Welcome to Colab!

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view, and the command palette.


```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
import graphviz
import re
a=pd.read_csv("/content/heart_disease_data.csv")
x=a.drop(columns=["target"],axis=1)
y=a["target"]
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8)
model=DecisionTreeClassifier()
model.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
print(f'x_train : {x_train.shape}')
print(f'y_train : {y_train.shape}')
print(f'x_test : {x_test.shape}')
print(f'y_test : {y_test.shape}')
```

```
x_train : (484, 13)
y_train : (484,)
x_test : (122, 13)
y_test : (122,)
```

```
s=model.predict(x_train)
t=accuracy_score(s,y_train)
print(t)
```

```
1.0
```

```
s=model.predict(x_test)
t=accuracy_score(s,y_test)
print(t)
```

```
0.9836065573770492
```

```
from sklearn.metrics import confusion_matrix, classification_report
y_pred = model.predict(x_test)
# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
# Create a Pandas DataFrame for the confusion matrix
conf_matrix_df = pd.DataFrame(
conf_matrix,
columns=['Predicted Negative', 'Predicted Positive'],
index=['Actual Negative', 'Actual Positive']
)
# Calculate and add row/column totals
conf_matrix_df['Total'] = conf_matrix_df.sum(axis=1)
conf_matrix_df.loc['Total'] = conf_matrix_df.sum()
# Print the DataFrame
print("Confusion Matrix for Testing Data:")
print(conf_matrix_df)
```

```
Confusion Matrix for Testing Data:
                  Predicted Negative  Predicted Positive  Total
Actual Negative                 52                   0     52
Actual Positive                  2                  68     70
Total                           54                  68    122
```

```
from sklearn.metrics import confusion_matrix, classification_report
y_predt = model.predict(x_train)
# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_train, y_predt)
# Create a Pandas DataFrame for the confusion matrix
conf_matrix_df = pd.DataFrame(
conf_matrix,
columns=['Predicted Negative', 'Predicted Positive'],
index=['Actual Negative', 'Actual Positive']
)
# Calculate and add row/column totals
conf_matrix_df['Total'] = conf_matrix_df.sum(axis=1)
conf_matrix_df.loc['Total'] = conf_matrix_df.sum()
# Print the DataFrame
print("Confusion Matrix for Traning Data:")
print(conf_matrix_df)
```

```
Confusion Matrix for Traning Data:
                  Predicted Negative  Predicted Positive  Total
Actual Negative                224                   0    224
Actual Positive                  0                 260    260
Total                          224                 260    484
```

```
y_predict = model.predict(x_test)
from sklearn.metrics import accuracy_score,classification_report
print(accuracy_score(y_test,y_predict))
print(classification_report(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
0.9836065573770492
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        52
           1       1.00      0.97      0.99        70

    accuracy                           0.98       122
   macro avg       0.98      0.99      0.98       122
weighted avg       0.98      0.98      0.98       122
```

col_0	0	1
target		
0	52	0
1	2	68

```
from sklearn.model_selection import RandomizedSearchCV
#Using max_depth, criterion will suffice for DT Models, rest all will remain constant
parameters = {'max_depth' : (np.arange(1,500))
, 'criterion' : ('gini','entropy')
, 'max_features' : ('sqrt', 'log2')
, 'min_samples_split' : (np.arange(2,500))
}
```

```
DT_grid = RandomizedSearchCV(DecisionTreeClassifier(), param_distributions = parameters, cv = 5, verbose = True)
DT_grid.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

▸ RandomizedSearchCV
▸ estimator: DecisionTreeClassifier
▸ DecisionTreeClassifier

```
DT_grid.best_estimator_
```

▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=438, max_features='log2', min_samples_split=65)

```
DT_Model = DecisionTreeClassifier(max_depth=438, max_features='log2', min_samples_split=65)
```

```
DT_Model.fit(x_train,y_train)
```

```
DT_Model.fit(x_train,y_train)
```

```
DecisionTreeClassifier(max_depth=438, max_features='log2', min_samples_split=65)
```

```
print (f'Train Accuracy - : {DT_Model.score(x_train,y_train):.3f}')
print (f'Test Accuracy - : {DT_Model.score(x_test,y_test):.3f}')
```

```
Train Accuracy - : 0.829
Test Accuracy - : 0.762
```

```
from sklearn.metrics import confusion_matrix, classification_report
y_pred = DT_Model.predict(x_test)
# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
# Create a Pandas DataFrame for the confusion matrix
conf_matrix_df = pd.DataFrame(
conf_matrix,
columns=['Predicted Negative', 'Predicted Positive'],
index=['Actual Negative', 'Actual Positive']
)
# Calculate and add row/column totals
conf_matrix_df['Total'] = conf_matrix_df.sum(axis=1)
conf_matrix_df.loc['Total'] = conf_matrix_df.sum()
# Print the DataFrame
print("Confusion Matrix for Testing Data:")
print(conf_matrix_df)
```

```
Confusion Matrix for Testing Data:
                 Predicted Negative  Predicted Positive  Total
Actual Negative                  42                  10     52
Actual Positive                  18                  52     70
Total                            60                  62    122
```

```
from sklearn.metrics import confusion_matrix, classification_report
y_predt = DT_Model.predict(x_train)
# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_train, y_predt)
# Create a Pandas DataFrame for the confusion matrix
conf_matrix_df = pd.DataFrame(
conf_matrix,
columns=['Predicted Negative', 'Predicted Positive'],
index=['Actual Negative', 'Actual Positive']
)
# Calculate and add row/column totals
conf_matrix_df['Total'] = conf_matrix_df.sum(axis=1)
conf_matrix_df.loc['Total'] = conf_matrix_df.sum()
# Print the DataFrame
print("Confusion Matrix for Testing Data:")
print(conf_matrix_df)
```

```
Confusion Matrix for Testing Data:
                 Predicted Negative  Predicted Positive  Total
Actual Negative                 213                  11    224
Actual Positive                  13                 247    260
Total                           226                 258    484
```

```
y_predict = DT_Model.predict(x_test)
from sklearn.metrics import accuracy_score,classification_report
print(accuracy_score(y_test,y_predict))
print(classification_report(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
0.8688524590163934
              precision    recall  f1-score   support

           0       0.89      0.79      0.84        52
           1       0.86      0.93      0.89        70

    accuracy                           0.87       122
   macro avg       0.87      0.86      0.86       122
weighted avg       0.87      0.87      0.87       122
```

col_0	0	1
target		
0	41	11
1	5	65

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 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

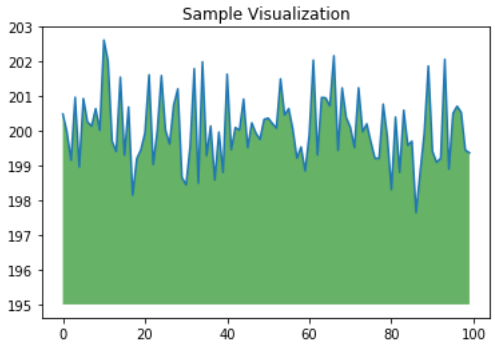
With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

```
import numpy as np
from matplotlib import pyplot as plt

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization")
plt.show()
```



You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

More Resources

Working with Notebooks in Colab

- Overview of Colaboratory
- Guide to Markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms
- Interactive widgets

Working with Data

- Loading data: Drive, Sheets, and Google Cloud Storage
- Charts: visualizing data
- Getting started with BigQuery

Machine Learning Crash Course

These are a few of the notebooks from Google's online Machine Learning course. See the full course website for more.

- Intro to Pandas DataFrame
- Linear regression with tf.keras using synthetic data

Using Accelerated Hardware

- TensorFlow with GPUs
- TensorFlow with TPUs

Featured examples

- NeMo Voice Swap: Use Nvidia's NeMo conversational AI Toolkit to swap a voice in an audio fragment with a computer generated one.
- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.