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?

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

Discover Gemini's advanced capabilities

- Play with Gemini multimodal outputs, mixing text and images in an iterative way.
- Discover the multimodal Live API (demo here).
- Learn how to analyze images and detect items in your pictures using Gemini (bonus, there's a 3D version as well!).
- Unlock the power of Gemini thinking model, capable of solving complex task with its inner thoughts.

Explore complex use cases

- Use Gemini grounding capabilities to create a report on a company based on what the model can find on internet.
- Extract invoices and form data from PDF in a structured way.
- Create illustrations based on a whole book using Gemini large context window and Imagen.

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.

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

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.

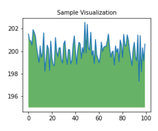
You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

```
import numpy as np
import IPython.display as display
from matplotlib import pyplot as plt
import io
import base64

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

fig = plt.figure(figsize=(4, 3), facecolor='w')
plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
plt.title("Sample Visualization", fontsize=10)

data = io.BytesIO()
plt.savefig(data)
image = F"{base64.b64encode(data.getvalue()).decode()}"
alt = "Sample Visualization"
display.display(display.Markdown(F"""![{alt}]({image})"""))
plt.close(fig)
```



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.

For example, if you find yourself waiting for **pandas** code to finish running and want to go faster, you can switch to a GPU Runtime and use libraries like RAPIDS cuDF that provide zero-code-change acceleration.

To learn more about accelerating pandas on Colab, see the 10 minute guide or US stock market data analysis demo.

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 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 Colab
- 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
- Intro to RAPIDS cuDF to accelerate pandas
- Linear regression with tf.keras using synthetic data

Using Accelerated Hardware

- TensorFlow with GPUs
- TPUs in Colab

Featured examples

- 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.

```
import pandas as pd
```

```
df=pd.read_csv("/content/German_Credit_Card_Dataset.csv")
```

```
df
```

	checkin_acc	duration	credit_history	amount	savings_acc	present_emp_since	inst_rate	personal_status	residing_since	age	inst_plans	num_credits	job	status
0	A11	6	A34	1169	A65	A75	4	A93	4	67	A143	2	A173	0
1	A12	48	A32	5951	A61	A73	2	A92	2	22	A143	1	A173	1
2	A14	12	A34	2096	A61	A74	2	A93	3	49	A143	1	A172	0
3	A11	42	A32	7882	A61	A74	2	A93	4	45	A143	1	A173	0
4	A11	24	A33	4870	A61	A73	3	A93	4	53	A143	2	A173	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	A14	12	A32	1736	A61	A74	3	A92	4	31	A143	1	A172	0
996	A11	30	A32	3857	A61	A73	4	A91	4	40	A143	1	A174	0
997	A14	12	A32	804	A61	A75	4	A93	4	38	A143	1	A173	0
998	A11	45	A32	1845	A61	A73	4	A93	4	23	A143	1	A173	1
999	A12	45	A34	4576	A62	A71	3	A93	4	27	A143	1	A173	0

1000 rows × 14 columns

```
df.info()
df.shape
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   checkin_acc         1000 non-null   object
 1   duration            1000 non-null   int64
 2   credit_history      1000 non-null   object
 3   amount              1000 non-null   int64
 4   savings_acc         1000 non-null   object
 5   present_emp_since   1000 non-null   object
 6   inst_rate           1000 non-null   int64
 7   personal_status     1000 non-null   object
 8   residing_since      1000 non-null   int64
 9   age                 1000 non-null   int64
 10  inst_plans          1000 non-null   object
 11  num_credits         1000 non-null   int64
 12  job                 1000 non-null   object
 13  status              1000 non-null   int64
dtypes: int64(7), object(7)
memory usage: 109.5+ KB
```

	checkin_acc	duration	credit_history	amount	savings_acc	present_emp_since	inst_rate	personal_status	residing_since	age	inst_plans	num_credits	job	status
0	A11	6	A34	1169	A65	A75	4	A93	4	67	A143	2	A173	0
1	A12	48	A32	5951	A61	A73	2	A92	2	22	A143	1	A173	1
2	A14	12	A34	2096	A61	A74	2	A93	3	49	A143	1	A172	0
3	A11	42	A32	7882	A61	A74	2	A93	4	45	A143	1	A173	0
4	A11	24	A33	4870	A61	A73	3	A93	4	53	A143	2	A173	1

```
df['checkin_acc'].unique()
```

```
array(['A11', 'A12', 'A14', 'A13'], dtype=object)
```

```
x_features = list(df.columns)
x_features.remove('status')
encoded_df = pd.get_dummies(df[x_features],drop_first=True)
print(list(encoded_df.columns))
```

```
['duration', 'amount', 'inst_rate', 'residing_since', 'age', 'num_credits', 'checkin_acc_A12', 'checkin_acc_A13', 'checkin_acc_A14', 'credit_history_A31', 'credit_history_A32', 'credit_history_A33', 'credit_history_A34', 'savings_acc_A62', 'savings_acc_A63', 'savings_acc_A64', 'savings_acc_A65', 'present_emp_since_A72', 'present_emp_since_A73', 'present_emp_since_A74', 'present_emp_since_A75', 'personal_status_A92', 'personal_status_A93', 'personal_status_A94', 'inst_plans_A142', 'inst_plans_A143', 'job_A172', 'job_A173', 'job_A174']
```

```
encoded_df
```

	duration	amount	inst_rate	residing_since	age	num_credits	checkin_acc_A12	checkin_acc_A13	checkin_acc_A14	credit_history_A31	...	present_emp_since_A74	present_emp_since_A75	personal_status_A92	personal_status_A93	personal_status_A94	inst_plans_A142	inst_plans_A143	job_A172	job_A173	job_A174
0	6	1169	4	4	67	2	False	False	False	False	...	False	True	False	True	False	False	True	False	True	False
1	48	5951	2	2	22	1	True	False	False	False	...	False	False	True	False	False	False	True	False	True	False
2	12	2096	2	3	49	1	False	False	True	False	...	True	False	False	True	False	False	True	True	False	False
3	42	7882	2	4	45	1	False	False	False	False	...	True	False	False	True	False	False	True	False	True	False
4	24	4870	3	4	53	2	False	False	False	False	...	False	False	False	True	False	False	True	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	12	1736	3	4	31	1	False	False	True	False	...	True	False	True	False	False	False	True	True	False	False
996	30	3857	4	4	40	1	False	False	False	False	...	False	False	False	False	False	False	True	False	False	True
997	12	804	4	4	38	1	False	False	True	False	...	False	True	False	True	False	False	True	False	True	False
998	45	1845	4	4	23	1	False	False	False	False	...	False	False	False	True	False	False	True	False	True	False
999	45	4576	3	4	27	1	True	False	False	False	...	False	False	False	True	False	False	True	False	True	False

1000 rows × 29 columns

```
x = encoded_df
y = df['status']


from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)


from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier (criterion='gini',max_depth=3)
dt.fit(x_train,y_train)
```

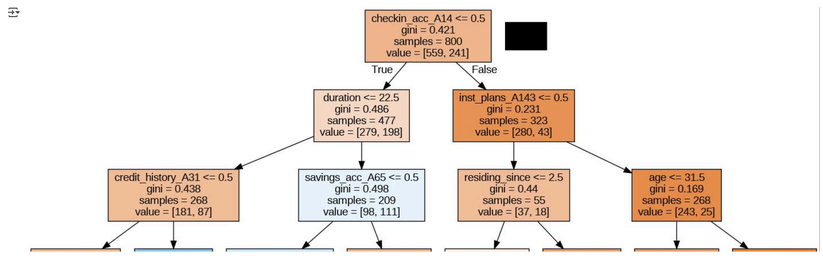
```
DecisionTreeClassifier(max_depth=3)
```

```
pred_y=dt.predict(x_test)


from sklearn import metrics
print("Confusion Matrix is\n",metrics.confusion_matrix(y_test,pred_y))
print("Accuracy is", metrics.accuracy_score(y_test,pred_y))
```

```
Confusion Matrix is
 [[120  21]
 [ 31  28]]
Accuracy is 0.74
```

```
from sklearn.tree import export_graphviz
import pydotplus as pdot
from IPython.display import Image
export_graphviz(dt,out_file="tree.odt",feature_names=x_train.columns,filled=True)
graph=pdot.graphviz.graph_from_dot_file("tree.odt")
graph.write_jpg("tree.png")
Image(filename="tree.png")
```



Start coding or generate with AI.