
macro avg	0.70	0.69	0.70	231
weighted avg	0.72	0.73	0.73	231

```
In [117... knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
# Make predictions
y_pred = knn.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('KNeighbors', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
              f"{key}: \n{value}")
```

Confusion Matrix for KNeighbors

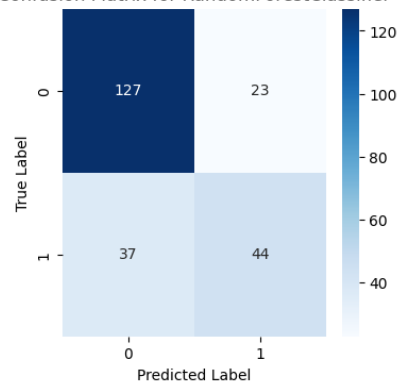


Model Name:
 KNeighbors
 Accuracy: 0.7273
 Precision: 0.7406
 Recall: 0.7273
 F1 Score: 0.6873

	precision	recall	f1-score	support
0	0.72	0.95	0.82	150
1	0.78	0.31	0.44	81
accuracy			0.73	231
macro avg	0.75	0.63	0.63	231
weighted avg	0.74	0.73	0.69	231

```
In [118]: from sklearn.model_selection import GridSearchCV
# Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
# Define the hyperparameters for grid search
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]
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier,
param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_rf_classifier = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_rf_classifier.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
              f"{key}: \n{value}")
# Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

Confusion Matrix for RandomForestClassifier



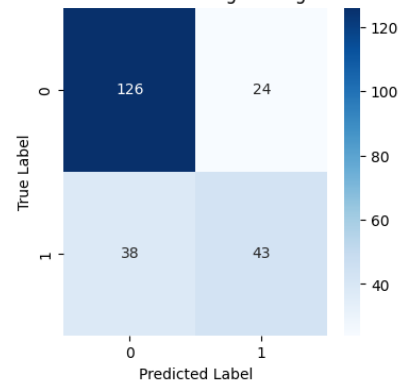
Model Name:
RandomForestClassifier
Accuracy: 0.7403
Precision: 0.7331
Recall: 0.7403
F1 Score: 0.7338

	precision	recall	f1-score	support
0	0.77	0.85	0.81	150
1	0.66	0.54	0.59	81
accuracy			0.74	231
macro avg	0.72	0.69	0.70	231
weighted avg	0.73	0.74	0.73	231

Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}

```
In [119]: # Define the Logistic Regression model
LR = LogisticRegression(random_state=42)
# Define the hyperparameters for grid search
param_grid = {
    'C': [0.01, 0.1, 1, 10], # Regularization strength
    'solver': ['liblinear', 'lbfgs'], # Solver type
    'max_iter': [100, 200, 300] # Maximum iterations for convergence
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=LR, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_LR = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_LR.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

Confusion Matrix for LogisticRegression



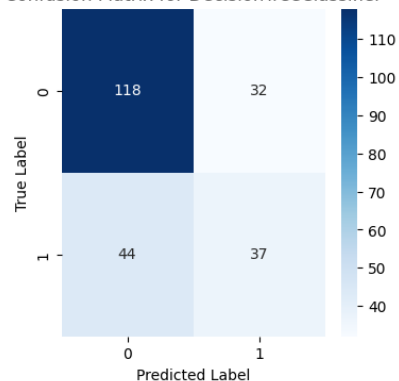
Model Name:
 LogisticRegression
 Accuracy: 0.7316
 Precision: 0.7239
 Recall: 0.7316
 F1 Score: 0.7249

	precision	recall	f1-score	support
0	0.77	0.84	0.80	150
1	0.64	0.53	0.58	81
accuracy			0.73	231
macro avg	0.71	0.69	0.69	231
weighted avg	0.72	0.73	0.72	231

Best hyperparameters found by GridSearchCV:
 {'C': 0.1, 'max_iter': 100, 'solver': 'liblinear'}

```
In [120]: # Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)
# Define the hyperparameters for grid search
param_grid = {
    'criterion': ['gini', 'entropy'], # The function to measure the quality of a split
    'max_depth': [None, 10, 20, 30], # The maximum depth of the tree
    'min_samples_split': [2, 5, 10], # The minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4], # The minimum number of samples required to be at a leaf node
    'max_features': [None, 'sqrt', 'log2'] # The number of features to consider when looking for the best split
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=DT, param_grid=param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_DT = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

Confusion Matrix for DecisionTreeClassifier



```

Model Name:
DecisionTreeClassifier
Accuracy: 0.6710
Precision: 0.6610
Recall: 0.6710
F1 Score: 0.6642

```

	precision	recall	f1-score	support
0	0.73	0.79	0.76	150
1	0.54	0.46	0.49	81
accuracy			0.67	231
macro avg	0.63	0.62	0.62	231
weighted avg	0.66	0.67	0.66	231

```

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2}

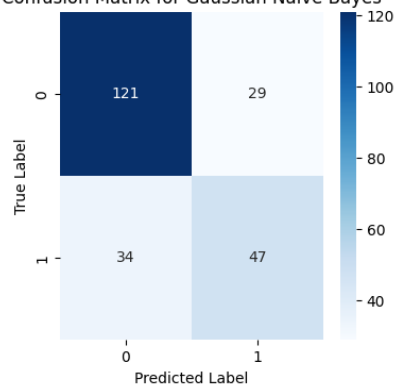
```

```

In [121]: # Define the Gaussian Naive Bayes model
gnb = GaussianNB()
# Define the hyperparameters for grid search
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6] # Smoothing parameter
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=gnb, param_grid=param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_gnb = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_gnb.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Gaussian Naive Bayes', y_test,
    y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
            f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for Gaussian Naive Bayes



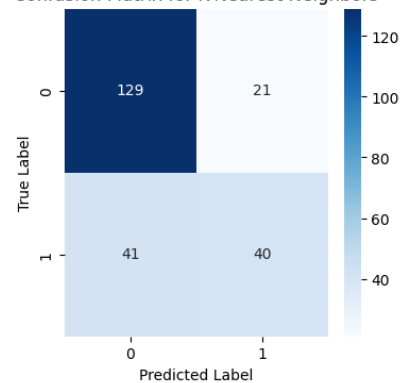
Model Name:
 Gaussian Naive Bayes
 Accuracy: 0.7273
 Precision: 0.7238
 Recall: 0.7273
 F1 Score: 0.7252

	precision	recall	f1-score	support
0	0.78	0.81	0.79	150
1	0.62	0.58	0.60	81
accuracy			0.73	231
macro avg	0.70	0.69	0.70	231
weighted avg	0.72	0.73	0.73	231

Best hyperparameters found by GridSearchCV:
 {'var_smoothing': 1e-09}

```
In [122]: # Define the K-Nearest Neighbors model
knn = KNeighborsClassifier()
# Define the hyperparameters for grid search
param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9], # Number of neighbors to use
    'weights': ['uniform', 'distance'], # Weight function used in prediction
    'metric': ['euclidean', 'manhattan'], # Distance metric
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'] # Algorithm to compute the nearest neighbors
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_knn = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_knn.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('K-Nearest Neighbors', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

Confusion Matrix for K-Nearest Neighbors



Model Name:
K-Nearest Neighbors
Accuracy: 0.7316
Precision: 0.7227
Recall: 0.7316
F1 Score: 0.7211

	precision	recall	f1-score	support
0	0.76	0.86	0.81	150
1	0.66	0.49	0.56	81
accuracy			0.73	231
macro avg	0.71	0.68	0.68	231
weighted avg	0.72	0.73	0.72	231

Best hyperparameters found by GridSearchCV:
{'algorithm': 'auto', 'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'uniform'}

In [123..

```
# Define the Support Vector Machine model
SVM = SVC(random_state=42)

# Smaller hyperparameter grid to make GridSearch feasible
param_grid = {
    'C': [0.1, 1],          # Reduced number of values
    'kernel': ['linear', 'rbf'], # Removed 'poly' to save time
    'gamma': ['scale'],      # Only one option
    'class_weight': [None]   # Single option
}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=SVM,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best estimator
best_SVM = grid_search.best_estimator_

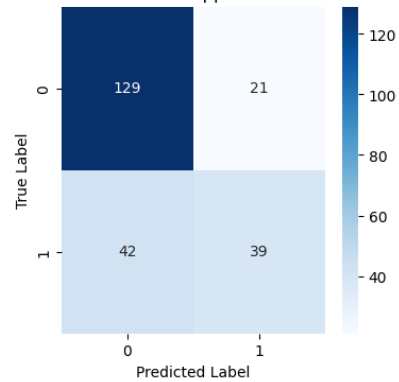
# Make predictions
y_pred = best_SVM.predict(X_test)

# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test, y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")

# Print the best hyperparameters
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

Confusion Matrix for Support Vector Machine



Model Name:

Support Vector Machine

Accuracy: 0.7273

Precision: 0.7178

Recall: 0.7273

F1 Score: 0.7159

	precision	recall	f1-score	support
0	0.75	0.86	0.80	150
1	0.65	0.48	0.55	81
accuracy			0.73	231
macro avg	0.70	0.67	0.68	231
weighted avg	0.72	0.73	0.72	231

Best hyperparameters found by GridSearchCV:

{'C': 1, 'class_weight': None, 'gamma': 'scale', 'kernel': 'linear'}

```
In [124... 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.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, chi2, RFE
```

```
In [164... X = df.drop('Outcome', axis=1)
y = df['Outcome']
```

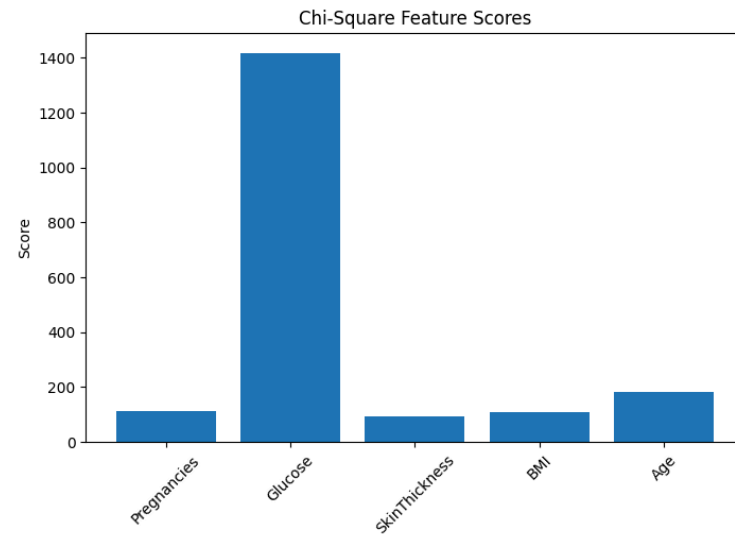
```
In [165... # TRAIN/TEST SPLIT
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
```

```
In [166... # FEATURE SELECTION 1: CHI-SQUARE

# Chi-Square requires non-negative values
chi_selector = SelectKBest(score_func=chi2, k=5)
X_chi = chi_selector.fit_transform(X, y)
selected_features_chi = X.columns[chi_selector.get_support()]
print("Chi-Square Selected Features:", list(selected_features_chi))
```

Chi-Square Selected Features: ['Pregnancies', 'Glucose', 'SkinThickness', 'BMI', 'Age']

```
In [167... # Plot Chi-Square scores
plt.figure(figsize=(8,5))
plt.bar(selected_features_chi, chi_selector.scores_[chi_selector.get_support()])
plt.title("Chi-Square Feature Scores")
plt.ylabel("Score")
plt.xticks(rotation=45)
plt.show()
```



```
In [175... # TRAIN/TEST SPLIT

X_train_chi, X_test_chi, y_train, y_test = train_test_split(
    X[selected_features_chi], y, test_size=0.2, random_state=42, stratify=y
)

X_train_scaled, X_test_scaled, _, _ = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
```

```
In [169... # SCALE FEATURES FOR MODELS AND RFE
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
```

```
In [176... # FEATURE SELECTION 2: RFE (LogisticRegression)

# Scaling only for RFE
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

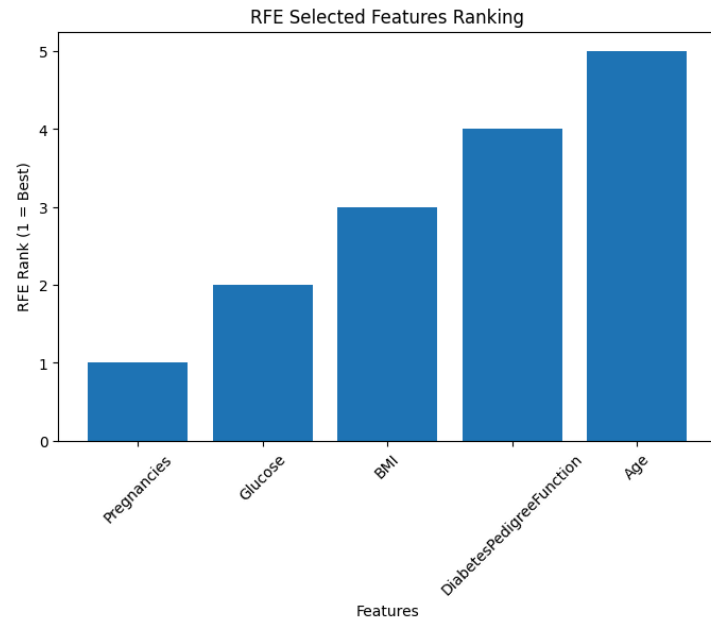
X_train_scaled, X_test_scaled, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

lr_for_rfe = LogisticRegression(max_iter=500)
rfe_selector = RFE(lr_for_rfe, n_features_to_select=5)
X_train_rfe = rfe_selector.fit_transform(X_train_scaled, y_train)
X_test_rfe = rfe_selector.transform(X_test_scaled)
selected_features_rfe = X_scaled.columns[rfe_selector.get_support()]
print("RFE Selected Features:", list(selected_features_rfe))
```

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

In [177... `# Plot RFE feature ranking`

```
plt.figure(figsize=(8,5))
plt.bar(X_scaled.columns[rfe_selector.get_support()], np.arange(1, sum(rfe_selector.get_support()+1)))
plt.title("RFE Selected Features Ranking")
plt.xlabel("Features")
plt.ylabel("RFE Rank (1 = Best)")
plt.xticks(rotation=45)
plt.show()
```



In [178... `# Define models (with class_weight='balanced' where needed)`

```
models = {
    "Logistic Regression": LogisticRegression(max_iter=500, class_weight='balanced'),
    "Decision Tree": DecisionTreeClassifier(random_state=42, class_weight='balanced'),
    "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42, class_weight='balanced'),
    "SVM": SVC(kernel='rbf', random_state=42, class_weight='balanced'),
    "KNN": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB()
}
```

In [179... `# TRAIN & EVALUATE ON CHI-SQUARE FEATURES`

```
print("\n=== MODELS ON CHI-SQUARE FEATURES ===")
X_train_chi, X_test_chi, y_train, y_test = train_test_split(
    X[selected_features_chi], y, test_size=0.2, random_state=42, stratify=y
)

for name, model in models.items():
    model.fit(X_train_chi, y_train)
    y_pred = model.predict(X_test_chi)
    print(f"\n--- {name} ---")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"Precision: {precision_score(y_test, y_pred, zero_division=0):.4f}")
    print(f"Recall: {recall_score(y_test, y_pred, zero_division=0):.4f}")
    print(classification_report(y_test, y_pred, zero_division=0))
```

=== MODELS ON CHI-SQUARE FEATURES ===

--- Logistic Regression ---

Accuracy: 0.7078

Precision: 0.5714

Recall: 0.6667

	precision	recall	f1-score	support
0	0.80	0.73	0.76	100
1	0.57	0.67	0.62	54
accuracy			0.71	154
macro avg	0.69	0.70	0.69	154
weighted avg	0.72	0.71	0.71	154

--- Decision Tree ---

Accuracy: 0.7078

Precision: 0.5763

Recall: 0.6296

	precision	recall	f1-score	support
0	0.79	0.75	0.77	100
1	0.58	0.63	0.60	54
accuracy			0.71	154
macro avg	0.68	0.69	0.69	154
weighted avg	0.71	0.71	0.71	154

--- Random Forest ---

Accuracy: 0.7403

Precision: 0.6346

Recall: 0.6111

	precision	recall	f1-score	support
0	0.79	0.81	0.80	100
1	0.63	0.61	0.62	54
accuracy			0.74	154
macro avg	0.71	0.71	0.71	154
weighted avg	0.74	0.74	0.74	154

--- SVM ---

Accuracy: 0.7078

Precision: 0.5714

Recall: 0.6667

	precision	recall	f1-score	support
0	0.80	0.73	0.76	100
1	0.57	0.67	0.62	54
accuracy			0.71	154
macro avg	0.69	0.70	0.69	154
weighted avg	0.72	0.71	0.71	154

--- KNN ---

Accuracy: 0.7208

Precision: 0.6000

Recall: 0.6111

	precision	recall	f1-score	support
0	0.79	0.78	0.78	100
1	0.60	0.61	0.61	54
accuracy			0.72	154
macro avg	0.69	0.70	0.69	154
weighted avg	0.72	0.72	0.72	154

--- Naive Bayes ---

Accuracy: 0.7013

Precision: 0.5741

Recall: 0.5741

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

0	0.77	0.77	0.77	100
1	0.57	0.57	0.57	54
accuracy			0.70	154
macro avg	0.67	0.67	0.67	154
weighted avg	0.70	0.70	0.70	154

In [180]: *# Train & Evaluate Models using RFE features*

```
print("\n=== MODELS ON RFE FEATURES ===")
for name, model in models.items():
    model.fit(X_train_rfe, y_train)
    y_pred = model.predict(X_test_rfe)
    print(f"\n--- {name} ---")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"Precision: {precision_score(y_test, y_pred, zero_division=0):.4f}")
    print(f"Recall: {recall_score(y_test, y_pred, zero_division=0):.4f}")
    print(classification_report(y_test, y_pred, zero_division=0))
```

=== MODELS ON RFE FEATURES ===

--- Logistic Regression ---

Accuracy: 0.7338

Precision: 0.6032

Recall: 0.7037

	precision	recall	f1-score	support
0	0.82	0.75	0.79	100
1	0.60	0.70	0.65	54
accuracy			0.73	154
macro avg	0.71	0.73	0.72	154
weighted avg	0.75	0.73	0.74	154

--- Decision Tree ---

Accuracy: 0.7597

Precision: 0.6667

Recall: 0.6296

	precision	recall	f1-score	support
0	0.81	0.83	0.82	100
1	0.67	0.63	0.65	54
accuracy			0.76	154
macro avg	0.74	0.73	0.73	154
weighted avg	0.76	0.76	0.76	154

--- Random Forest ---

Accuracy: 0.7792

Precision: 0.7083

Recall: 0.6296

	precision	recall	f1-score	support
0	0.81	0.86	0.83	100
1	0.71	0.63	0.67	54
accuracy			0.78	154
macro avg	0.76	0.74	0.75	154
weighted avg	0.78	0.78	0.78	154

--- SVM ---

Accuracy: 0.7597

Precision: 0.6232

Recall: 0.7963

	precision	recall	f1-score	support
0	0.87	0.74	0.80	100
1	0.62	0.80	0.70	54
accuracy			0.76	154
macro avg	0.75	0.77	0.75	154
weighted avg	0.78	0.76	0.76	154

--- KNN ---

Accuracy: 0.7468

Precision: 0.6531

Recall: 0.5926

	precision	recall	f1-score	support
0	0.79	0.83	0.81	100
1	0.65	0.59	0.62	54
accuracy			0.75	154
macro avg	0.72	0.71	0.72	154
weighted avg	0.74	0.75	0.74	154

--- Naive Bayes ---

Accuracy: 0.7338

Precision: 0.6226

Recall: 0.6111

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

0	0.79	0.80	0.80	100
1	0.62	0.61	0.62	54
accuracy			0.73	154
macro avg	0.71	0.71	0.71	154
weighted avg	0.73	0.73	0.73	154