**Mini Project

Problem Statement: Implement a mini-project involving a real-world ML task.

Suggested topics: Weather prediction, Stock price trend analysis, Heart disease prediction, Fake news detection

15

Activities:

Data collection & cleaning

Model development

Present findings**

Heart Disease Prediction Using Machine Learning

Problem Statement

Predict whether a patient has heart disease based on clinical and demographic features such as age, cholesterol, blood pressure, and heart rate.

```
# Step 1: Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Step 2: Load Dataset
# Make sure heart-disease.csv is in your current directory
df = pd.read_csv("heart-disease.csv")
```

```
# Step 3: Basic Inspection
print("✅ Dataset Loaded Successfully!")
print("Shape of data:", df.shape)
print("\nFirst 5 rows:\n", df.head())
```

```
✅ Dataset Loaded Successfully!
Shape of data: (303, 14)

First 5 rows:
    age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

```
# Step 4: Check for Missing Values
print("\nMissing Values:\n", df.isnull().sum())
```

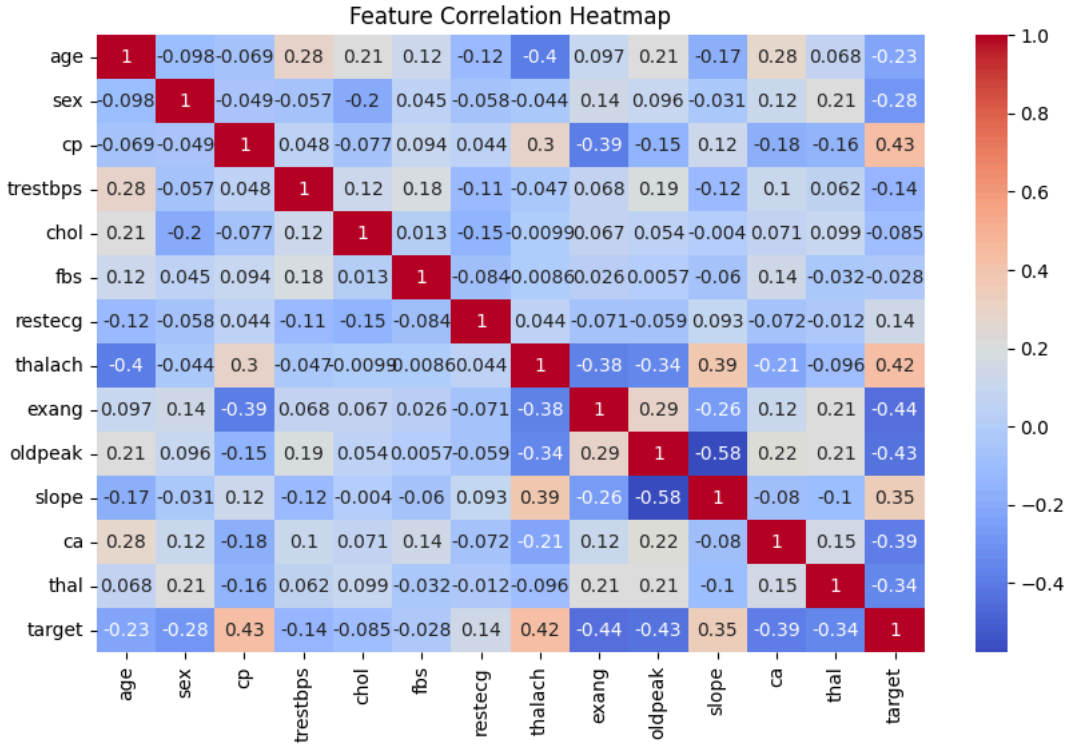
```
Missing Values:
 age         0
sex          0
cp           0
trestbps     0
chol         0
fbs          0
```

```
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

```
# Step 5: Data Info and Statistics
print("\nData Types:\n")
print(df.info())
print("\nStatistical Summary:\n")
print(df.describe())
```

```
Data Types:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
None

Statistical Summary:

              age         sex          cp    trestbps        chol         fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000   
mean    54.366337    0.683168    0.966997  131.623762  246.264026    0.148515   
std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356198   
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000000   
25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000000   
50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000000   
75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000000   
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000   

          restecg     thalach       exang     oldpeak       slope          ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000   
mean     0.528053  149.646865    0.326733    1.039604    1.399340    0.729373   
std      0.525860   22.905161    0.469794    1.161075    0.616226    1.022606   
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000   
25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000000   
50%      1.000000  153.000000    0.000000    0.800000    1.000000    0.000000   
75%      1.000000  166.000000    1.000000    1.600000    2.000000    1.000000   
max      2.000000  202.000000    1.000000    6.200000    2.000000    4.000000   

             thal      target  
count  303.000000  303.000000  
mean     2.313531    0.544554  
std      0.612277    0.498835  
min      0.000000    0.000000  
25%      2.000000    0.000000  
50%      2.000000    1.000000  
75%      3.000000    1.000000  
max      3.000000    1.000000
```

```
# Step 6: Correlation Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()
```




```
# ========================================
# MODEL DEVELOPMENT
# ========================================
# Step 7: Split Features and Target
X = df.drop('target', axis=1)
y = df['target']

# Step 8: 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
)

# Step 9: Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 10: Initialize Models
lr = LogisticRegression(max_iter=1000, random_state=42)
rf = RandomForestClassifier(random_state=42)

# Step 11: Train Models
lr.fit(X_train, y_train)
rf.fit(X_train, y_train)
```

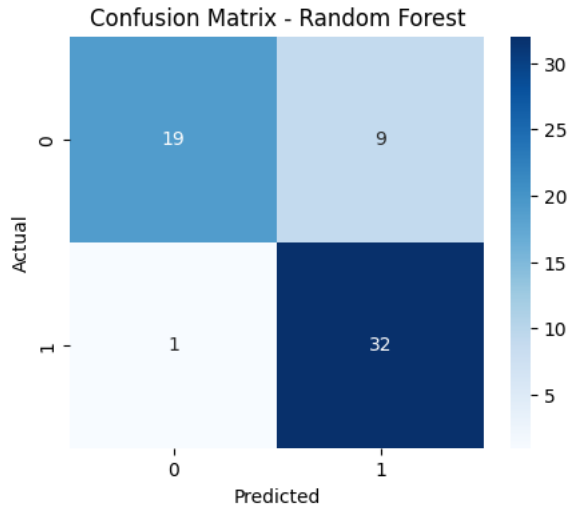
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# ========================================
# MODEL EVALUATION
# ========================================
```

```
# Step 12: Predictions
y_pred_lr = lr.predict(X_test)
y_pred_rf = rf.predict(X_test)

# Step 13: Evaluate Models
print("\n=== Logistic Regression ===")
print("Accuracy:", round(accuracy_score(y_test, y_pred_lr), 4))
print(classification_report(y_test, y_pred_lr))

print("\n=== Random Forest ===")
print("Accuracy:", round(accuracy_score(y_test, y_pred_rf), 4))
print(classification_report(y_test, y_pred_rf))

# Step 14: Confusion Matrix - Random Forest
```

```
cm = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
=== Logistic Regression ===
Accuracy: 0.8033
              precision    recall  f1-score   support

           0       0.86      0.68      0.76        28
           1       0.77      0.91      0.83        33

    accuracy                           0.80        61
   macro avg       0.82      0.79      0.80        61
weighted avg       0.81      0.80      0.80        61


=== Random Forest ===
Accuracy: 0.8361
              precision    recall  f1-score   support

           0       0.95      0.68      0.79        28
           1       0.78      0.97      0.86        33

    accuracy                           0.84        61
   macro avg       0.87      0.82      0.83        61
weighted avg       0.86      0.84      0.83        61
```




```
# ==========================================
# HYPERPARAMETER TUNING (GridSearchCV)
# ==========================================

# Step 15: Define Parameter Grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Step 16: Apply GridSearch with 5-Fold Cross Validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=kfold,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

▸ **GridSearchCV**

▸ **best_estimator_: RandomForestClassifier**

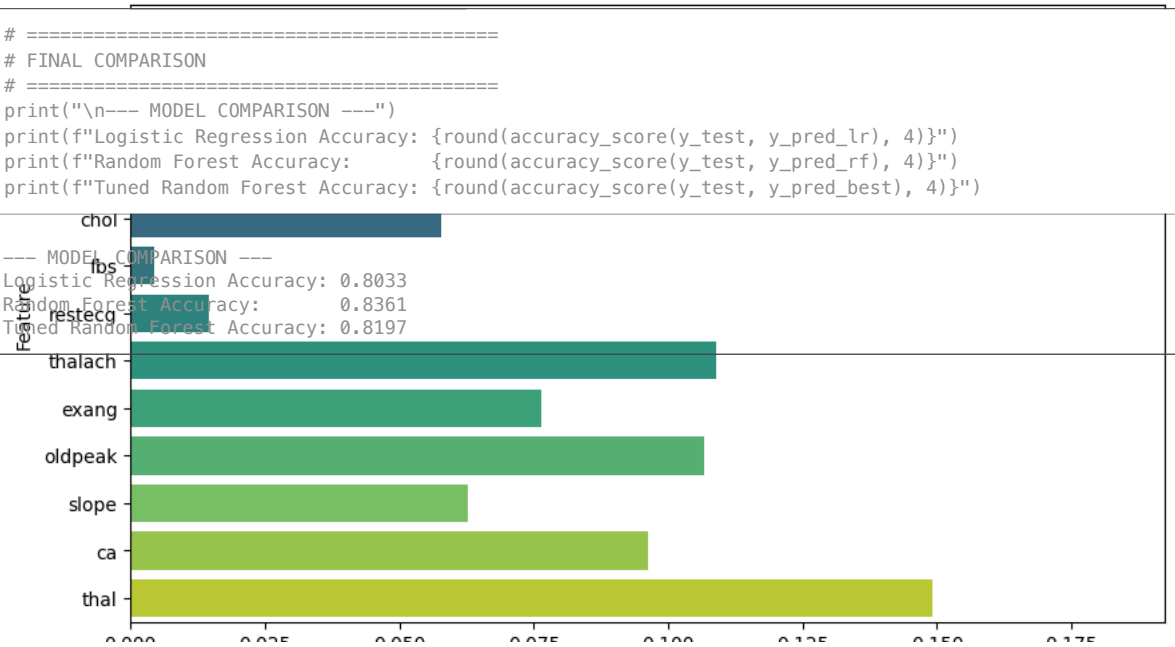
▸ RandomForestClassifier

```
# Step 17: Display Best Parameters and Accuracy
print("\nBest Parameters from Grid Search:")
print(grid_search.best_params_)
print("Best Cross-Validation Accuracy:", round(grid_search.best_score_, 4))

# Step 18: Evaluate Best Model
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)

print("\n=== Tuned Random Forest ===")
print("Accuracy:", round(accuracy_score(y_test, y_pred_best), 4))
print(classification_report(y_test, y_pred_best))
```

```
Best Parameters from Grid Search:
{'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
Best Cross-Validation Accuracy: 0.8551

=== Tuned Random Forest ===
Accuracy: 0.8197
              precision    recall  f1-score   support

           0       0.95      0.64      0.77        28
           1       0.76      0.97      0.85        33

    accuracy                           0.82        61
   macro avg       0.85      0.81      0.81        61
weighted avg       0.85      0.82      0.81        61
```

```
# ========================================
# FEATURE IMPORTANCE VISUALIZATION
# ========================================

importances = best_model.feature_importances_
features = X.columns

plt.figure(figsize=(10,6))
sns.barplot(x=importances, y=features, palette='viridis')
plt.title("Feature Importance - Tuned Random Forest")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.show()
```

```
/tmp/ipython-input-641666014.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable t

  sns.barplot(x=importances, y=features, palette='viridis')
```




```
# ========================================
# FINAL COMPARISON
# ========================================
print("\n--- MODEL COMPARISON ---")
print(f"Logistic Regression Accuracy: {round(accuracy_score(y_test, y_pred_lr), 4)}")
print(f"Random Forest Accuracy:       {round(accuracy_score(y_test, y_pred_rf), 4)}")
print(f"Tuned Random Forest Accuracy: {round(accuracy_score(y_test, y_pred_best), 4)}")
```

```
--- MODEL COMPARISON ---
Logistic Regression Accuracy: 0.8033
Random Forest Accuracy:       0.8361
Tuned Random Forest Accuracy: 0.8197
```