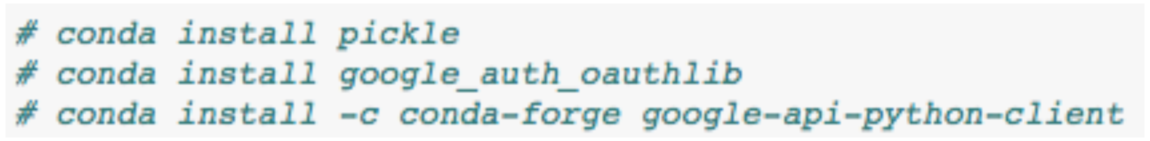
Intro to PyMail2

Create and send e-mails using Python from Jupyter Notebook.

PyMail2 because an actual PyMail exists out there

Program requirements

- Python
- Anaconda/Jupyter Notebook
- The following packages:
  - pickle
  - google_auth_oauthlib
  - google-api-python-client



```
# conda install pickle
# conda install google_auth_oauthlib
# conda install -c conda-forge google-api-python-client
```

Sections

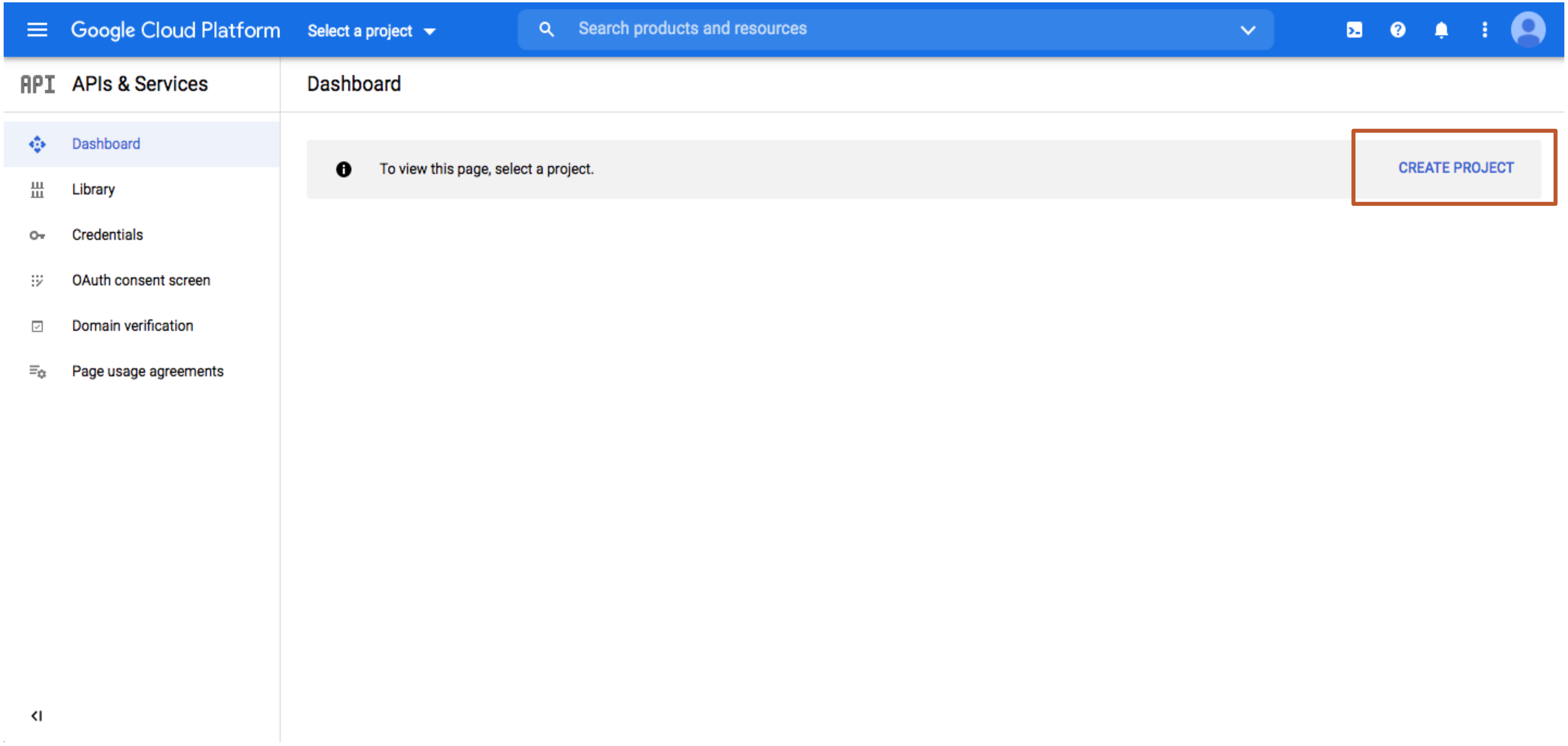
Create Google API .json File

0. First, create an account or login to Google Developers Console

Link: https://console.developers.google.com/

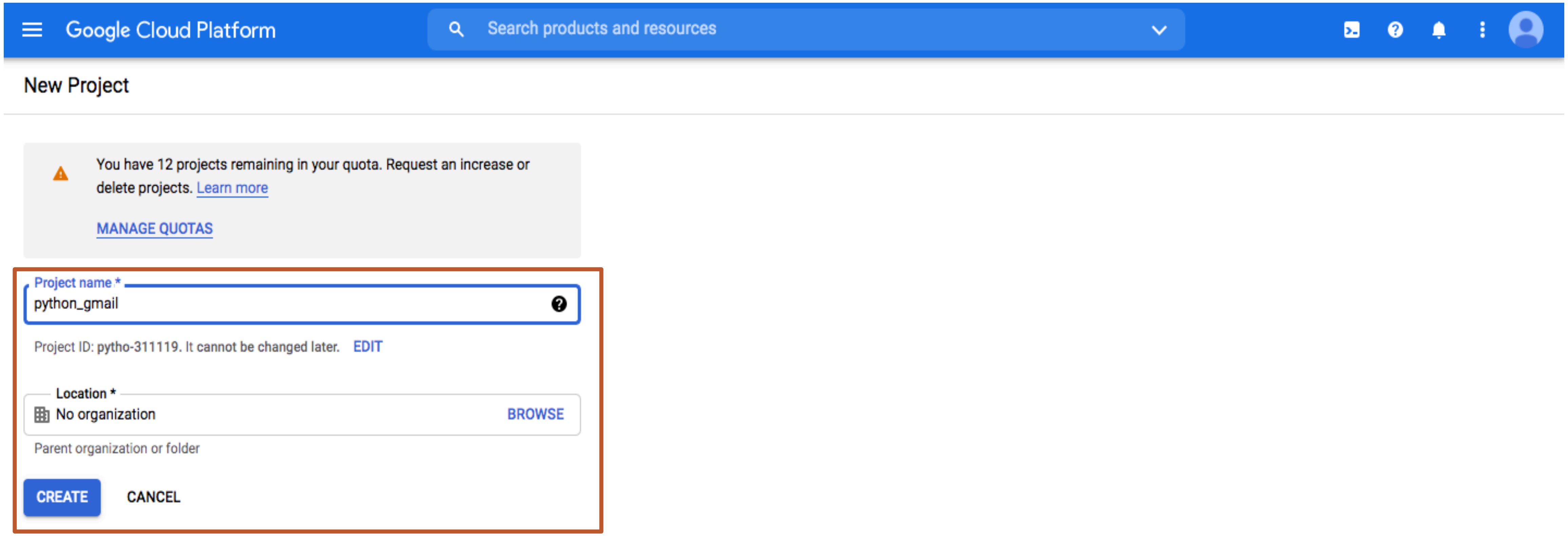
The next few slides will provide step-by-step instructions to generate the .json file required to access Google’s API.

1. From your dashboard, create project.



To view this page, select a project.

2. Name the project



New Project

You have 12 projects remaining in your quota. Request an increase or delete projects. Learn more

MANAGE QUOTAS

Project name *
python_gmail

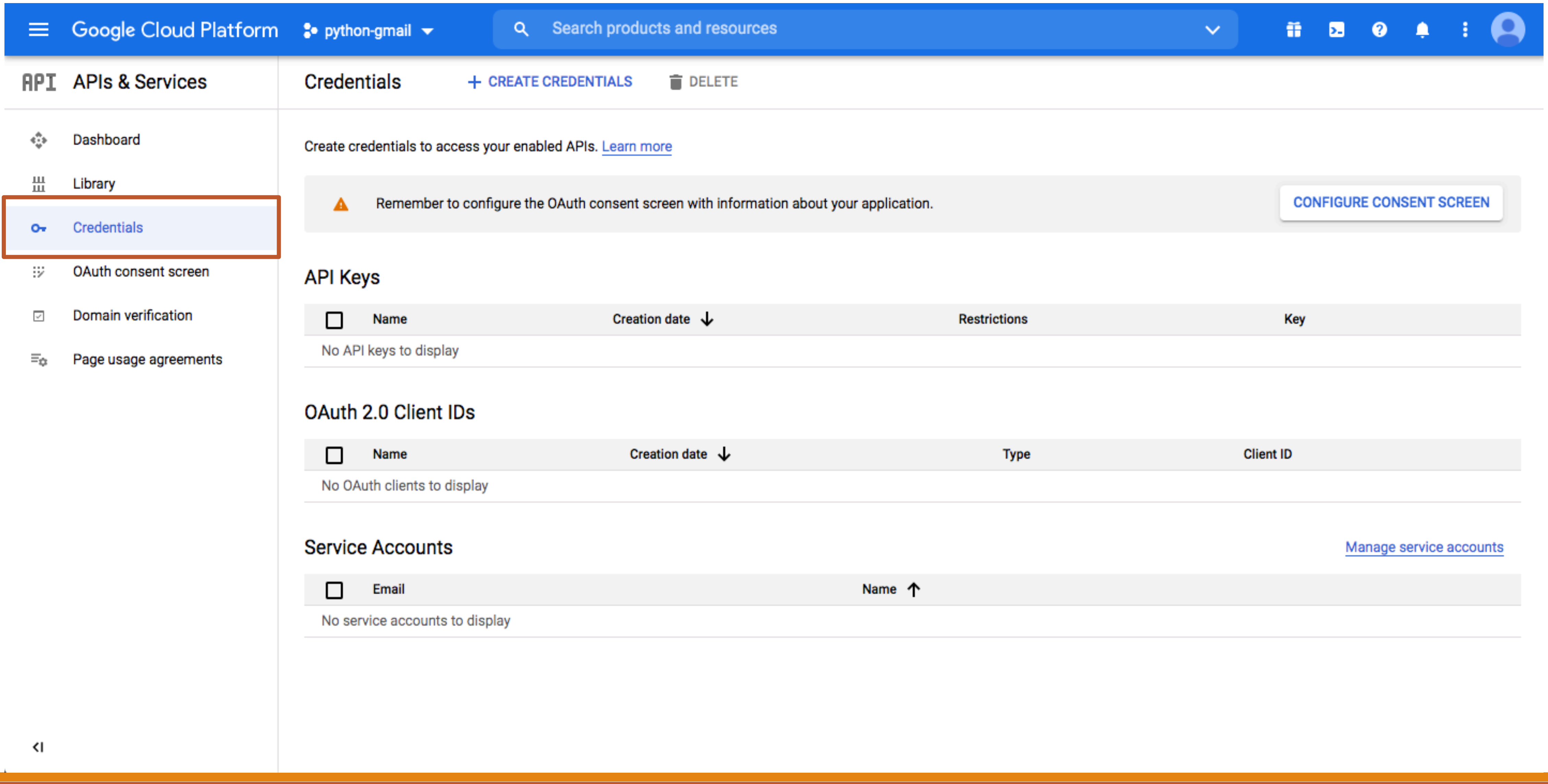
Project ID: pytho-311119. It cannot be changed later. EDIT

Location *
No organization BROWSE

Parent organization or folder

CREATE CANCEL

3. Once the project is created, select ‘Credentials’ from sidebar.



Create credentials to access your enabled APIs. Learn more

Remember to configure the OAuth consent screen with information about your application.

API Keys

Name	Creation date ↓	Restrictions	Key
No API keys to display			

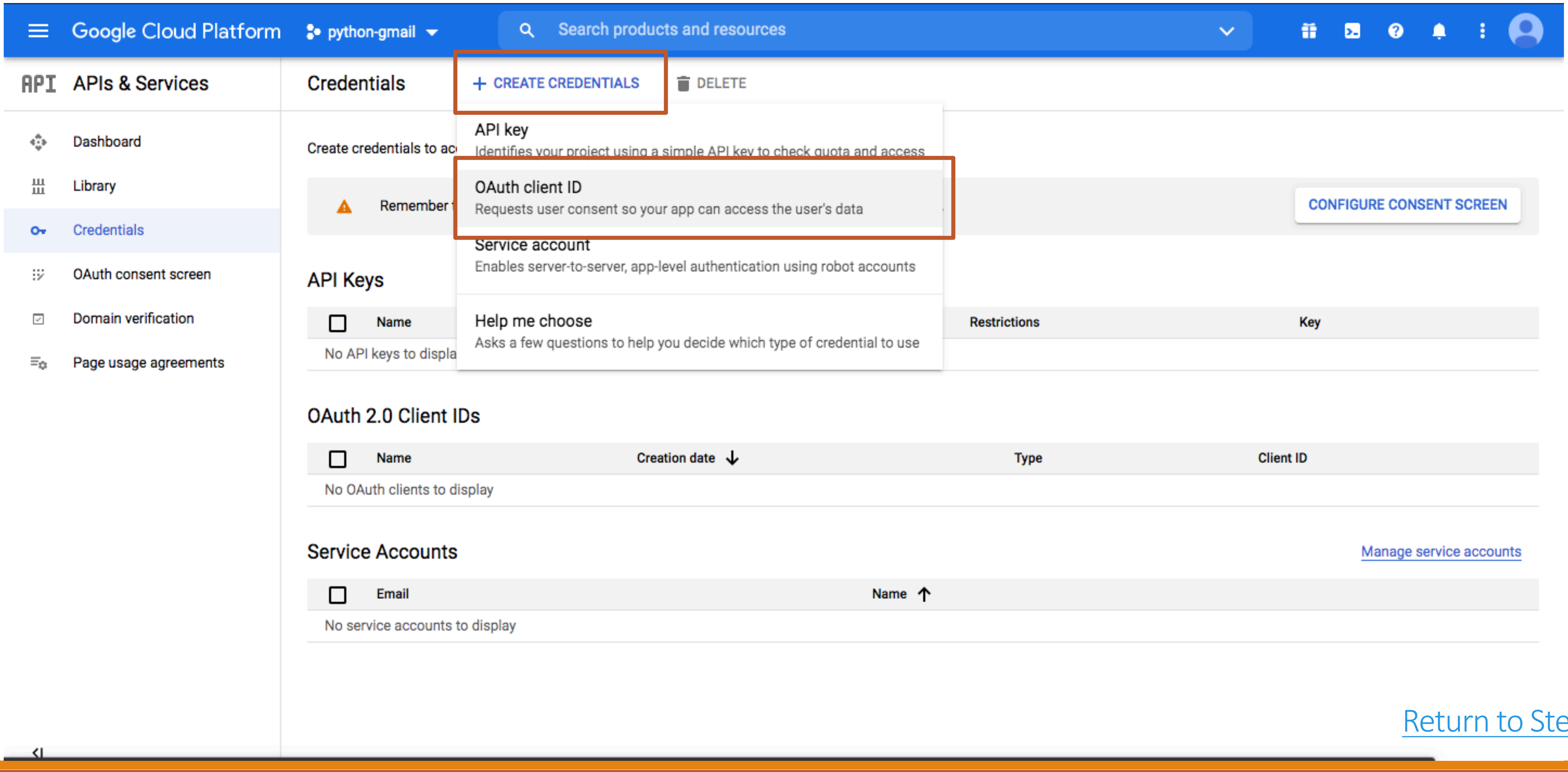
OAuth 2.0 Client IDs

Name	Creation date ↓	Type	Client ID
No OAuth clients to display			

Service Accounts

Email	Name ↑
No service accounts to display	

4. Select ‘Create Credentials’ then ‘OAuth client ID’



Google Cloud Platform   python-gmail   Search products and resources

APIs & Services   Credentials   + CREATE CREDENTIALS   DELETE

Dashboard
Library
Credentials
OAuth consent screen
Domain verification
Page usage agreements

Create credentials to ac

Remember

CONFIGURE CONSENT SCREEN

API key
Identifies your project using a simple API key to check quota and access

OAuth client ID
Requests user consent so your app can access the user's data

Service account
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

API Keys

Name	Restrictions	Key
No API keys to displa		

OAuth 2.0 Client IDs

Name	Creation date ↓	Type	Client ID
No OAuth clients to display			

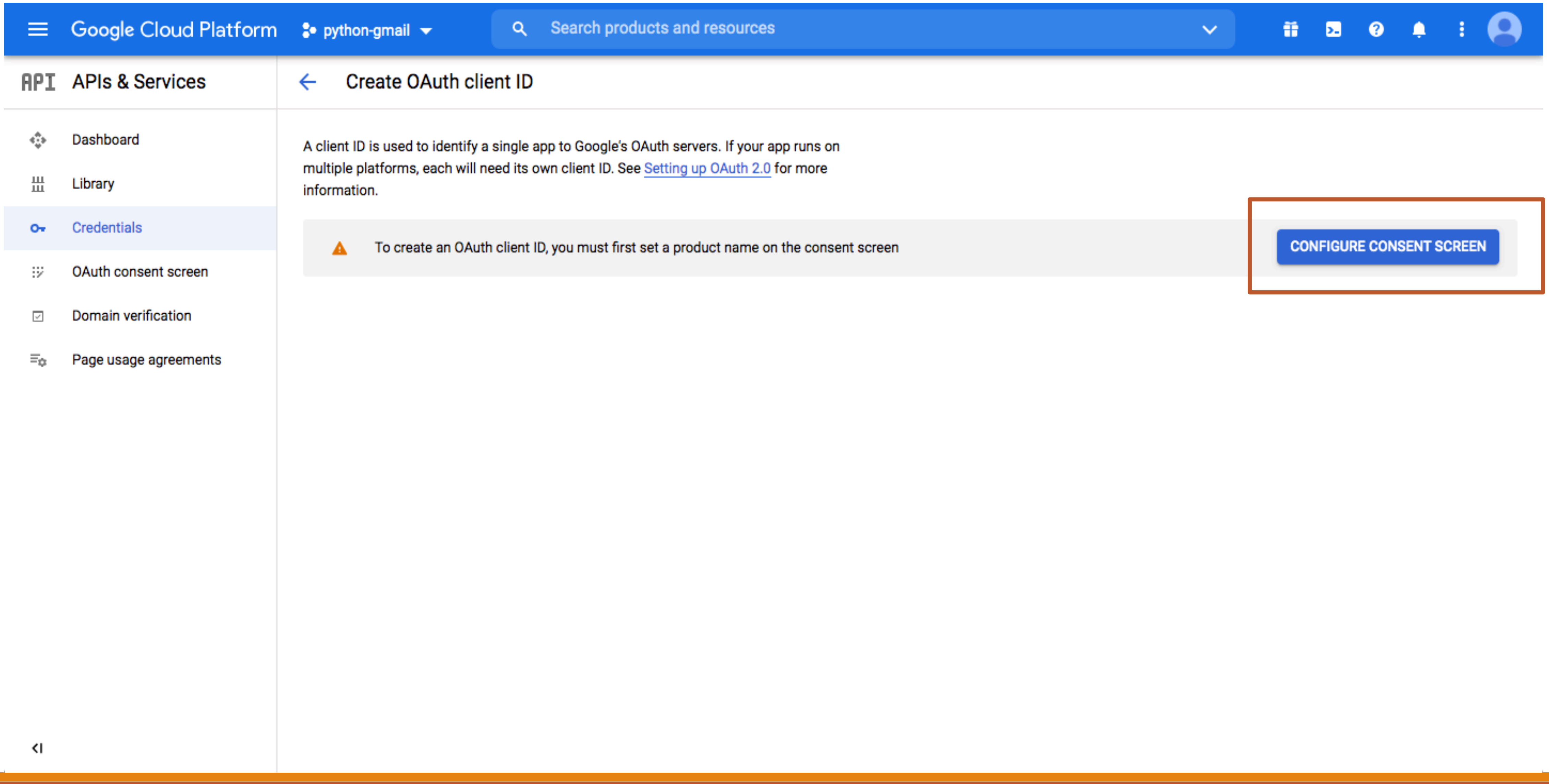
Service Accounts

Manage service accounts

Email	Name ↑
No service accounts to display	

Return to Step 10

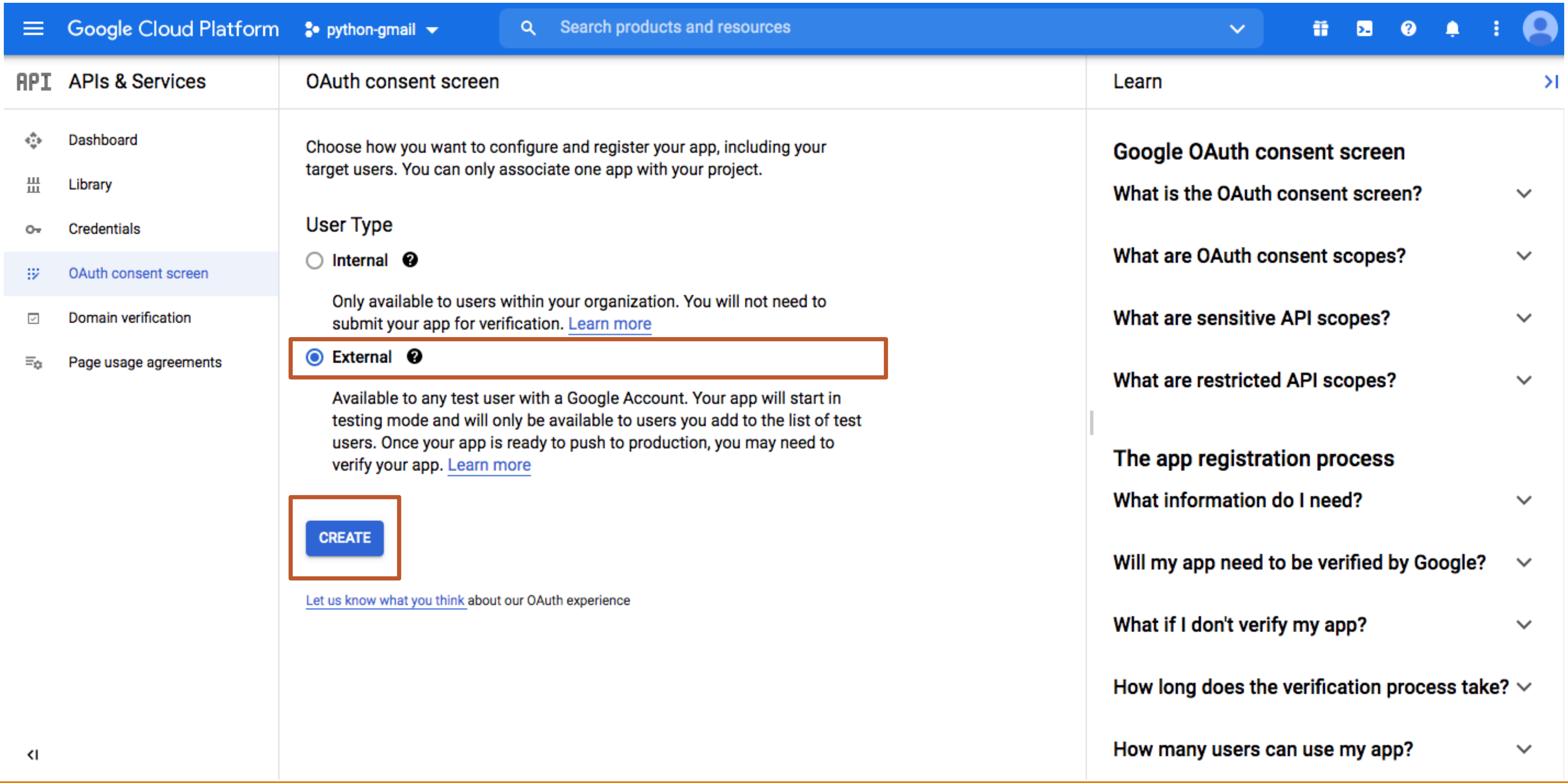
5. If no product exists, you will have to create a product first.



A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See Setting up OAuth 2.0 for more information.

To create an OAuth client ID, you must first set a product name on the consent screen

6. Select ‘External’ then create.



OAuth consent screen

Choose how you want to configure and register your app, including your target users. You can only associate one app with your project.

User Type

Internal

Only available to users within your organization. You will not need to submit your app for verification. Learn more

External [✓]

Available to any test user with a Google Account. Your app will start in testing mode and will only be available to users you add to the list of test users. Once your app is ready to push to production, you may need to verify your app. Learn more

CREATE

Let us know what you think about our OAuth experience

Google OAuth consent screen

What is the OAuth consent screen?

What are OAuth consent scopes?

What are sensitive API scopes?

What are restricted API scopes?

The app registration process

What information do I need?

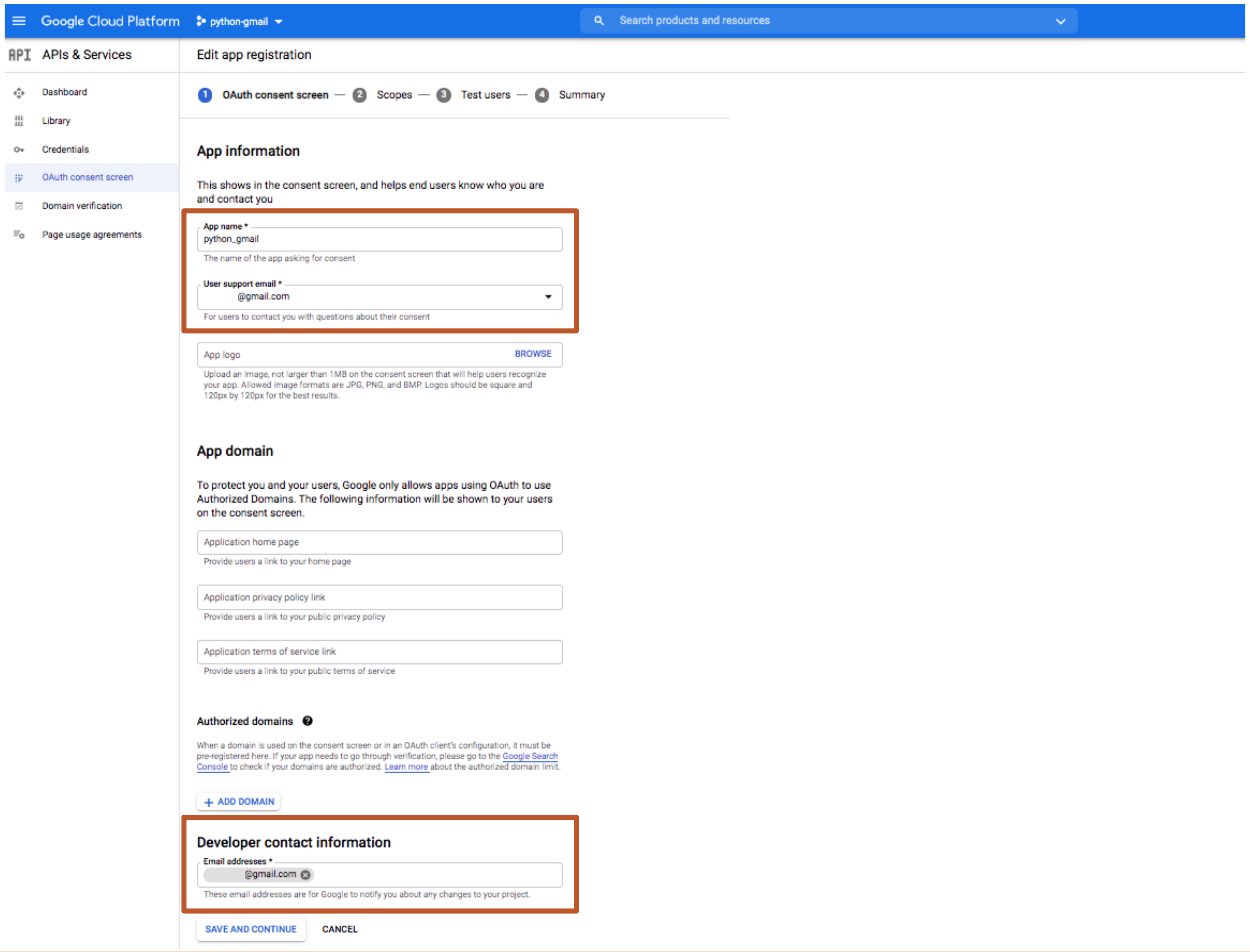
Will my app need to be verified by Google?

What if I don't verify my app?

How long does the verification process take?

How many users can use my app?

7. Fill in necessary product information and continue.



App information

This shows in the consent screen, and helps end users know who you are and contact you

App name *
python_gmail
The name of the app asking for consent

User support email *
@gmail.com
For users to contact you with questions about their consent

App logo BROWSE
Upload an image, not larger than 1MB on the consent screen that will help users recognize your app. Allowed image formats are JPG, PNG, and BMP. Logos should be square and 120px by 120px for the best results.

App domain

To protect you and your users, Google only allows apps using OAuth to use Authorized Domains. The following information will be shown to your users on the consent screen.

Application home page
Provide users a link to your home page

Application privacy policy link
Provide users a link to your public privacy policy

Application terms of service link
Provide users a link to your public terms of service

Authorized domains

When a domain is used on the consent screen or in an OAuth client's configuration, it must be pre-registered here. If your app needs to go through verification, please go to the Google Search Console to check if your domains are authorized. Learn more about the authorized domain limit.

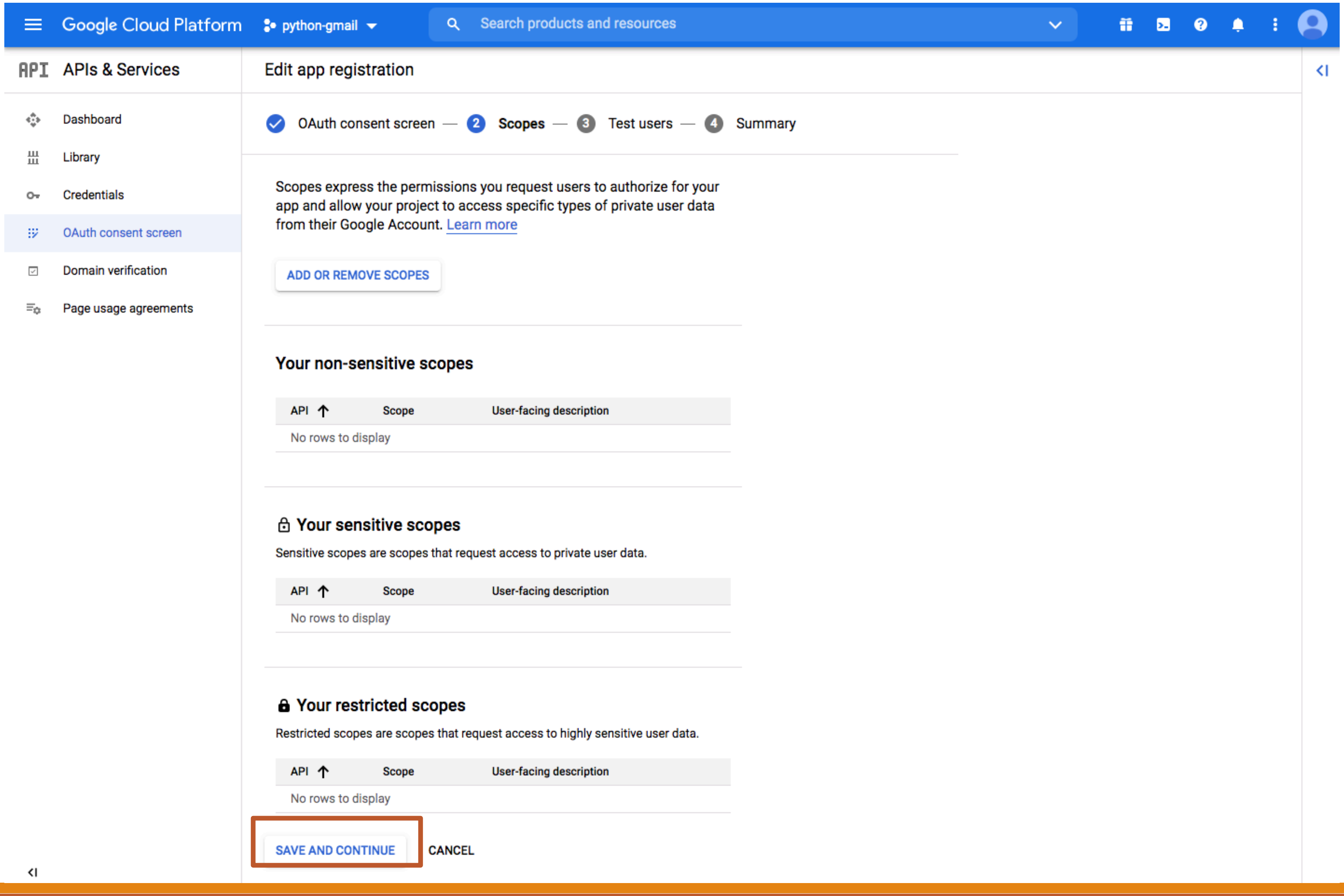
+ ADD DOMAIN

Developer contact information

Email addresses *
@gmail.com
These email addresses are for Google to notify you about any changes to your project.

SAVE AND CONTINUE CANCEL

8. (next page) Continue.



Edit app registration

OAuth consent screen — 2 Scopes — 3 Test users — 4 Summary

Scopes express the permissions you request users to authorize for your app and allow your project to access specific types of private user data from their Google Account. Learn more

ADD OR REMOVE SCOPES

Your non-sensitive scopes

API ↑	Scope	User-facing description
No rows to display		

Your sensitive scopes

Sensitive scopes are scopes that request access to private user data.

API ↑	Scope	User-facing description
No rows to display		

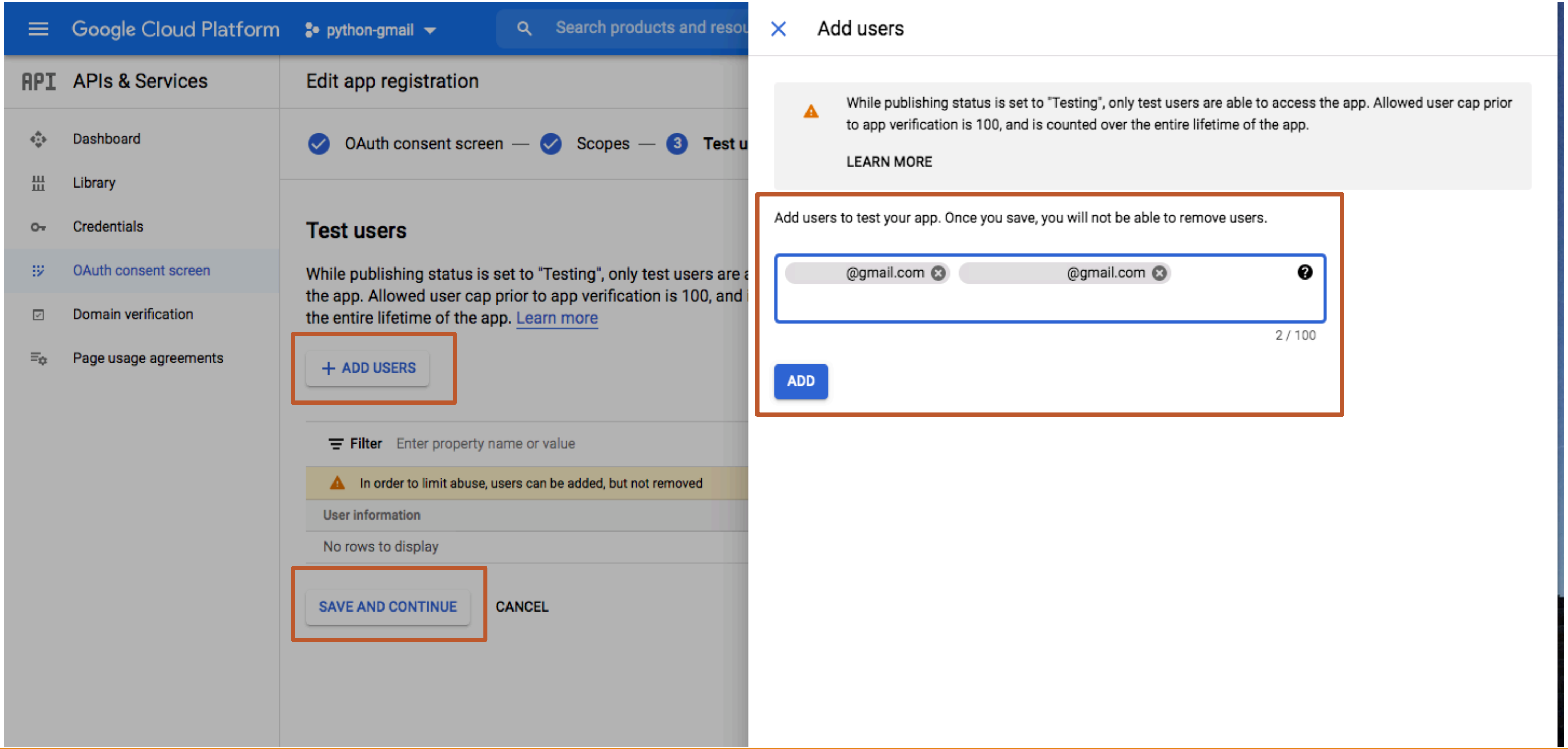
Your restricted scopes

Restricted scopes are scopes that request access to highly sensitive user data.

API ↑	Scope	User-facing description
No rows to display		

SAVE AND CONTINUE CANCEL

9. Add test users for the product then continue.
Test users – email address you will be using to send emails from.



Test users

While publishing status is set to "Testing", only test users are a
the app. Allowed user cap prior to app verification is 100, and i
the entire lifetime of the app. Learn more

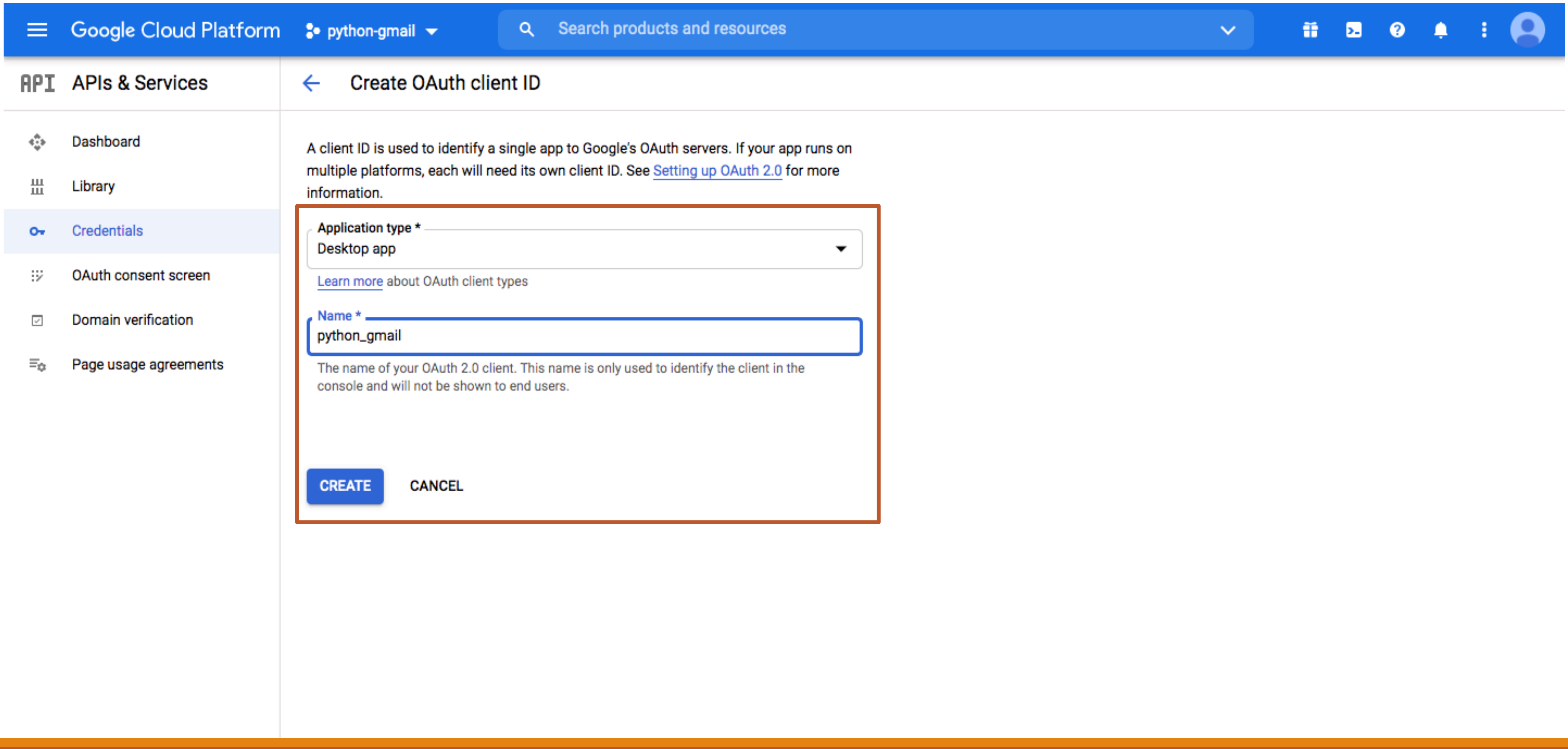
In order to limit abuse, users can be added, but not removed

No rows to display

While publishing status is set to "Testing", only test users are able to access the app. Allowed user cap prior to app verification is 100, and is counted over the entire lifetime of the app.

Add users to test your app. Once you save, you will not be able to remove users.

10. Once done, repeat step 4. Then, select application type and name the app.



Create OAuth client ID

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See Setting up OAuth 2.0 for more information.

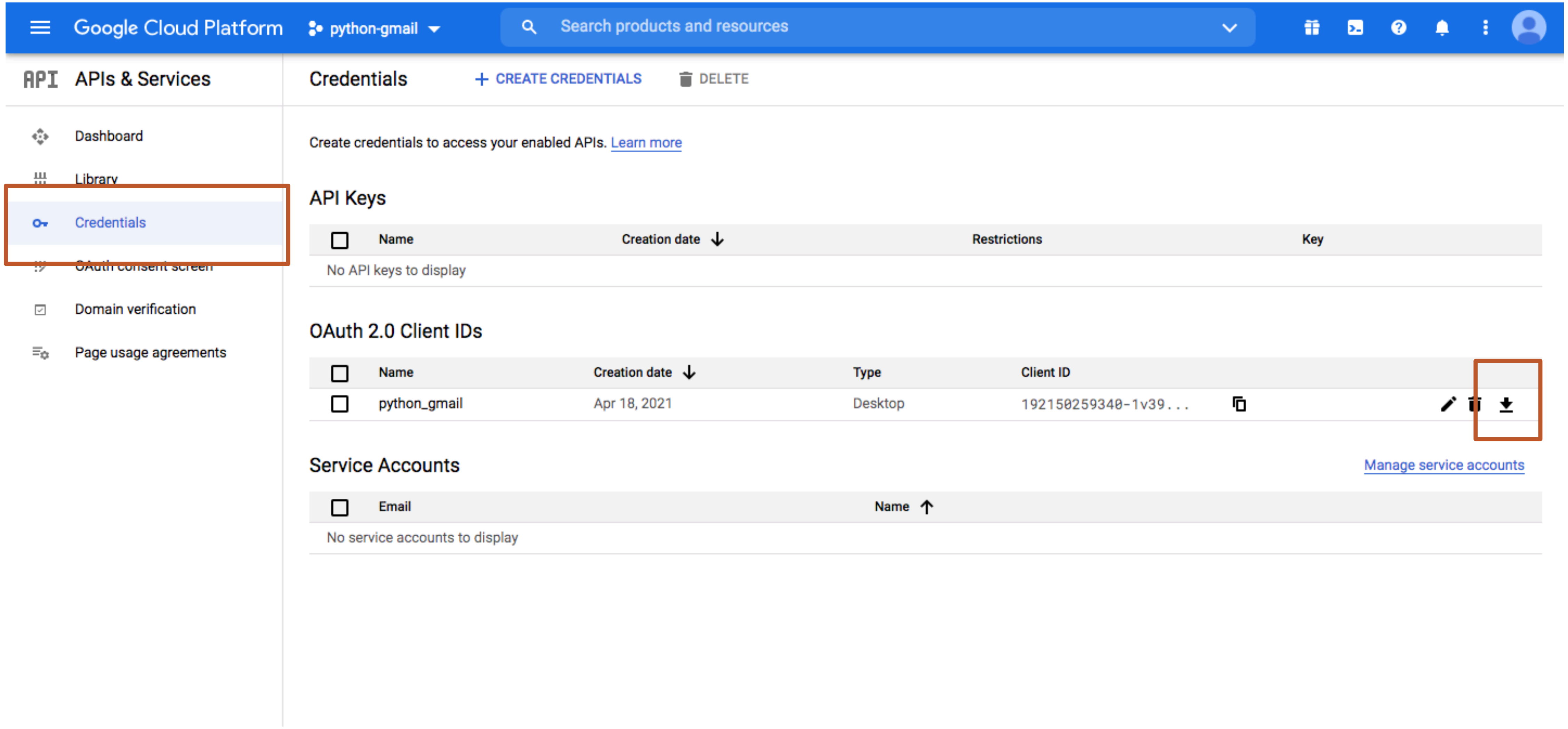
Application type *
Desktop app

Learn more about OAuth client types

Name *
python_gmail

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

11. Once complete, go back to Credentials and download the .json file. Rename the file as ‘credentials.json’



Create credentials to access your enabled APIs. Learn more

API Keys

Name	Creation date ↓	Restrictions	Key
No API keys to display			

OAuth 2.0 Client IDs

Name	Creation date ↓	Type	Client ID
python_gmail	Apr 18, 2021	Desktop	192150259340-1v39...

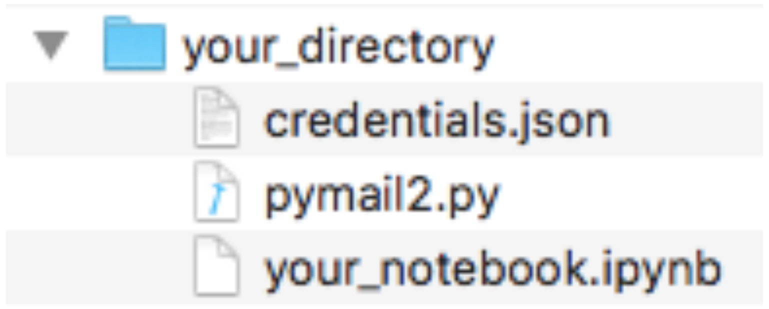
Service Accounts

Manage service accounts

Email	Name ↑
No service accounts to display	

Where to store credentials.json and pymail2.py?

1. Download pymail2.py from GitHub: https://github.com/csaw68/pymail2
2. Place pymail2.py in the same directory as Notebook.
3. Place credentials.json in the same directory as well.



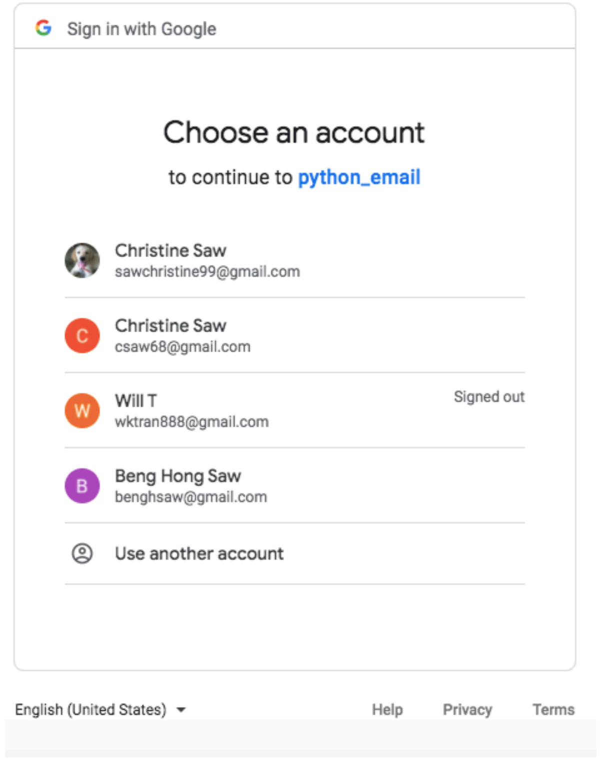
```
your_directory
    credentials.json
    pymail2.py
    your_notebook.ipynb
```

Accessing the API file

1. Import pymail2 then run the function `service_account_login`.

```python
import pymail2
service = pymail2.service_account_login()
```

2. The function will redirect you to Gmail. Make sure to login using the email that you added as a test user earlier.



Sign in with Google

Choose an account

to continue to python_email

Christine Saw
sawchristine99@gmail.com

Christine Saw
csaw68@gmail.com

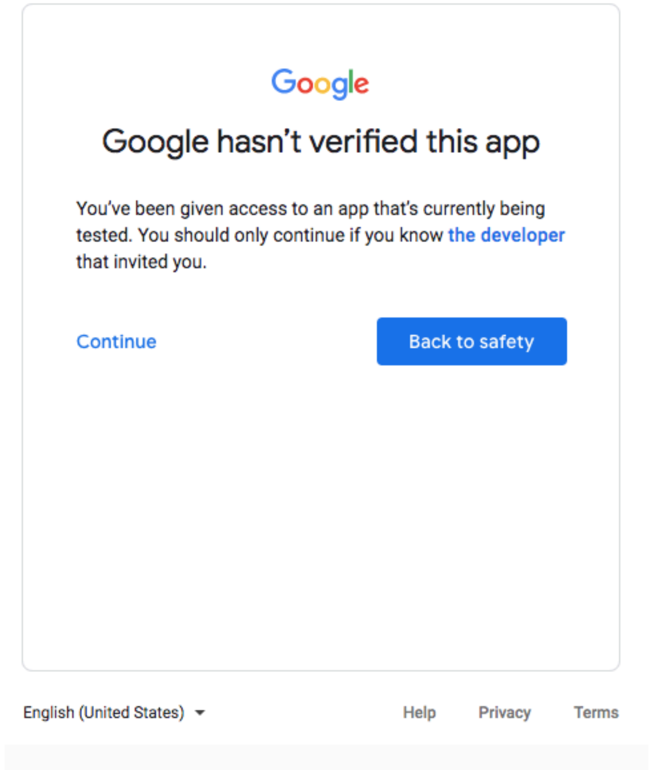
Will T
wktran888@gmail.com
Signed out

Beng Hong Saw
benghsaw@gmail.com

Use another account

English (United States) Help Privacy Terms

3. Give access to the application.



Google

Google hasn't verified this app

You've been given access to an app that's currently being tested. You should only continue if you know the developer that invited you.

Continue Back to safety

English (United States) Help Privacy Terms

4. The following message will display once completed.

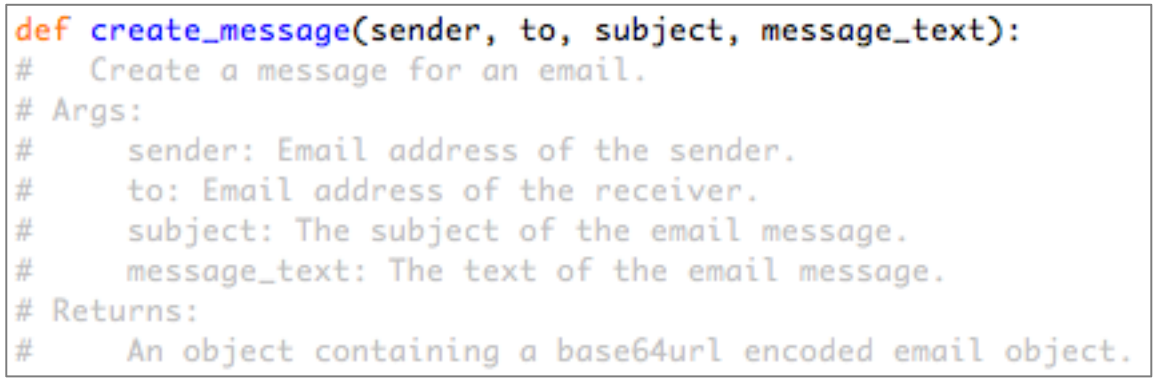
```
The authentication flow has completed. You may close this window.
```

Creating Email

1. The function `create_message` takes in 4 inputs.

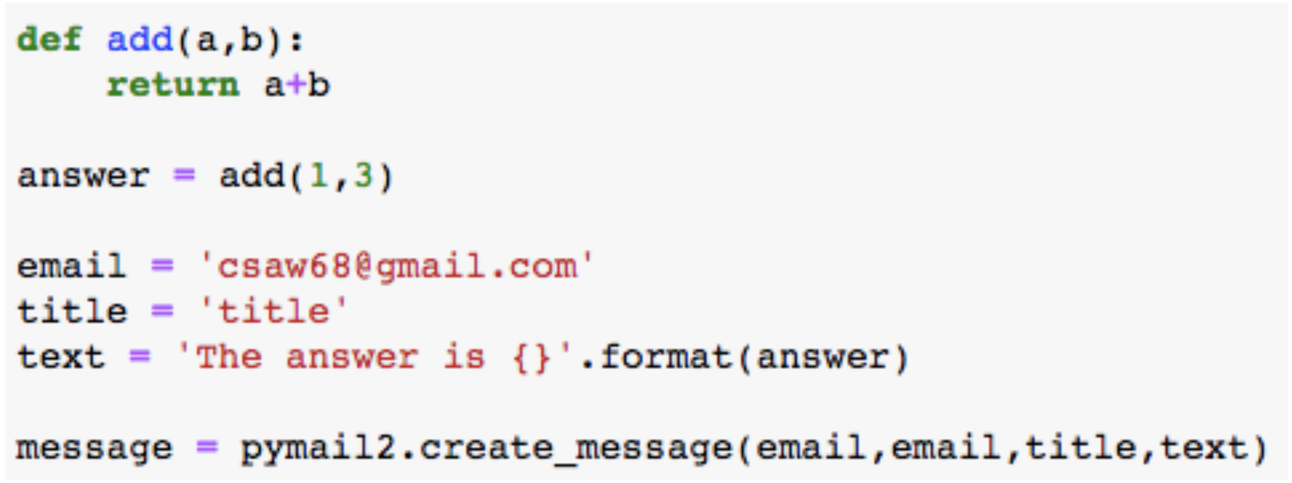
`sender`	:	sender's email address
`to`	:	receiver's email address
`subject`	:	subject title of email
`message_text`	:	message content of email (must be in string)

Use this function to create the email message.



```
def create_message(sender, to, subject, message_text):
#   Create a message for an email.
# Args:
#     sender: Email address of the sender.
#     to: Email address of the receiver.
#     subject: The subject of the email message.
#     message_text: The text of the email message.
# Returns:
#     An object containing a base64url encoded email object.
```

2. In this example, the program will sum two integers and create an email with the answer to be sent to the sender themselves.



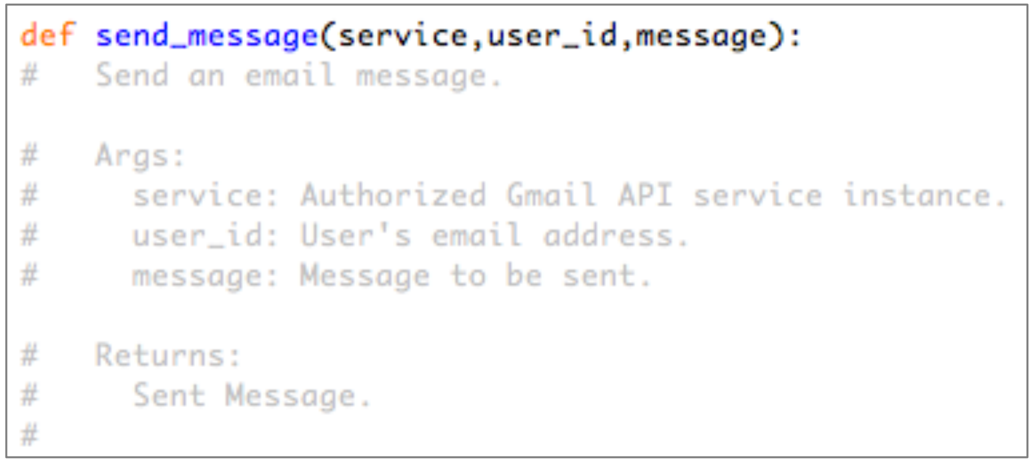
```
def add(a,b):
    return a+b

answer = add(1,3)

email = 'csaw68@gmail.com'
title = 'title'
text = 'The answer is {}'.format(answer)

message = pymail2.create_message(email,email,title,text)
```

Sending Email

1. The function `send_message` takes in 3 inputs.

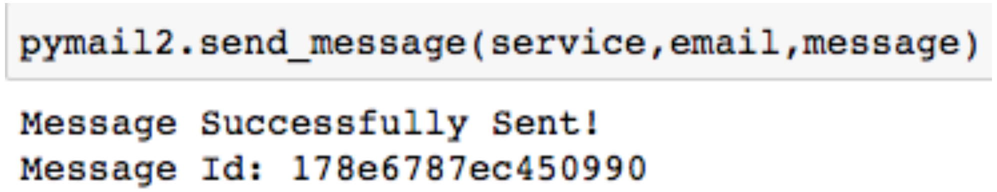
`service` : returned instance from `service_account_login`
`user_id` : email address used to create Google Console account
`message` : returned encoded object from `create_message`

Use this function to send the email created.



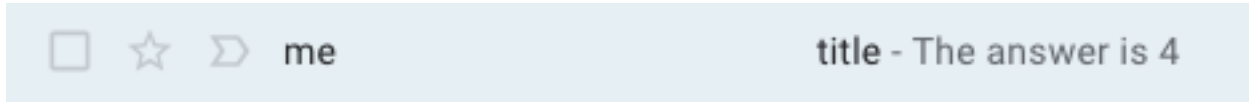
```
def send_message(service,user_id,message):
#   Send an email message.

#   Args:
#     service: Authorized Gmail API service instance.
#     user_id: User's email address.
#     message: Message to be sent.

#   Returns:
#     Sent Message.
#
```

2. In this example, the program sends the created message earlier.



```
pymail2.send_message(service,email,message)
```

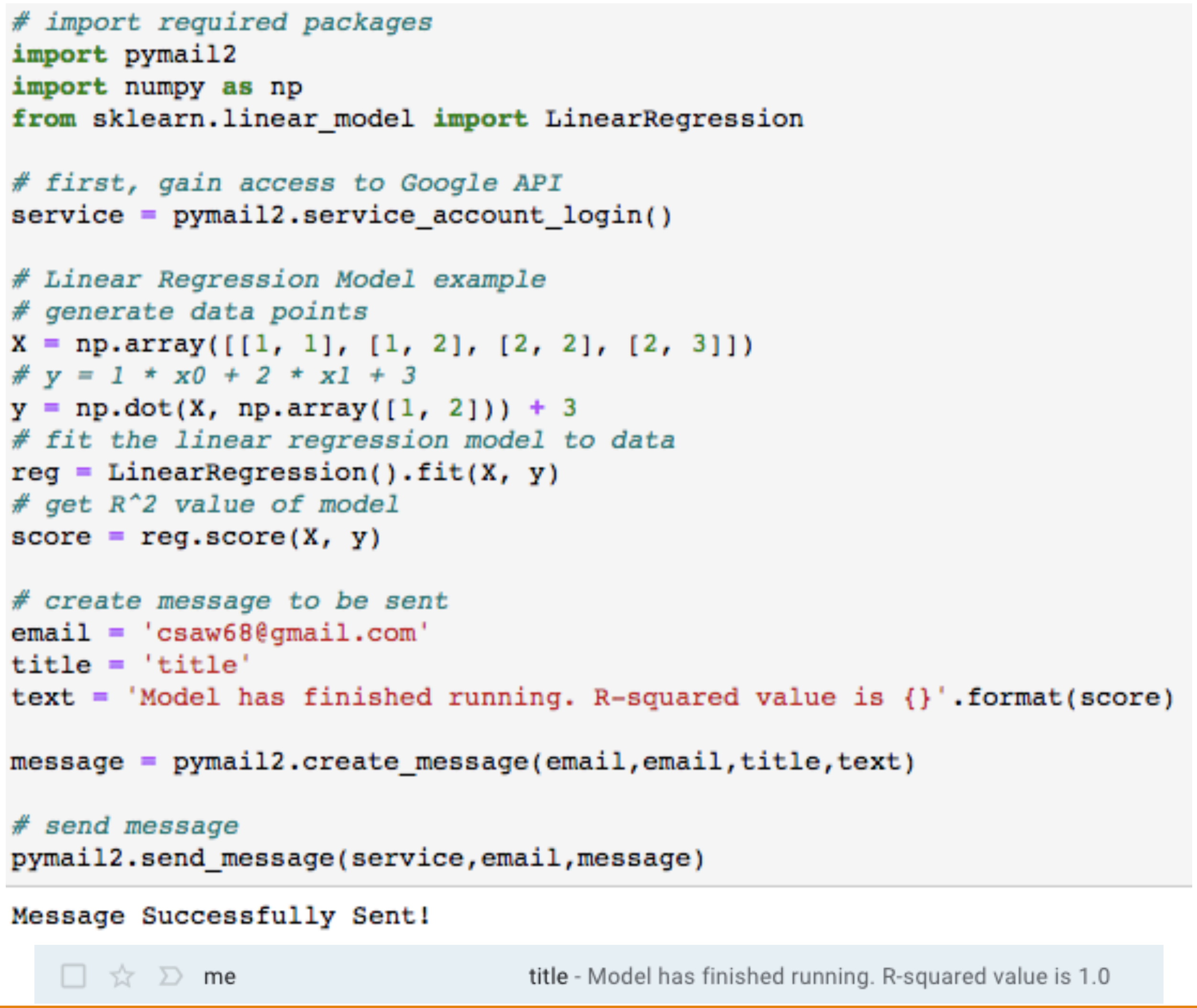
```
Message Successfully Sent!
Message Id: 178e6787ec450990
```

Received email: 

me title - The answer is 4

Example (putting it all together!)

In the following example, a Linear Regression model is first ran then the R^2 value is sent to the user once it completes running.



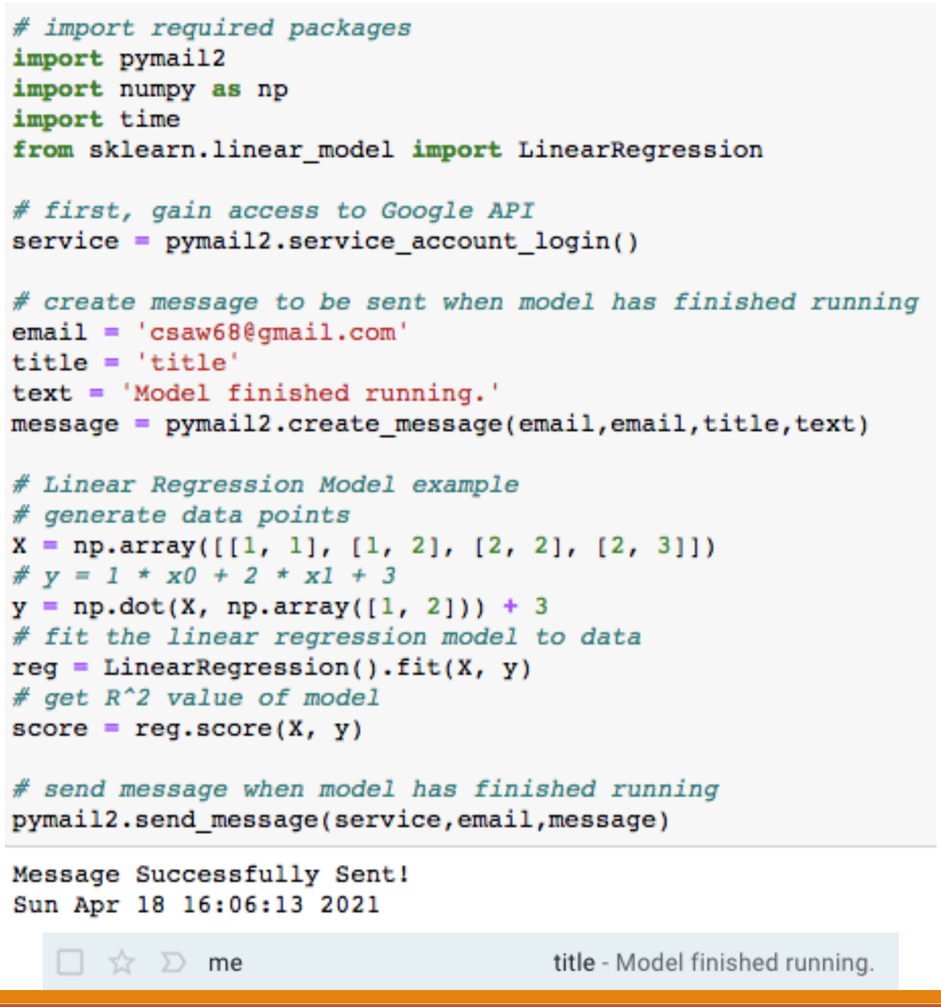
```
# import required packages
import pymail2
import numpy as np
from sklearn.linear_model import LinearRegression

# first, gain access to Google API
service = pymail2.service_account_login()

# Linear Regression Model example
# generate data points
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
# y = 1 * x0 + 2 * x1 + 3
y = np.dot(X, np.array([1, 2])) + 3
# fit the linear regression model to data
reg = LinearRegression().fit(X, y)
# get R^2 value of model
score = reg.score(X, y)

# create message to be sent
email = 'csaw68@gmail.com'
title = 'title'
text = 'Model has finished running. R-squared value is {}'.format(score)

message = pymail2.create_message(email,email,title,text)

# send message
pymail2.send_message(service,email,message)
```

```
Message Successfully Sent!
```

me   title - Model has finished running. R-squared value is 1.0

Example (notification when program completes running)

In the following example, a Linear Regression model is ran, then an email is sent to the user to alert them once the program finishes executing. If a program takes too long to run and we don't want to wait in front of our screen, this is a perfect setting to use pymail2.



```python
# import required packages
import pymail2
import numpy as np
import time
from sklearn.linear_model import LinearRegression

# first, gain access to Google API
service = pymail2.service_account_login()

# create message to be sent when model has finished running
email = 'csaw68@gmail.com'
title = 'title'
text = 'Model finished running.'
message = pymail2.create_message(email,email,title,text)

# Linear Regression Model example
# generate data points
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
# y = 1 * x0 + 2 * x1 + 3
y = np.dot(X, np.array([1, 2])) + 3
# fit the linear regression model to data
reg = LinearRegression().fit(X, y)
# get R^2 value of model
score = reg.score(X, y)

# send message when model has finished running
pymail2.send_message(service,email,message)
```

```
Message Successfully Sent!
Sun Apr 18 16:06:13 2021
```

me   title - Model finished running.

Thank you! ☺