



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DIRECCIÓN
GENERAL DE
EDUCACIÓN
CONTINUA

AMBIENTES VIRTUALES Y CONTROL DE VERSIONES





AMBIENTES VIRTUALES EN PYTHON

¿Qué es un ambiente virtual?

Un **ambiente virtual** es un entorno aislado que te permite instalar versiones específicas de paquetes de Python sin afectar otros proyectos. Es útil cuando trabajas en múltiples proyectos que dependen de diferentes versiones de librerías.

¿Cómo crear y activar un ambiente virtual en Windows?

Los pasos para crear y activar un ambiente virtual en Windows son los siguientes:

1. Abrir el Símbolo del Sistema o PowerShell:

- Para ello presiona **Windows + R**, escribe **cmd** o **powershell** y presiona **Enter**.

2. Navegar a la carpeta de tu proyecto:

- Si tus archivos de proyecto están en **C:\MisProyectos\mi_proyecto**, ejecuta:

```
cd C:\MisProyectos\mi_proyecto
```

3. Crear un ambiente virtual:

- Para ello debes ejecutar el siguiente comando para crear un ambiente virtual:

```
python -m venv mi_entorno
```

4. Activar el ambiente virtual:

- En Windows

```
mi_entorno\Scripts\activate
```



- En macOS/Linux

```
source mi_entorno/bin/activate
```

Después de activar el ambiente, puedes instalar los paquetes necesarios usando **pip**:

```
pip install numpy pandas matplotlib
```

Cuando termines de trabajar en tu proyecto, puedes desactivar el ambiente virtual con:

```
deactivate
```

Cuando el ambiente está activado, el nombre del ambiente aparecerá antes de cada línea de comando:

```
(mi_entorno) C:\MisProyectos\mi_proyecto>
```

Pregunta para Reflexionar: ¿Qué ventajas ofrece usar ambientes virtuales en un proyecto colaborativo donde cada miembro podría tener diferentes dependencias?





CONTROL DE VERSIONES CON GIT Y GITHUB

Qué es Git y por qué es útil

Git es una herramienta de control de versiones que permite a los desarrolladores trabajar de manera colaborativa, seguir cambios en el código y gestionar versiones de manera efectiva.

Cómo Instalar Git en tu PC (Windows)

Los pasos para crear y activas un ambiente virtual en Windows son los siguientes:

1. Descargar Git:

- Ve a la página oficial de Git: <https://git-scm.com/downloads>
- Descarga la versión para Windows.

2. Instalar Git:

- Abre el archivo descargado y sigue las instrucciones del asistente de instalación.
- Asegúrate de seleccionar la opción "**Git Bash Here**" durante la instalación, lo que te permitirá usar Git desde cualquier carpeta.

3. Configurar Git:

- Una vez instalado, abre **Git Bash** o **PowerShell** y configura tu nombre de usuario y correo electrónico (esto es necesario para registrar los cambios que hagas en los repositorios):

```
git config --global user.name "Tu Nombre"  
git config --global user.email "tuemail@ejemplo.com"
```

Cómo crear una cuenta en GitHub

1. Ve a <https://github.com> y crea una cuenta.
2. Una vez creada la cuenta, podrás crear repositorios donde almacenar tu código.



Pregunta para Reflexionar: ¿Cuál es la diferencia entre Git y GitHub?





CICLO DE GIT: ETAPAS Y COMANDOS

El ciclo básico de trabajo con **Git** pasa por tres etapas:

1. Directorio de trabajo (**Working Directory**)
2. Área de preparación (**Staging Area**)
3. Repositorio (**Repository**)

A lo largo de estas etapas, los archivos pasan por diferentes estados, en que podemos ir usando comandos que nos permiten gestionarlas.

1. Directorio de Trabajo (**Working Directory**)

- **Descripción:** Es el lugar donde están los archivos en tu máquina local y donde realizas modificaciones. Aquí editas, creas, eliminas o modifcas archivos de tu proyecto. En esta etapa, Git aún no tiene control sobre los cambios.
- **Estado de los archivos:** Los archivos en esta etapa se consideran **modificados**, pero **no rastreados o no añadidos** al control de versiones.
- **Comandos:**
 - **Verificar el estado actual del proyecto:** Puedes usar el comando `git status` para ver qué archivos han sido modificados y cuáles están listos para ser añadidos al control de versiones.

`git status`

- **Modificaciones de archivos:** Cualquier modificación que hagas en los archivos está en el **directorio de trabajo**, pero aún no forma parte del repositorio.

2. Área de Preparación (**Staging Area**)

- **Descripción:** Después de modificar un archivo, debes decirle a Git qué cambios deben ser considerados para el próximo **commit**. Esto lo haces con el comando **git add**, que mueve los archivos modificados al **staging**. En esta etapa, los archivos están en un estado temporal, listos para ser confirmados en el repositorio.





- **Estado de los archivos:** Los archivos añadidos al **staging** están en un estado **preparado**. Esto significa que están listos para ser guardados en el próximo **commit**.
- **Comandos:**
 - **Añadir un archivo al área de staging:** El comando **git add** se utiliza para mover los archivos modificados del directorio de trabajo al área de preparación (staging). Esto no guarda los cambios definitivamente, sino que los marca como listos para el próximo commit.

```
git add nombre_archivo
```

3. **Repositorio (Repository)**

- **Descripción:** Una vez que los archivos están en el **staging**, puedes hacer un **commit** para guardarlos de manera definitiva en el repositorio de Git. El repositorio es donde Git guarda el historial de todos los cambios confirmados (commits) y rastrea las versiones de los archivos.
- **Estado de los archivos:** Los archivos que han sido confirmados están ahora en el **repositorio** y forman parte del historial del proyecto. Puedes volver a cualquier versión anterior desde aquí.
- **Comandos:**
 - **Guardar los cambios con git commit:** Despues de añadir los archivos al área de staging, puedes usar el comando git commit para confirmar esos cambios en el repositorio.

```
git commit -m "Descripción de los cambios"
```

Esto guarda una "foto" del estado actual de los archivos en el repositorio, junto con un mensaje descriptivo.

- **Subir los cambios a un repositorio remoto con git push:** Si estás utilizando un repositorio remoto como GitHub, puedes subir los cambios al servidor remoto utilizando el comando **git push**.

```
git push origin nombre_rama
```



4. Resumen Visual del Ciclo de Git:

- Modificas archivos → git add → Área de Staging
- git commit → Repositorio Local (commit confirmado)
- git push → Repositorio Remoto (GitHub/GitLab)

Este ciclo básico de **add -> commit -> push -> pull** es el núcleo de Git y permite gestionar los cambios de manera controlada y colaborativa.

Veamos otro Ejemplo del Ciclo

1. Modificas archivos en tu directorio de trabajo (Working Directory).
 - **Estado de los archivos:** Modificados.
 - **Comando:** Trabajas directamente en tu editor o IDE.
 - **Verificar estado:**

`git status`

2. Añades los archivos modificados al área de staging (Staging Area).

- **Comando:**

`git add nombre_archivo`

o

`git add .`

3. Confirmas los archivos en el repositorio con un commit (Repository).

- **Comando:**

`git commit -m "Descripción de los cambios"`

4. Subes los cambios al repositorio remoto (si usas GitHub).

- **Comando:**

`git push origin nombre_rama`



5. Recibes los cambios remotos y los sincronizas con tu repositorio local.

- Si otros colaboradores han hecho cambios, puedes recibir esos cambios en tu máquina local usando:

```
git pull origin nombre_rama
```

¿Dónde está nuestro Git?

Cuando inicias y usas **Git** para realizar control de versiones, lo haces dentro de una **carpeta específica** que actúa como tu proyecto o repositorio local. Git no rastrea automáticamente todos los archivos de tu equipo, solo aquellos que están dentro de la carpeta donde has inicializado Git.

Esto implica que para hacer todo el ciclo de **Git** (add, commit, push, etc.), debes haber inicializado un repositorio de Git dentro de una **carpeta** específica en tu sistema de archivos.

Supongamos que estamos trabajando en un archivo Excel en el disco C, los pasos para trabajar con Git son los siguientes:

1. Crea una carpeta para tu proyecto en Disco C:

- Si tu archivo Excel está en el Disco C, necesitas crear una carpeta que servirá como el repositorio Git. Por ejemplo, supongamos que tienes un archivo llamado **planilla.xlsx** en el directorio C:\Proyectos\.

2. Inicia Git en esa carpeta:

- Abre la consola de **Windows** y navega hasta esa carpeta usando el comando cd:

```
cd C:\Proyectos
```

Luego, inicializa un repositorio de **Git** en esa carpeta:

```
git init
```

Este comando crea una carpeta oculta llamada .git en la que Git guardará todo el historial y la información sobre los archivos del proyecto.



3. Agrega tu archivo al repositorio:

- Para incluir nuestro archivo Excel en Git usamos el comando:

```
git add planilla.xlsx
```

4. Haz un commit de los cambios:

- Una vez que has añadido el archivo al área de staging, puedes guardar los cambios con un commit:

```
git commit -m "Añadir la planilla Excel al repositorio"
```

5. Opcional: Subir a un repositorio remoto (GitHub):

- Si deseas, puedes subir el repositorio local a GitHub o GitLab para mantenerlo en la nube. Primero, necesitas crear un repositorio en GitHub y luego conectarlo a tu repositorio local:

```
git remote add origin https://github.com/usuario/repositorio.git  
git push -u origin master
```

6. Explicación más Detallada:

- Carpeta Local (Repositorio Local):** Git solo rastrea los archivos que están dentro de la carpeta en la que lo has inicializado. En este caso, la carpeta sería C:\Proyectos y el archivo que estás rastreando es planilla.xlsx. Esto significa que todos los cambios que realices en este archivo (como actualizaciones, eliminaciones, o restauraciones) estarán bajo el control de Git.
- Repositorio de Git:** La carpeta en la que iniciaste Git (en este caso, C:\Proyectos) es ahora un "repositorio de Git". Dentro de esa carpeta puedes realizar todo el ciclo de Git (add, commit, push, etc.).

¿Qué pasa si tengo archivos en otras carpetas?

Git solo rastrea los archivos que están en la carpeta donde lo has inicializado. Si tienes archivos en diferentes carpetas, no estarán bajo el control de Git a menos que:





1. Muevas esos archivos a la carpeta donde tienes inicializado Git.
2. Inicialices un nuevo repositorio Git en la carpeta en que están esos archivos.

Ahora Veamos un Ejemplo en que Ana y Luis Trabajan colaborativamente

1 Ana crea el repositorio inicial

1. **Iniciar un repositorio en su máquina local:** Ana empieza el proyecto creando un repositorio de Git en su computadora:

```
git init
```

2. **Agregar archivos al repositorio:** Ana añade todos los archivos de su proyecto al repositorio. Por ejemplo, supongamos que tiene un archivo llamado **análisis.py**.

```
git add análisis.py
```

3. **Hacer un commit inicial:** El commit guarda un punto específico en el historial de Git con todos los archivos que Ana ha añadido.

```
git commit -m "Primer commit: Estructura básica del análisis"
```

4. **Crear un repositorio en GitHub:** Ana crea un repositorio en GitHub desde la interfaz web de GitHub, llamándolo **proyecto_análisis**.
5. **Vincular el repositorio local con GitHub:** Ana conecta su repositorio local al remoto en GitHub:

```
git remote add origin https://github.com/Ana/proyecto_análisis.git
```

6. **Subir los cambios a GitHub:** Ana sube el proyecto a GitHub para que otros, como Luis, puedan trabajar en él:

```
git push -u origin master
```





2 Luis clona el repositorio de Ana y trabaja en él

1. **Clonar el repositorio:** Luis clona el repositorio de Ana en su máquina para obtener la misma copia del código:

```
git clone https://github.com/Ana/proyecto_analisis.git
```

2. **Navegar al directorio del proyecto:** Una vez clonado, Luis navega a la carpeta del proyecto en su computadora:

```
cd proyecto_analisis
```

3. **Activar su ambiente virtual y trabajar en el proyecto:** Luis crea y activa su propio ambiente virtual para trabajar en el proyecto sin interferir con otras configuraciones en su máquina:

```
python -m venv mi_entorno_luis  
mi_entorno_luis\Scripts\activate  
pip install numpy pandas matplotlib
```

4. **Luis hace cambios en el código:** Luis edita **análisis.py** y añade una nueva función que calcula estadísticas. Luego, agrega este archivo a su repositorio local:

```
git add análisis.py
```

5. **Hacer un commit de sus cambios:** Luis guarda sus cambios localmente con un commit:

```
git commit -m "Añadida función de cálculo de estadísticas"
```

6. **Subir sus cambios a GitHub:** Luis sube sus cambios al repositorio remoto en GitHub:

```
git push origin master
```



3 Ana actualiza su copia con los cambios de Luis

1. **Ana obtiene los cambios de Luis:** Ana, al darse cuenta de que Luis ha hecho cambios en GitHub, puede actualizar su copia del proyecto local:

```
git pull origin master
```

2. **Resolver conflictos:** Si Ana y Luis editaron la misma parte del código, Git puede detectar un conflicto. Ana resolverá el conflicto editando el archivo manualmente y luego hará un commit para guardar la resolución:

```
git add análisis.py  
git commit -m "Resuelto conflicto con la función de estadísticas"
```

Pregunta para Reflexionar: ¿Por qué es importante hacer commits frecuentemente y subirlos a GitHub cuando trabajas en equipo?





ANACONDA: GESTIÓN DE ENTORNOS Y PAQUETES

¿Qué es Anaconda?

Anaconda es una plataforma que facilita la creación y gestión de entornos virtuales, así como la instalación de paquetes de ciencia de datos. Es ampliamente utilizada por científicos de datos y desarrolladores.

Instalación de Anaconda desde la Consola de Windows

1. Descargar Anaconda:

- Ve a la página oficial de Anaconda y descarga la versión de Windows (<https://www.anaconda.com/download>)

2. Instalación desde el instalador:

- Abre el archivo descargado (normalmente un archivo .exe) y sigue las instrucciones del instalador. Asegúrate de marcar la opción que añade **Anaconda** a tu **PATH**, para poder usarlo desde la consola.

3. Verificar la instalación:

- Una vez completada la instalación, abre la consola de Windows y verifica la instalación con el siguiente comando:

```
conda --version
```

Principales productos de Anaconda

1. Jupyter Notebook

- **Jupyter Notebook** es una aplicación web que permite crear y compartir documentos que contienen código ejecutable, visualizaciones, y texto explicativo. Es muy utilizado para realizar análisis interactivos de datos, y su principal ventaja es que permite ejecutar código en celdas de manera incremental.

• Usos comunes:

- Visualización de datos.
- Documentación de análisis de datos.
- Prototipado de algoritmos de machine learning.





2. Conda

- **Conda** es el gestor de paquetes y entornos virtuales de Anaconda. Permite instalar paquetes y crear entornos virtuales de manera sencilla.
- **Comando**

```
conda create --name mi_entorno numpy pandas
```

3. Spyder

- **Spyder** es un entorno de desarrollo integrado (IDE) para Python que está enfocado en la ciencia de datos. Incluye un editor de código, consola interactiva y herramientas de depuración.

Veamos Cómo Usar Anaconda en el Proyecto Colaborativo de Ana y Luis

1. Luis crea un entorno con Anaconda: Luis puede usar Anaconda en lugar de venv para gestionar su entorno:

```
conda create --name entorno_analisis python=3.8
conda activate entorno_analisis
```

2. Instalar paquetes con conda:

```
conda install numpy pandas matplotlib
```

3. Usar Jupyter Notebook: Luis puede abrir Jupyter Notebooks para visualizar los datos:

```
jupyter notebook
```

4. Google Colab: Colaboración en la Nube

- Veamos como Ana y Luis también podrían usar **Google Colab** para trabajar en la nube sin instalar nada en sus máquinas.



Pasos para usar Google Colab

1. Subir el cuaderno a Google Colab: Ana crea un cuaderno (notebook) en Google Colab:
 - Accede a <https://colab.research.google.com>
 - Sube un archivo .ipynb o crea uno nuevo.
2. Conectar a Google Drive: Ana monta su Google Drive para acceder a los datos:

```
from google.colab import drive  
drive.mount('/content/drive')
```

3. Guardar y compartir el proyecto: Ana guarda el cuaderno en su Google Drive y lo comparte con Luis para que también colabore en Colab.

Pregunta para Reflexionar: ¿Qué ventajas ofrece Google Colab sobre el trabajo local en equipo?





EJEMPLO DE CONTROL DE VERSIONES CON GIT Y GITHUB: PROYECTO COLABORATIVO

Volvamos a analizar el proyecto colaborativo de **Ana** y **Luis**, en que trabajarán juntos en un análisis de datos, usando **Git** y **GitHub** para gestionar el proyecto y Google Colab para la parte de análisis.

Pasos del Proyecto:

1. Ana comienza el proyecto:

- Crea un repositorio en GitHub llamado **proyecto_analisis_datos**.
- Clona el repositorio en su máquina local:

```
git clone https://github.com/Ana/proyecto_analisis_datos.git
```

2. Ana configura el ambiente virtual:

- Crea un ambiente virtual:

```
python -m venv entorno_analisis
```

- Activa el ambiente:

```
entorno_analisis\Scripts\activate
```

- Instala las librerías necesarias:

```
pip install numpy pandas matplotlib
```

3. Ana agrega los archivos del proyecto:

- Crea el archivo **analisis.py** donde cargará y analizará los datos.
- Agrega el archivo al repositorio.

```
git add analisis.py
```



- Realiza un commit con un mensaje descriptivo:

```
git commit -m "Añadido el archivo de análisis inicial"
```

- Sube los cambios a GitHub

```
git push origin master
```

4. Luis colabora en el proyecto:

- Luis clona el repositorio de Ana

```
git clone https://github.com/Ana/proyecto_analisis_datos.git
```

5. Luis agrega nuevos análisis:

- Luis crea una nueva función en **analisis.py** que calcula el promedio de una columna de datos, luego añade y comete los cambios:

```
git add analisis.py
git commit -m "Añadido el cálculo de promedio"
git push origin master
```

6. Ana obtiene los cambios de Luis:

- Ana actualiza su repositorio local con los cambios de Luis:

```
git pull origin master
```

Ejemplo completo del archivo **analisis.py** con datos:

Datos de prueba: Vamos a trabajar con datos de un CSV de ejemplo llamado **datos_ventas.csv**:



Producto,Cantidad,Precio
Producto1,10,5.5
Producto2,20,7.5
Producto3,15,6.0

Código en analysis.py:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Cargar los datos
df = pd.read_csv('datos_ventas.csv')

# Mostrar los primeros registros
print("Primeros registros del dataset:")
print(df.head())

# Función para calcular el precio total
def calcular_precio_total(cantidad, precio):
    return cantidad * precio

# Aplicar la función al DataFrame
df['Precio_Total'] = df.apply(lambda row: calcular_precio_total(row['Cantidad'], row['Precio']), axis=1)

# Mostrar el DataFrame con el precio total
print("\nDataFrame con el Precio Total:")
print(df)

# Crear un gráfico de barras
plt.bar(df['Producto'], df['Precio_Total'])
plt.xlabel('Producto')
plt.ylabel('Precio Total')
plt.title('Precio Total por Producto')
plt.show()
```

Comentarios en el código:

1. Cargar los datos:

- Usamos `pd.read_csv` para cargar los datos desde un archivo CSV.
- La función `head()` nos permite ver los primeros registros del dataset.



2. Calcular el precio total:

- Definimos una función `calcular_precio_total` que toma la cantidad y el precio de cada producto y calcula el precio total.
- Luego aplicamos esta función a cada fila del DataFrame usando `apply()`.

3. Visualización con Matplotlib:

- Creamos un gráfico de barras para visualizar el precio total por producto.

Salida esperada del código:

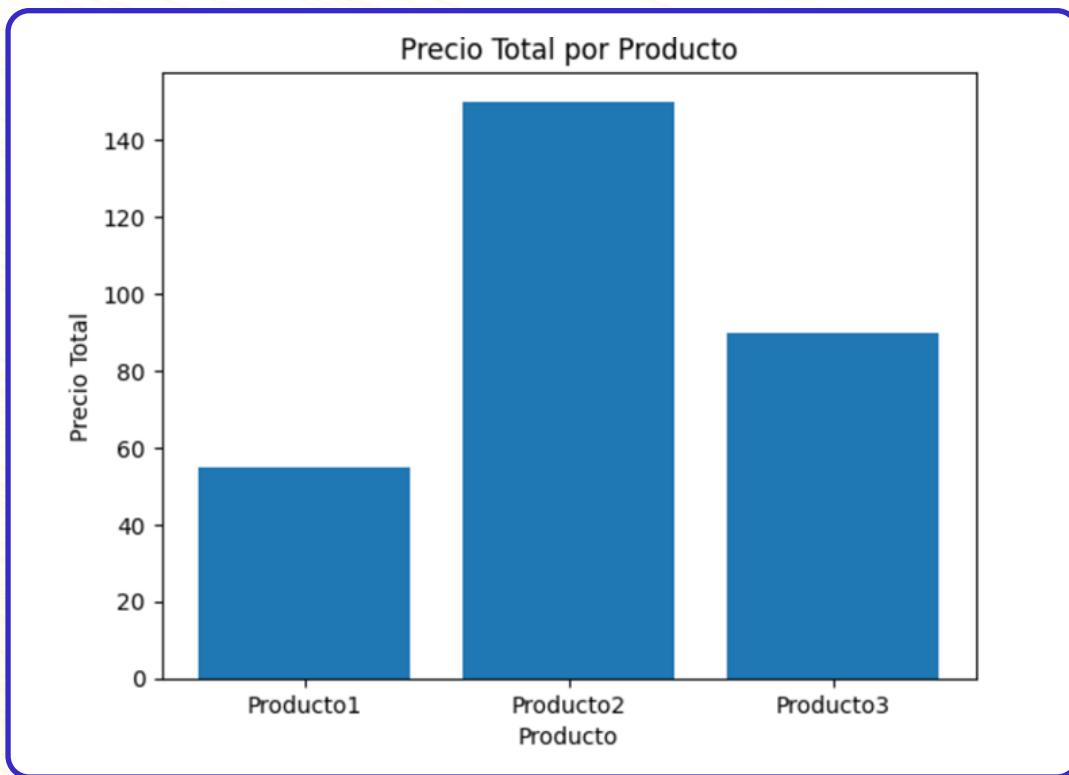
Primeros registros del dataset:

	Producto	Cantidad	Precio
0	Producto1	10	5.5
1	Producto2	20	7.5
2	Producto3	15	6.0

DataFrame con el Precio Total:

	Producto	Cantidad	Precio	Precio_Total
0	Producto1	10	5.5	55.0
1	Producto2	20	7.5	150.0
2	Producto3	15	6.0	90.0

El gráfico de barras mostrará la cantidad total por producto.



7. Colaboración en Google Colab:

- Ana y Luis pueden continuar el análisis en Google Colab, trabajando de manera remota sin necesidad de instalar nada.

Pasos para Colaborar en Google Colab:

1. Ana sube el análisis a Google Colab:

- Ana crea un cuaderno en Google Colab y monta su Google Drive para acceder a los datos:

```
from google.colab import drive  
drive.mount('/content/drive')
```

2. Luis accede al cuaderno compartido:

- Ana comparte el cuaderno de Google Colab con Luis, quien puede acceder, hacer cambios y guardarlos automáticamente en Google Drive.



3. Visualización de datos en Google Colab:

- Ambos pueden visualizar los datos y gráficos sin necesidad de configuraciones locales, solo utilizando el navegador.

Qué sucede si Luis o Ana agregan nuevos datos al proyecto

Imaginemos que **Ana** ha conseguido nuevos datos de ventas y quiere agregarlos al archivo **datos_ventas.csv** que ya existe en el repositorio. A continuación, veremos cómo lo haría y cómo **Luis** puede actualizar su copia local para trabajar con esos nuevos datos.

1. Ana agrega nuevos datos

Ana ha recibido más datos de ventas para agregar al archivo **datos_ventas.csv**. Los nuevos datos son:

```
Producto,Cantidad,Precio
Producto4,25,8.0
Producto5,30,9.0
```

Pasos para agregar los nuevos datos:

1. Actualizar el archivo de datos:

- Ana abre el archivo **datos_ventas.csv** y agrega las nuevas filas de datos. El archivo actualizado ahora se ve así:

Nuevo archivo datos_ventas.csv:

```
Producto,Cantidad,Precio
Producto1,10,5.5
Producto2,20,7.5
Producto3,15,6.0
Producto4,25,8.0
Producto5,30,9.0
```



2. Guardar los cambios y hacer un commit:

- Ana guarda el archivo y lo añade al repositorio de Git usando los siguientes comandos:

```
git add datos_ventas.csv  
git commit -m "Agregados nuevos datos de ventas para Producto4 y Producto5"
```

3. Subir los cambios a GitHub

- Ana sube el archivo actualizado a GitHub:

```
git push origin master
```

2. Luis obtiene los nuevos datos de Ana

Luis necesita los nuevos datos para continuar trabajando en su copia del proyecto. Aquí están los pasos que debe seguir para obtener los cambios realizados por Ana.

Pasos para actualizar su copia local:

1. Actualizar la copia local de Luis:

- Luis debe asegurarse de estar en la rama correcta (master) y luego ejecutar el siguiente comando para obtener los cambios de Ana:

```
git pull origin master
```

Esto descargará el archivo **datos_ventas.csv** actualizado que Ana subió a GitHub.

2. Revisar los nuevos datos en su copia local:

- Ahora Luis puede abrir el archivo **datos_ventas.csv** y ver que se han agregado los productos **Producto4** y **Producto5**.

3. Ajustar el código para manejar los nuevos datos

Ahora que ambos tienen los nuevos datos, necesitan ajustar su código en **analisis.py** para incluir los nuevos productos en el análisis. Aquí está el código actualizado:





```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Cargar los datos actualizados
df = pd.read_csv('datos_ventas.csv')

# Mostrar los primeros registros
print("Primeros registros del dataset actualizado:")
print(df.head())

# Función para calcular el precio total
def calcular_precio_total(cantidad, precio):
    return cantidad * precio

# Aplicar la función al DataFrame
df['Precio_Total'] = df.apply(lambda row: calcular_precio_total(row['Cantidad'], row['Precio']), axis=1)

# Mostrar el DataFrame con el precio total
print("\nDataFrame con el Precio Total actualizado:")
print(df)

# Crear un gráfico de barras con los nuevos datos
plt.bar(df['Producto'], df['Precio_Total'])
plt.xlabel('Producto')
plt.ylabel('Precio Total')
plt.title('Precio Total por Producto (Datos Actualizados)')
plt.show()
```

Comentarios en el código actualizado:

1. Carga de los datos actualizados:

- El código ahora trabaja con el archivo **datos_ventas.csv** actualizado, que incluye los productos 4 y 5.

2. Cálculo y visualización de los nuevos datos:

- El cálculo del **Precio Total** sigue siendo el mismo, pero ahora incluirá los nuevos productos en el análisis y en el gráfico de barras.



Salidas:

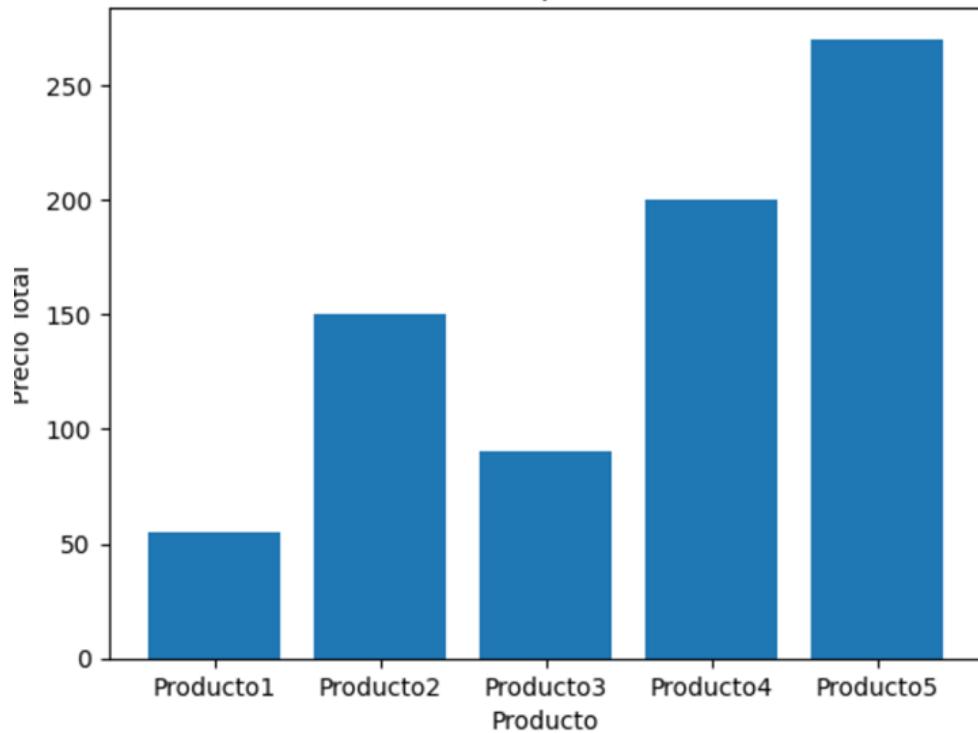
Primeros registros del dataset:

	Producto	Cantidad	Precio
0	Producto1	10	5.5
1	Producto2	20	7.5
2	Producto3	15	6.0
3	Producto4	25	8.0
4	Producto5	30	9.0

DataFrame con el Precio Total:

	Producto	Cantidad	Precio	Precio_Total
0	Producto1	10	5.5	55.0
1	Producto2	20	7.5	150.0
2	Producto3	15	6.0	90.0
3	Producto4	25	8.0	200.0
4	Producto5	30	9.0	270.0

Precio Total por Producto





4. ¿Qué pasa si ambos editan el archivo al mismo tiempo?

Si tanto **Ana** como **Luis** editan el archivo **datos_ventas.csv** al mismo tiempo, **Git** detectará un **conflicto** cuando intenten hacer un **push**. A continuación, veremos cómo resolver este conflicto.

Pasos para resolver un conflicto de Git:

1. Luis intenta hacer un push:

- Después de hacer cambios en el archivo **datos_ventas.csv**, Luis intenta hacer un push, pero recibe un mensaje de error:

git push origin master

Error:

```
! [rejected] master -> master (fetch first)
error: failed to push some refs to 'https://github.com/Ana/proyecto_analisis_datos.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
```

2. Resolver el conflicto con un pull:

- Luis necesita primero descargar los cambios de Ana antes de hacer su propio push:

git pull origin master

3. Git detecta un conflicto:

- Git detectará un conflicto si Luis y Ana han editado el mismo archivo en diferentes líneas. Luis verá algo como esto en el archivo **datos_ventas.csv**:

```
<<<<< HEAD
Producto3,15,6.0
=====
Producto3,20,6.0
>>>>> origin/master
```



4. Resolver el conflicto manualmente:

- Luis debe decidir cómo combinar los cambios. Si acepta el cambio de Ana, el archivo debería verse así:

```
Producto,Cantidad,Precio
Producto1,10,5.5
Producto2,20,7.5
Producto3,15,6.0
Producto4,25,8.0
Producto5,30,9.0
```

5. Guardar y hacer un commit del archivo resuelto:

- Una vez que el conflicto se ha resuelto, Luis debe guardar el archivo y hacer un nuevo commit:

```
git add datos_ventas.csv
git commit -m "Resuelto conflicto con los datos de Producto3"
```

6. Finalmente, hacer el push:

- Ahora Luis puede subir sus cambios a GitHub:

```
git push origin master
```