```
from google.colab import files
uploaded = files.upload()
```

Choose Files E Commerce Dataset.csv

- **E Commerce Dataset.csv**(text/csv) - 1247 bytes, last modified: 5/8/2025 - 100% done

```
Saving E Commerce Dataset.csv to E Commerce Dataset.csv
```

```
import pandas as pd

df = pd.read_csv('E Commerce Dataset.csv', header=1)
print(df.columns)


df.columns = df.columns.str.strip()
print(df.columns)
df = pd.read_csv("E Commerce Dataset.csv", header=None)
df.head(10)
```

```
Index(['Unnamed: 0', 'Data', 'Variable', 'Discerption'], dtype='object')
Index(['Unnamed: 0', 'Data', 'Variable', 'Discerption'], dtype='object')
```

	0	1	2	3
0	NaN	NaN	NaN	NaN
1	NaN	Data	Variable	Discerption
2	NaN	E Comm	CustomerID	Unique customer ID
3	NaN	E Comm	Churn	Churn Flag
4	NaN	E Comm	Tenure	Tenure of customer in organization
5	NaN	E Comm	PreferredLoginDevice	Preferred login device of customer
6	NaN	E Comm	CityTier	City tier
7	NaN	E Comm	WarehouseToHome	Distance in between warehouse to home of customer
8	NaN	E Comm	PreferredPaymentMode	Preferred payment method of customer
9	NaN	E Comm	Gender	Gender of customer

Next steps: Generate code with df View recommended plots New interactive sheet

```
df = pd.read_csv("E Commerce Dataset.csv", skiprows=3)
df.columns = df.columns.str.strip()
df.head()
```

	Unnamed: 0	E Comm	Churn	Churn Flag
0	NaN	E Comm	Tenure	Tenure of customer in organization
1	NaN	E Comm	PreferredLoginDevice	Preferred login device of customer
2	NaN	E Comm	CityTier	City tier
3	NaN	E Comm	WarehouseToHome	Distance in between warehouse to home of customer
4	NaN	E Comm	PreferredPaymentMode	Preferred payment method of customer

Next steps: Generate code with df View recommended plots New interactive sheet

```
print(df.columns)
```

```
Index(['Unnamed: 0', 'E Comm', 'Churn', 'Churn Flag'], dtype='object')
```

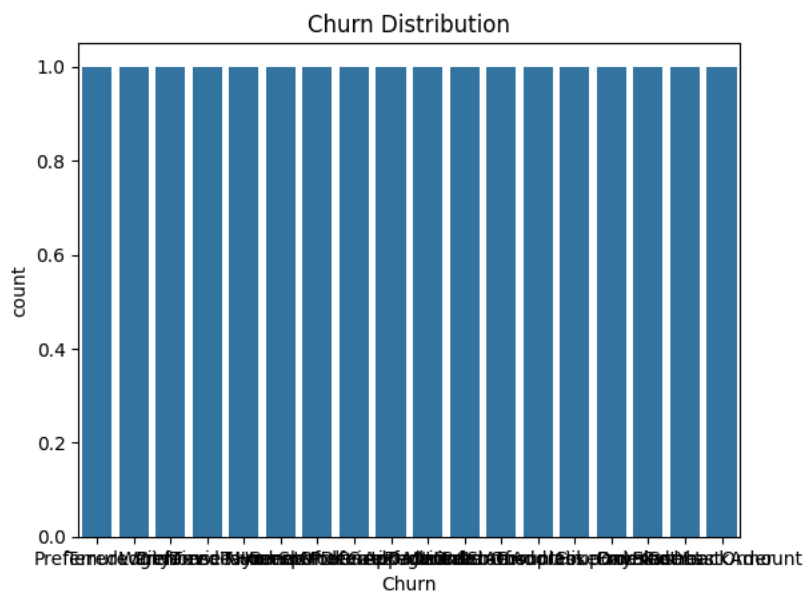
```
df.info()
df.describe()
df.columns
```

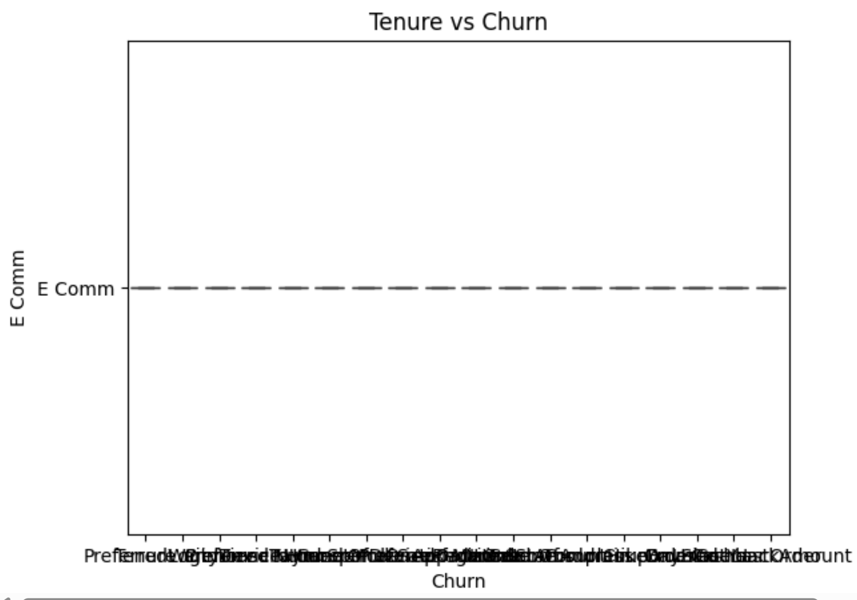
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  0 non-null      float64
 1   E Comm      18 non-null     object
 2   Churn       18 non-null     object
 3   Churn Flag  18 non-null     object
dtypes: float64(1), object(3)
memory usage: 708.0+ bytes
Index(['Unnamed: 0', 'E Comm', 'Churn', 'Churn Flag'], dtype='object')
```

```
print("Missing values:\n", df.isnull().sum())
print("\nDuplicates:", df.duplicated().sum())
```

```
Missing values:
 Unnamed: 0    18
E Comm         0
Churn          0
Churn Flag     0
dtype: int64

Duplicates: 0
```

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

sns.countplot(x='Churn', data=df)
plt.title('Churn Distribution')
plt.show()


sns.boxplot(x='Churn', y='E Comm', data=df)
plt.title('Tenure vs Churn')
plt.show()
```





```python
target = 'Churn'
features = [col for col in df.columns if col != target]
X = df[features]
y = df[target]
```

```python
cat_cols = X.select_dtypes(include='object').columns
X[cat_cols] = X[cat_cols].apply(lambda col: col.str.strip())
X[cat_cols] = X[cat_cols].apply(lambda col: col.astype('category').cat.codes)
```

```
<ipython-input-9-b7aebfb4dcd9>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
  X[cat_cols] = X[cat_cols].apply(lambda col: col.str.strip())
<ipython-input-9-b7aebfb4dcd9>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
  X[cat_cols] = X[cat_cols].apply(lambda col: col.astype('category').cat.codes)
```

```python
X = pd.get_dummies(X, drop_first=True)
```

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/extmath.py:1101: RuntimeWarning: invalid value encountered in divide
  updated_mean = (last_sum + new_sum) / updated_sample_count
/usr/local/lib/python3.11/dist-packages/sklearn/utils/extmath.py:1106: RuntimeWarning: invalid value encountered in divide
  T = new_sum / new_sample_count
/usr/local/lib/python3.11/dist-packages/sklearn/utils/extmath.py:1126: RuntimeWarning: invalid value encountered in divide
  new_unnormalized_variance -= correction**2 / new_sample_count
```

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```python
import pandas as pd

X_train_df = pd.DataFrame(X_train)

X_train_df.isnull().sum()
```

```
    0
0  14
1   0
2   0

dtype: int64
```

```python
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression

imputer = SimpleImputer(strategy='mean')

X_train_imputed = imputer.fit_transform(X_train)

model = LogisticRegression()
model.fit(X_train_imputed, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarning: Skipping features without any observed values:
  warnings.warn(
▾ LogisticRegression
LogisticRegression()
```

```
import numpy as np

print("Original X_train shape:", X_train.shape)

X_train_no_nan = X_train[~np.isnan(X_train).any(axis=1)]
y_train_no_nan = y_train[~np.isnan(X_train).any(axis=1)]

print("Filtered X_train shape:", X_train_no_nan.shape)

if X_train_no_nan.shape[0] == 0:
    print("Warning: No data left after removing rows with NaN values.")
else:
    model = LogisticRegression()
    model.fit(X_train_no_nan, y_train_no_nan)
```

```
Original X_train shape: (14, 3)
Filtered X_train shape: (0, 3)
Warning: No data left after removing rows with NaN values.
```

```
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)

X_test_imputed = imputer.transform(X_test)

model = LogisticRegression()
model.fit(X_train_imputed, y_train)

y_pred = model.predict(X_test_imputed)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarning: Skipping features without any observed values:
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarning: Skipping features without any observed values:
  warnings.warn(
```

```
from sklearn.metrics import classification_report, confusion_matrix

y_pred = model.predict(X_test_imputed)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0]]
                      precision    recall  f1-score   support

            CityTier       0.00      0.00      0.00       0.0
              Gender       0.00      0.00      0.00       1.0
          OrderCount       0.00      0.00      0.00       0.0
    PreferedOrderCat       0.00      0.00      0.00       1.0
PreferredLoginDevice       0.00      0.00      0.00       1.0
PreferredPaymentMode       0.00      0.00      0.00       0.0
   SatisfactionScore       0.00      0.00      0.00       0.0
              Tenure       0.00      0.00      0.00       1.0

            accuracy                           0.00       4.0
           macro avg       0.00      0.00      0.00       4.0
        weighted avg       0.00      0.00      0.00       4.0

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-define
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined a
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-define
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined a
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-define
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined a
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression


pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression())
])


pipeline.fit(X_train, y_train)

sample = X.iloc[[0]]
prediction = pipeline.predict(sample)
print("Prediction:", prediction)
```

```
Prediction: ['OrderCount']
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarning: Skipping features without any observed values:
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2732: UserWarning: X has feature names, but SimpleImputer wa
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarning: Skipping features without any observed values:
  warnings.warn(
```

```
def preprocess_input(input_dict):
    input_df = pd.DataFrame([input_dict])
    input_df = pd.get_dummies(input_df)
    input_df = input_df.reindex(columns=X.columns, fill_value=0)
    return scaler.transform(input_df)



def predict_churn(input_dict):
    processed = preprocess_input(input_dict)
    prediction = model.predict(processed)
    return "Churn" if prediction[0] == 1 else "Not Churn"



!pip install gradio
import gradio as gr
```

```
Collecting gradio
  Downloading gradio-5.29.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<25.0,>=22.0 (from gradio)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.10.0 (from gradio)
  Downloading gradio_client-1.10.0-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.30.2)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.18)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.4)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
```

```
Collecting ruff>=0.9.3 (from gradio)
  Downloading ruff-0.11.8-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (25 kB)
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.2 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio) (2025.3
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0-
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.4.26)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradi
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->grad
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (202
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->g
```

```python
import gradio as gr

# Define the churn_app function
def churn_app(input1, input2, input3, input4):
    # Your code to handle inputs and predict churn
    # For example:
    prediction = "Churn prediction based on inputs"
    return prediction

# Define the interface
interface = gr.Interface(
    fn=churn_app,
    inputs=["text", "number", "number", "text"],
    outputs="text",
    title="Customer Churn Predictor"
)

# Launch the interface
interface.launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automa

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://441b1cf3833db2cfa9.gradio.live
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the wor
```

Customer Churn Predictor

input1

input2

0

input3

0

input4

output

Flag

Clear Submit

Use via API · Built with Gradio · Settings

Double-click (or enter) to edit

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

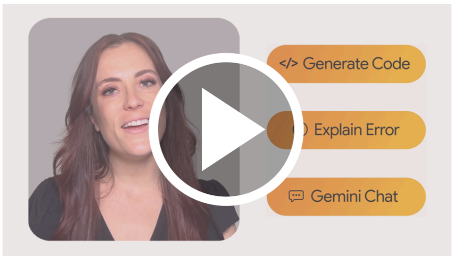
- Play with Gemini multimodal outputs, mixing text and images in an iterative way.
- Discover the multimodal Live API (demo here).
- Learn how to analyse images and detect items in your pictures using Gemini (bonus, there's a 3D version as well!).
- Unlock the power of the Gemini thinking model, capable of solving complex tasks 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 the Internet.
- Extract invoices and form data from PDFs in a structured way.
- Create illustrations based on a whole book using Gemini large context window and Imagen.

To learn more, take a look at 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 that 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
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-eb250fa3eac1> in <cell line: 0>()
----> 1 seconds_in_a_week = 7 * seconds_in_a_day
      2 seconds_in_a_week

NameError: name 'seconds_in_a_day' is not defined
```

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 find out 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.

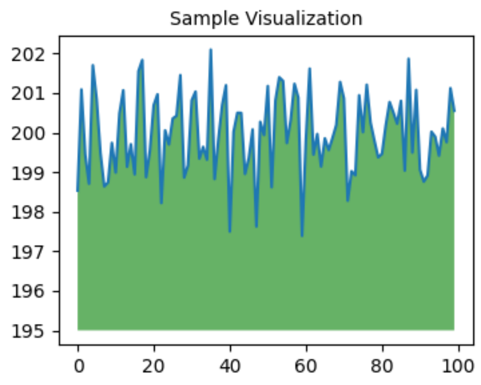
Colab notebooks are Jupyter notebooks that are hosted by Colab. To find out more about the Jupyter project, see jupyter.org.

Data science

With Colab you can harness the full power of popular Python libraries to analyse and visualise data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualise 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 find out 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"data:image/png;base64,{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 that 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: visualising 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 film 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.