**DOCUMENTATION OF STREAMLIT**

**INTRODUCTION:**

Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. In just a few minutes you can build and deploy powerful data apps. So, let's get started!

**INSTALLATION:**

1. Set up your Python development environment.
2. Run: pip install streamlit
3. Validate the installation by running our Hello app: streamlit hello

**RUN STREAMLIT:**

1. Streamlit run filename.py

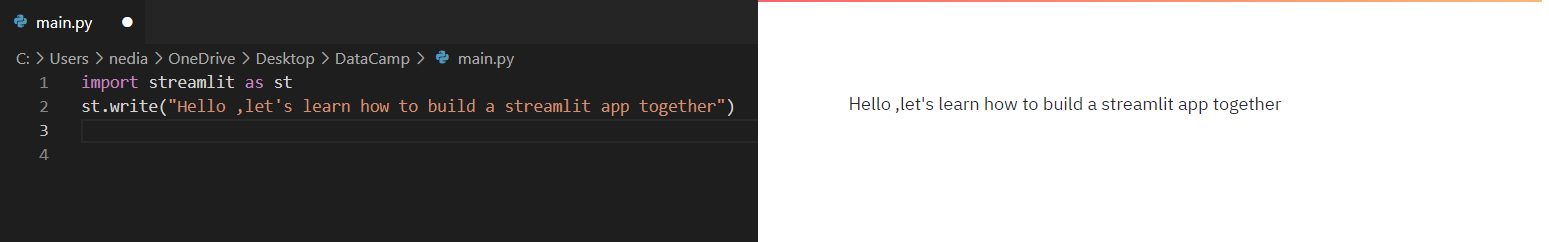
**DISPLAY TEXTS WITH STREAMLIT:**

In the beginning, we will see how to add text to your Streamlit app, and what the different commands are to add texts.

st.write(): This function is used to add anything to a web app, from formatted string to charts in matplotlib figure, Altair charts, plotly figure, data frame, Keras model, and others.

import streamlit as st

st.write("Hello, let's learn how to build a Streamlit app together")



st.title(): This function allows you to add the title of the app. st.header(): This function is used to set header of a section. st.markdown(): This function is used to set a markdown of a section. st.subheader(): This function is used to set sub-header of a section. st.caption(): This function is used to write caption. st.code(): This function is used to set a code. st.latex(): This function is used to display mathematical expressions formatted as LaTeX.

import streamlit as st

st.title("this is the app title")

st.header("this is the markdown")

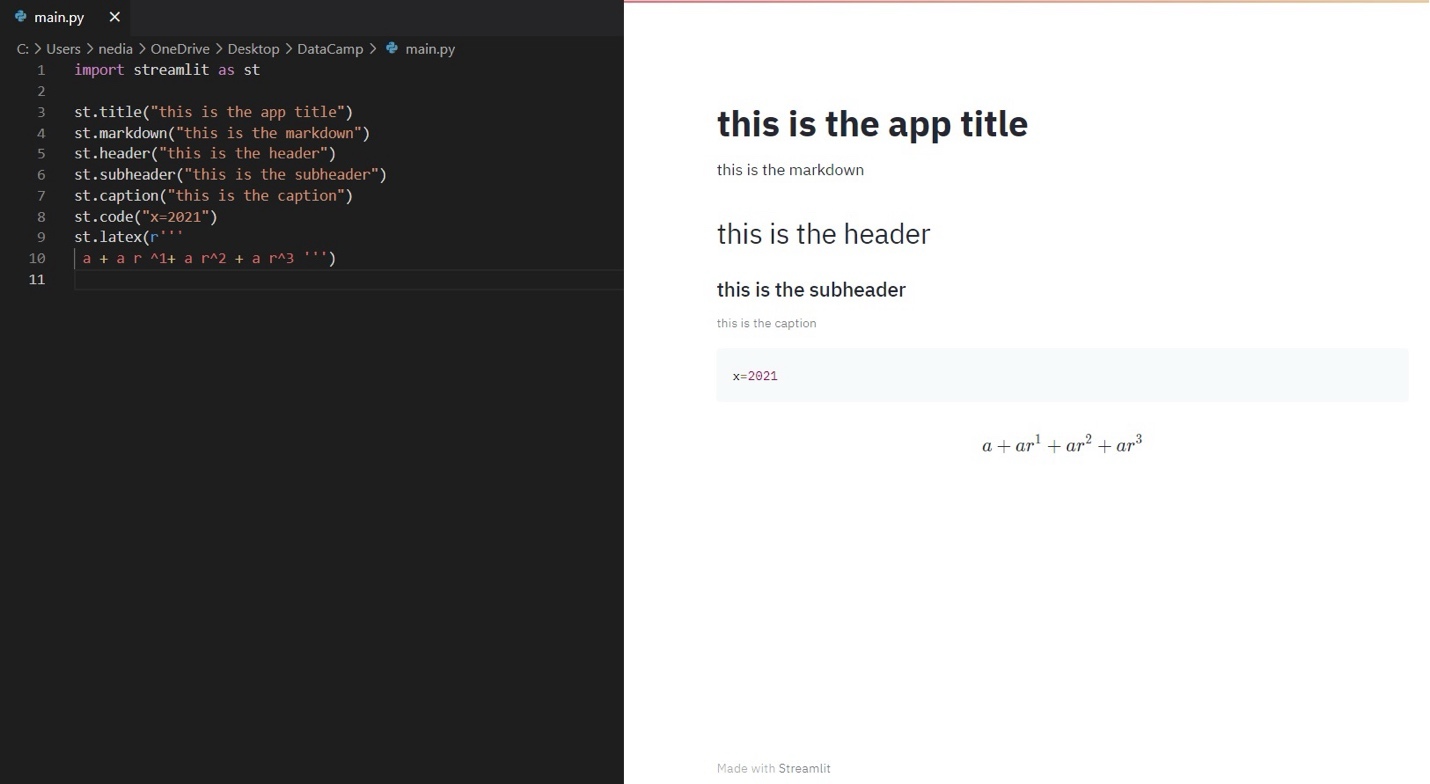
st.markdown("this is the header")

st.subheader("this is the subheader")

st.caption("this is the caption")

st.code("x=2021")

st.latex(r''' a+a r^1+a r^2+a r^3 ''')



**DISPLAY AN IMAGE, VIDEO OR AUDIO FILE WITH STREAMLIT:**

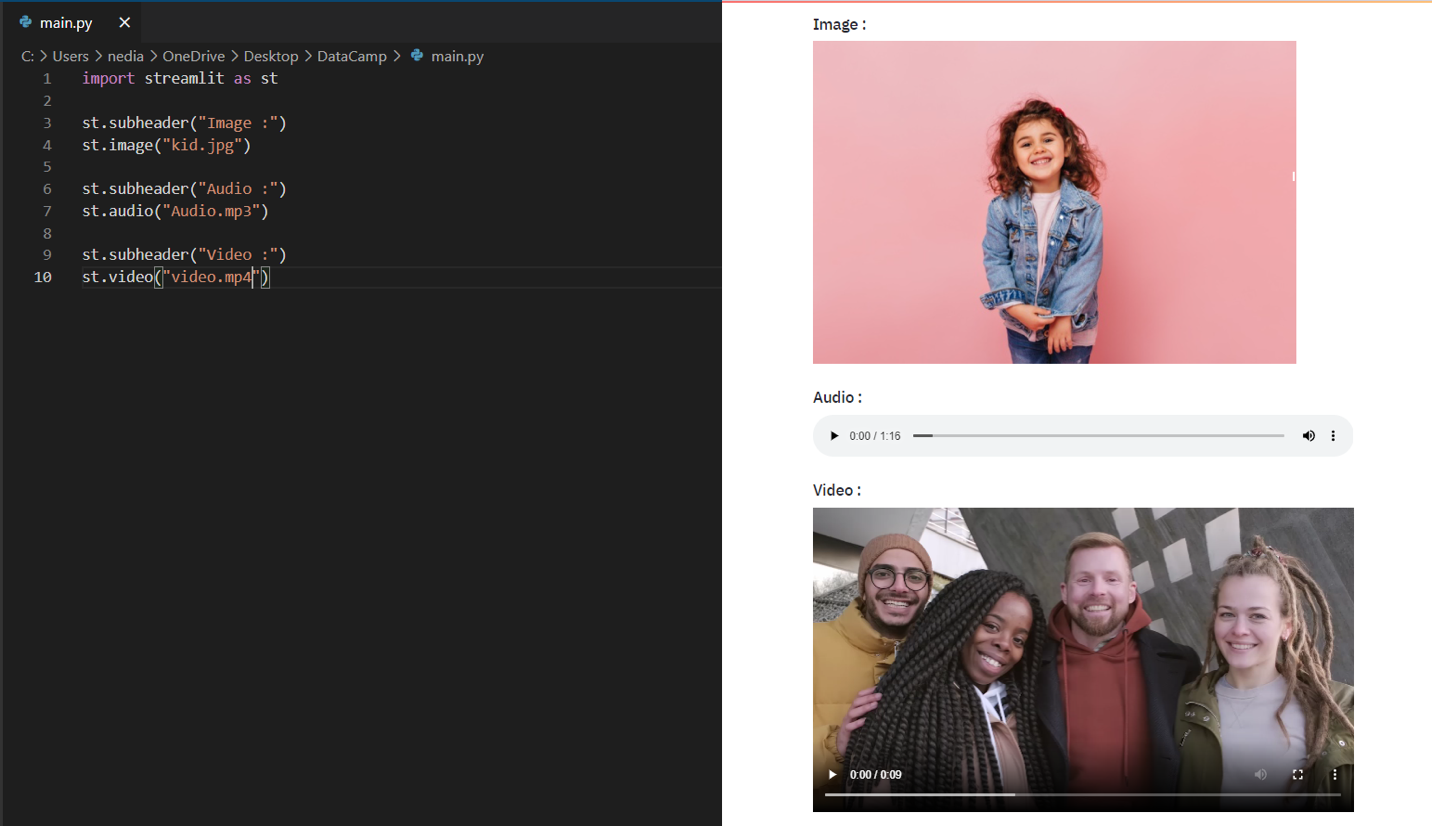
You can't find functions as easy as Streamlit functions to display images, videos, and audio files. Let's take a look at how to display media with Streamlit !

st.image(): This function is used to display an image. st.audio(): This function is used to display an audio. st.video(): This function is used to display a video.

st.image("kid.jpg")

st.audio("Audio.mp3")

st.video("video.mp4")



*Input Widget:*

Widgets are the most important user interface components. Streamlit has various widgets that allow you to bake interactivity directly into your apps with buttons, sliders, text inputs, and more.

st.checkbox(): This function returns a Boolean value. When the box is checked, it returns a True value, otherwise a False value. st.button(): This function is used to display a button widget. st.radio(): This function is used to display a radio button widget. st.selectbox(): This function is used to display a select widget. st.multiselect(): This function is used to display a multiselect widget. st.select\_slider(): This function is used to display a select slider widget. st.slider(): This function is used to display a slider widget.

import streamlit as st

st.checkbox('yes')

st.button('Click')

st.radio('Pick your gender', ['Male', 'Female'])

st.selectbox('Pick your gender', ['Male', 'Female'])

st.multiselect('choose a planet', ['Jupiter', 'Mars', 'neptune'])

st.select\_slider('Pick a mark', ['Bad', 'Good', 'Excellent'])

st.slider('Pick a number', 0, 50)

A screenshot of a computer

Description automatically generated

st.number\_input(): This function is used to display a numeric input widget. st.text\_input(): This function is used to display a text input widget. st.date\_input(): This function is used to display a date input widget to choose a date. st.time\_input(): This function is used to display a time input widget to choose a time. st.text\_area(): This function is used to display a text input widget with more than a line text. st.file\_uploader(): This function is used to display a file uploader widget. st.color\_picker(): This function is used to display color picker widget to choose a color.

import streamlit as st

st.number\_input('Pick a number', 0,10)

st.text\_input('Email address')

st.date\_input('Travelling date')

st.time\_input('School time')

st.text\_area('Description')

st.file\_uploader('Upload a photo')

st.color\_picker('Choose your favorite color')

A screenshot of a computer

Description automatically generated

**DISPLAY PROGRESS AND STATUS WITH STREAMLIT:**

Now we will see how we can add a progress bar and status messages such as error and success to our app.

st.balloons(): This function is used to display balloons for celebration. st.progress(): This function is used to display a progress bar. st.spinner(): This function is used to display a temporary waiting message during execution.

import streamlit as st

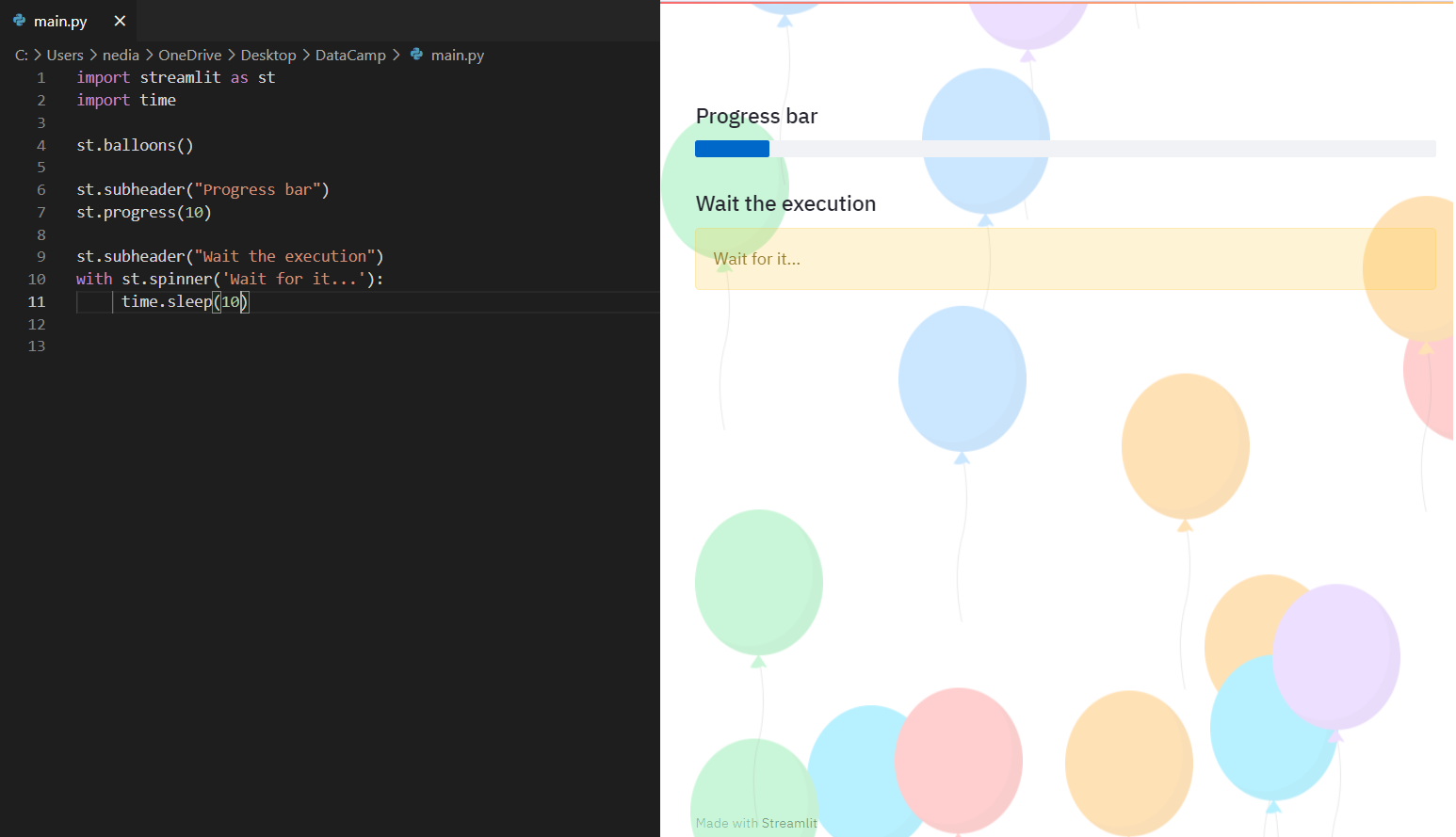
import time

st.balloons()

st.progress(10)

with st.spinner('Wait for it...'):

time.sleep(10)



st.success(): This function is used to display a success message. st.error(): This function is used to display an error message. st.warnig(): This function is used to display a warning message. st.info(): This function is used to display an informational message. st.exception(): This function is used to display an exception message.

import streamlit as st

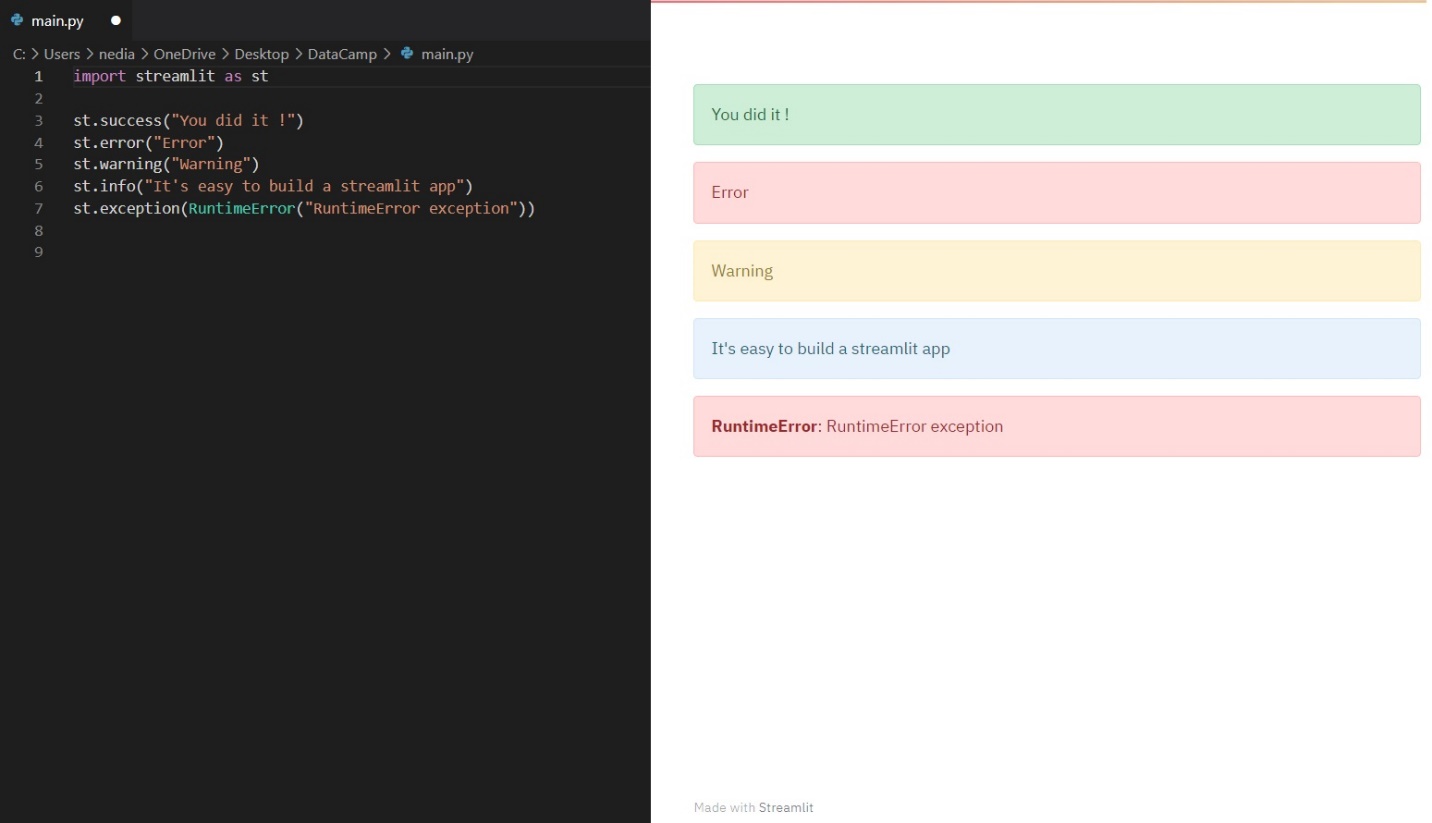
st.success("You did it !")

st.error("Error")

st.warning("Warning")

st.info("It's easy to build a streamlit app")

st.exception(RuntimeError("RuntimeError exception"))



**SIDEBAR AND CONTAINER:**

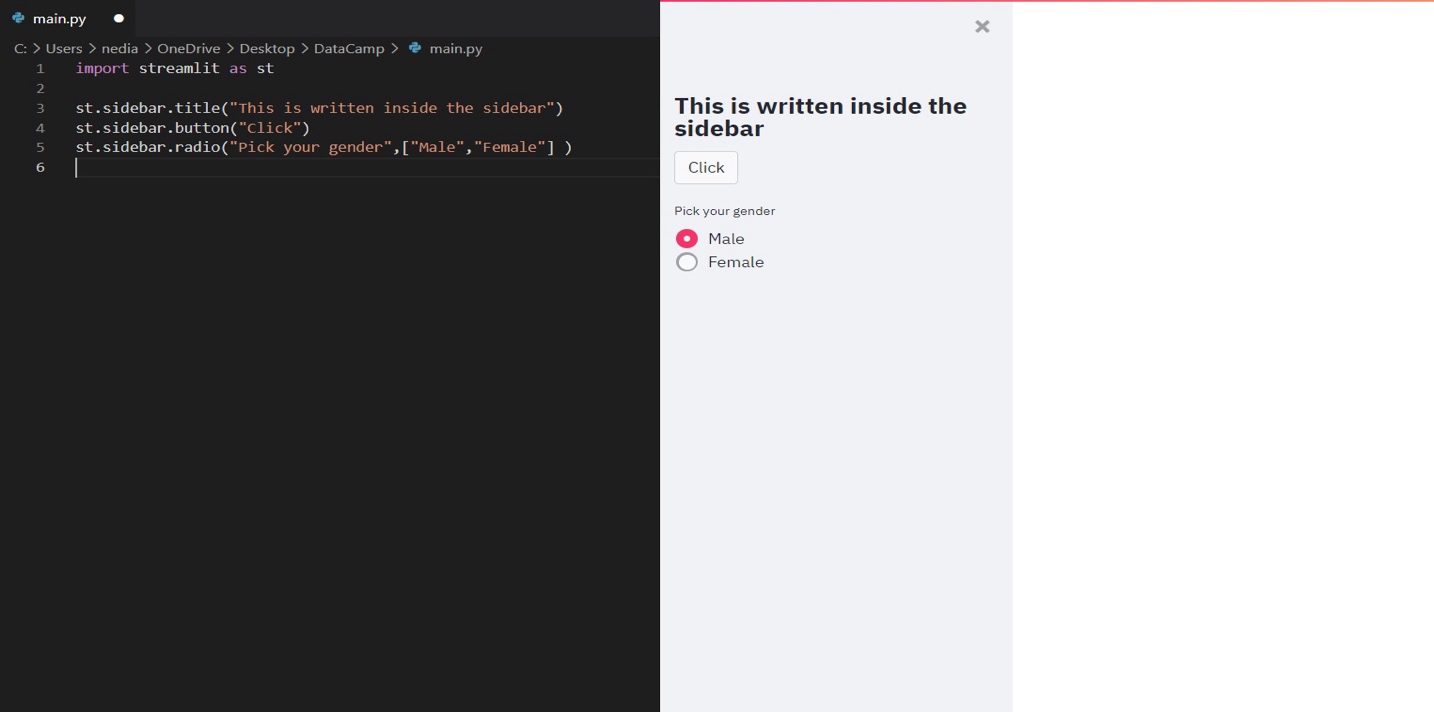
You can also create a sidebar or a container on your page to organize your app. The hierarchy and arrangement of pages on your app can have a large impact on your user experience. By organizing your content, you allow visitors to understand and navigate your site, which helps them find what they're looking for and increases the likelihood that they'll return in the future.

*Sidebar:*

Passing an element to st.sidebar() will make this element pinned to the left, allowing users to focus on the content in your app.

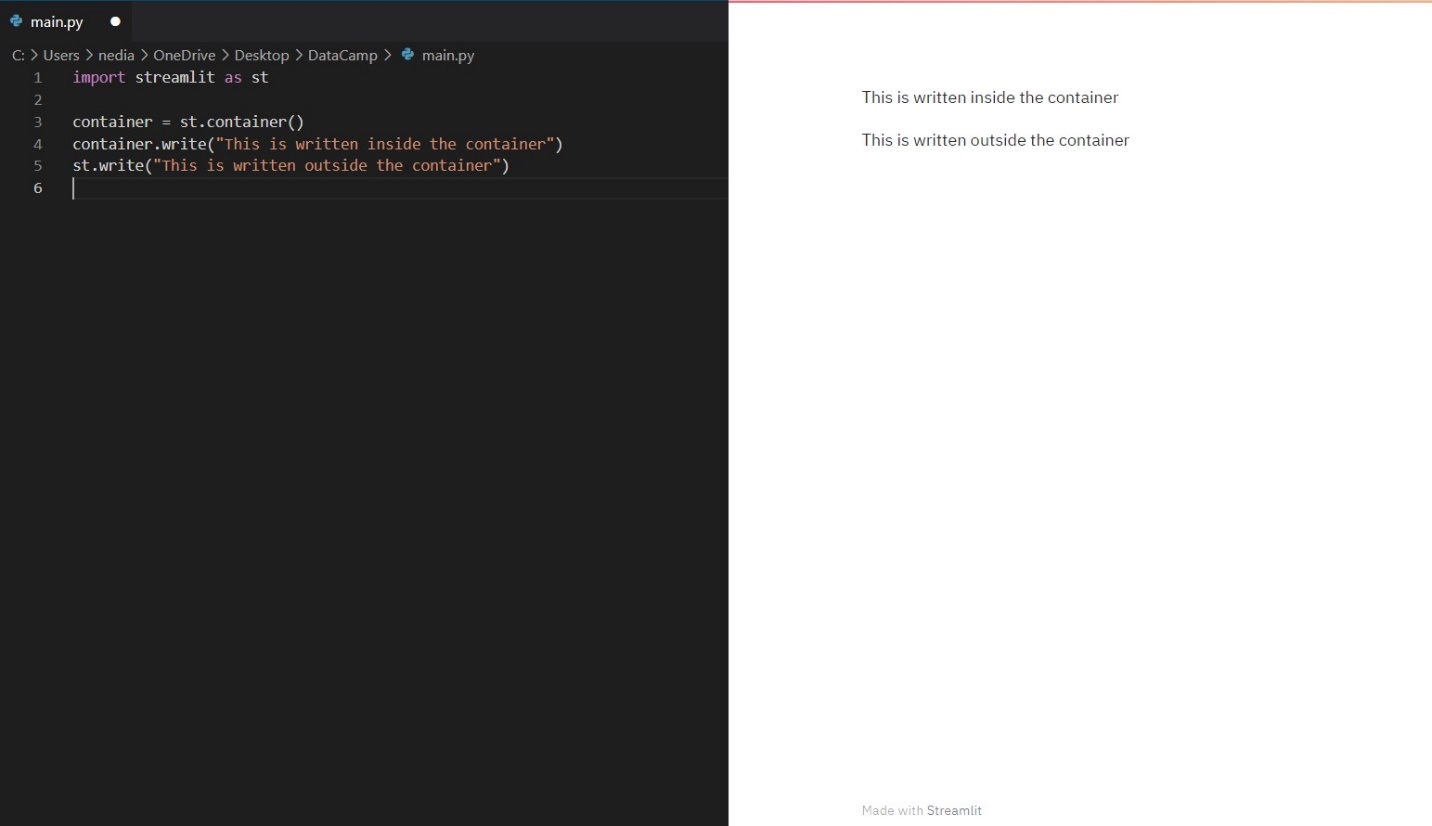
But st.spinner() and st.echo() are not supported with st.sidebar.

As you see, you can create a sidebar in your app interface and put elements inside it that will make your app more organized and easier to understand.



*Container:*

st.container() is used to create an invisible container where you can put elements in order to create a useful arrangement and hierarchy



**DISPLAY GRAPHS WITH STREAMLIT:**

*Why do we need visualization?*

Data visualization helps to tell stories by curating data into a format that's easier to understand, highlighting the trends and outliers. A good visualization tells a story, removing the noise from data and highlighting the useful information. However, it's not simply as easy as dressing up a graph to make it look better or slapping on the "info" part of an infographic. Effective data visualization is a delicate balancing act between form and function. The plainest graph could be too boring to draw attention or convey a powerful message, and the most stunning visualization could utterly fail at conveying the right message. The data and the visuals need to work together, and there's an art to combining great analysis with great storytelling.

Do you think giving you the data of one million points in a table/database file and asking you to provide your inferences by just seeing the data on that table is feasible? Unless you're a super human, it's not possible. This is when we make use of data visualization—it gives us a clear idea of what the information means by giving it visual context through maps or graphs. That's the power of Streamlit visualization.

st.pyplot(): This function is used to display a matplotlib.pyplot figure.

import streamlit as st

import matplotlib.pyplot as plt

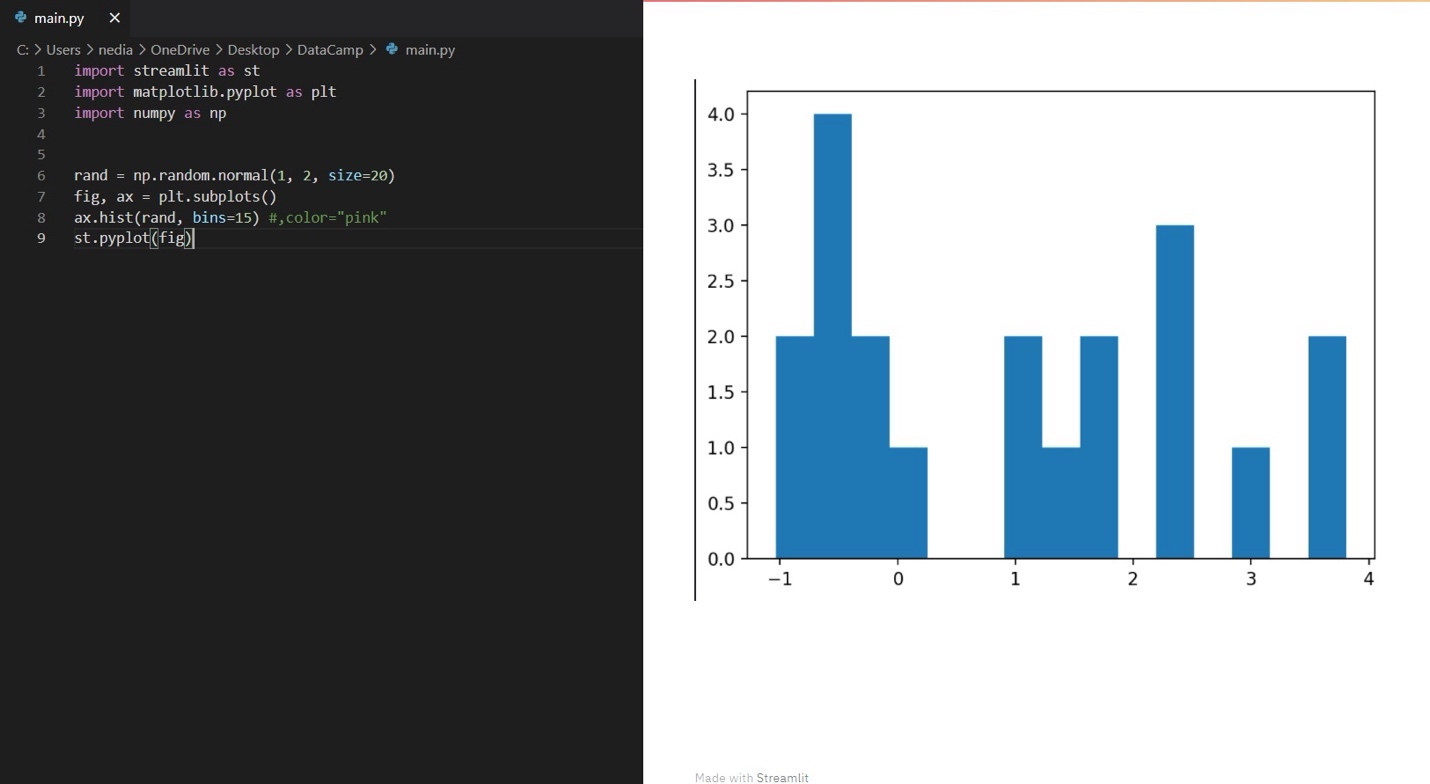
import numpy as np

rand=np.random.normal(1, 2, size=20)

fig, ax = plt.subplots()

ax.hist(rand, bins=15)

st.pyplot(fig)



st.line\_chart(): This function is used to display a line chart.

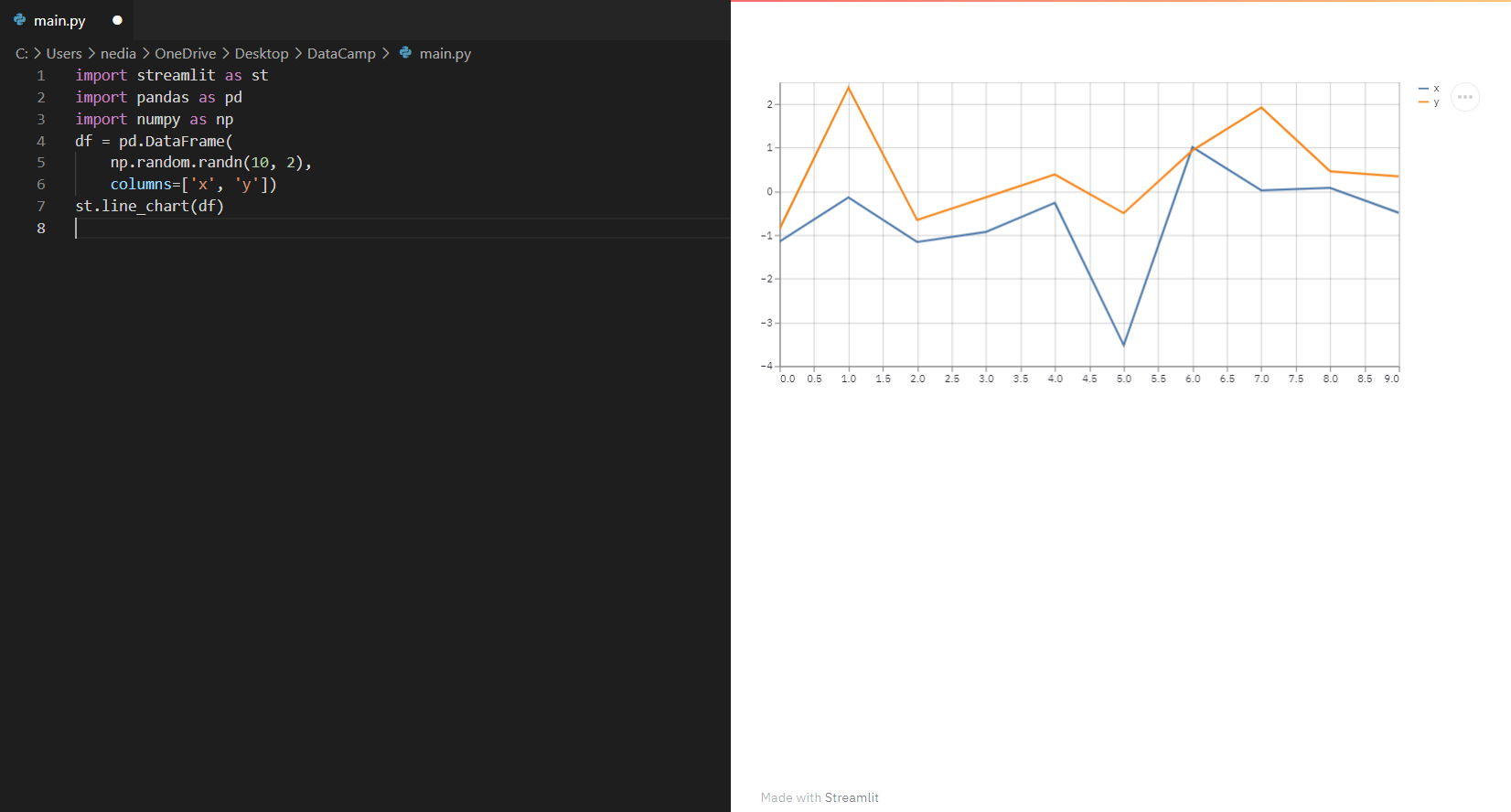
import streamlit as st

import pandas as pd

import numpy as np

df= pd.DataFrame( np.random.randn(10, 2), columns=['x', 'y'])

st.line\_chart(df)



st.bar\_chart(): This function is used to display a bar chart.

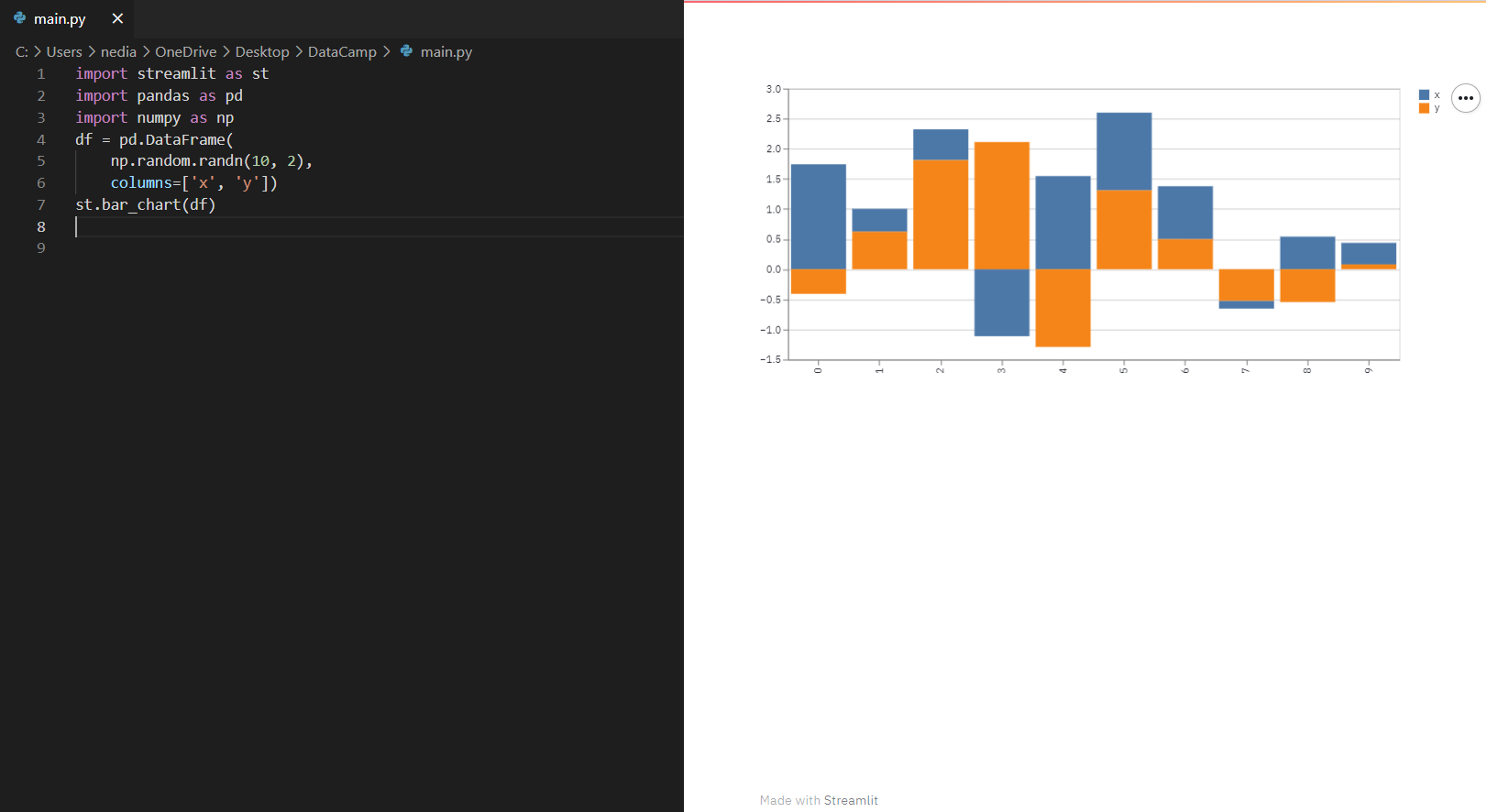
import streamlit as st

import pandas as pd

import numpy as np

df= pd.DataFrame( np.random.randn(10, 2), columns=['x', 'y'])

st.bar\_chart(df)



st.area\_chart(): This function is used to display an area chart.

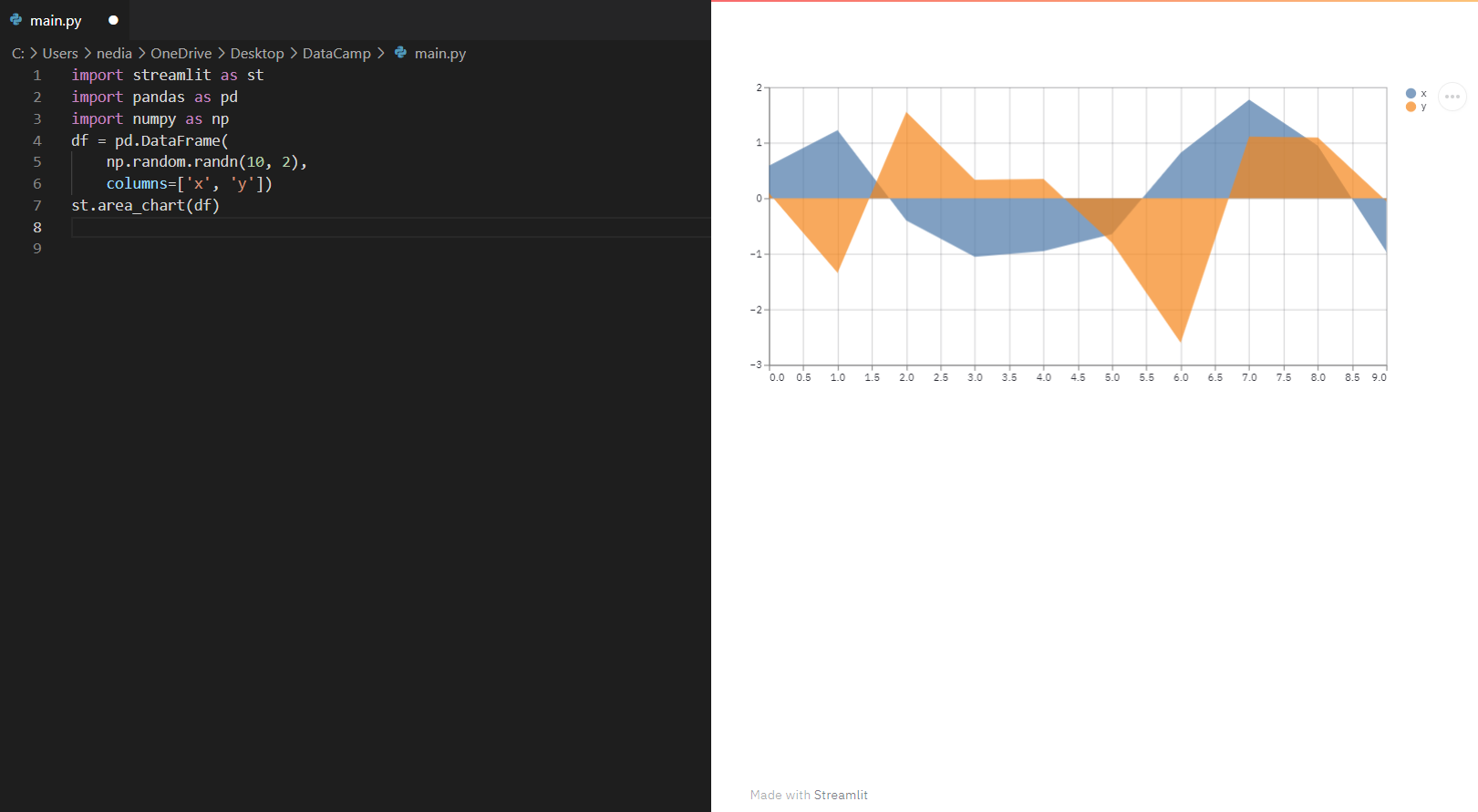
import streamlit as st

import pandas as pd

import numpy as np

df= pd.DataFrame( np.random.randn(10, 2), columns=['x', 'y'])

st.area\_chart(df)



st.altair\_chart(): This function is used to display an altair chart.

import streamlit as st

import numpy as np

import pandas as pd

import altair as alt​

df = pd.DataFrame( np.random.randn(500, 3), columns=['x','y','z'])​

c = alt.Chart(df).mark\_circle().encode( x='x' , 'y'=y , size='z', color='z', tooltip=['x', 'y', 'z'])

st.altair\_chart(c, use\_container\_width=True)

st.altair\_chart(): This function is used to display an altair chart.

import streamlit as st

import numpy as np

import pandas as pd

import altair as alt

​df = pd.DataFrame( np.random.randn(500, 3), columns=['x','y','z'])​

c = alt.Chart(df).mark\_circle().encode( x='x' , y='y' , size='z', color='z', tooltip=['x', 'y', 'z'])

st.altair\_chart(c, use\_container\_width=True)



**DISPLAY MAPS WITH STREAMLIT:**

st.map(): This function is used to display maps in the app. However, it requires the values of latitude and longitude and these values should not be null/NA.

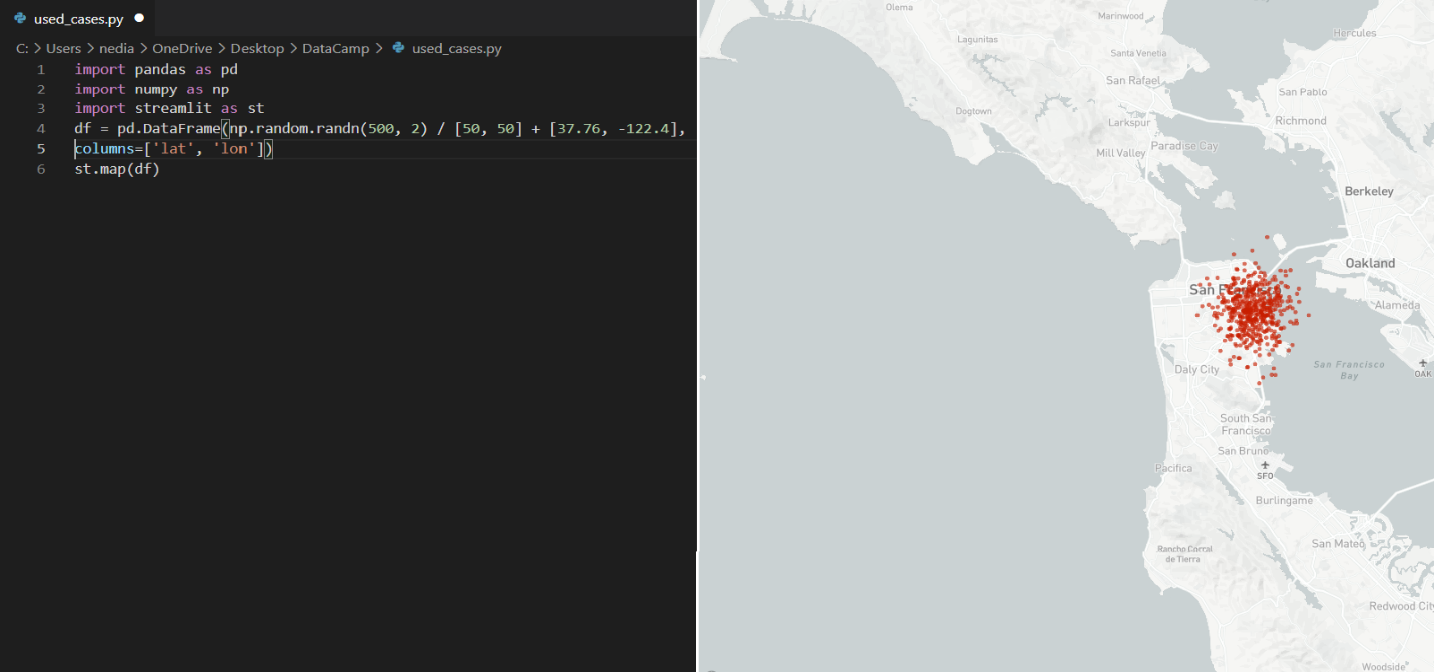
import pandas as pd

import numpy as np

import streamlit as st

df = pd.DataFrame(np.random.randn(500, 2) / [50, 50] + [37.76, -122.4],columns=['lat', 'lon'])

st.map(df)



**BUILD A MACHINE LEARNING APPLICATION**

In this section, I will walk you through a project I made about loan prediction.

The main profit of loans comes directly from the loan's interest. The loan companies grant a loan after an intensive process of verification and validation. However, they still don't have assurance if the applicant is able to repay the loan with no difficulties. In this tutorial, we will build a predictive model (Random Forest Classifier) to predict the loan status of an applicant. Our mission is to prepare a web app to make it available in production.

import streamlit as st

import pandas as pd

import numpy as np

import pickle # to load a saved model

import base64 # to open .gif files in streamlit app

In this app, we will use multiple widgets as sliders: selectbox and radio in the sidebar menu, for which we will prepare some Python functions.The example will be a simple demo that has two pages. On the homepage, it will show the data that we selected, whereas the Exploration page will allow you to visualize variables in plots, and the Prediction page will contain variables with a button named Predict that will allow you to estimate the loan status. The code below gives you a selectbox on the sidebar which allows you to select a page. The data is cached so that it does not need to reload constantly.

@st.cache is a caching mechanism that allows your app to stay performant even when loading data from the web, manipulating large datasets, or performing expensive computations.

@st.cache(suppress\_st\_warning=True)

def get\_fvalue(val):

feature\_dict = {"No":1, "Yes":2}

for key, value in feature\_dict.items():

if val == key:

return value

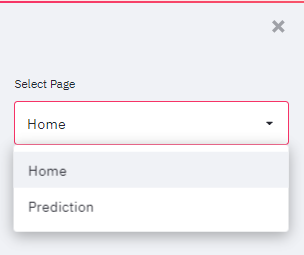
def get\_value(val, my\_dict):

for key, value in my\_dict.items():

if val == key:

return value

app\_mode = st.sidebar.selectbox('Select Page', ['Home', 'Prediction']) # two pages



In the Home page, we will visualize: presentation picture / the dataset / histogram of applicant income and loan amount.

Note: We will use if/elif/else to switch between pages.

We will load the loan\_dataset.csv in variable data that will allow us to show a few lines of it in the Home page.

If app\_mode == ‘Home’:

st.title(‘LOAN PREDICTION:’)

st.image(‘loan\_image.jpg’)

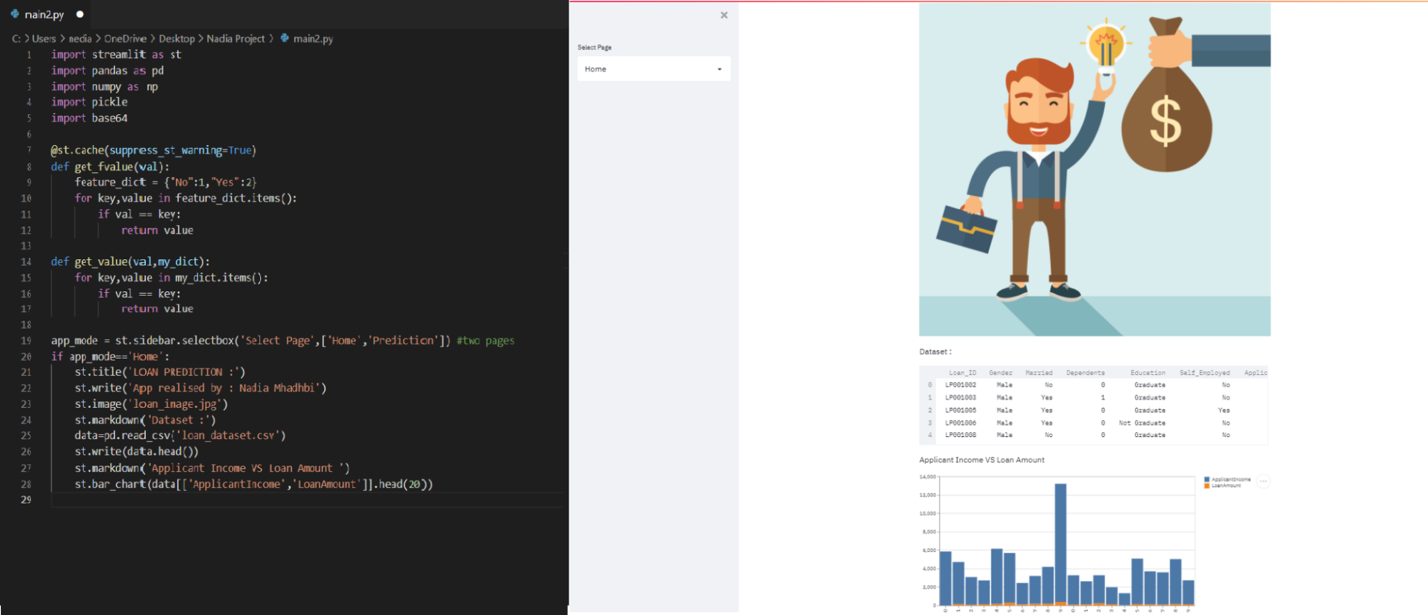
st.markdown(‘Dataset:’)

data = pd.read\_csv(‘loan\_dataset.csv’)

st.write(data.head())

st.markdown(‘Applicant Income VS Loan Amount’)

st.bar\_chart(data[[‘ApplicantIncome', 'LoanAmount']].head(20))



Then in the Prediction page:

elif app\_mode == 'Prediction':

st.image('slider-short-3.jpg')

st.subheader('Sir/Madame , YOU need to fill all necessary information in order to get a reply to your loan request !')

st.sidebar.header("Information about the client :")

gender\_dict = {"Male": 1, "Female": 2}

feature\_dict = {"No": 1, "Yes": 2}

edu = {'Graduate': 1, 'Not Graduate': 2}

prop = {'Rural': 1, 'Urban': 2, 'Semiurban': 3}

ApplicantIncome = st.sidebar.slider('ApplicantIncome', 0, 10000, 0)

CoapplicantIncome = st.sidebar.slider('CoapplicantIncome', 0, 10000, 0)

LoanAmount = st.sidebar.slider('LoanAmount in K $', 9.0, 700.0, 200.0)

Loan\_Amount\_Term = st.sidebar.selectbox('Loan\_Amount\_Term - Months', (12.0, 36.0, 60.0, 84.0, 120.0, 180.0, 240.0, 300.0, 360.0))

Credit\_History = st.sidebar.radio('Credit\_History', (0.0, 1.0))

Gender = st.sidebar.radio('Gender', tuple(gender\_dict.keys()))

Married = st.sidebar.radio('Married', tuple(feature\_dict.keys()))

Self\_Employed = st.sidebar.radio('Self Employed', tuple(feature\_dict.keys()))

Dependents = st.sidebar.radio('Dependents', options=['0', '1', '2', '3+'])

Education = st.sidebar.radio('Education', tuple(edu.keys()))

Property\_Area = st.sidebar.radio('Property\_Area', tuple(prop.keys()))

class\_0, class\_3, class\_1, class\_2 = 0, 0, 0, 0

if Dependents == '0':

class\_0 = 1

elif Dependents == '1':

class\_1 = 1

elif Dependents == '2':

class\_2 = 1

else:

class\_3 = 1

Rural, Urban, Semiurban = 0, 0, 0

if Property\_Area == 'Urban':

Urban = 1

elif Property\_Area == 'Semiurban':

Semiurban = 1

else:

Rural = 1

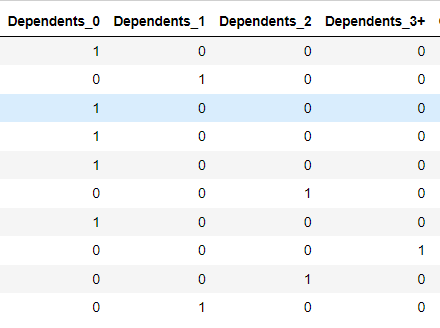
We wrote two functions get\_value(val,my\_dict) and get\_fvalue(val) and dictionaries as feature\_dict to manipulate st.sidebar.radio() with non-numeric variables. It's optional, you can easily do something like this:

Build a machine learning application 3

Let's see why we did that.

Note: Machine learning algorithms cannot handle categorical variables. In the dataset, I did some feature engineering. For example, the column Married has two variables 'Yes' and 'No' and I did a Label Encoding ( Take a look to better understand ) so "NO" will be equal to 1 and "Yes" to 2. The function get\_fvalue(val) will easily return the value (1/2) depending what the client has chosen. Same for the function get\_value(val,my\_dict) . The difference between the two functions is that the first works on yes/no features and the second one is in the general case when we have multiple variables ( example: Gender ).

As we can see the variable Dependents has four categories '0','1' , '2' and '3+' and we cannot convert something like that into a numeric variable, and we have '+3' that means Dependents can take 3,4,5 ... We did a One Hot Enconding ( Take a look to better understand ) Thus , we created a sidebar radio containing the four elements and each one has a binary variable, if the client chose '0' class\_0 will be equal to 1 and the others will be equal to 0.



Also we did One Hot Encoding for Property\_Area that's why we created 3 variables (Rural,Urban,Semiurban) ,When Rural takes 1 the others will be equal to 0.



So we have seen both—when we label or one hot encoding our features and how to deal with it to successfully created a working Streamlit app.

data1 = {

'Gender': Gender,

'Married': Married,

'Dependents': [class\_0, class\_1, class\_2, class\_3],

'Education': Education,

'ApplicantIncome': ApplicantIncome,

'CoapplicantIncome': CoapplicantIncome,

'Self Employed': Self\_Employed,

'LoanAmount': LoanAmount,

'Loan\_Amount\_Term': Loan\_Amount\_Term,

'Credit\_History': Credit\_History,

'Property\_Area': [Rural, Urban, Semiurban],

}

feature\_list = [

ApplicantIncome,

CoapplicantIncome,

LoanAmount,

Loan\_Amount\_Term,

Credit\_History,

get\_value(Gender, gender\_dict),

get\_fvalue(Married),

data1['Dependents'][0],

data1['Dependents'][1],

data1['Dependents'][2],

data1['Dependents'][3],

get\_value(Education, edu),

get\_fvalue(Self\_Employed),

data1['Property\_Area'][0],

data1['Property\_Area'][1],

data1['Property\_Area'][2]

]

single\_sample = np.array(feature\_list).reshape(1, -1)

Now we will store our variables in a dictionary because we wrote get\_value(val,my\_dict) and get\_fvalue(val) to deal with dictionaries. After that, the input—what the client will choose as input in our Streamlit app—will be arranged in a list named feature\_list then to a numpy variable named single\_sample.

Note: The inputs of features must be arranged in the same order of dataset columns (e.g. Married cannot take the input of Gender).

if st.button("Predict"):

file\_ = open("6m-rain.gif", "rb")

contents = file\_.read()

data\_url = base64.b64encode(contents).decode("utf-8")

file\_.close()

file = open("green-cola-no.gif", "rb")

contents = file.read()

data\_url\_no = base64.b64encode(contents).decode("utf-8")

file.close()

loaded\_model = pickle.load(open('Random\_Forest.sav', 'rb'))

prediction = loaded\_model.predict(single\_sample)

if prediction[0] == 0:

st.error('According to our calculations, you will not get the loan from the bank')

st.markdown(f'<img src="data:image/gif;base64,{data\_url\_no}" alt="cat gif">', unsafe\_allow\_html=True)

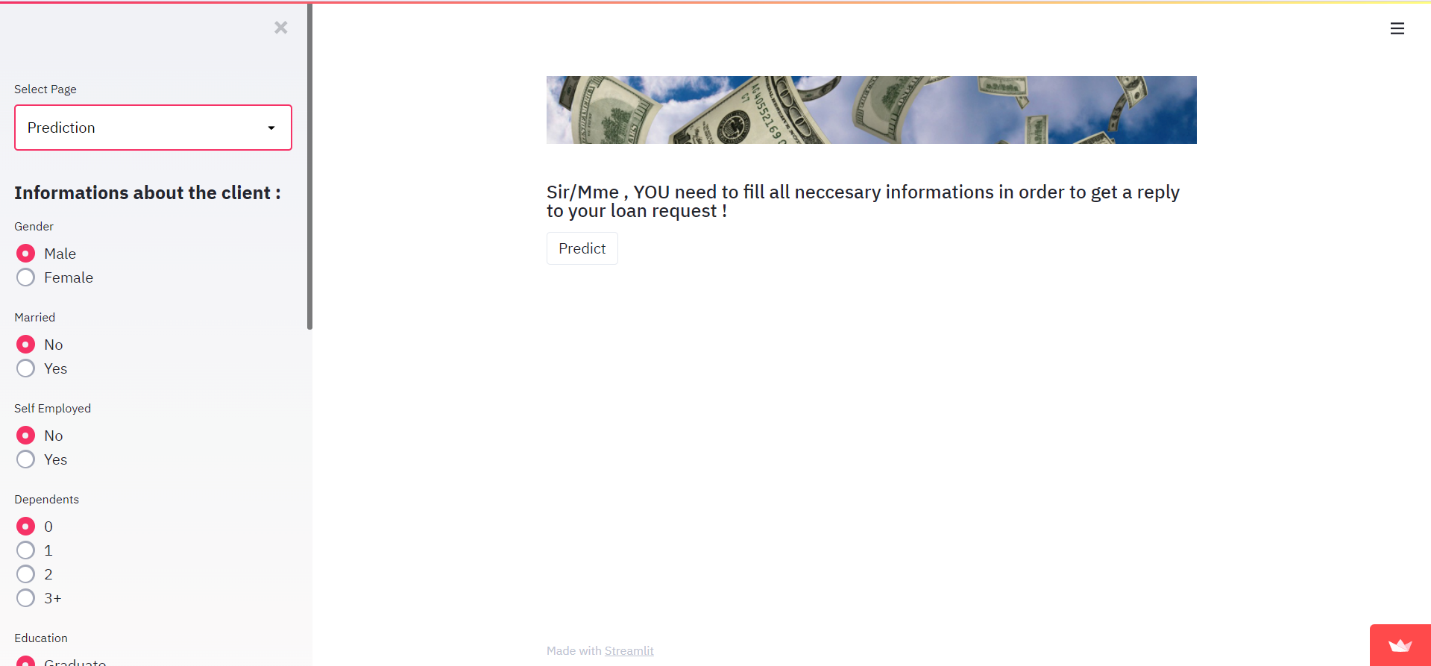
elif prediction[0] == 1:

st.success('Congratulations! You will get the loan from the bank')

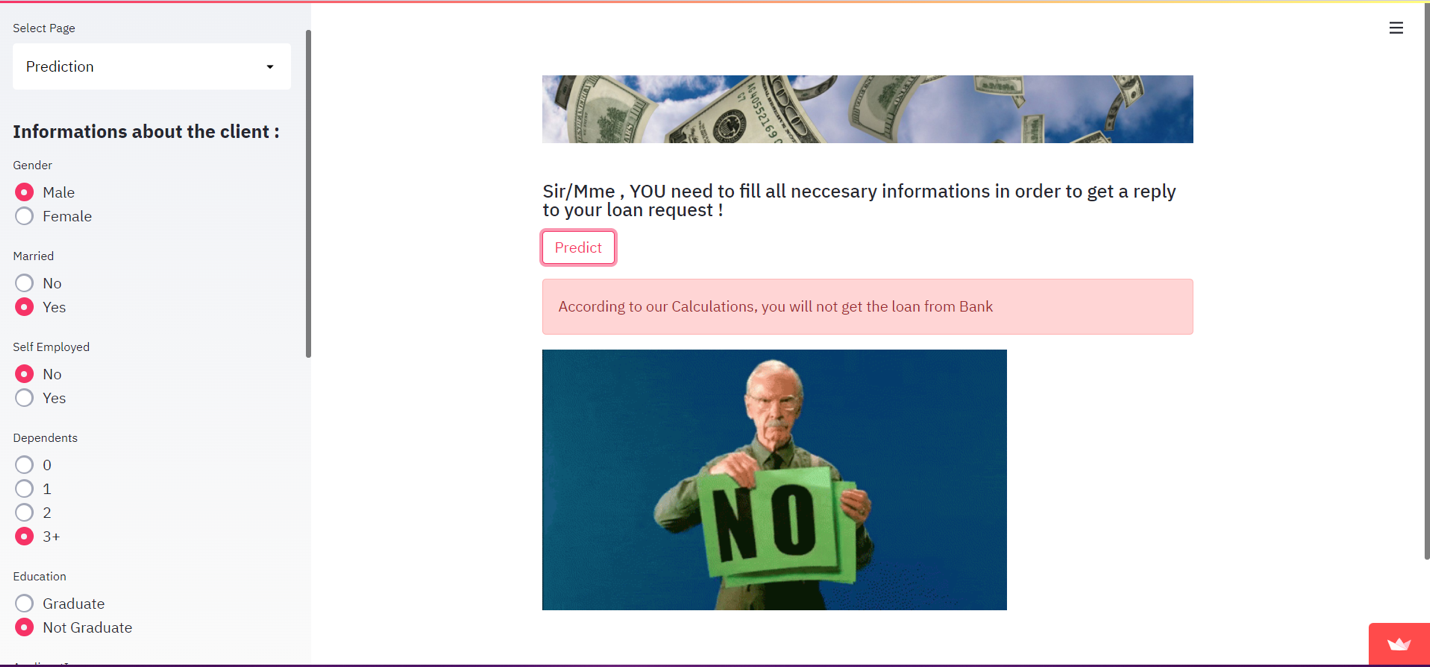
st.markdown(f'<img src="data:image/gif;base64,{data\_url}" alt="cat gif">', unsafe\_allow\_html=True)

Finally, we will load our saved RandomForestClassifier model in loaded\_model and its prediction, which is 0 or 1 (classification problem) in prediction. The .gif files will be stored in file and file\_. Depending on the value of prediction, we will have two cases, "Success" or "Failed," to get a loan from the bank.

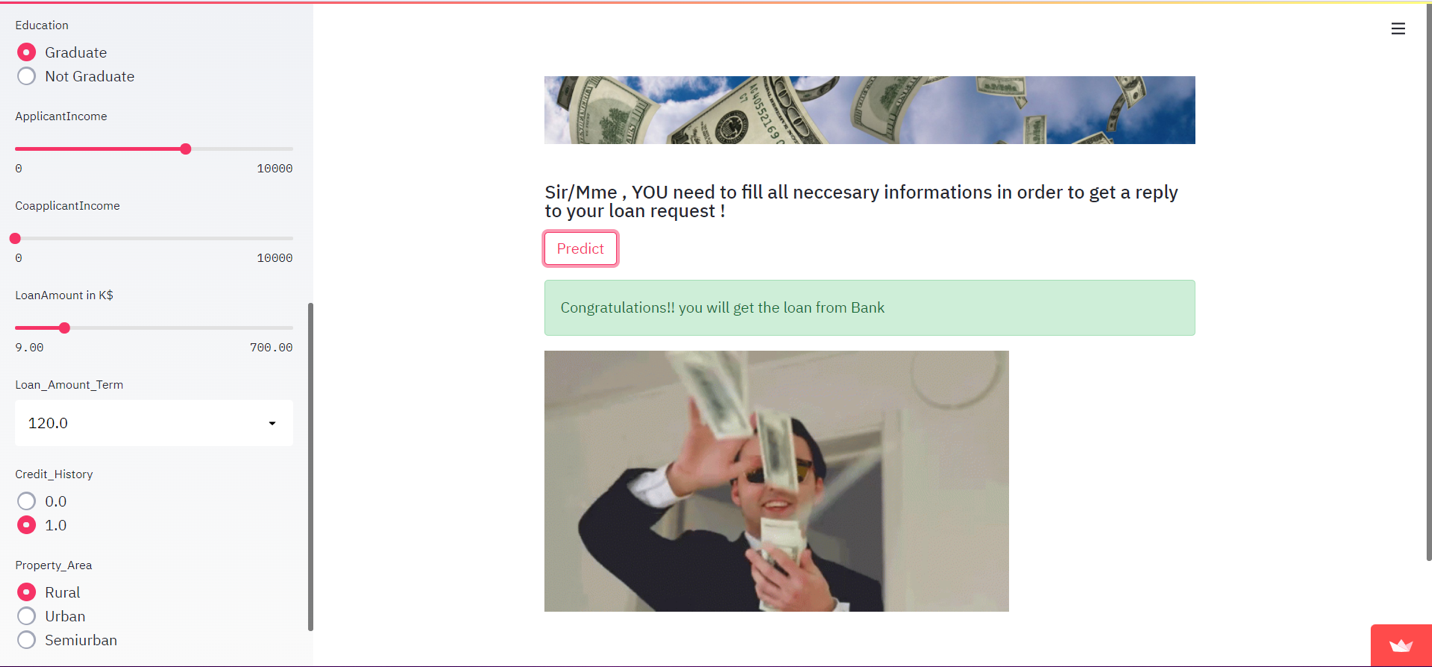
This is our Prediction page:



In the case of FAILURE, the output will look like this:



In the case of SUCCESS, the output will look like this:



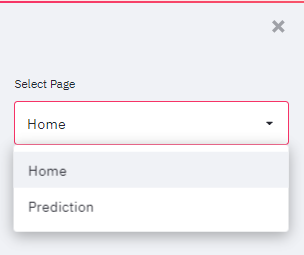
**HOW TO DEPLOY A STREAMLIT APP**

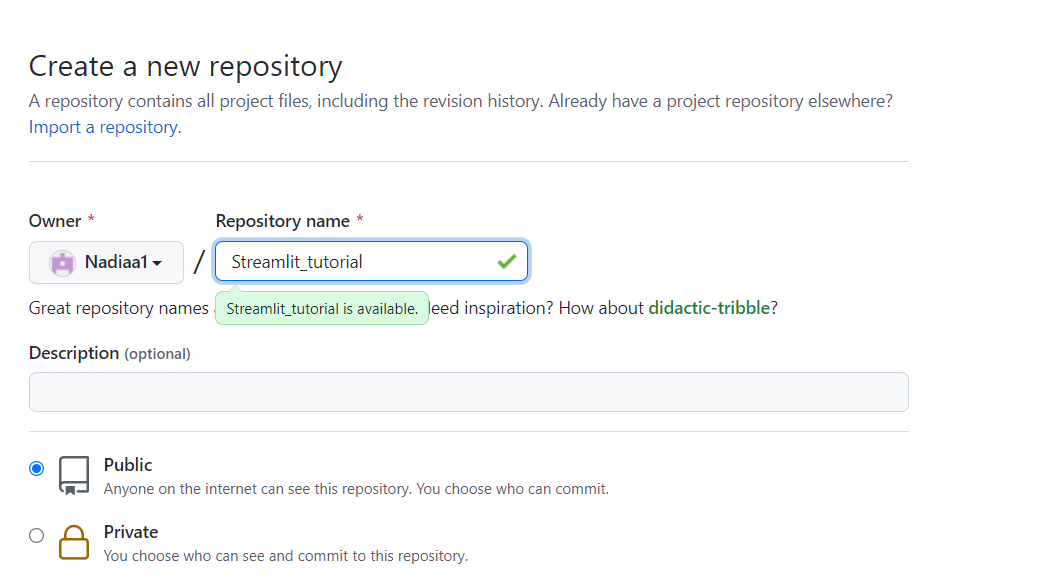
Deployment is the mechanism through which applications are delivered from developers to users.

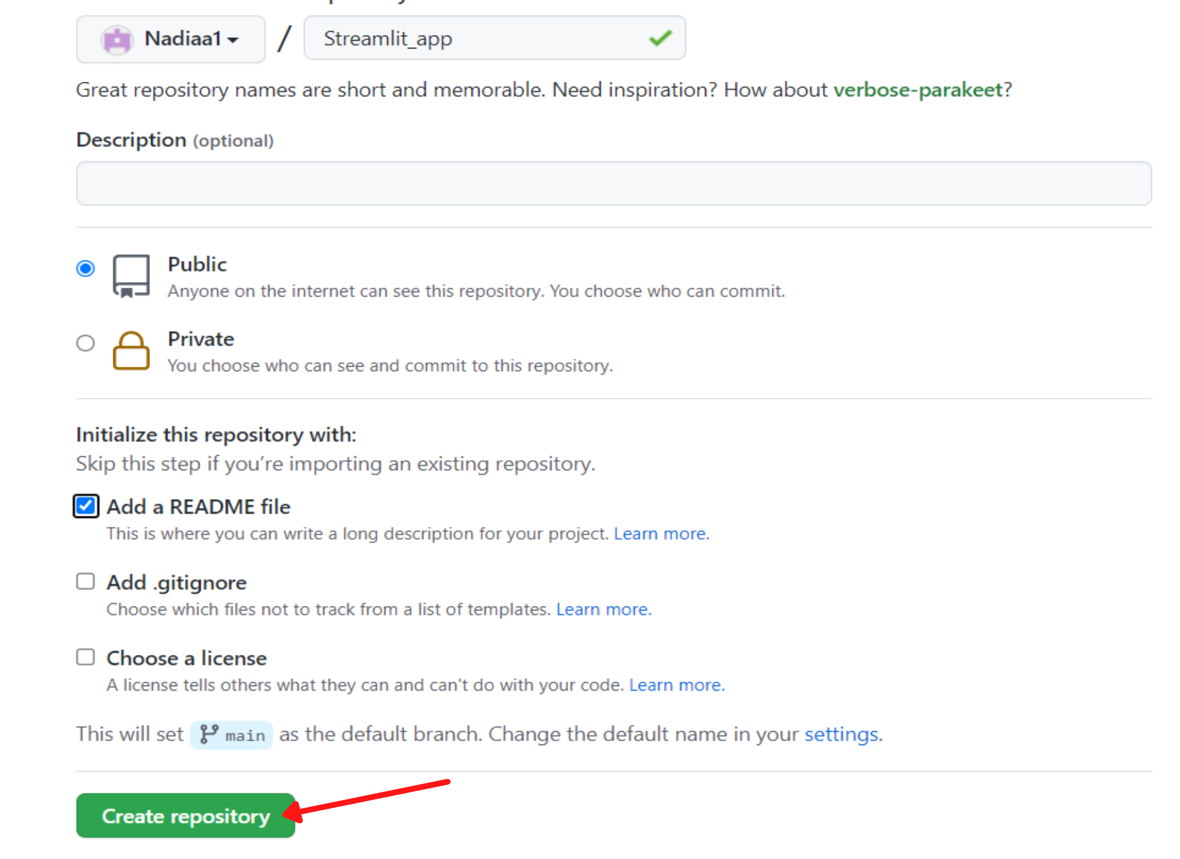
Deploying an application is the process of copying, configuring, and enabling a specific application to a specific base URL. Once the deployment process has finished, the application becomes publicly accessible on the base URL. The server carries out this two-step process by first staging the application, and then activating it after successful staging.

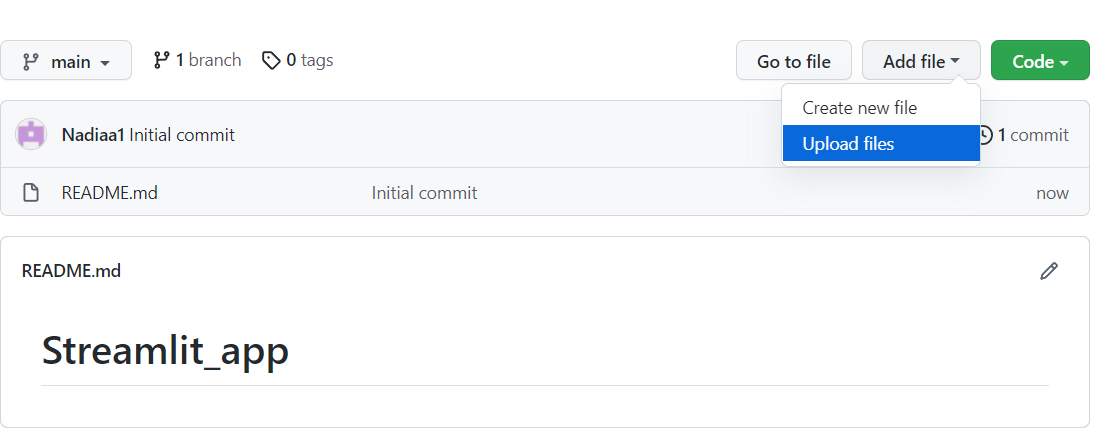
Let's learn how to deploy a Streamlit app!

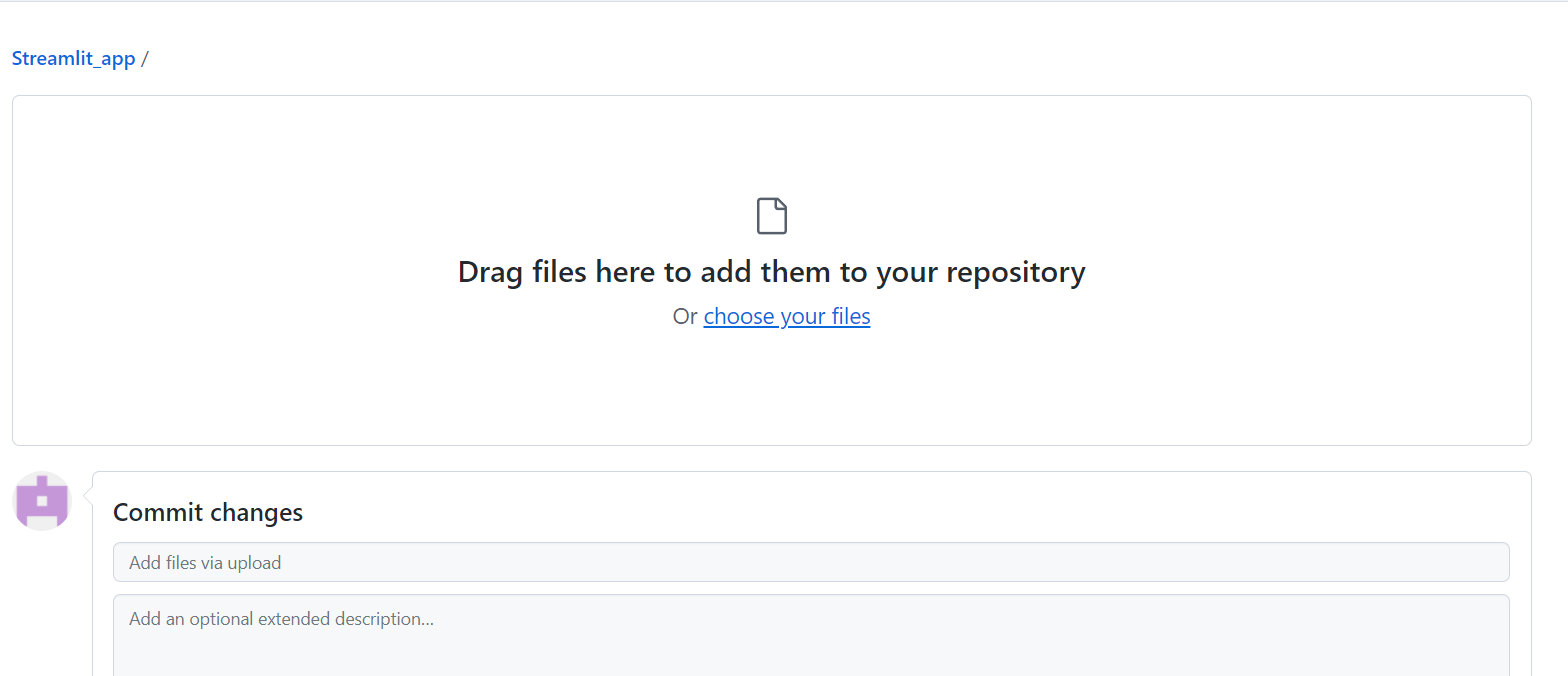
Before you try to deploy your app, you need to create a new repository on your GitHub where you need to put your app code and dependencies.



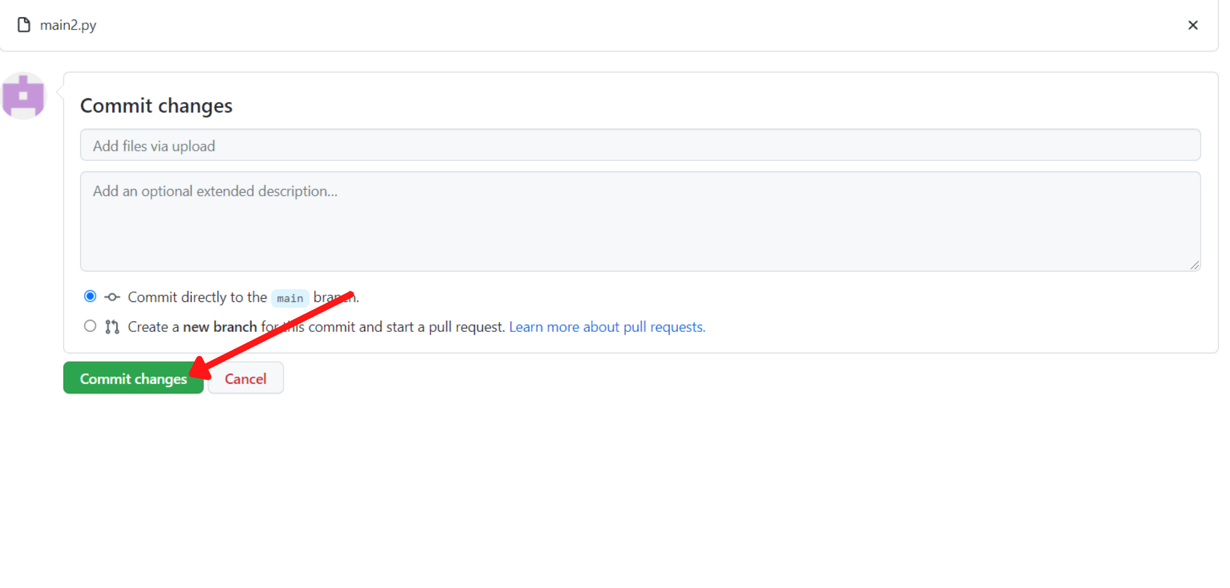






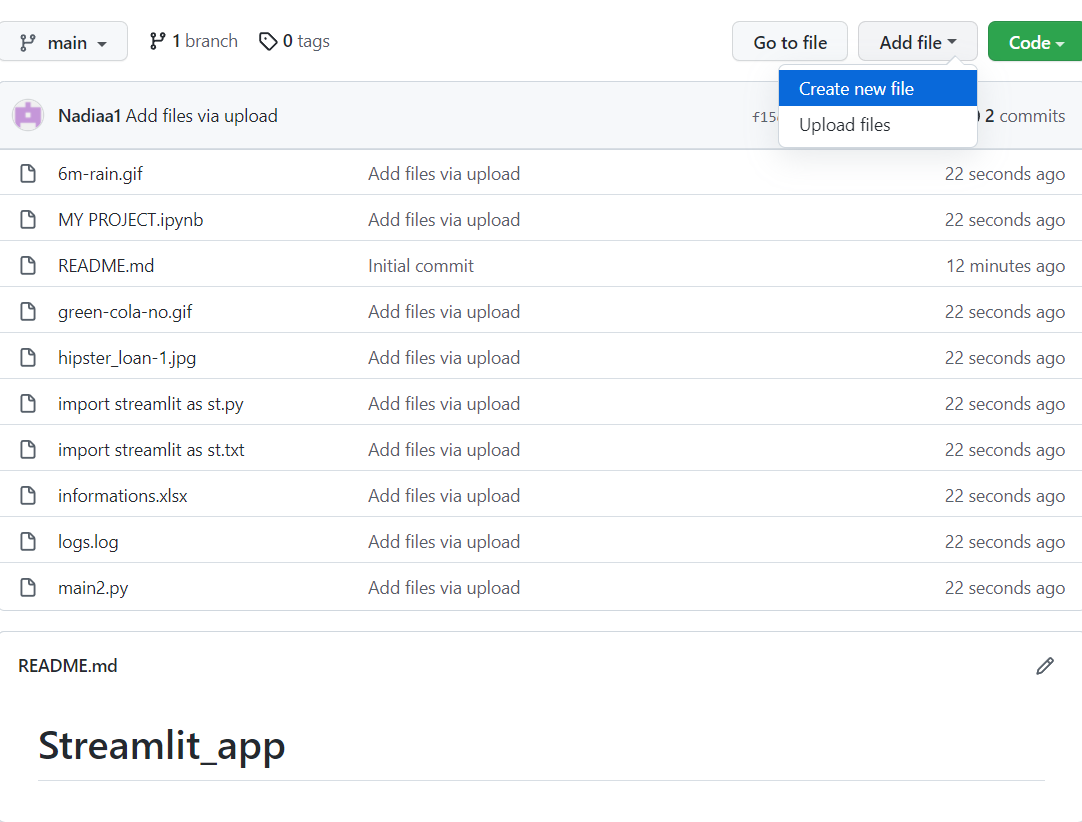


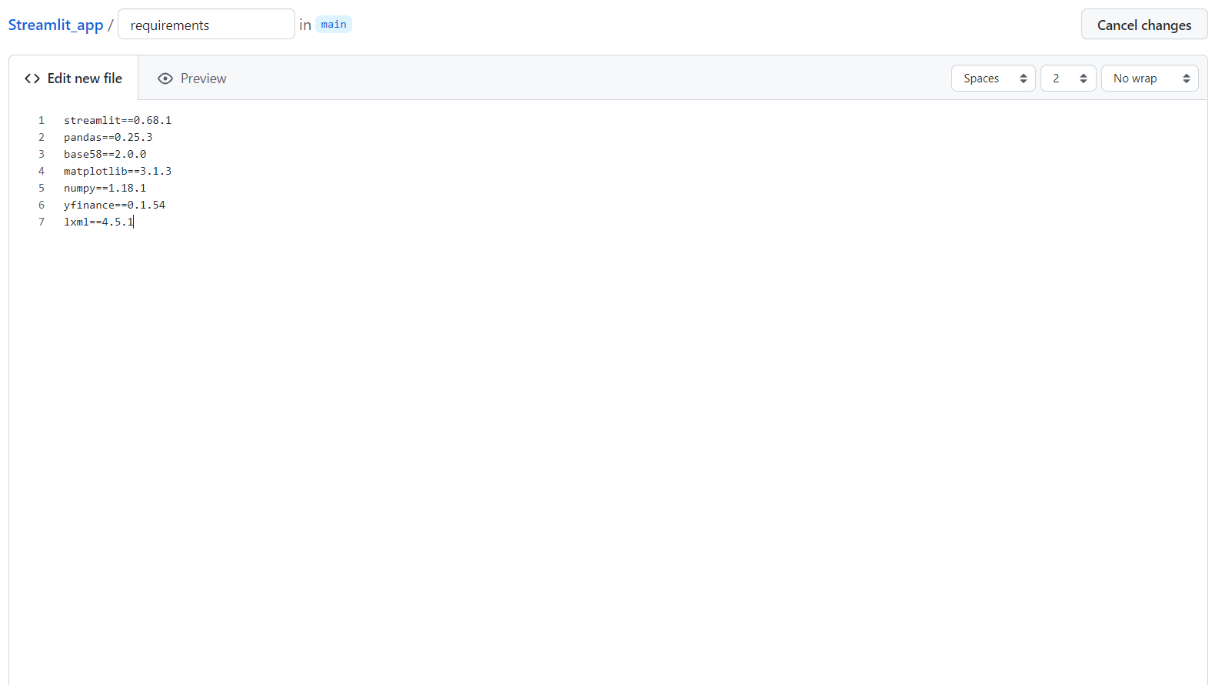
Then click on commit changes to save them:



After creating a repository and uploading files, you need to create a new file named requirements where you have to put the libraries you used in your app.

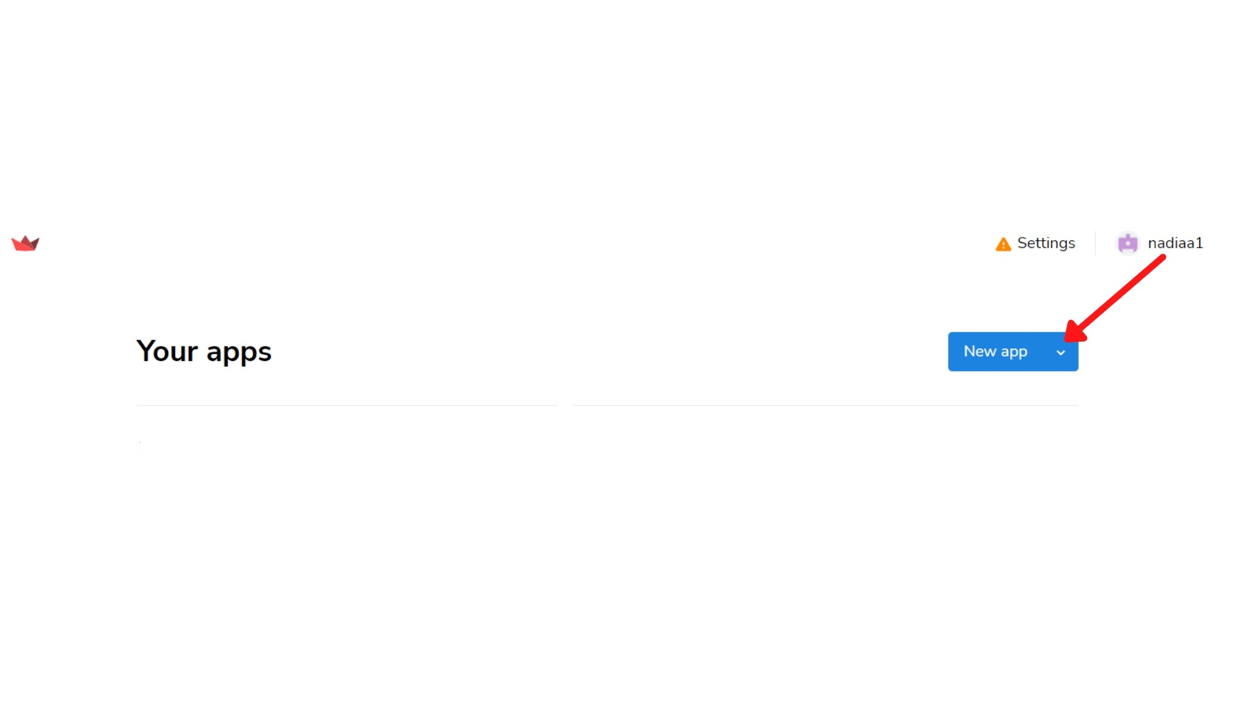
First, click on create new file.

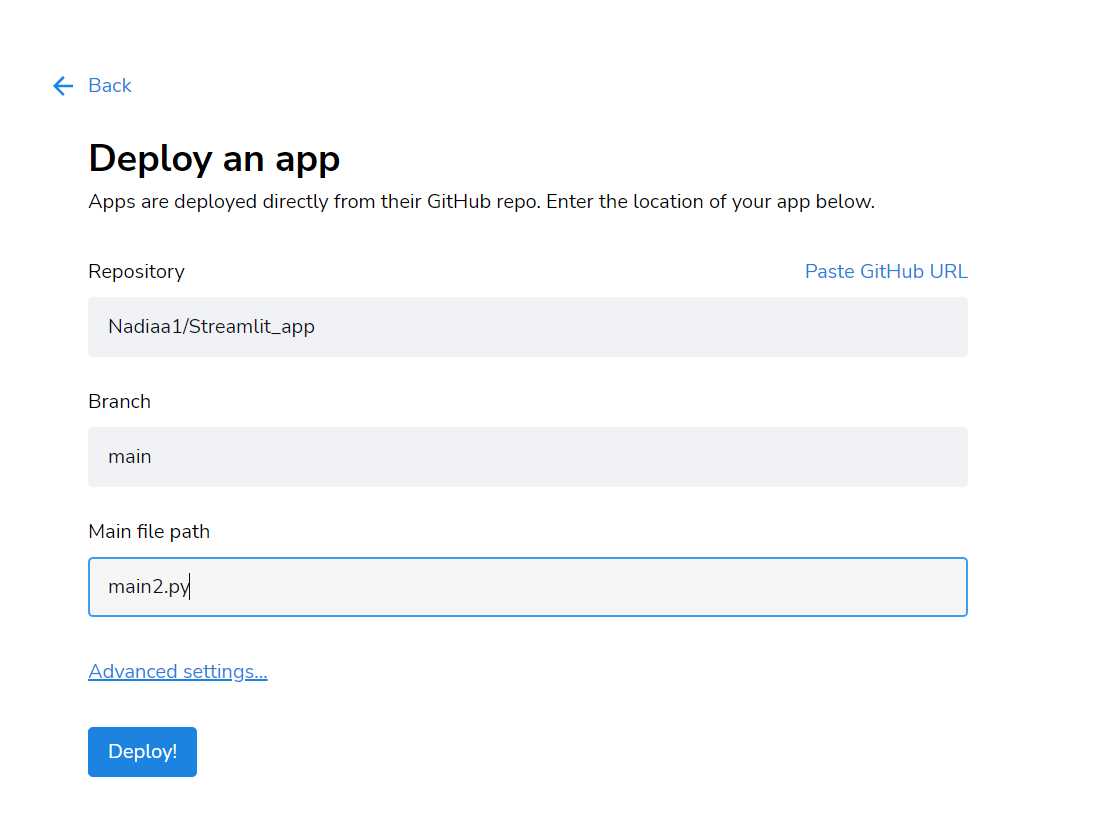


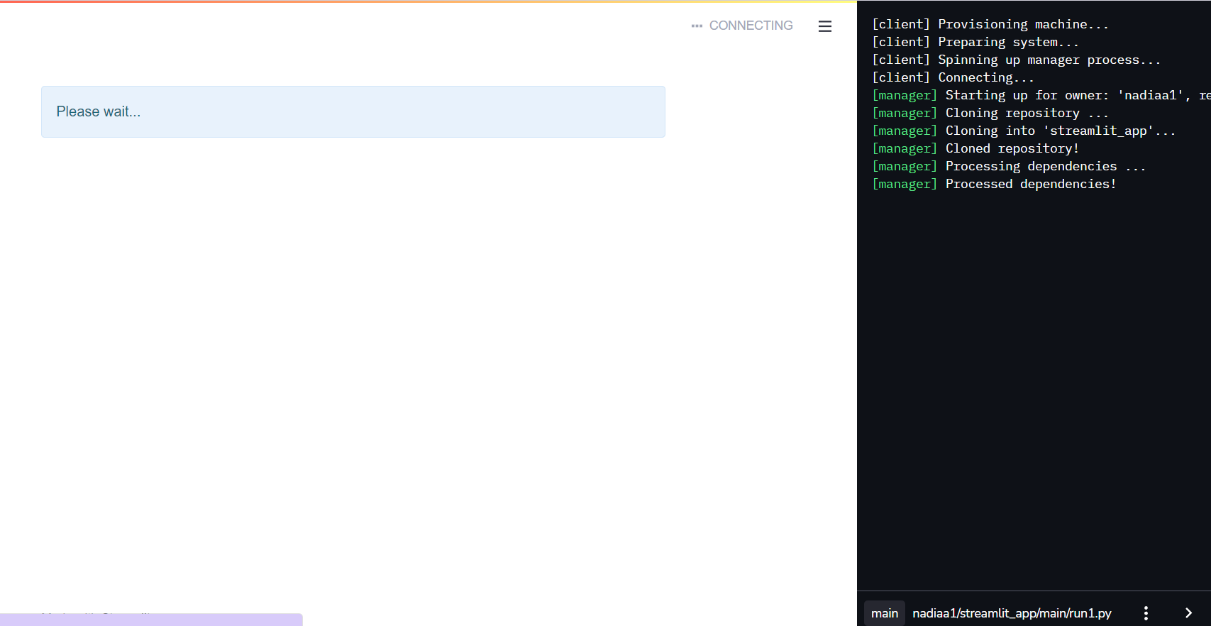


Now you're close to deploying your app, all you need is to visit this [**link**](https://share.streamlit.io/).

Then follow these steps:







Click on Deploy and wait for a moment!

A page will open automatically in your browser! This page is your project app realized with Streamlit.

Congratulations, you've successfully deployed your app!