**REPO Y CURSO INTRODUCTORIO UTILIZADO PARA APRENDER Y TOMAR IDEAS:**

REPO <https://github.com/marcopeix/MachineLearningModelDeploymentwithStreamlit>

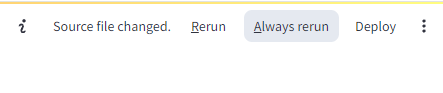
CURSO UDEMY:<https://www.udemy.com/course/machine-learning-model-deployment-with-streamlit/>

**COMO CORRER APP WEB DE STREAMLIT LOCALMENTE**

* pararse en la carpeta donde está el script con el código de streamlit
* abrir consola con el env que tiene instalado streamlit
* correr la siguiente linea de comando: **streamlit run your\_script.py**
* listo, automáticamente se debería abrir una pestaña en el navegador preteterminado en el local host: <http://localhost:8501/>

**RERUN DEL FRONT AL MOMENTO DE HACER UN CAMBIO EN EL CÓDIGO QUE LO GENERA Y GUARDAR ESE CAMBIO**

Cuando cambia el código se puede configurar stremalit para que siempre haga rerun de la app y guardar cambios en el script.py y así mostrar siempre la pantalla del código guardado



**CÓMO HACER DEPLOY DE LAS APPS – DOCU OFICIAL STREAMLIT**

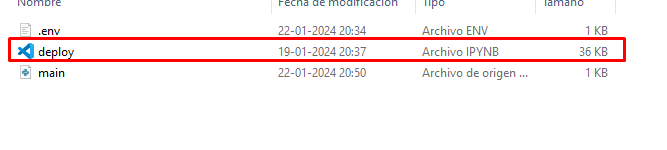
Link con lecturas de interés de cómo hacerlo en diferentes servicios de diferentes nubes

<https://docs.streamlit.io/knowledge-base/tutorials/deploy>

**CÓMO HACER DEPLOY EN CLOUD RUN – (seguir misma lógica que deploy de flask app en cloud run)**

Tanto para flask como para streamlit tengo desarrollado un código para hacer el deploy en un cloud run. Correr este notebook ghace todo el proceso, solo hay que definir los parámetros y luego correr los códigos e intermanente creará el dockerfile y el cloudbuild necesarios para hacer el deploy en un cloud run

Correr notebook deploy



**Fuentes:**

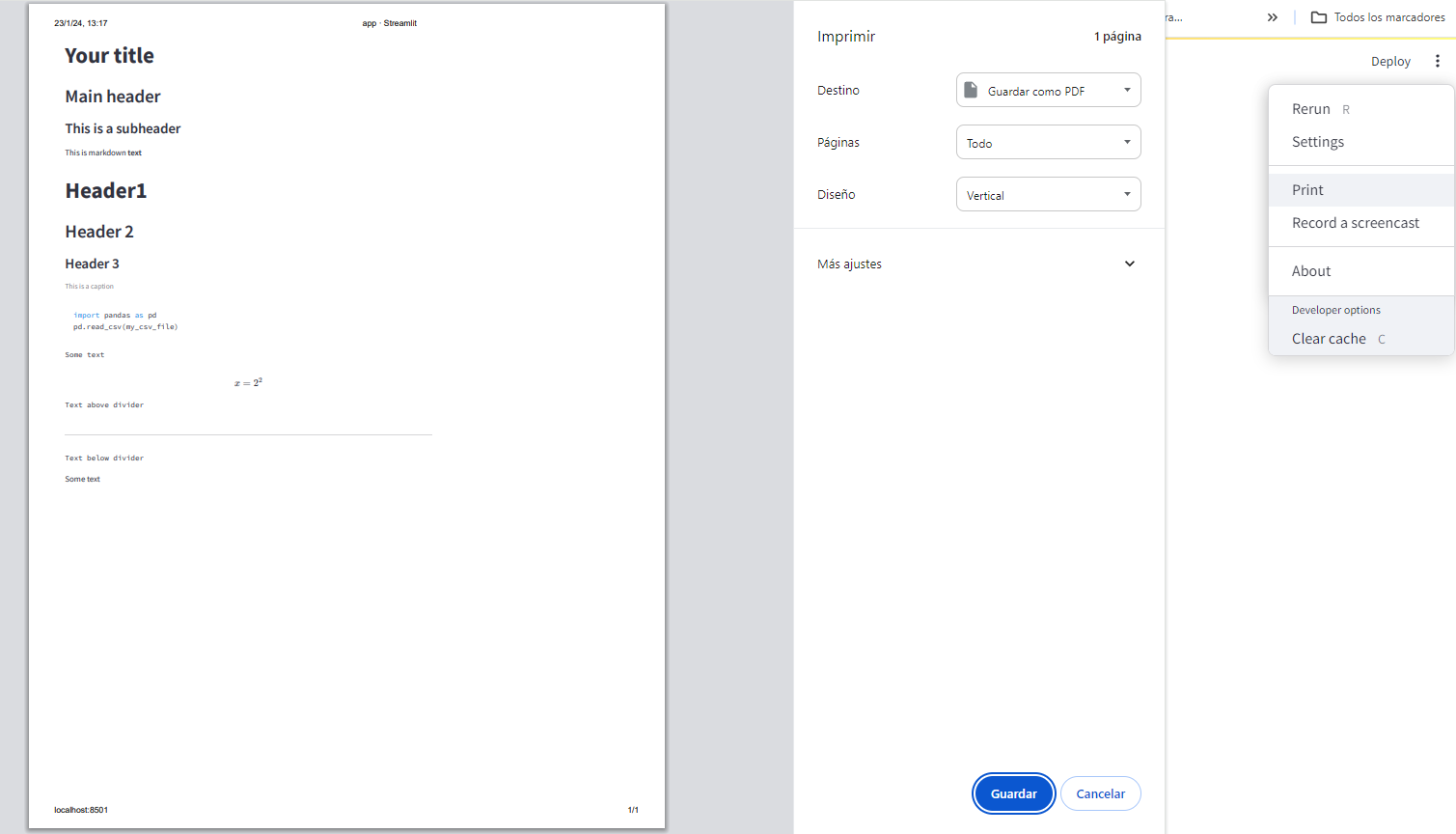
- deploy streamlit app in cloud run: https://medium.com/@faizififita1/how-to-deploy-your-streamlit-web-app-to-google-cloud-run-ba776487c5fe

- deploy streamlit app into google app engine: https://dev.to/whitphx/how-to-deploy-streamlit-apps-to-google-app-engine-407o

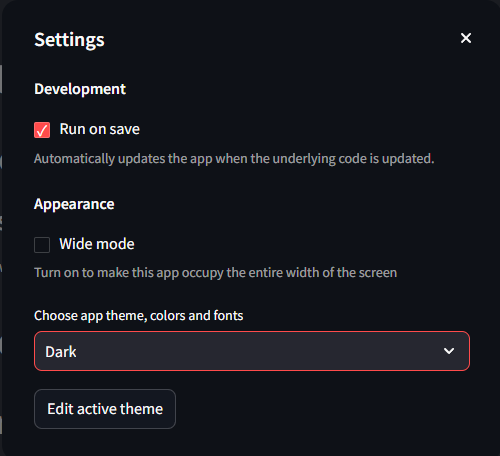
- deploy a flask app into a cloud run: my previous codes

**PERMITE IMPRIMIR LA PÁGINA WEB COMO UN PDF DE FORMA SIMPLE**

Entonces puedo diseñar una web de forma muy simple y además guardarla como un pdf (pareciera que sin perder las dimensiones) de forma extremadamente fácil



**OPCIONES DE CONFIGURACIÓN DE LA APP DESDE EL UI**

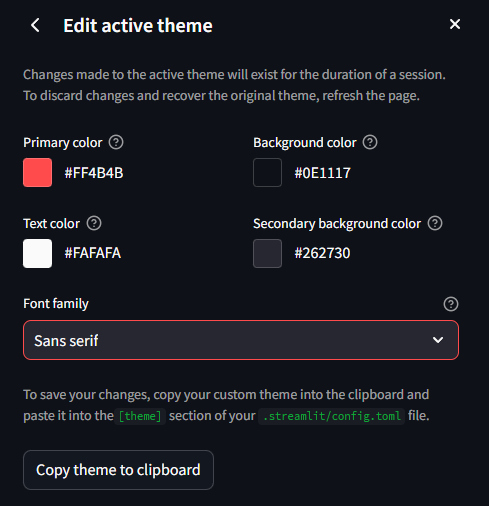


- development: run on save. Permite actualizar automáticamente la web al hacer un cambio en el código python que lo genera. En lugar de tener que cerrar el localhost y volver a abrirlo

- Opciones de apariencia:

- Permite ajustar la pantalla a que el texto ocupe toda la pantalla o esté centrado

- Permite migrar entre modo light y modo dark solo con un click!!. Además permite editar la tonalidad de los colores light y dark también solo cambiando los códigos de los colores desde la misma app



**RESUMEN GENERAL STREAMLIT:**

- cambiar temas light y dark con un solo click. Cambiar tonalidad de los colores del tema con un par de clicks

- tiene un nivel de customización suficiente para hacer una web para mostrar resultados y probar modelos de ml. Obviamente no tiene el nivel de customización que permite un desarrollo desde cero pero el tradeoff que ofrece entre tiempo de desarrollo (y curva de aprendizaje) y grado de personalización de la pantalla es muy aceptable para realizar pruebas rápidas y mostrar visualmente resultados

- con el nivel de personalización que ofrece es más que suficiente para mostrar la inferencia de modelos de ml y a la vez para que el usuario haga pruebas de la inferencia de los modelos

**CARACTERÍSTICAS DE STREAMLIT NECESARIAS DE CONOCER PARA DESARROLLAR CON LA HERRAMIENTA**

- Se corre el código de top a bottom

- Cada vez que el usuario interactúa con la interfaz de usuario, por ejemplo, ingresando un valor en una celda se ejecuta todo el código nuevamente para actualizar lo mostrado en la pantalla

- Si se quiere que el usuario ingrese varios valores y que no se ejecute hasta haber ingresado todos los valores es necesario utilizar un formulario, para que al enviar al formulario se actualize la pantalla

- Hacer KPIs son muy fáciles de hacer con una función nativa de streamlit en comparación de flask+html que requieren utilizar css para obtener componentes de kpis de calidad comparable a la de streamlit

**REFRESH PAGE: VOLVER A CORRER TODO EL CODIGO DEL SCRIPT**

Rerun streamlit:

<https://docs.streamlit.io/library/api-reference/control-flow/st.rerun>

es un comando que se debe correr porque lo que permite volver a correr el script pero necesita un trigger manual

Por otro lado, existe esa librería para autorefresh comentada en todos los foros de streamlit como la solución:

Foro con ejemplos: <https://discuss.streamlit.io/t/streamlit-autorefresh/14519>

Package: <https://github.com/kmcgrady/streamlit-autorefresh>

**MULTIPAGE Y SESION STATE**

- Pages are defined as .py files in a pages/ directory. **Cuando se crean los scripts en el folder page, automáticamente se crea un sidebar con los nombres de las páginas**

- Filenames are composed of four different parts:

- A number — if the file is prefixed with a number.

- A separator — could be \_, -, space, or any combination thereof.

- A label — which is everything up to, but not including, .py.

- The extension — which is always .py.

- st.set\_page\_config works at the page level. When you set a title or favicon using st.set\_page\_config, this applies to the current page only.

**- Pages share the same st.session\_state. So, with this way yo can share variable between diferents pages**

**- Session state: Para poder conservar variables entre páginas es necesario utilizar sesión state obliatoriamente.**

**-** Importante siempre hay que inicializar los sesion state. SI NO SE INICIALIZA RETORNA UN ERROR

- Funcionan con la misma sintaxis que un diccionario de python

- EL PRINCIPAL PROBLEMAS DE LAS MULTI PÁGINAS ES QUE:

- PUEDO GUARDAR VARIABLE DE ESTADO PARA LUEGO SER MOSTRADAS EN OTRA PÁGINA. TODO OK. PERO LUEGO, SI EN ESTA OTRA PÁGINA EL USUARIO REALIZA ACCIONES, LOS VALORES DE LAS VARIABLES DE ESTADO SE PIERDEN

Ej. Página main guardo 2 sesion state y paso a la página auxiliar, en esta página se ven los 2 sesion state guardado en la página main. Pero luego, si quiero interactuar con la pantalla auxiliar y guardar variables solo para dicha página, los valores de los sesión state de la página main se pierden

- SOLUCIÓN: <https://stackoverflow.com/questions/74968179/session-state-is-reset-in-streamlit-multipage-app>

- EJEMPLO DE LA SOLUCIÓN: <https://github.com/marcopeix/MachineLearningModelDeploymentwithStreamlit/blob/master/30_multipage_capstone/home.py>

Por mi parte prefiero la solución sencilla. Esta es. En la página main utilizar TODAS LAS SESSION STATE, incluso si se utilizan en otra página.

Además, lo recomendable es que si se utiliza multiple páginas, idealmente desde la main se definen todas las session states y desde el resto de páginas estas simplemente se consuma

DE TODAS FORMAS LO MEJOR ES NO COMPLICARSE LA VIDA UTILIZANDO MULTIPLE PÁGINAS SI NO ES NECESARIO

SUGERENCIA ORDENAR CÓDIGOS

La idea es tener todo en funciones donde si yo quiero mostrar un gráfico, kpi, etc pueda llamar una función que lo genera y tener el objeto a mostrar en una variable de python. Luego cuando se quiere ordenar la interfaz agregando tab, columns, headers, etc uno se puede concentrar en modificar la UI y solo llama la variable que tiene el objeto a motrar.

Por ejemplo, para probar que todo funciona se llaman los elementos a mostrar

    ########### SHOW VALUES, PLOTS IN UI STREAMLIT ###########

    # "metrics" - "show values of the prediction"

    st.metric(label = "Datetime last update webpage", value = FECHA\_ULTIMA\_ACTUALIZACION)

    st.metric(label = "Datetime Prediction Model", value = FECHA\_PREDICCION)

    st.metric(label = "Value Predicted", value = VALOR\_PREDICCION)

    st.metric(label = "Alert Level Prediction", value = ALERTA\_PREDICCION)

    # figure plotly traffic light alert

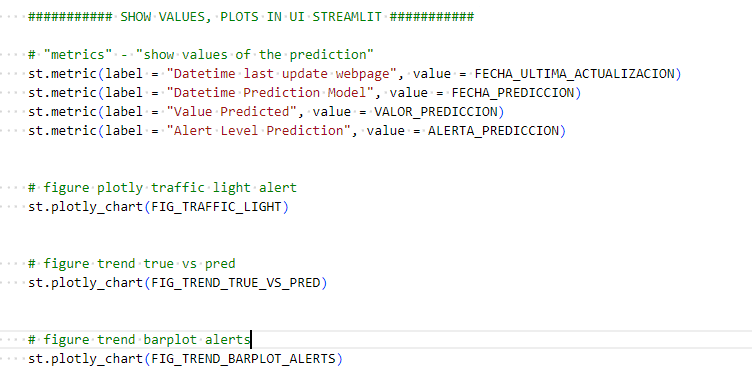
    st.plotly\_chart(FIG\_TRAFFIC\_LIGHT)

    # figure trend true vs pred

    st.plotly\_chart(FIG\_TREND\_TRUE\_VS\_PRED)

    # figure trend barplot alerts

    st.plotly\_chart(FIG\_TREND\_BARPLOT\_ALERTS)



Luego, se ordenan, por ejemplo en columnas

**ORDEN CARPETAS:**

1\_hello\_world: contiene código mostrar un mensaje hello world y hacer en deploy de dicha app en un cloud run

2\_template\_elements: contiene código donde se muestra diferentes elementos útiles para una app de data science (tablas de kpis, mostrar gráficos con diferentes packages, etc)

3\_hello\_world\_gcp: contiene códigos para conectarse con servicios de GCP utilizados para el desarrollo de app de data science, principalmente bigquery, storage y vertex

4\_app\_dashboard\_ml\_models: contiene códigos de una app que muestra los resultados de un modelo de machine learning. La app se conecta con servicios cloud y se actualiza con x frecuencia mostrar nuevas inferencia del modelo

5\_app\_test\_ml\_models: contiene códigos de una app que permite al usuario ingresar los valores de entrada de un modelo de machine learning y muestra los resultados del modelo. Así permite testear con diferentes entradas la predicción que hace el modelo.

6\_multipage\_app: contiene código de cómo hacer una app de multiples páginas. Este es uno de los puntos débiles de streamlit ya que originalmente fue pensada como tener apps de solo una página. Han ido trabajando en agregar esta funcionalidad y mejorarla

**IMPORTANTE:**

- Para hacer el deploy en un cloud run basta con correr el notebook deploy y cambiar los parámetros y luego correr el código. Al correr el notebook se crea automáticamente un archivo dockerfile y cloudbuild.yml.

- Por lo tanto al desarrollar solo se necesita tener el script de python app.py que funciona como root de la app y el archivo requirements.txt con las versiones de los packages que necesita la app

**FALTA VER:**

- CONEXIÓN A CLOUD

- REFRESH AUTOMÁTICO DE LA PANTALLA

- REFRESH AUTOMÁTICO DE LA PANTALLA CON UN OBSERVER DESDE BIGQUERY

- CUANTO DURA LA APP SIN CAERSE DESDE:

- CLOUD RUN (ME IMAGINO QUE POCO :/ )

- CORRIENDO EN EL LOCALHOST

- CORRIENDO EN EL LOCAL EN ALGUNA MAQUINA VIRTUAL O ALGO . APACHE/HAMACHI/O ALGO QUE SE UTILIZÓ EN EL RAMO DE TI

- VER LOS COSTOS DE TENER LA APP EN UN COMPUTE ENGINE U OTRA FORMA DE DESPLIEGUE QUE HAGA QUE NO SE CAIGA LA APP

- SESSION STATE

En Streamlit, la **session state** se refiere a un objeto que puedes utilizar para almacenar y acceder a datos de manera persistente durante la duración de una sesión de la aplicación. Esto es útil cuando necesitas conservar el estado de algunas variables entre distintas ejecuciones de la aplicación o entre diferentes partes de la interfaz.

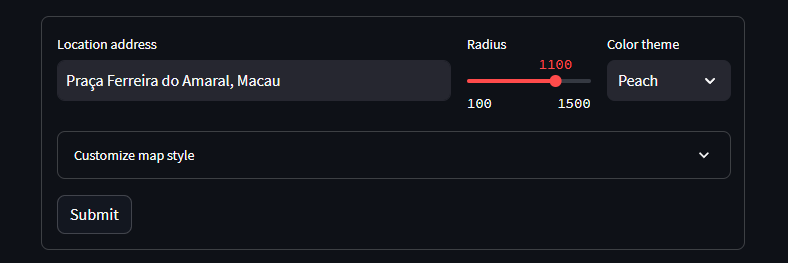
**LINKS**

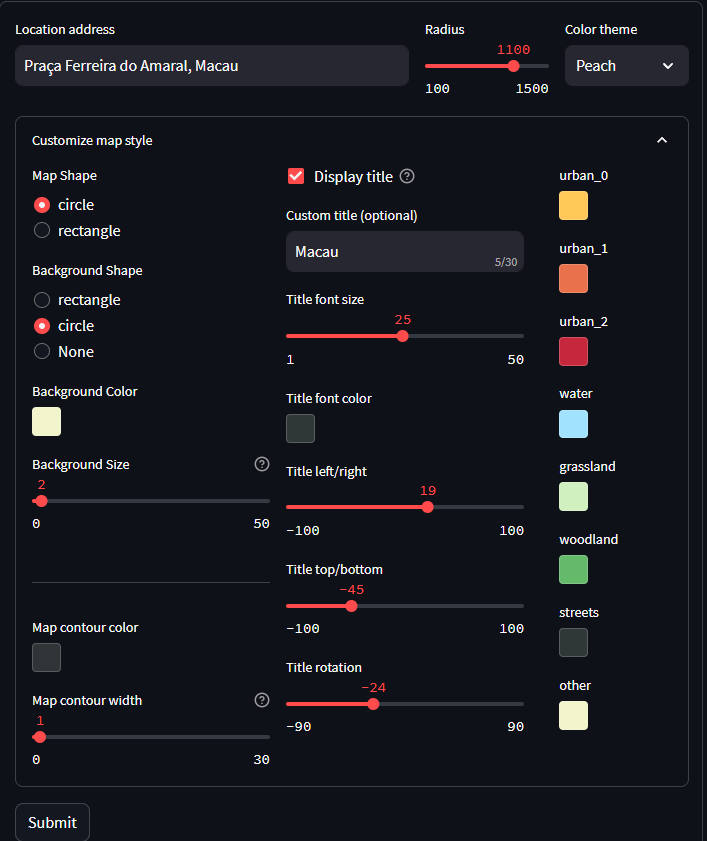
- cheat sheet streamlit: <https://cheat-sheet.streamlit.app/>

- streamlit roadmap: <https://roadmap.streamlit.app/>

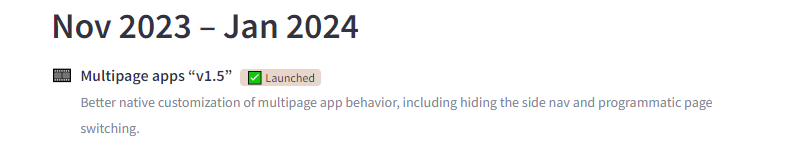
- stremlit multi page: <https://docs.streamlit.io/library/advanced-features/multipage-apps>

- streamlit gallery app to get ideas: <https://streamlit.io/gallery>

- streamlit beatiful example: 



Example multipage feature improve



- Otro ejemplo de app. GAN

<https://github.com/streamlit/demo-face-gan?tab=readme-ov-file>

- otro ejemplo de app. Visualizar información

<https://tdenzl-bulian-bulian-ifeiih.streamlit.app/#match-finder>