Welcome to Colab!

Explore the Gemini API

The Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be multimodal, so you can reason seamlessly across text, images, code, and audio.

How to get started

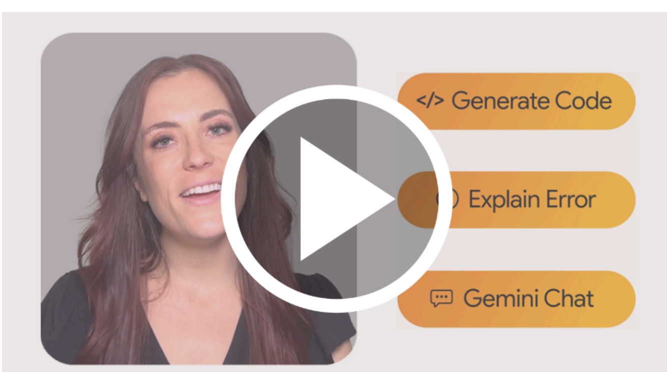
1. Go to Google AI Studio and log in with your Google account.
2. Create an API key.
3. Use a quickstart for Python, or call the REST API using curl.

Explore use cases

- Create a marketing campaign
- Analyze audio recordings
- Use System instructions in chat

To learn more, check out the Gemini cookbook or visit the Gemini API documentation.

Colab now has AI features powered by Gemini. The video below provides information on how to use these features, whether you're new to Python, or a seasoned veteran.



Load the dataset

```
# Load the dataset
file_path = "/content/heart.csv"  # Ensure the correct p
df = pd.read_csv(file_path)

# Display the first few rows
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thala
0	63	1	3	145	233	1	0	1
1	37	1	2	130	250	0	1	1
2	41	0	1	130	204	0	0	1
3	56	1	1	120	236	0	1	1
4	57	0	0	120	354	0	1	1

Next steps: code df recommended interactive

Summary Statistics

```
# Display information about the dataset
df.info()

# Summary statistics for numerical columns
df.describe()
```

```
<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
```

	age	sex	cp	trestbps
count	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762
std	9.082101	0.466011	1.032052	17.538143
min	29.000000	0.000000	0.000000	94.000000
25%	47.500000	0.000000	0.000000	120.000000
50%	55.000000	1.000000	1.000000	130.000000
75%	61.000000	1.000000	2.000000	140.000000
max	77.000000	1.000000	3.000000	200.000000

Check for Missing Values

```
# Check for missing values
missing_values = df.isnull().sum()
missing_values
```

	0
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

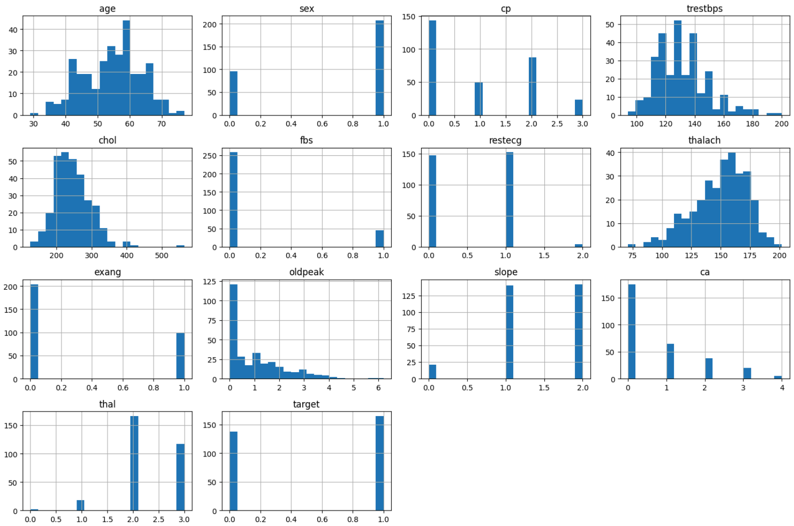
dtype: int64

Data Visualizations a. Distribution of the Target Variable

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x="target", data=df)
plt.title("Distribution of Target Variable")
plt.show()
```

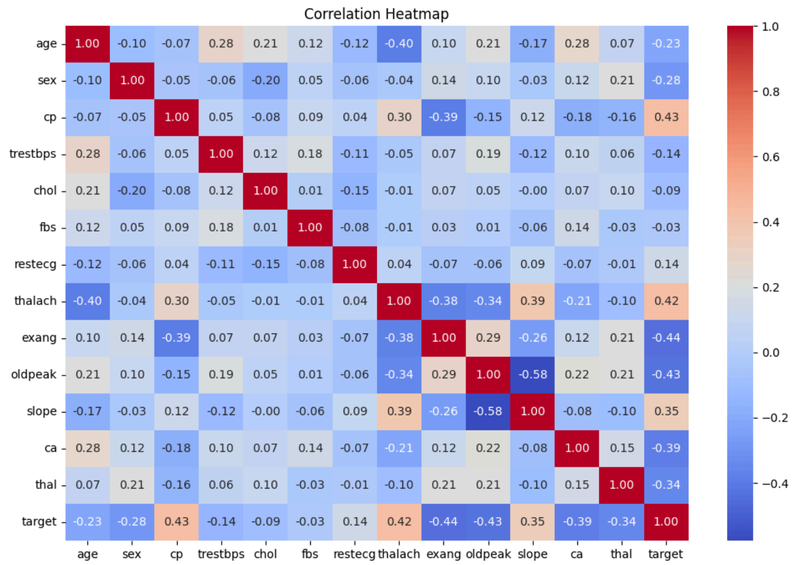
b. Histograms for Numerical Features

```
df.hist(figsize=(15, 10), bins=20)
plt.tight_layout()
plt.show()
```



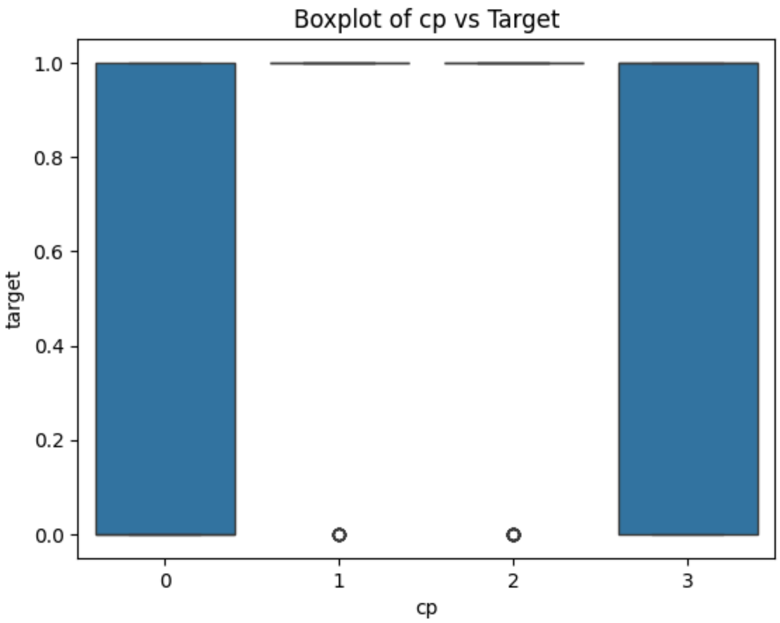
c. Correlation Heatmap

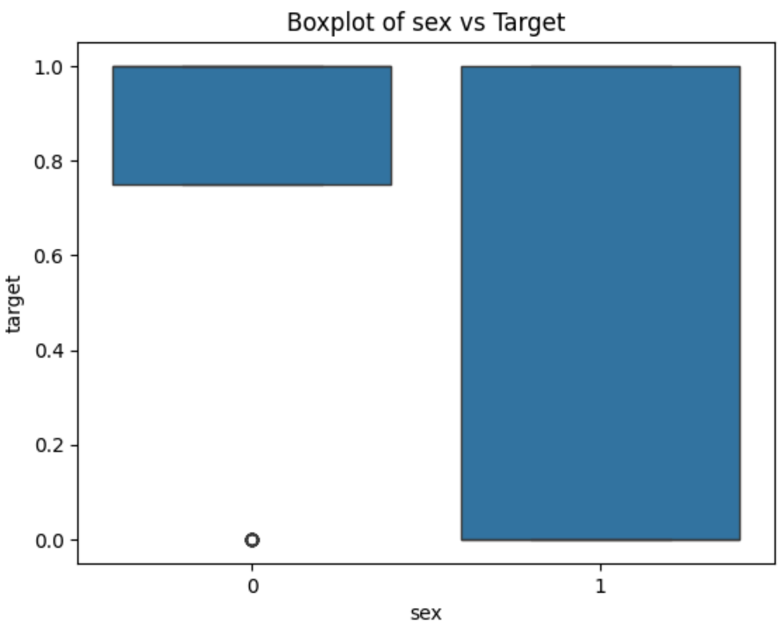
```
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwa
plt.title("Correlation Heatmap")
plt.show()
```

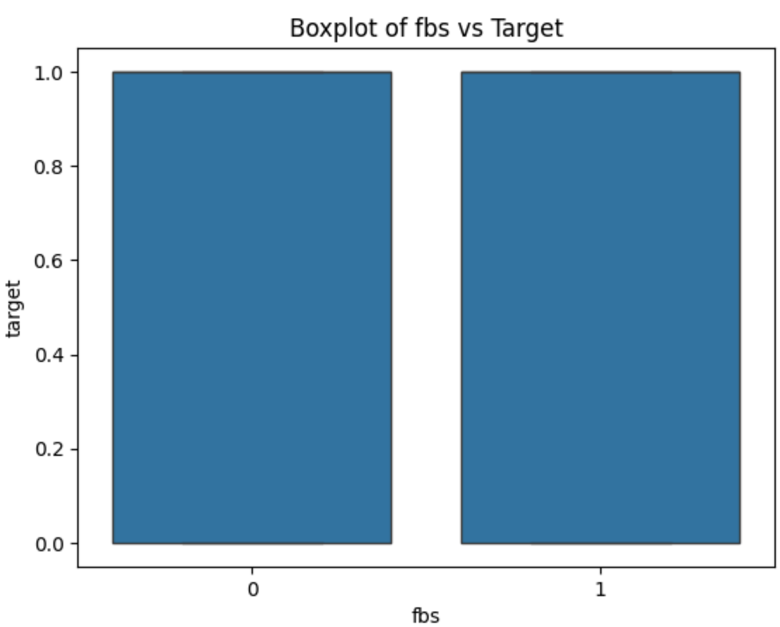


d. Boxplots for Categorical Variables

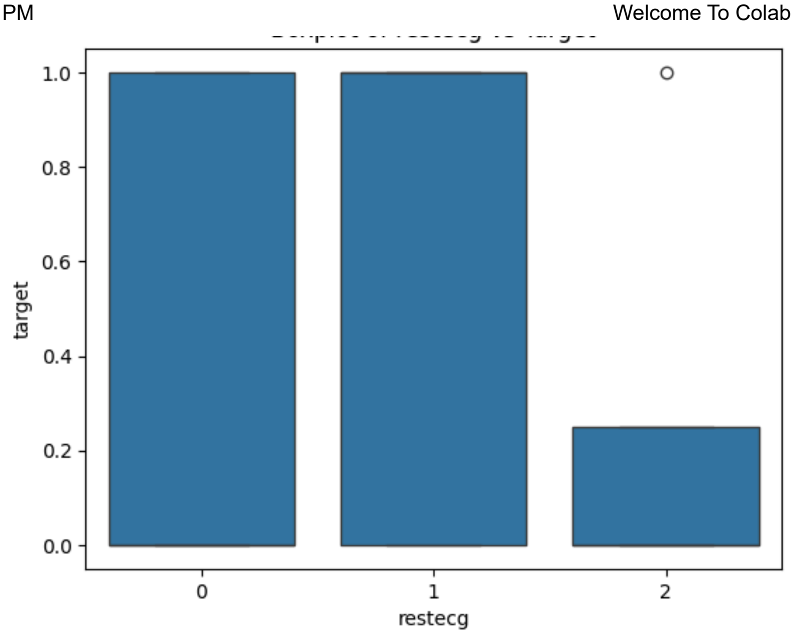
```
categorical_features = ["cp", "sex", "fbs", "restecg", "

for feature in categorical_features:
    sns.boxplot(x=feature, y="target", data=df)
    plt.title(f"Boxplot of {feature} vs Target")
    plt.show()
```

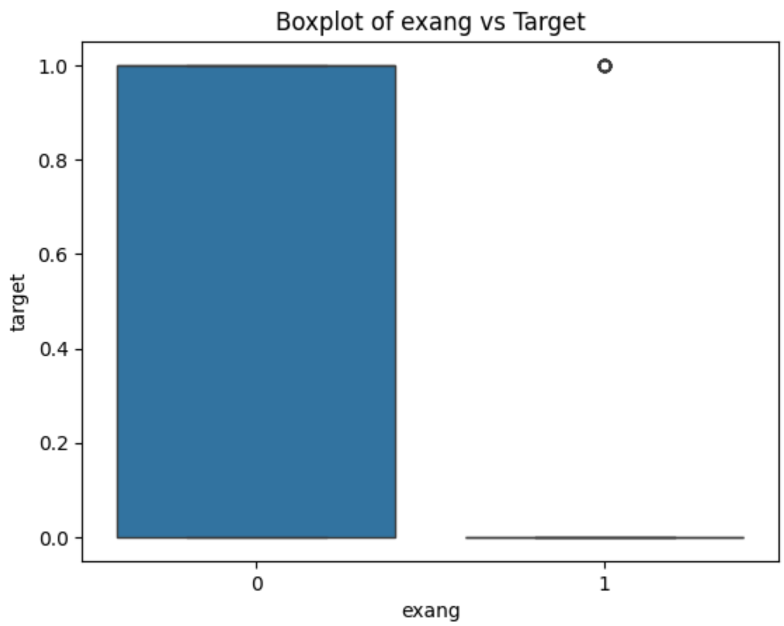


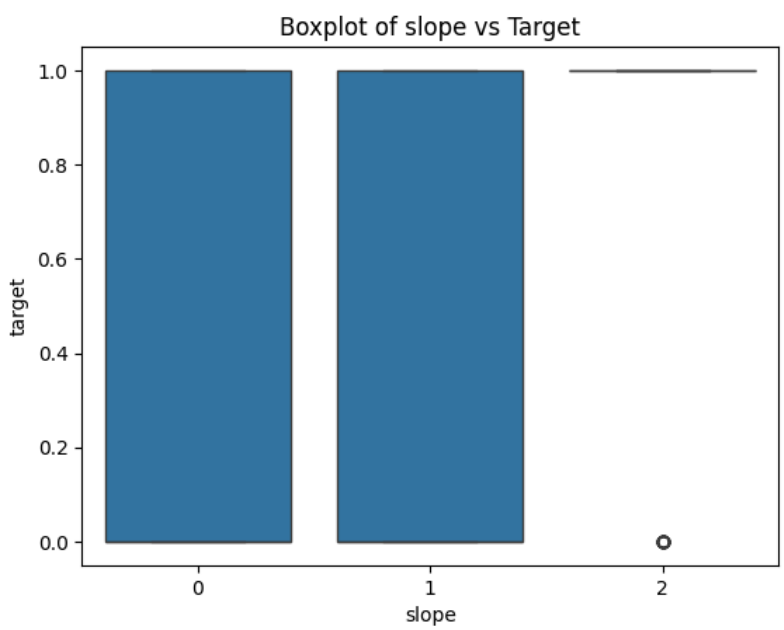


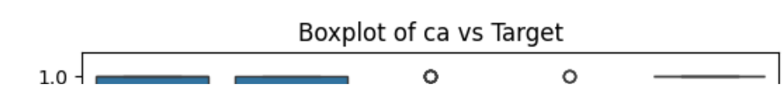


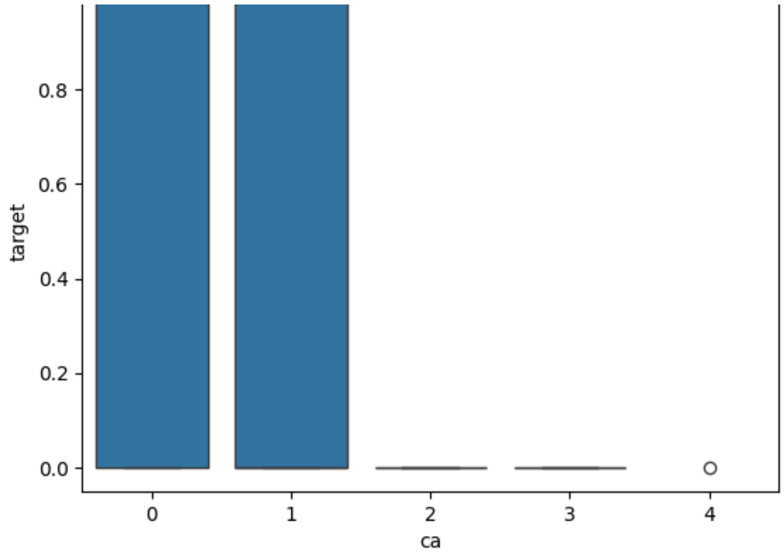
Boxplot of restecg vs Target

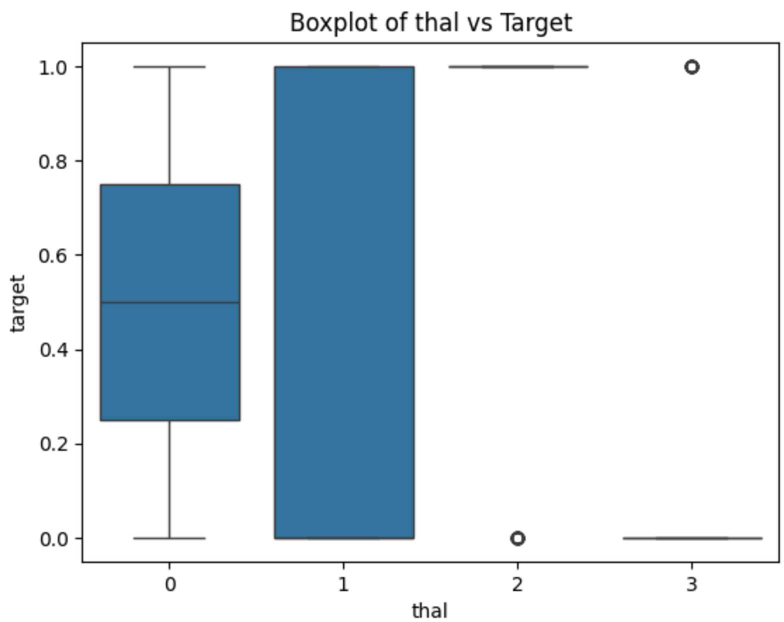

target
restecg
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2


Boxplot of exang vs Target
target
exang
1.0
0.8
0.6
0.4
0.2
0.0
0
1


Boxplot of slope vs Target
target
slope
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2


Boxplot of ca vs Target
1.0


target
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
ca


Boxplot of thal vs Target
target
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
thal

Handle Missing Values

```
# Handle missing values (if any)
df.fillna(df.median(), inplace=True)
```

Encode Categorical Variables

```
# One-hot encode categorical variables if needed
df = pd.get_dummies(df, columns=categorical_features, dr
df.head()
```

	age	trestbps	chol	thalach	oldpeak
0	0.952197	0.763956	-0.256334	0.015443	1.087338
1	-1.915313	-0.092738	0.072199	1.633471	2.122573
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705
4	0.290464	-0.663867	2.082050	0.583939	-0.379244

5 rows × 23 columns

Normalize/Standardize Features

```
from sklearn.preprocessing import StandardScaler

# Standardize numerical features
numerical_features = ["age", "trestbps", "chol", "thalac
scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numeric

df.head()
```

	age	sex	cp	trestbps	chol	fbs	reste
0	0.952197	1	3	0.763956	-0.256334	1	
1	-1.915313	1	2	-0.092738	0.072199	0	
2	-1.474158	0	1	-0.092738	-0.816773	0	
3	0.180175	1	1	-0.663867	-0.198357	0	
4	0.290464	0	0	-0.663867	2.082050	0	

Next steps: code df recommended interactive

Question no-3

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classificati

# Load a sample dataset (Iris dataset for demonstration)
data = load_iris()
X, y = data.data, data.target

# Split the dataset into training and testing sets (70/3
X_train, X_test, y_train, y_test = train_test_split(X, y

# Initialize the models
models = {
    "Logistic Regression": LogisticRegression(max_iter=2
    "Decision Tree": DecisionTreeClassifier(random_state
    "Random Forest": RandomForestClassifier(random_state
}

# Train each model and evaluate on the test set
for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)
    # Make predictions
    y_pred = model.predict(X_test)
    # Evaluate the model
    print(f"\n{name} Results:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.2
    print("Classification Report:")
```

```
print(classification_report(y_test, y_pred, target_n
```

```
Logistic Regression Results:
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   suppor

      setosa       1.00      1.00      1.00        1
  versicolor       1.00      1.00      1.00        1
   virginica       1.00      1.00      1.00        1

    accuracy                           1.00        4
   macro avg       1.00      1.00      1.00        4
weighted avg       1.00      1.00      1.00        4


Decision Tree Results:
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   suppor

      setosa       1.00      1.00      1.00        1
  versicolor       1.00      1.00      1.00        1
   virginica       1.00      1.00      1.00        1

    accuracy                           1.00        4
   macro avg       1.00      1.00      1.00        4
weighted avg       1.00      1.00      1.00        4


Random Forest Results:
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   suppor

      setosa       1.00      1.00      1.00        1
  versicolor       1.00      1.00      1.00        1
   virginica       1.00      1.00      1.00        1

    accuracy                           1.00        4
   macro avg       1.00      1.00      1.00        4
weighted avg       1.00      1.00      1.00        4
```

Double-click (or enter) to edit

Q-4

Generate randomly select 5 items from ... Close

```
from sklearn.metrics import accuracy_score, precision_sc
import matplotlib.pyplot as plt
import seaborn as sns
```

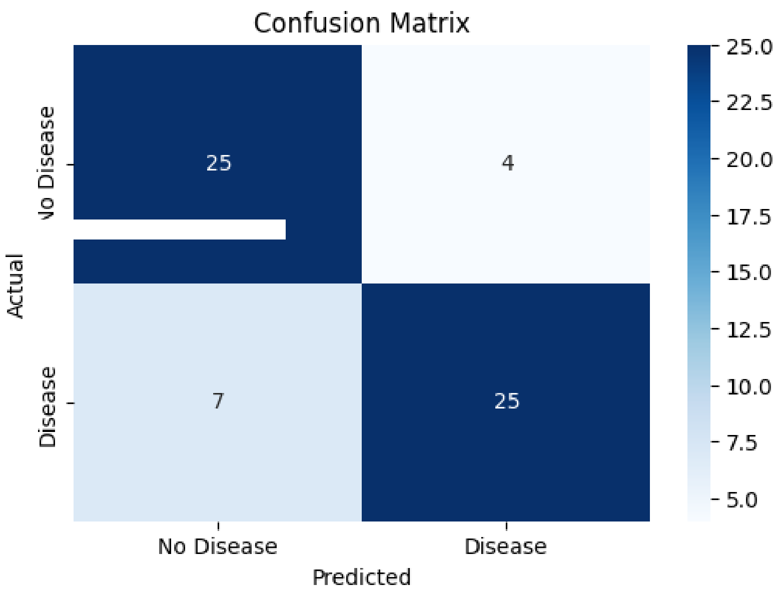
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Splitting data
X = df.drop(columns="target")
y = df["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y

# Train the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]  # For ROC cu
```

```
# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Display results
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

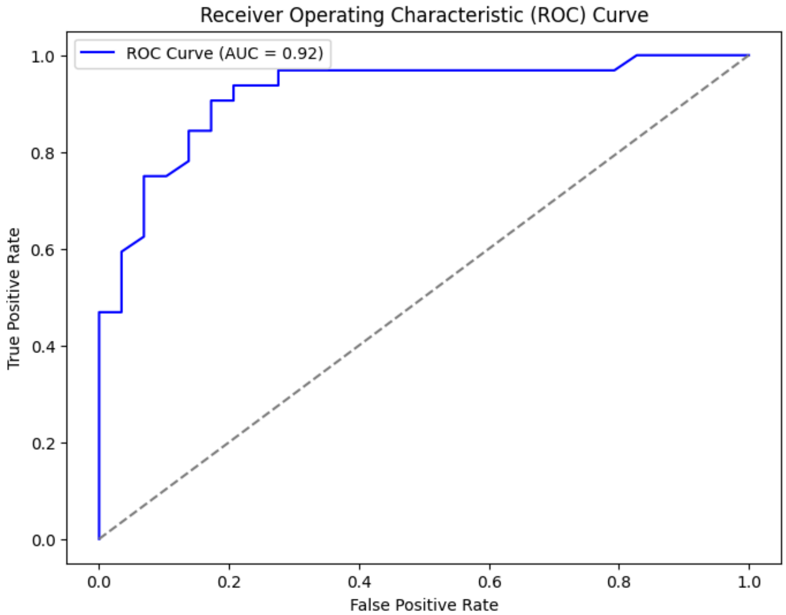
```
Accuracy: 0.82
Precision: 0.86
Recall: 0.78
F1-Score: 0.82
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xtick
```

```
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```




```
# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve
plt.legend()
plt.show()
```



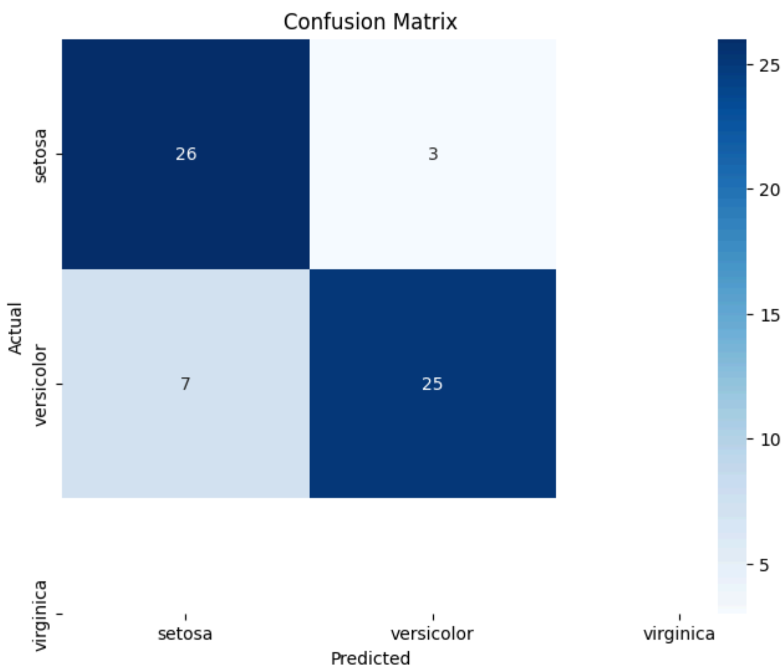
Question-5

Implement Grid Search

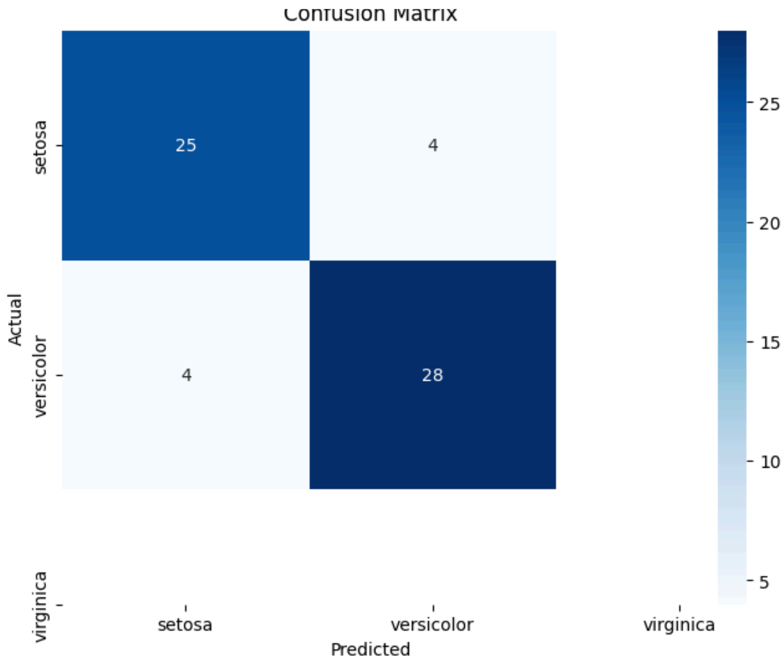
```
from sklearn.model_selection import GridSearchCV, Randomiz
from xgboost import XGBClassifier  # Ensure this is import
from sklearn.ensemble import RandomForestClassifier

# Define hyperparameter grids for each model
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 10],
    'learning_rate': [0.01, 0.1, 0.2],
```

```
    'subsample': [0.5, 0.7, 1.0],
}

# Perform Grid Search for Random Forest
print("Performing Grid Search for Random Forest...")
grid_rf = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid_rf,
    scoring='accuracy',
    cv=3,
    verbose=2,
    n_jobs=-1
)
grid_rf.fit(X_train, y_train)

# Perform Random Search for XGBoost
print("Performing Random Search for XGBoost...")
random_xgb = RandomizedSearchCV(
    estimator=XGBClassifier(use_label_encoder=False, eval_
    param_distributions=param_grid_xgb,
    n_iter=20,
    scoring='accuracy',
    cv=3,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
random_xgb.fit(X_train, y_train)

# Display the best parameters and their corresponding scor
print("\nBest parameters for Random Forest:", grid_rf.best
print("Best score for Random Forest:", grid_rf.best_score_

print("\nBest parameters for XGBoost:", random_xgb.best_pa
print("Best score for XGBoost:", random_xgb.best_score_)

# Evaluate the best models on the test set
best_rf = grid_rf.best_estimator_
best_xgb = random_xgb.best_estimator_

# Predict the values using the best models
y_pred_rf = best_rf.predict(X_test)
y_pred_xgb = best_xgb.predict(X_test)

# Evaluate the models
print("\nEvaluating the best Random Forest model...")
evaluate_classification_model(y_test, y_pred_rf, data.targ

print("\nEvaluating the best XGBoost model...")
evaluate_classification_model(y_test, y_pred_xgb, data.tar
```

```
Performing Grid Search for Random Forest...
Fitting 3 folds for each of 108 candidates, totallin
Performing Random Search for XGBoost...
Fitting 3 folds for each of 20 candidates, totalling
/usr/local/lib/python3.10/dist-packages/xgboost/core
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)

Best parameters for Random Forest: {'max_depth': 10,
Best score for Random Forest: 0.8141460905349794

Best parameters for XGBoost: {'subsample': 0.5, 'n_e
Best score for XGBoost: 0.7934156378600822

Evaluating the best Random Forest model...
Classification Metrics:
Accuracy: 0.84
Precision: 0.84
Recall: 0.84
F1-Score: 0.84

Confusion Matrix:
[[26  3]
 [ 7 25]]
```




```
Evaluating the best XGBoost model...
Classification Metrics:
Accuracy: 0.87
Precision: 0.87
Recall: 0.87
F1-Score: 0.87

Confusion Matrix:
[[25  4]
 [ 4 28]]
```

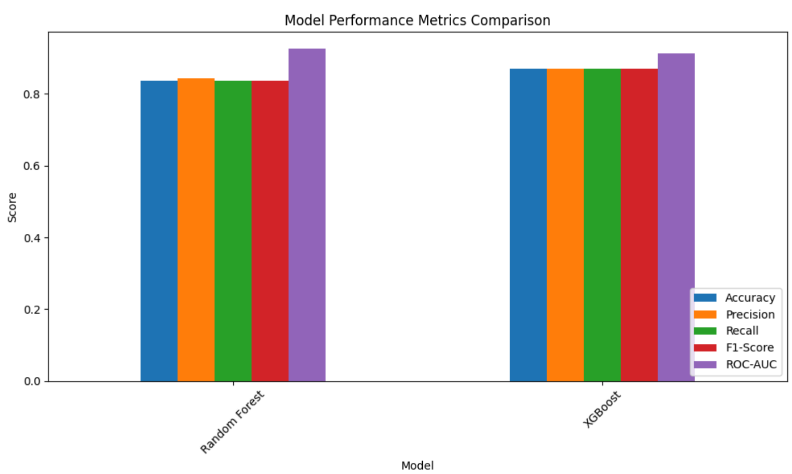

Confusion Matrix
setosa
versicolor
virginica
Actual
25
4
4
28
setosa
versicolor
virginica
Predicted
25
20
15
10
5

Question-6

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_sc
import pandas as pd

# Function to evaluate the model and collect metrics
def collect_metrics(model, X_test, y_test, model_name, n
    y_pred = model.predict(X_test)

    # Handle probability predictions (for ROC-AUC)
    y_pred_proba = model.predict_proba(X_test) if hasatt

    # Calculate metrics
    metrics = {
        'Model': model_name,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred, ave
        'Recall': recall_score(y_test, y_pred, average='
        'F1-Score': f1_score(y_test, y_pred, average='we
    }

    # Calculate ROC-AUC if it's a binary classification
    if y_pred_proba is not None:
        if num_classes == 2:  # Binary classification
            metrics['ROC-AUC'] = roc_auc_score(y_test, y
        else:  # Multi-class classification
            metrics['ROC-AUC'] = roc_auc_score(y_test, y
    else:
        metrics['ROC-AUC'] = np.nan  # If no probability

    # Display the classification report
    print(f"\nClassification Report for {model_name}:\n"
    print(classification_report(y_test, y_pred))

    return metrics

# Assuming `models` is a dictionary of trained models (e
# Example (replace with actual trained models):
models = {
    'Random Forest': best_rf,  # replace with your train
    'XGBoost': best_xgb  # replace with your trained XGB
}

# Collect metrics for all models
all_metrics = []
num_classes = len(np.unique(y_test))  # Get number of cl
```

```
for name, model in models.items():
    metrics = collect_metrics(model, X_test, y_test, nam
    all_metrics.append(metrics)

# Convert metrics into a DataFrame for comparison
metrics_df = pd.DataFrame(all_metrics)

# Print summary of performance
print("\nSummary of Model Performance:\n")
print(metrics_df)

# Plot metrics comparison
metrics_df.set_index('Model', inplace=True)
metrics_df.plot(kind='bar', figsize=(10, 6))
plt.title("Model Performance Metrics Comparison")
plt.ylabel("Score")
plt.xticks(rotation=45)
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()

# Challenges and Recommendations
print("\nChallenges Faced During Analysis:")
print("- Imbalanced dataset might skew performance metri
print("- Grid Search tuning can be computationally expen
print("- High model complexity (e.g., XGBoost) can lead

print("\nRecommendations for Improving the Model:")
print("- Address class imbalance using SMOTE or weighted
print("- Use feature engineering or selection to improve
print("- Apply advanced hyperparameter tuning techniques
print("- Combine models using ensemble techniques for be
```

```
Classification Report for Random Forest:

              precision    recall  f1-score   suppor

           0       0.79      0.90      0.84        2
           1       0.89      0.78      0.83        3

    accuracy                           0.84        6
   macro avg       0.84      0.84      0.84        6
weighted avg       0.84      0.84      0.84        6


Classification Report for XGBoost:

              precision    recall  f1-score   suppor

           0       0.86      0.86      0.86        2
           1       0.88      0.88      0.88        3

    accuracy                           0.87        6
   macro avg       0.87      0.87      0.87        6
weighted avg       0.87      0.87      0.87        6


Summary of Model Performance:

           Model  Accuracy  Precision    Recall  F1-
0  Random Forest  0.836066   0.842949  0.836066  0.8
1        XGBoost  0.868852   0.868852  0.868852  0.8
```




```
Challenges Faced During Analysis:
- Imbalanced dataset might skew performance metrics.
- Grid Search tuning can be computationally expensiv
- High model complexity (e.g., XGBoost) can lead to

Recommendations for Improving the Model:
- Address class imbalance using SMOTE or weighted lo
- Use feature engineering or selection to improve da
- Apply advanced hyperparameter tuning techniques li
- Combine models using ensemble techniques for bette
```

What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab] or [Colab Features You May Have Missed] to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see [Overview of Colab](). To create a new Colab notebook you can use the File menu above, or use the following link: [create a new Colab notebook]().

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see [jupyter.org]().

Data science